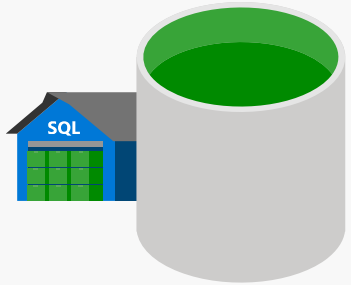
Azure SQL Data Warehouse



Contents

- Overview
- Architecture
- Scaling
- Schema Design
- Loading
- Querying
- High-Availability
- Backup & Disaster Recovery
- Monitoring, Management
- Security
- Sizing
- Migration
- Best Practices
- Pricing

Azure SQL Data Warehouse

Overview

Azure SQL Data Warehouse

Enables
transformative insights

Near real-time analytics over structured and unstructured data

Use tools you know and love; Excel Power BI, Tableau, Qlik or 20 others

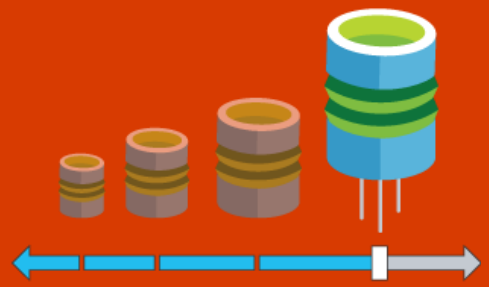
Seamless compatibility with Azure ingestion, machine learning, and Big Data services

Scales in minutes
with more freedom

Easily scale your workload in minutes

Decoupled storage and compute; pay for only the compute you use

Innovative "Pause" feature to stop compute and save even more



Helps secure
and protect

Multiple layers of data protection

Encryption at rest, in motion or in use

Tools to monitor data security

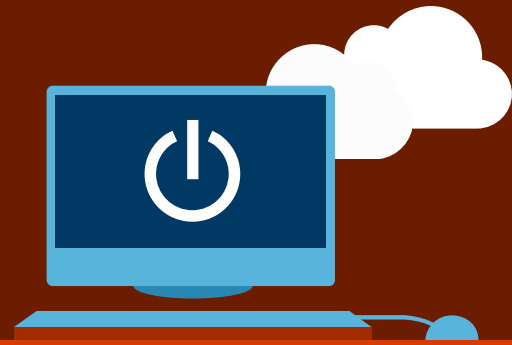
Protect user access to data with Azure Active Directory

Gets you up
& running quickly

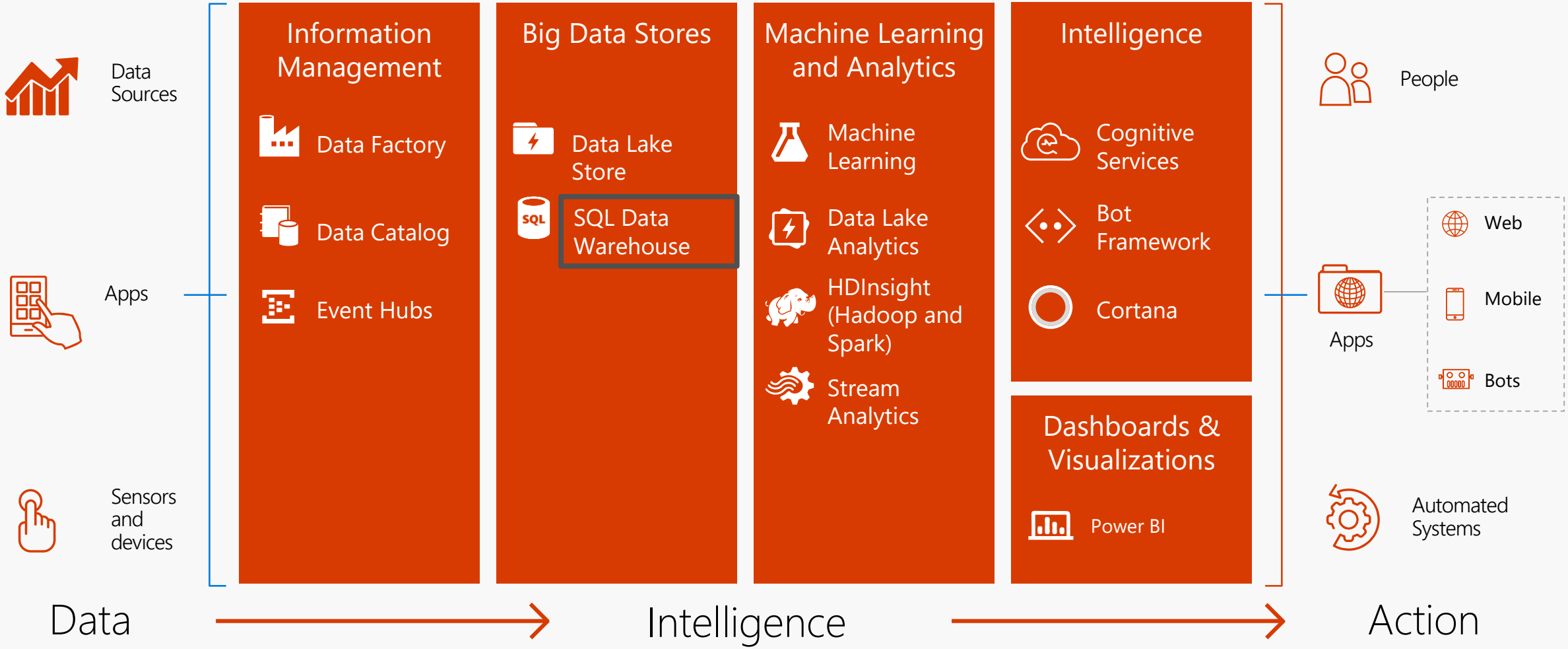
A single SQL-based view across data sources

Seamlessly ingest data from on-premises, cloud, devices, and apps

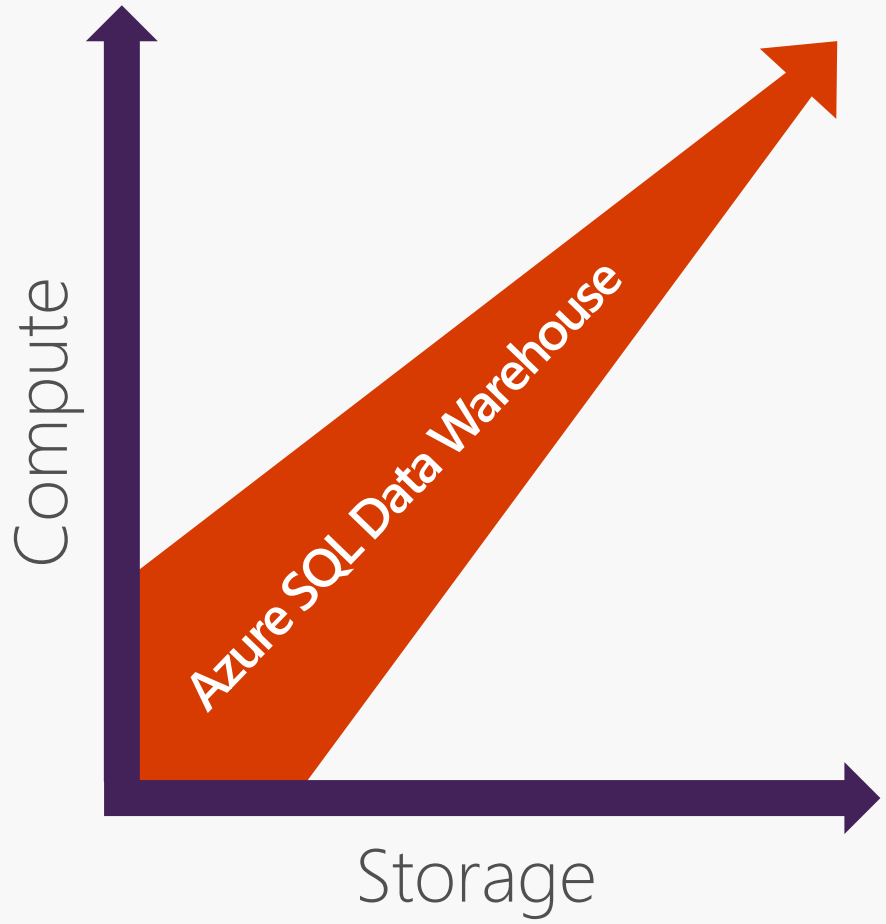
Built on SQL Server technology using tools you know love SSMS and Visual Studio SSDT

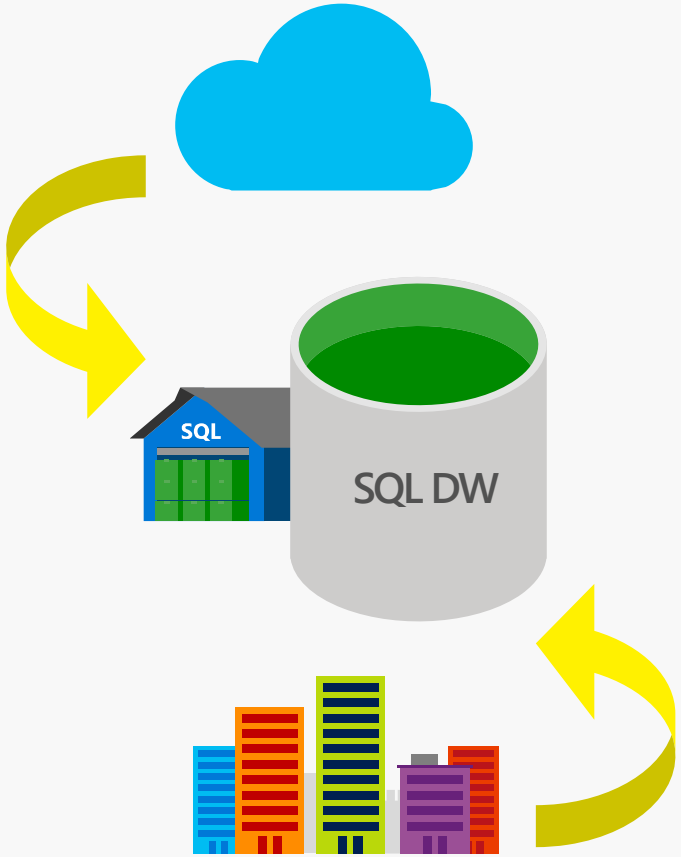


Cortana Intelligence Suite



Key differentiators



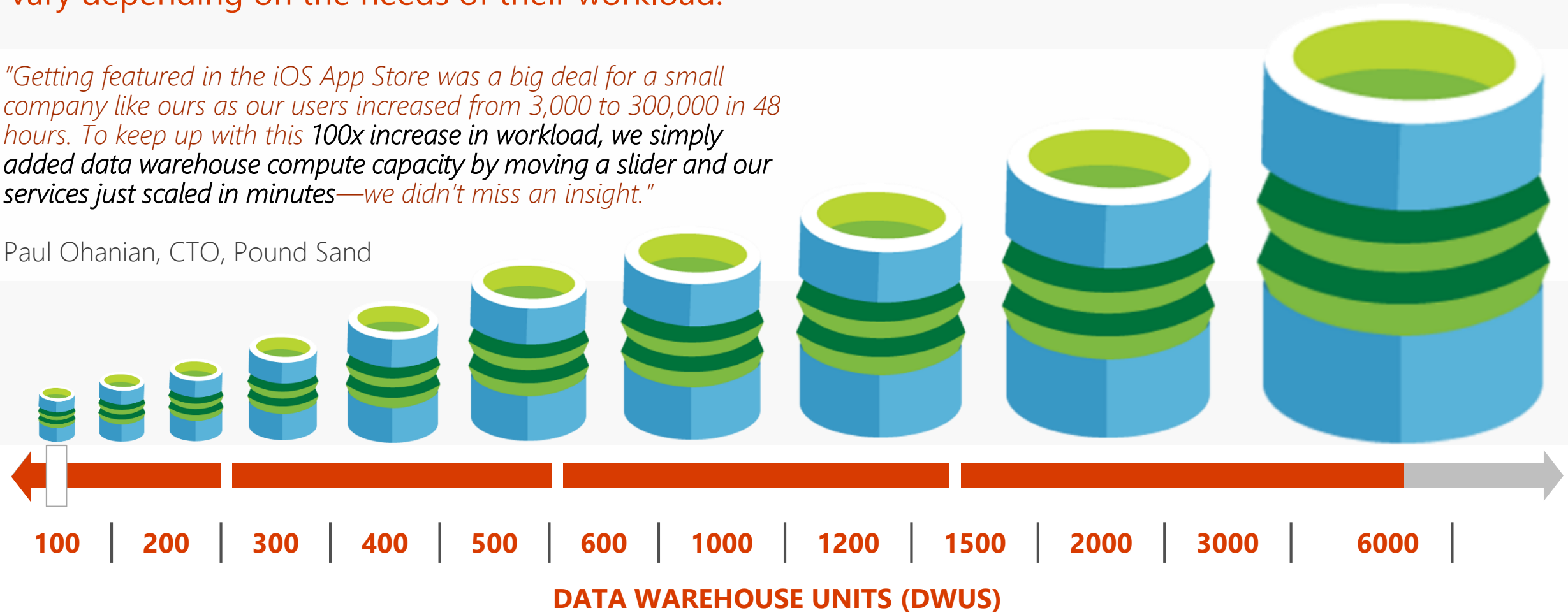


Scale compute on the fly when you need it

Data Warehouse Units (DWUs) are a measure of reserved compute performance or 'power.' A customer's DWU needs can vary depending on the needs of their workload.

"Getting featured in the iOS App Store was a big deal for a small company like ours as our users increased from 3,000 to 300,000 in 48 hours. To keep up with this ***100x increase in workload, we simply added data warehouse compute capacity by moving a slider and our services just scaled in minutes****—we didn't miss an insight."*

Paul Ohanian, CTO, Pound Sand



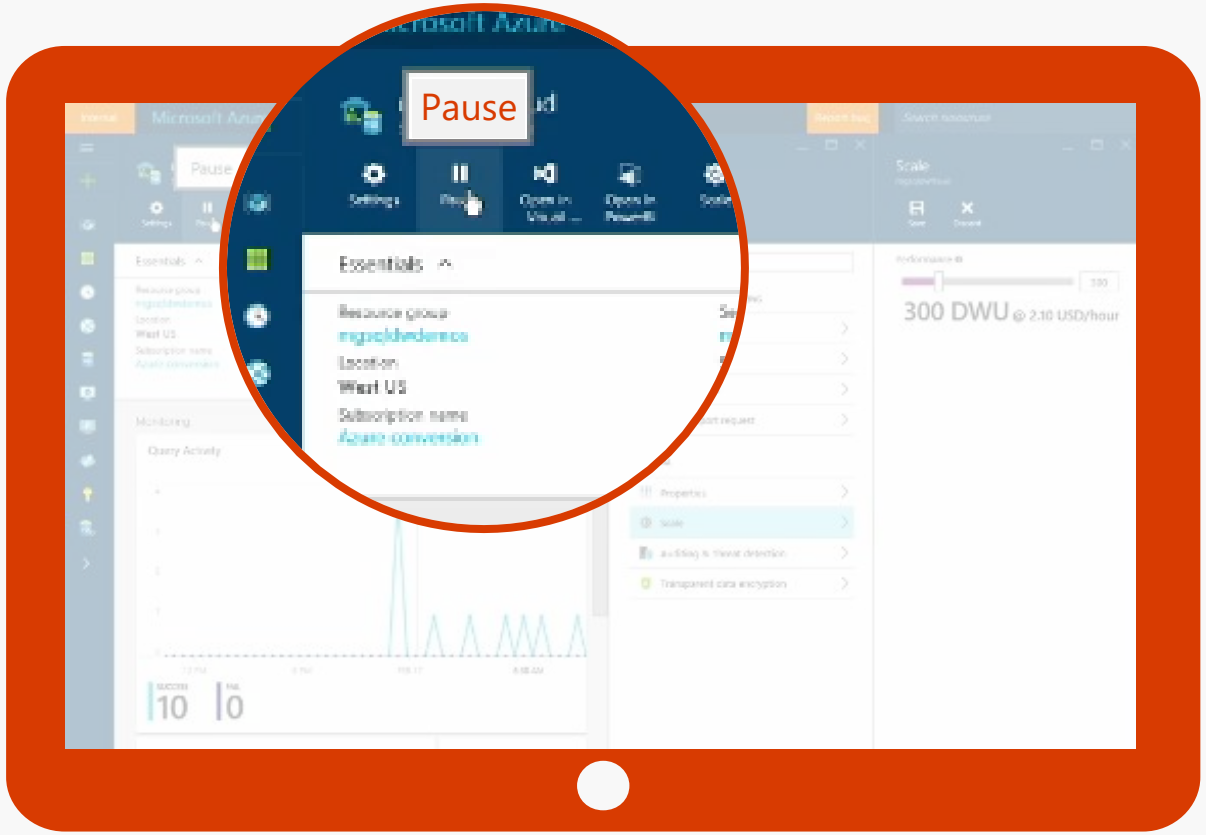
…pause compute when you don't need it

No need to over-provision or over-pay

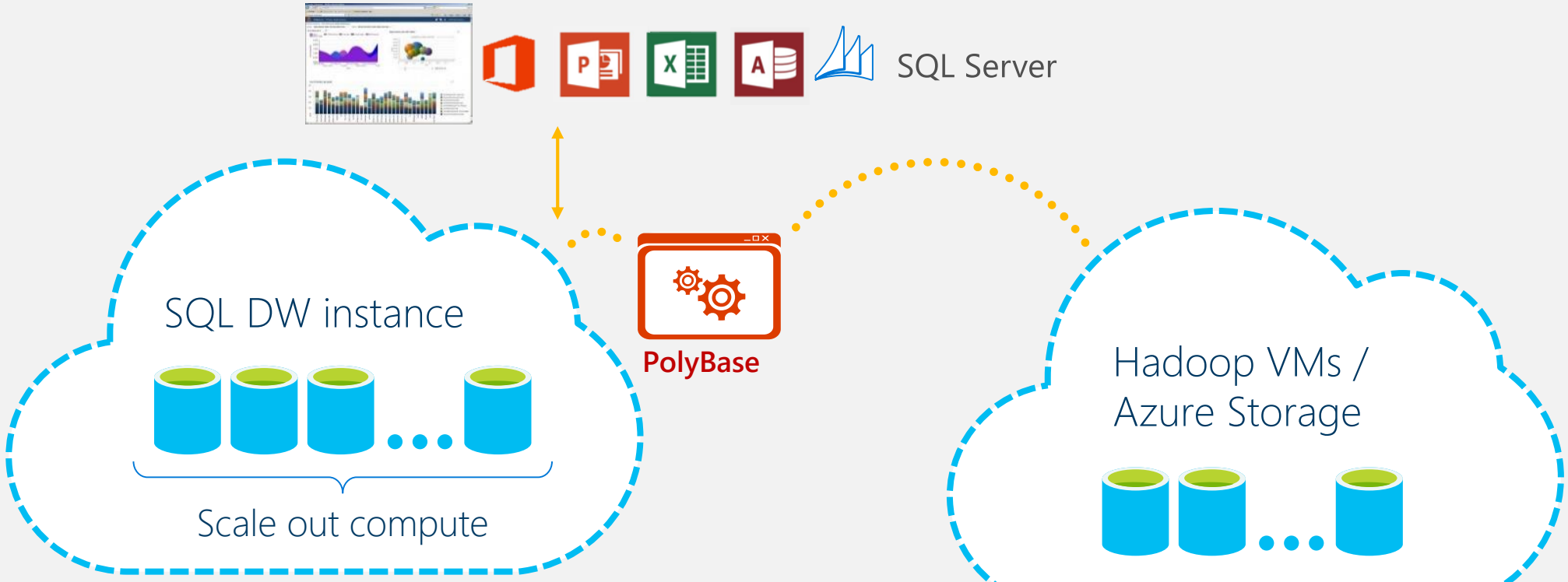
More freedom to right-size your environment

Bring all your data with no tradeoffs on what data to ingest - run compute only on the datasets that matter

"When we learned about the pause and resume capabilities of SQL Data Warehouse, ***we switched from AWS Redshift, migrating over 25TB of uncompressed data over a week for the simple reason of saving money****. To meet our business intelligence requirements, we load data once or twice a month and then build reports for our customers. Not having the data warehouse service running all the time is key for our business and our bottom line."*
Bill Sabo, Integral Analytics.



Query unstructured data via PolyBase/T-SQL

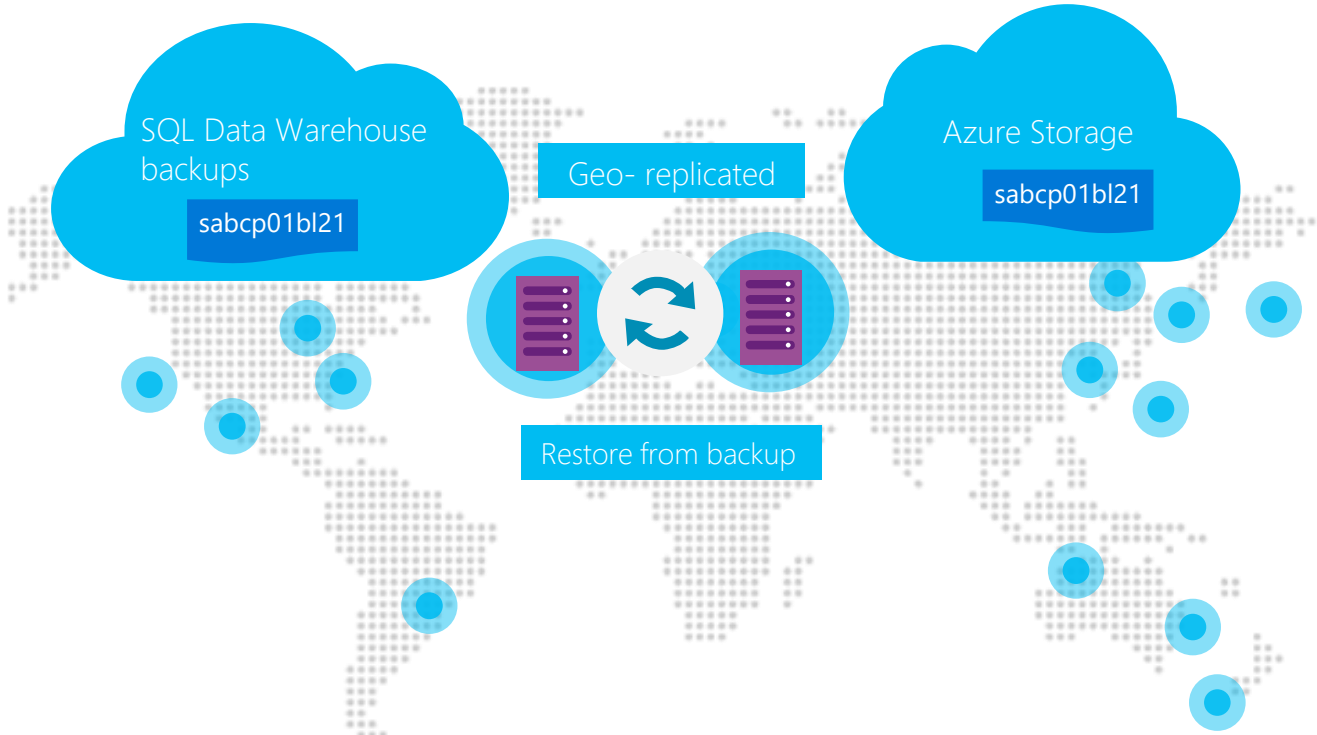


Any data, any size, anywhere

Automatic backup and geo-restore

Recover from data deletion or alteration or disaster

- Auto backups, every 4 to 8 hours
- Restore using REST API, PowerShell or Azure Portal
- Local and geo-redundant backups
- Backups retention policy:
  - Auto backup retention for 7 days



Auditing and Threat Detection

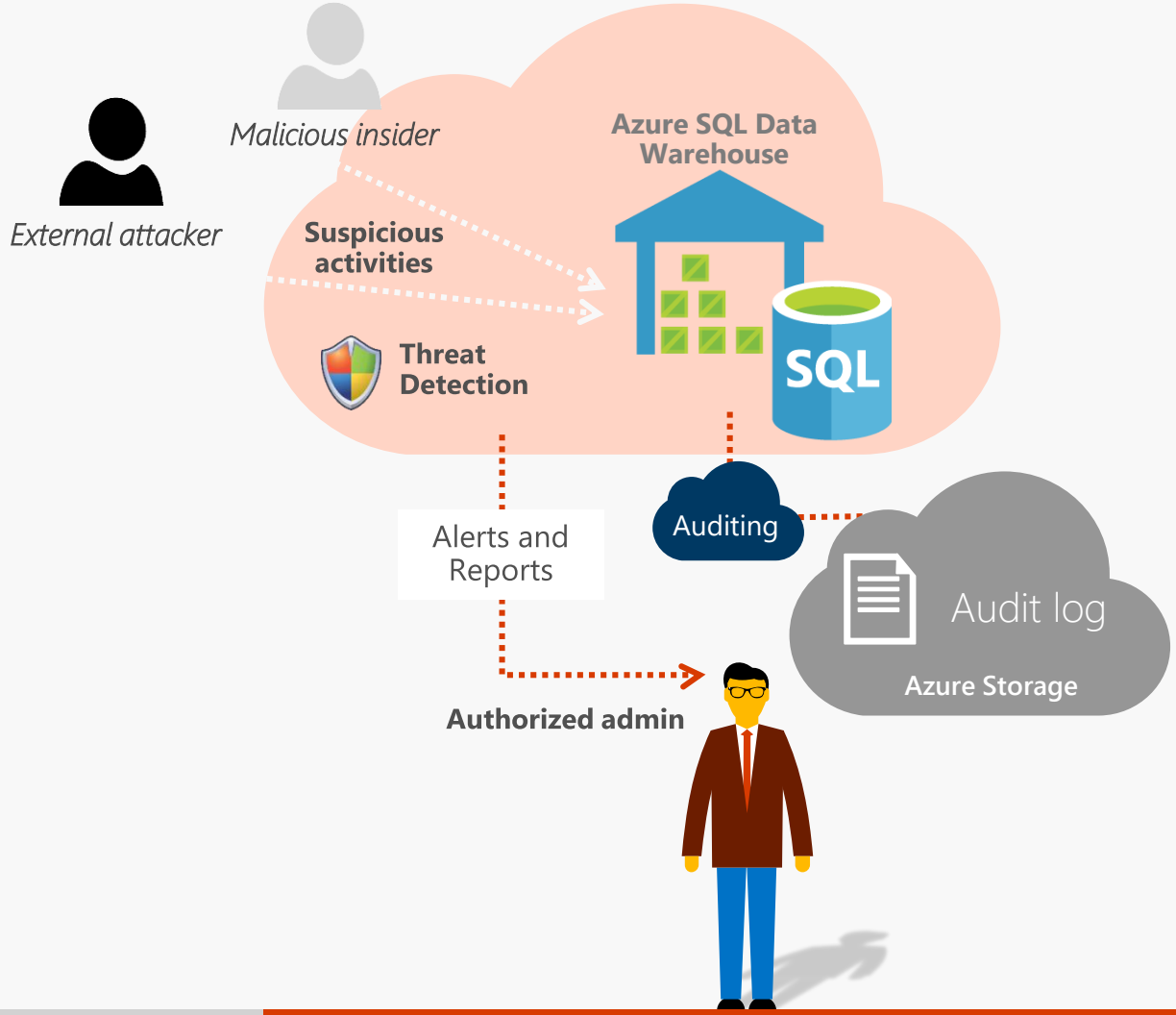
Gain real-time insights and streamline compliance-related tasks

RETAIN
an audit trail of selected events and activities

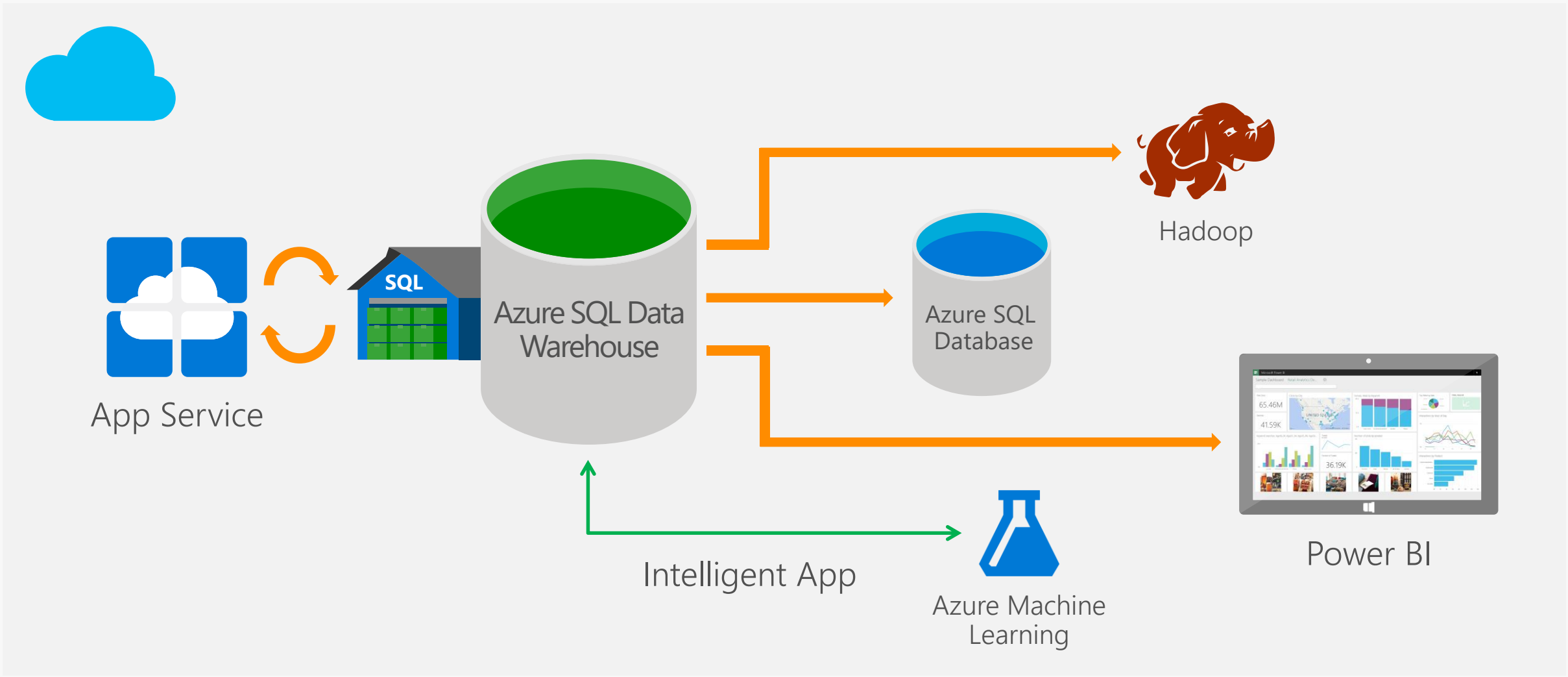
REPORT
on database activity—preconfigured reports and a dashboard help get you started quickly

ANALYZE
reports to find suspicious events, unusual activities, and trends

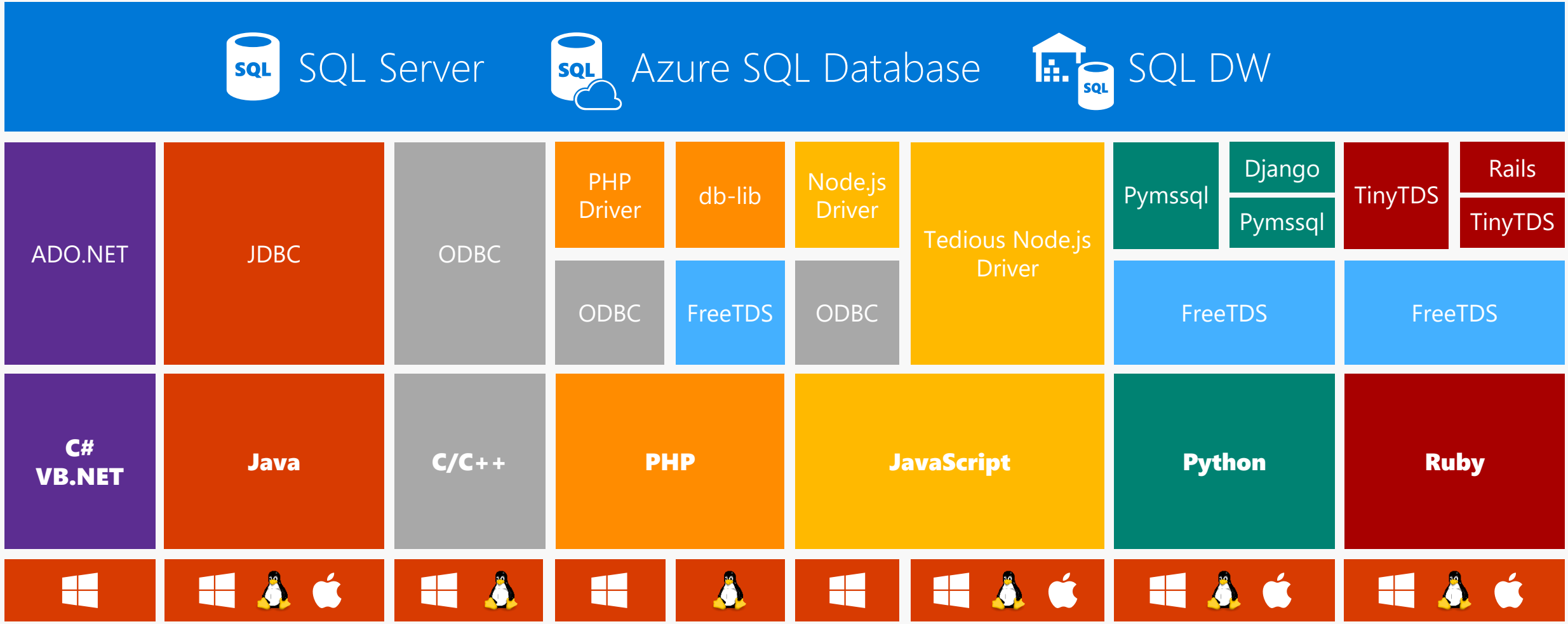
RECEIVE PROACTIVE ALERTS
about activities that might indicate potential security threats using the new Threat Detection feature



End to end platform built for the cloud



Your choice of language and tooling



SQL DW – Fit for Purpose

- Designed for **DW and not OLTP**. All the traditional DW workload characteristics apply.
- **Not good for singleton DML** heavy operations, Example: Clients issuing singleton update, insert, delete.
- Incremental data is loaded regularly by ETL/ELT process in **batch mode**. Not optimal for real time ingestion.
- **Low number of concurrent queries.**
- Where data runs in **Millions, Billions** of rows and **TBs** in Size.
- Main performance is driven from **Massively parallel processing Architecture.**

Azure SQL Data Warehouse

Architecture

MPP architecture - Concepts

- ***M**assively **P**arallel **P**rocessing*
- *A divide and conquer, **distributed computing** strategy*
- ***Shared Nothing** architecture*
- *Scale-out architecture (versus scale-up)*
- *Take one big problem & break it up & execute it individually*
- *Team approach "Many hands make light work"*
- *Multiple machines (RAM, CPU, Storage) with high bandwidth for single task*
- *Codeless parallel processing*
- *Best suited for large data processing*
- *Out of the box -*
  - Task Scheduling, Distribution, Communication & Co-ordination
  - High Availability, Fault Tolerance

MPP architecture - Concepts

Logical layer

- Holds application metadata, Does not persist application data
- Receives intermediate results and performs final aggregation

Physical layer

- Persists application data
- Performs query steps as instructed

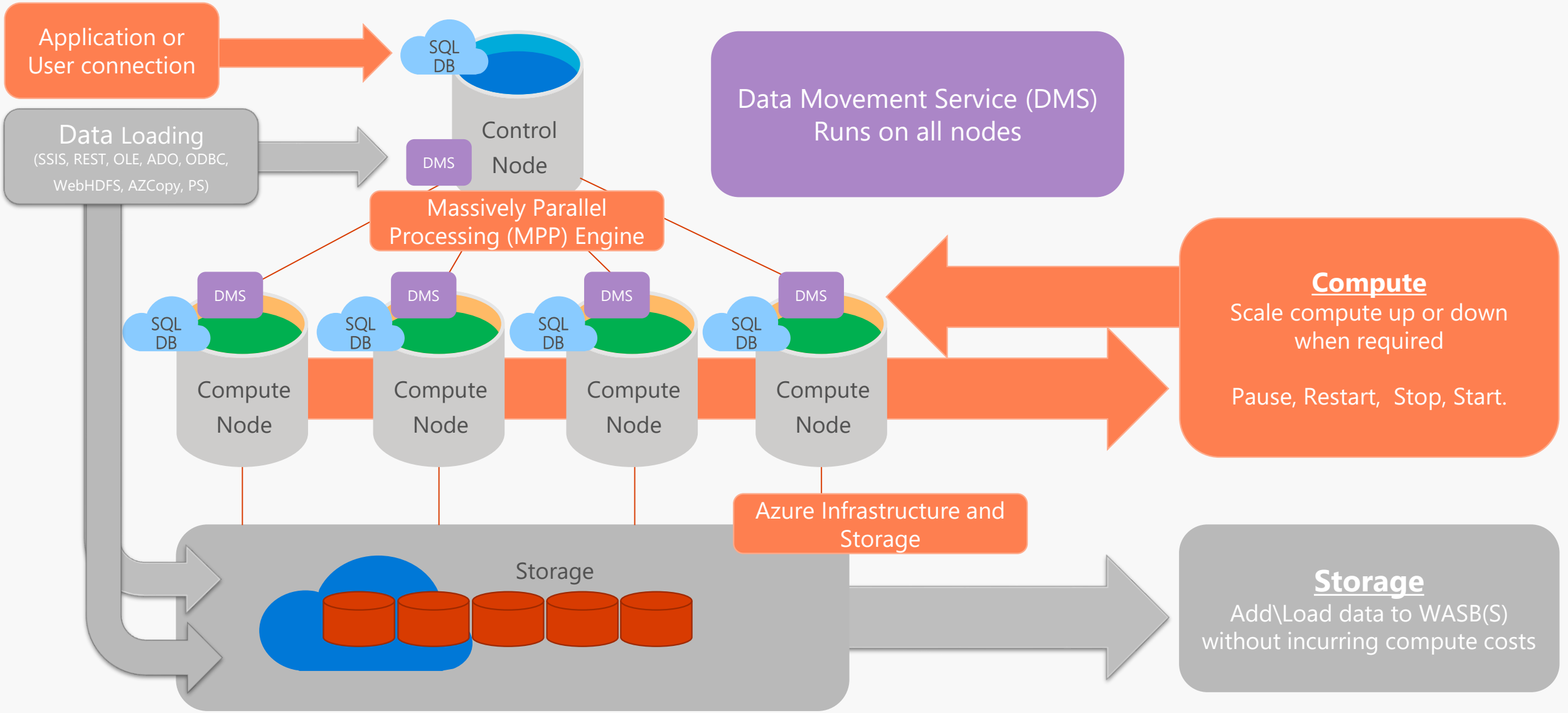
Distributed Query engine

- Parallel processing coordination

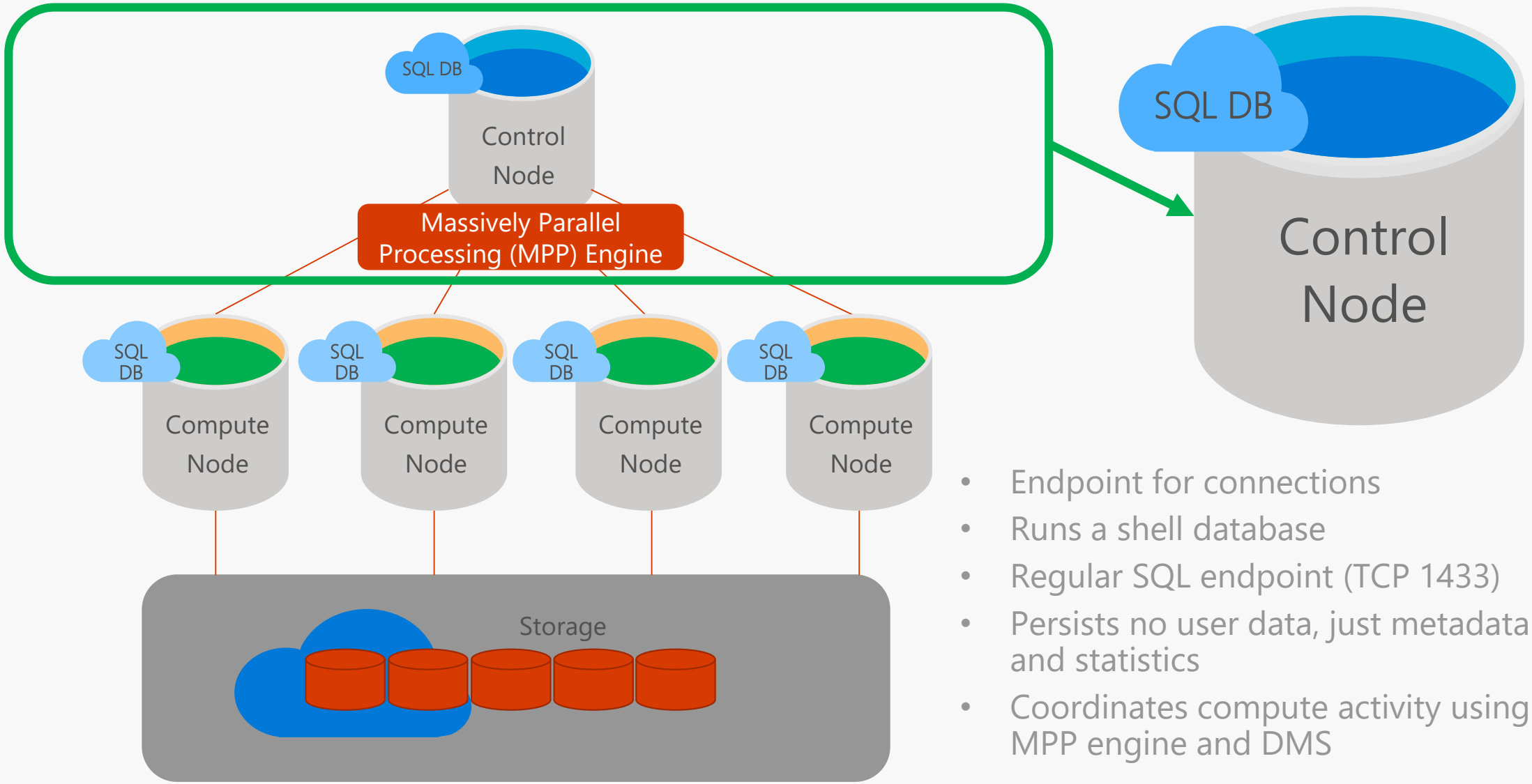
Lots of machines!

- Lots of small databases running in parallel

Azure SQL Data Warehouse Architecture



Azure SQL Data Warehouse – Control Node



- Endpoint for connections
- Runs a shell database
- Regular SQL endpoint (TCP 1433)
- Persists no user data, just metadata only and statistics
- Coordinates compute activity using MPP engine and DMS

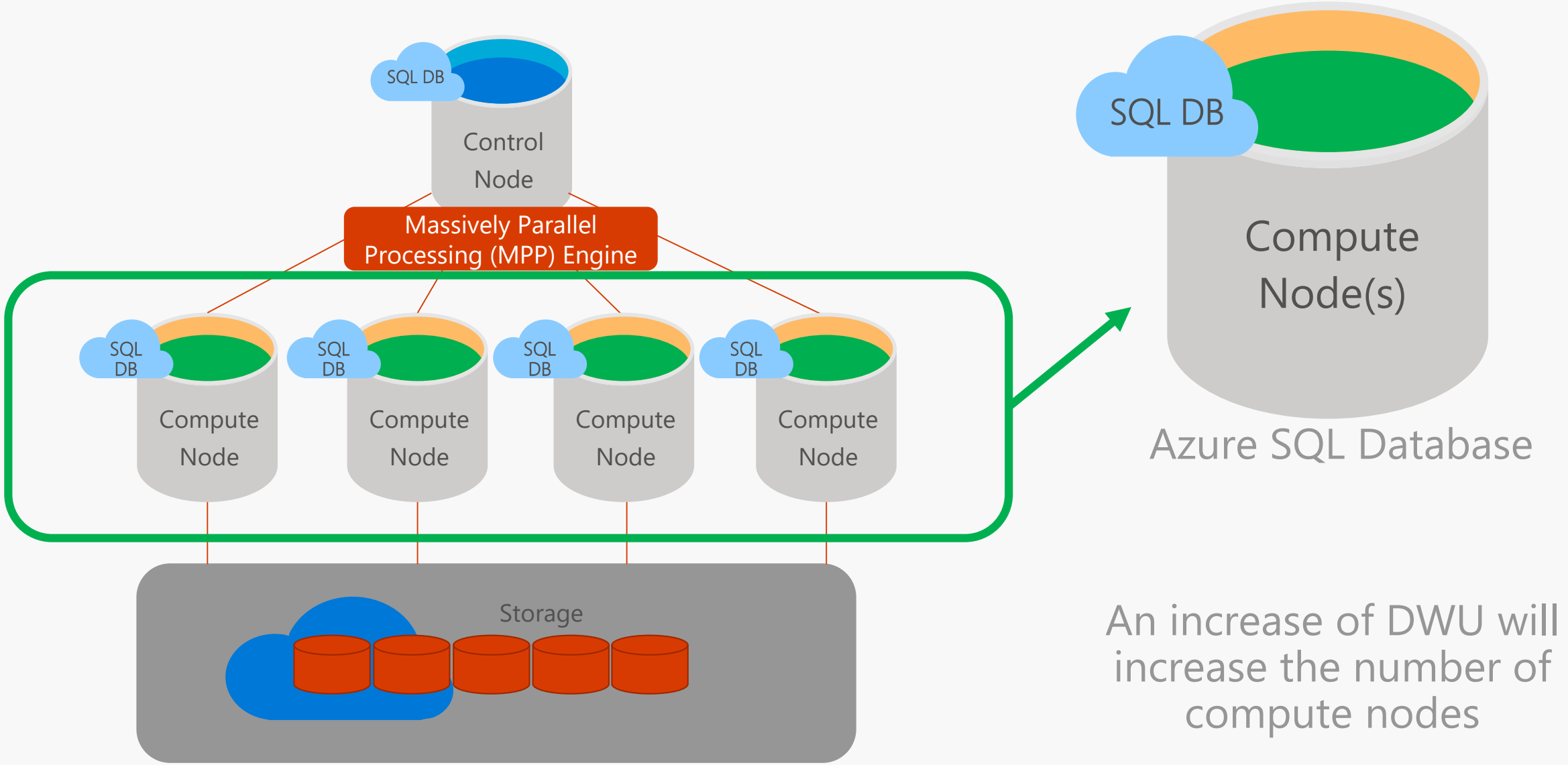
Control node

- Accepts requests from user and transforms it into separate queries that run on each compute node in parallel
- Coordinates all of the data movement and computation required to run parallel queries
- Interprets requests for scale-out
- Orchestrates actions / steps
- Final computation
- Returns result

✓ Runs SQL Server
✓ Stores metadata, statistics
✓ Performs final computation

Requires distributed query engine

Azure SQL Data Warehouse - Compute Nodes



An increase of DWU will increase the number of compute nodes

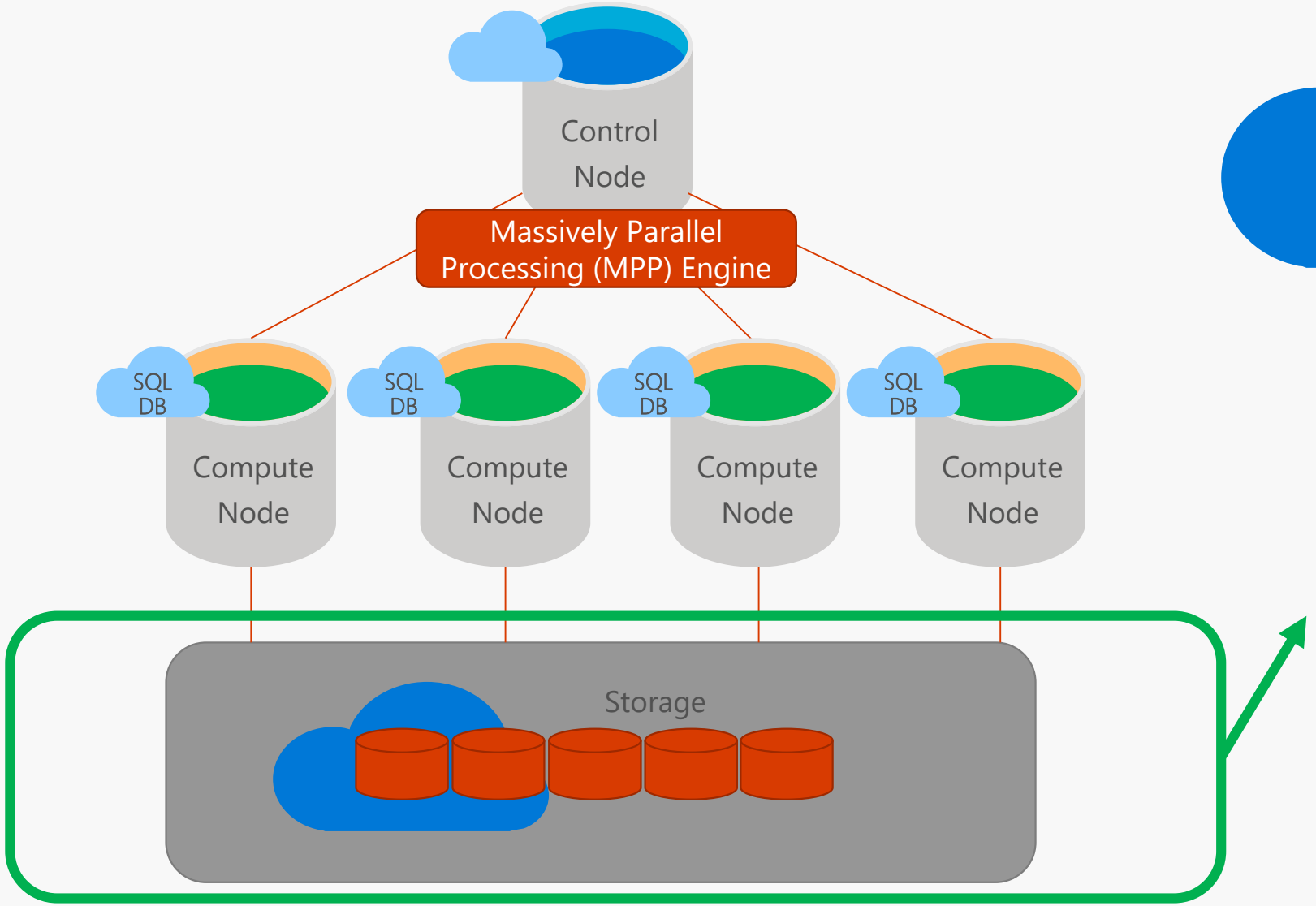
Compute node

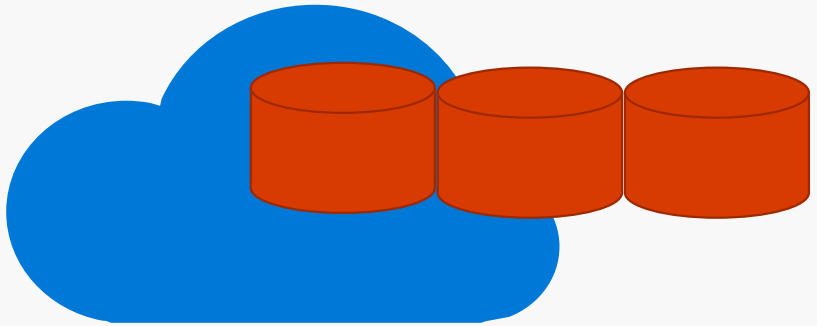
- Power behind the SQL DW; Performs the heavy lift
  - **Accepts requests from control node**
- Is a highly tuned SMP System
  - **Optimises requests from control node**
- SQL Databases that store data
- Workers that run parallel queries on data

No direct user interaction

Uses SQL Server

Azure SQL Data Warehouse – premium storage





- Premium RA-GRS storage
- SSD-based
- Better latency
- Reliable I/O – 400 MB/sec per node
- Persist the data without incurring compute costs

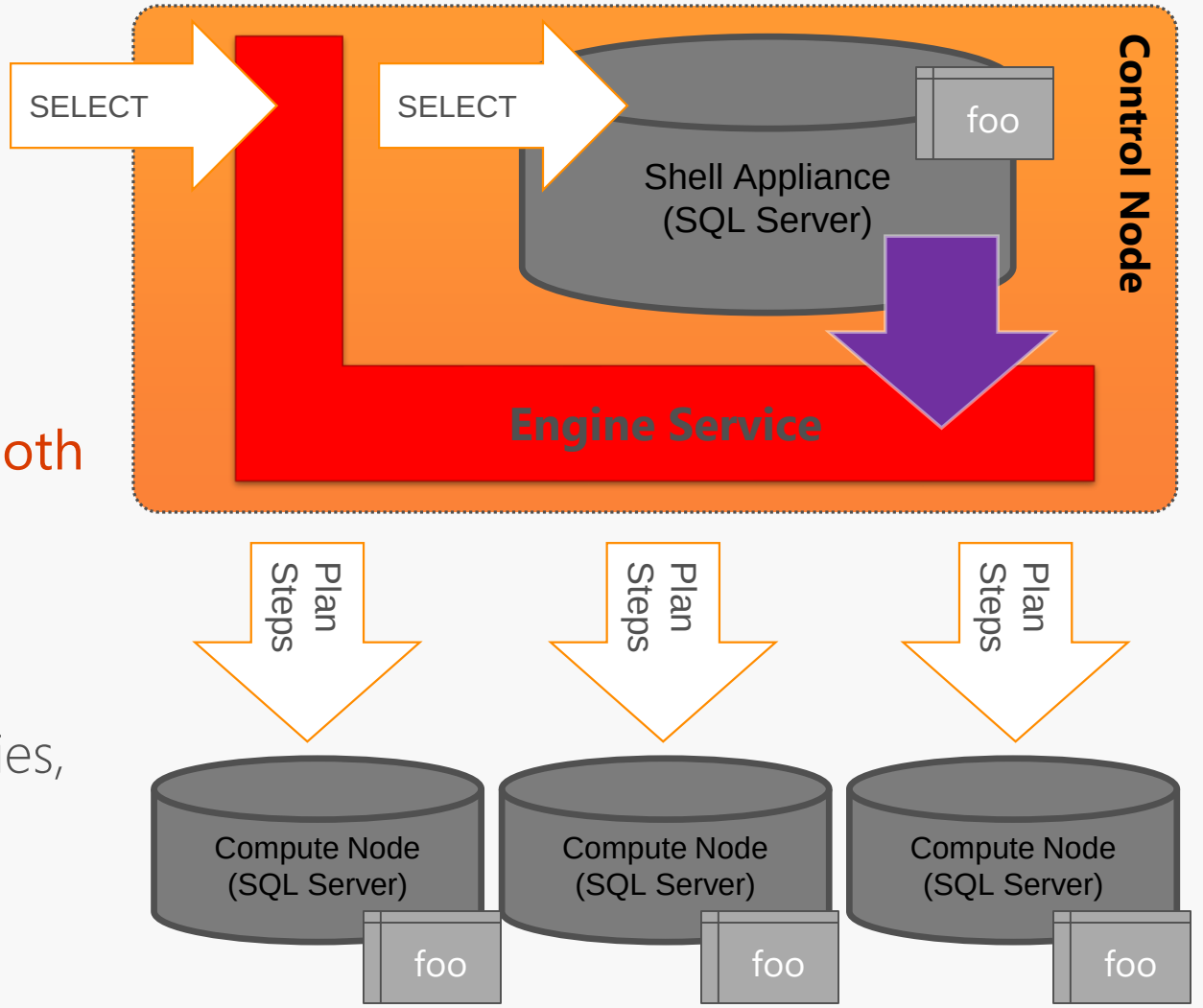
SQL DW Storage - Page blobs

- Azure SQL DW is backed by SSD based page blobs
- Page blobs – optimized for representing IaaS disks
- Supports random writes, up to 1 TB in size
- Premium page blobs – optimized for I/O intensive workloads
- Premium page blob is LRS (Locally Redundant Storage) only

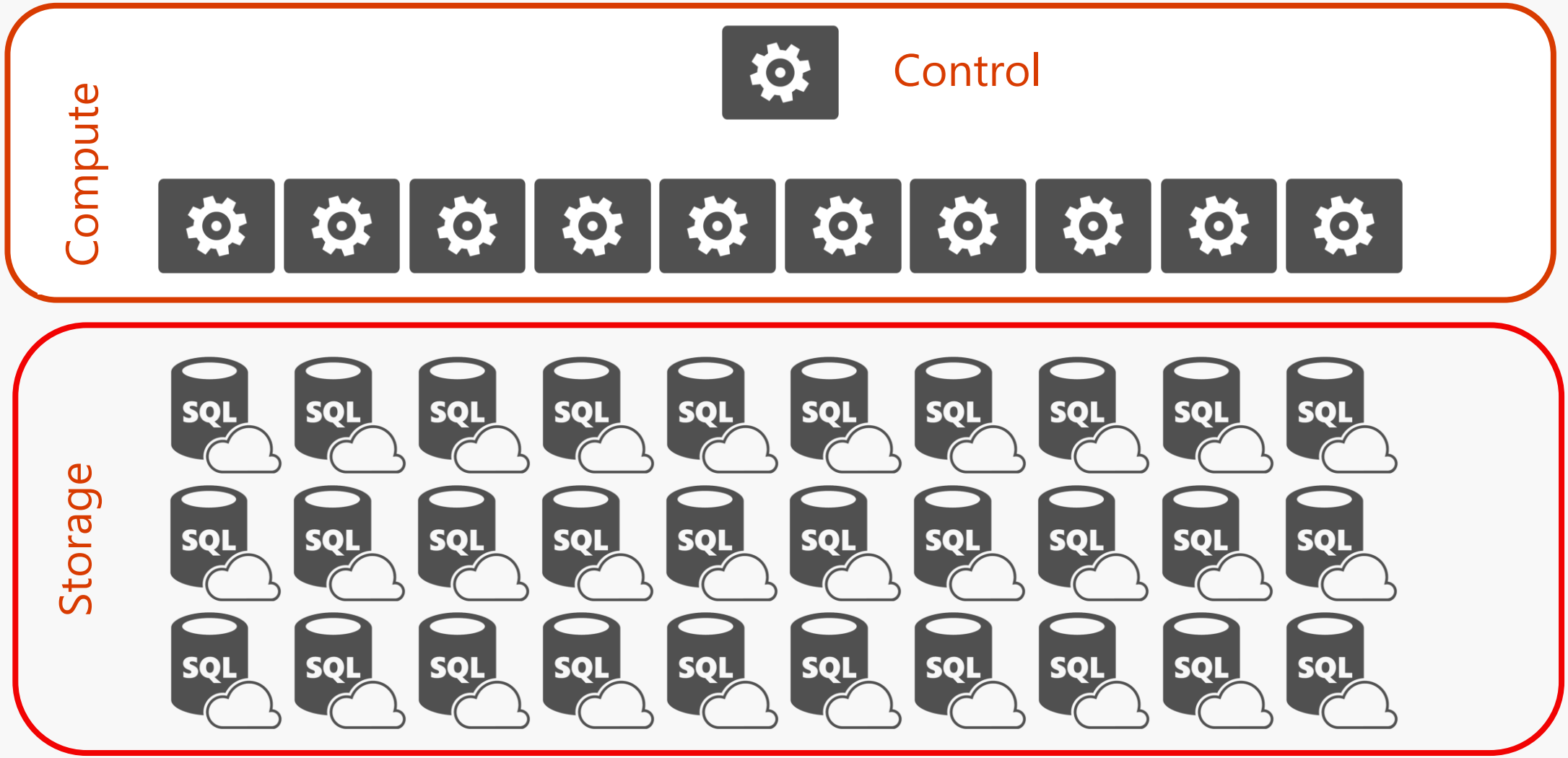
- *https://docs.microsoft.com/en-us/rest/api/storageservices/fileservices/understanding-block-blobs--append-blobs--and-page-blobs*

SQL DW Control Architecture

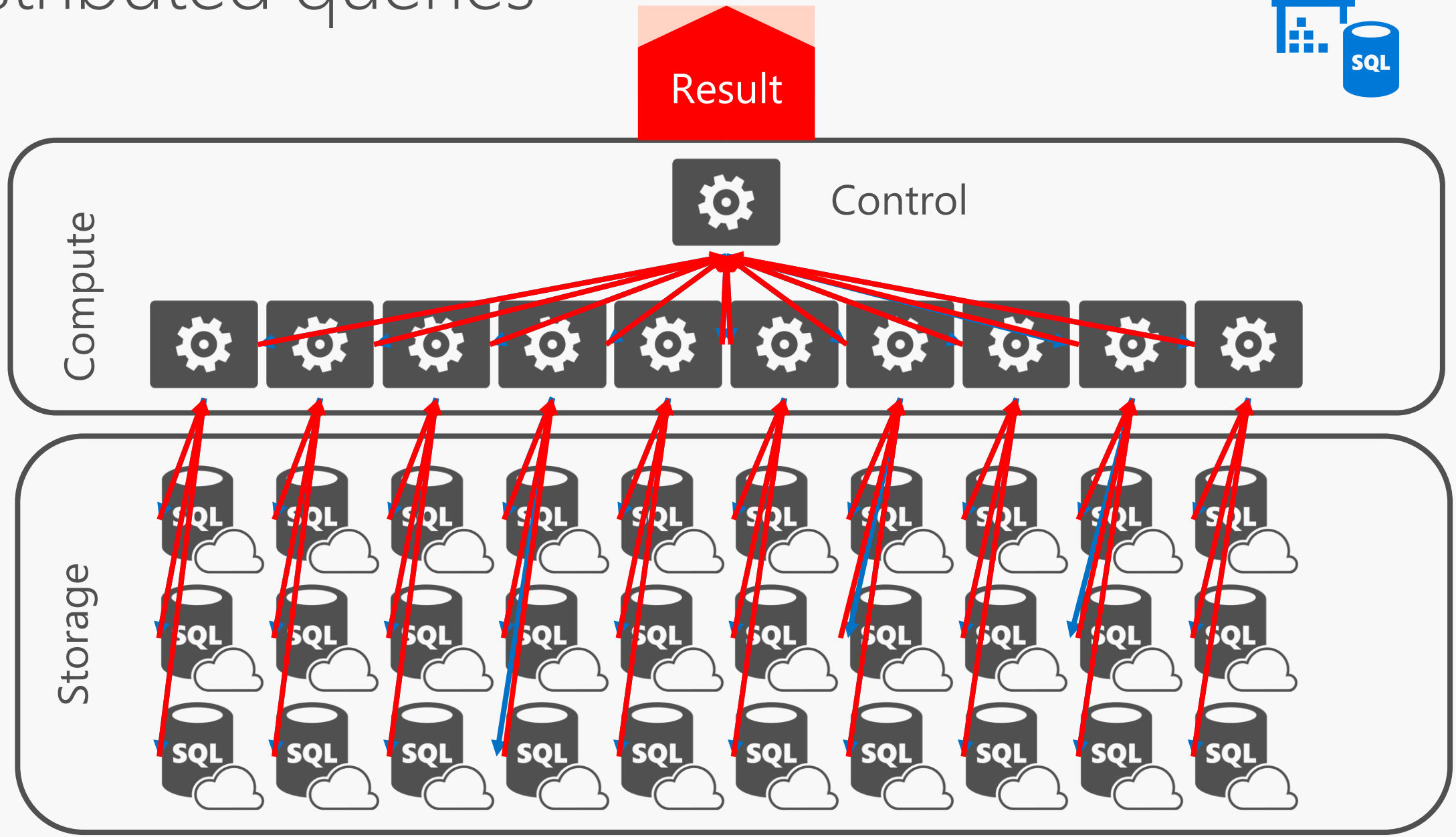
- SQL DW uses SQL Server on the Control node to run a "shell appliance"
- Every database with all its objects exists in the shell appliance as an empty "shell," lacking the user data (which sits on all the compute nodes)
- Every DDL operation is executed against both the shell and the compute nodes
- Large parts of basic RDBMS functionality provided by that shell
  - Authentication and authorization of queries, but also the full security system
  - Schema binding
  - Metadata catalog



Logical overview



Distributed queries



Simple Example

```
SELECT   COUNT_BIG(*)
FROM     dbo.[FactInternetSales];
```

```
SELECT   SUM(*)
FROM     dbo.[FactInternetSales];
```

Control

Compute

```
SELECT   COUNT_BIG(*)
FROM     dbo.[FactInternetSales];
```

```
SELECT   COUNT_BIG(*)
FROM     dbo.[FactInternetSales];
```

```
SELECT   COUNT_BIG(*)
FROM     dbo.[FactInternetSales];
```

```
SELECT   COUNT_BIG(*)
FROM     dbo.[FactInternetSales];
```

Partial Parallelism

- Performed in parallel across compute nodes
- Performed in series across the distributions
- Guarantees transactional behaviour
- Examples: INSERT, UPDATE, DELETE

Full Parallelism

- Performed in parallel across compute nodes
- Performed in parallel across the distributions
- Used for
  - New object creation
  - Reading data from a distributed table
- Examples:
  - CTAS (Create Table As Select)
  - SELECT Statements

Azure SQL Data Warehouse

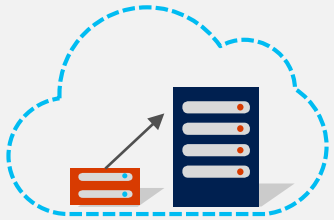
Scaling

Data warehouse unit (DWU)

Measure of power	Simply buy the query performance you need, not just hardware
Transparency	Quantified by workload objectives: how fast rows are scanned, loaded, copied
On demand	First DW service to offer compute power on demand, independent of storage

100 DWU*		
	Scan Rate	3.36M row/sec
	Loading Rate	130K row/sec
	Table Copy Rate	350K row/sec

Scan 1B rows*		
100 DWU	=	297 sec
400 DWU	=	74 sec
800 DWU	=	37 sec
1,600 DWU	=	19 sec

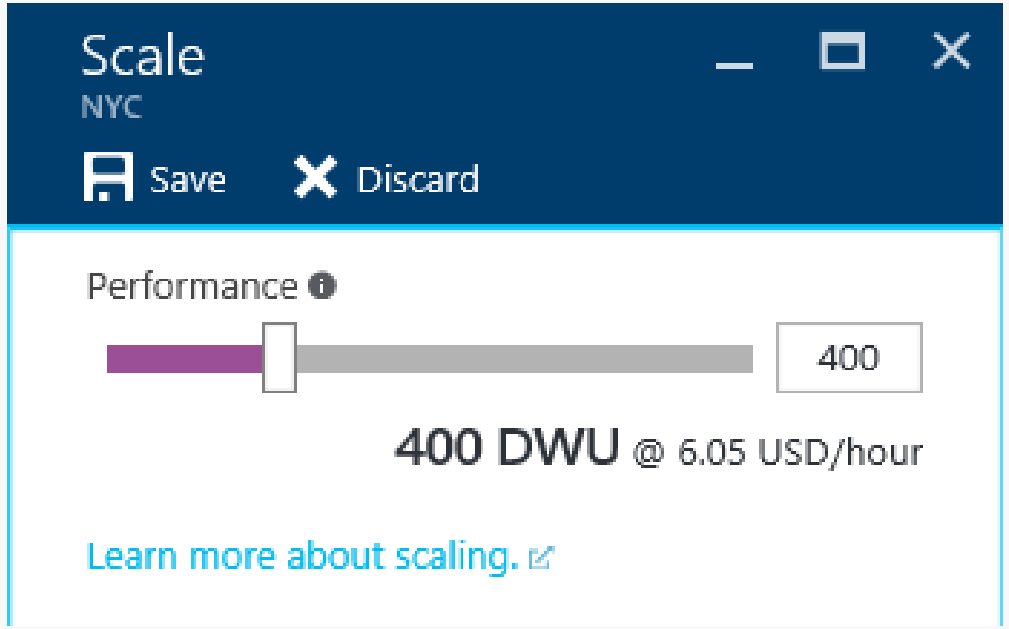


* Preliminary estimates; actual results may change

Data Warehouse Units (DWU)

DWU
DW100
DW200
DW300
DW400
DW500
DW600
DW1000
DW1200
DW1500
DW2000
DW3000
DW6000

```
ALTER DATABASE ContosoDW MODIFY
(service_objective = 'DW1000');
```



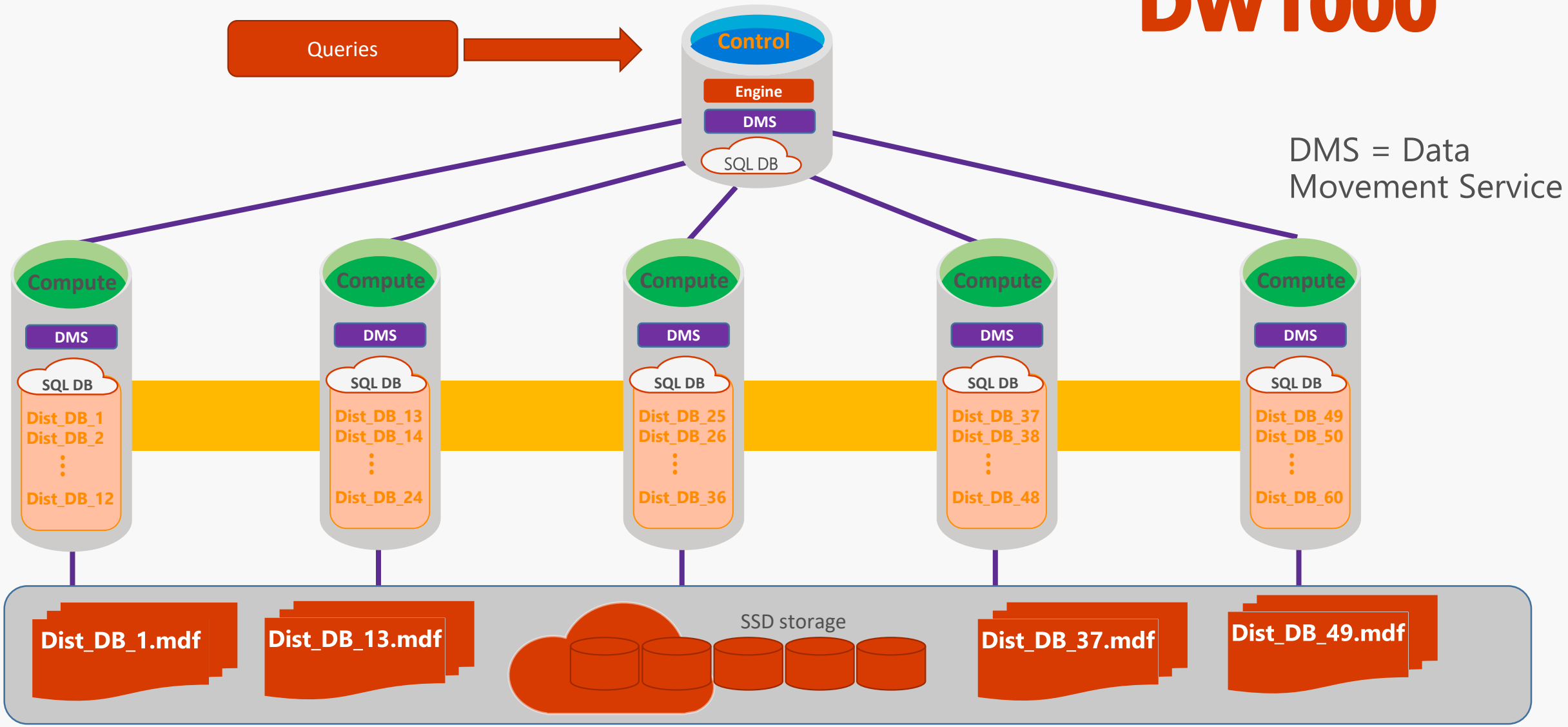
CPU

RAM

I/O

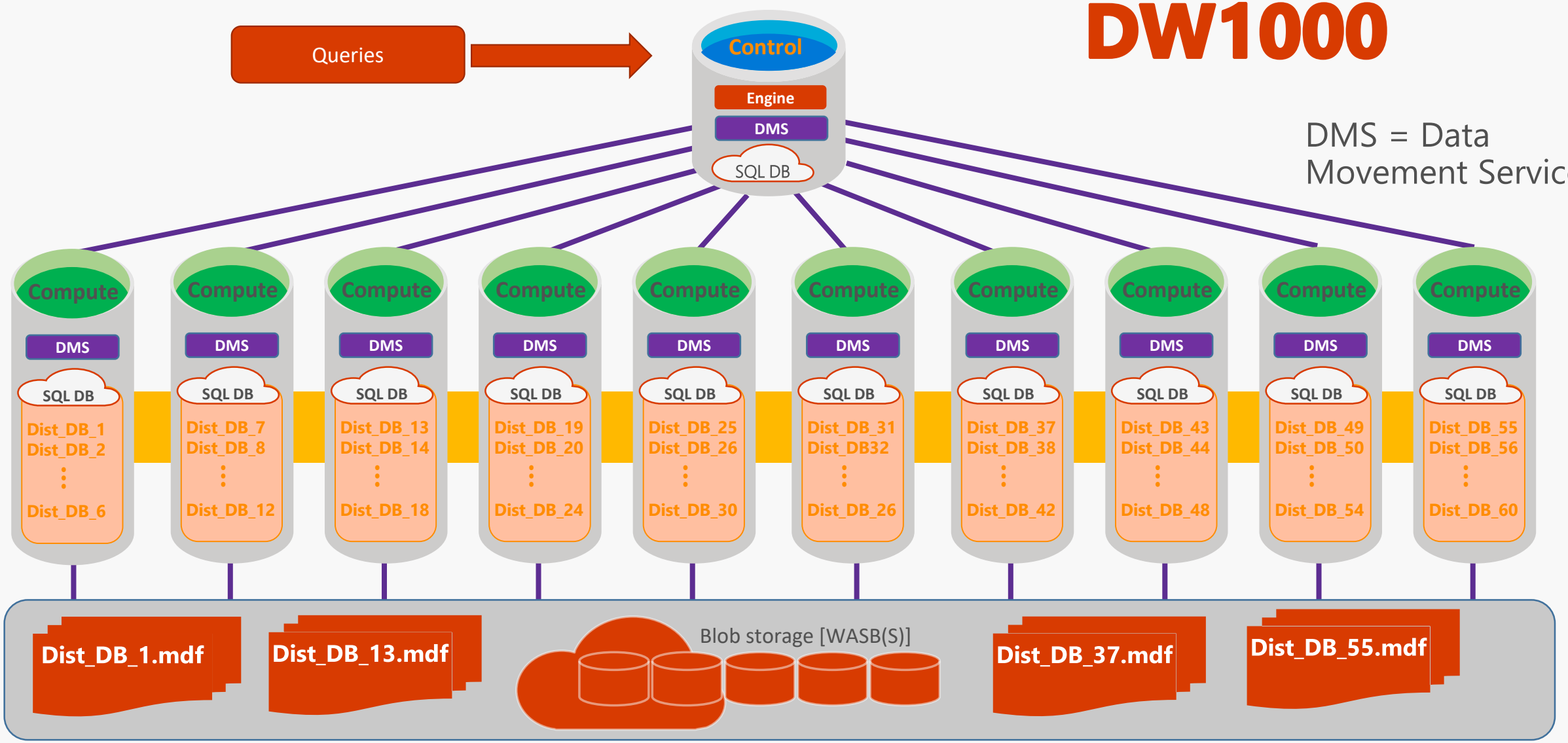
SQL DW Scaling

Changing to DW1000



SQL DW Scaling

Change DW100 to DW1000

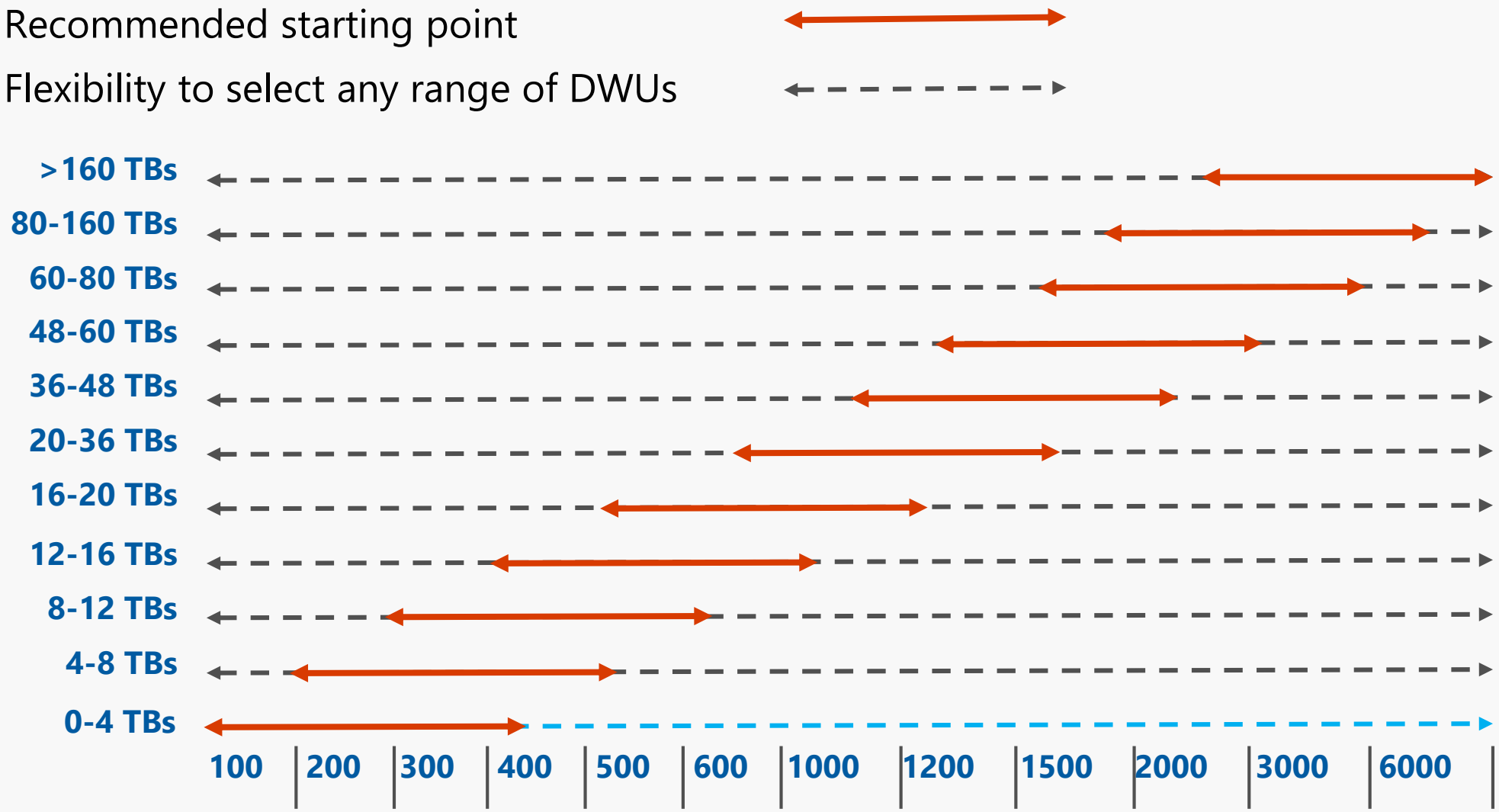


Azure SQL Data Warehouse

Sizing

Compute Sizing

Sizing by capacity



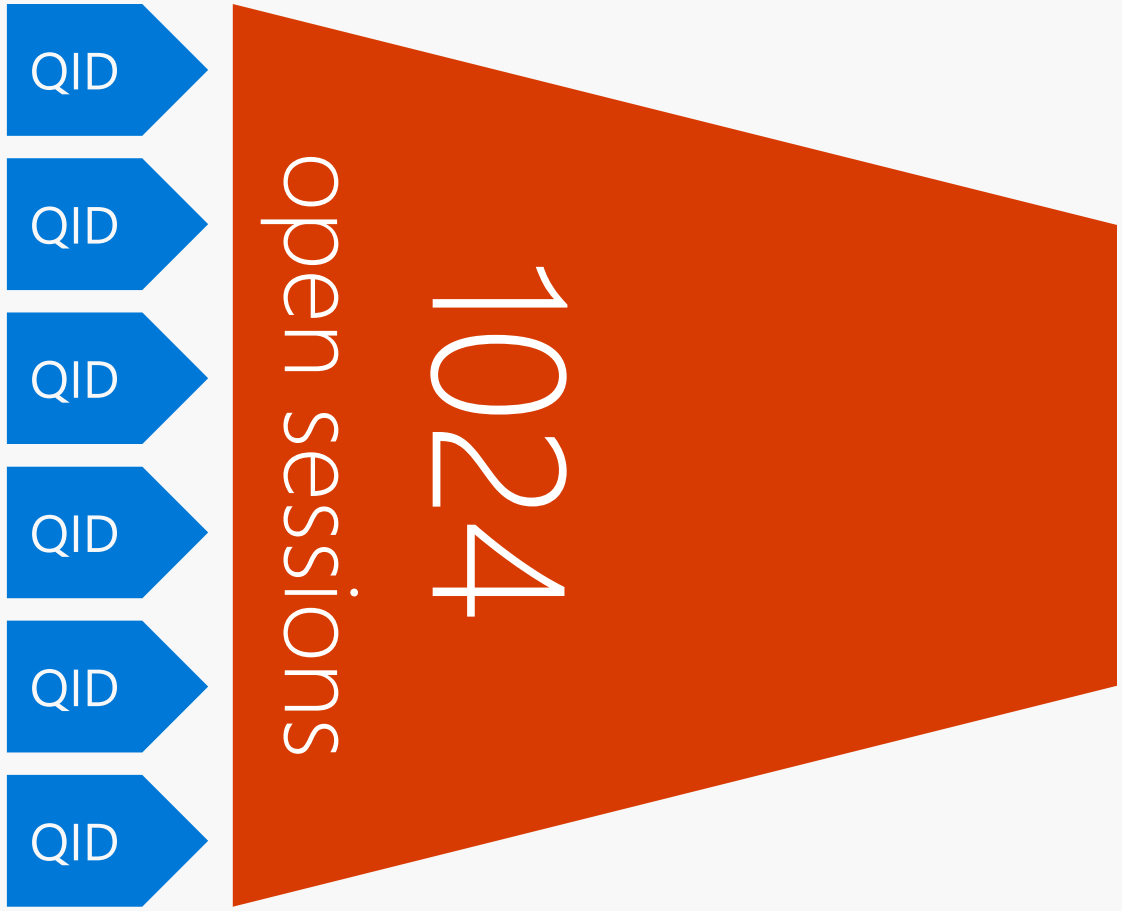
Compute sizing factors

Concurrency
Transaction size
Load
Memory

Data Warehouse Unit (DWU)

Compute sizing

Concurrency: queries

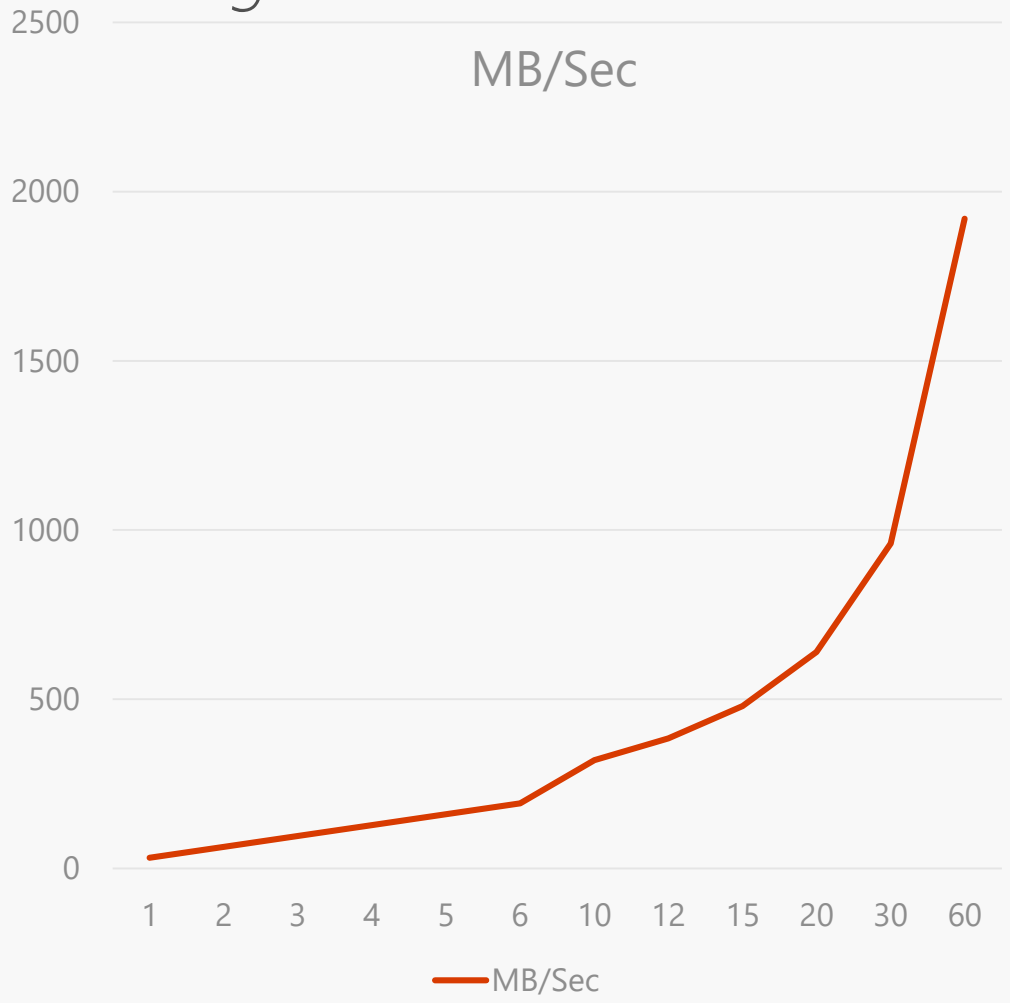


32

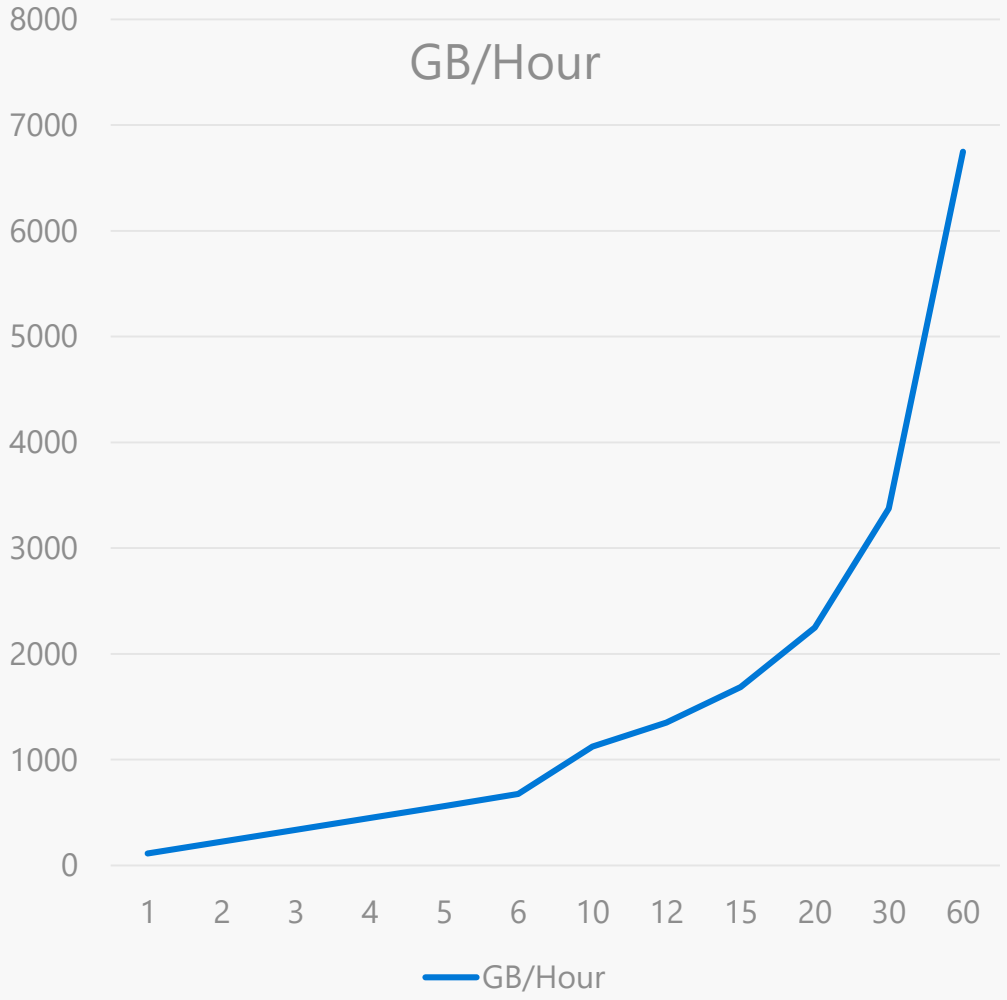
active queries

Compute sizing

Load scaling



30-32 MB/Sec/Node



110-115 GB/Hour/Node

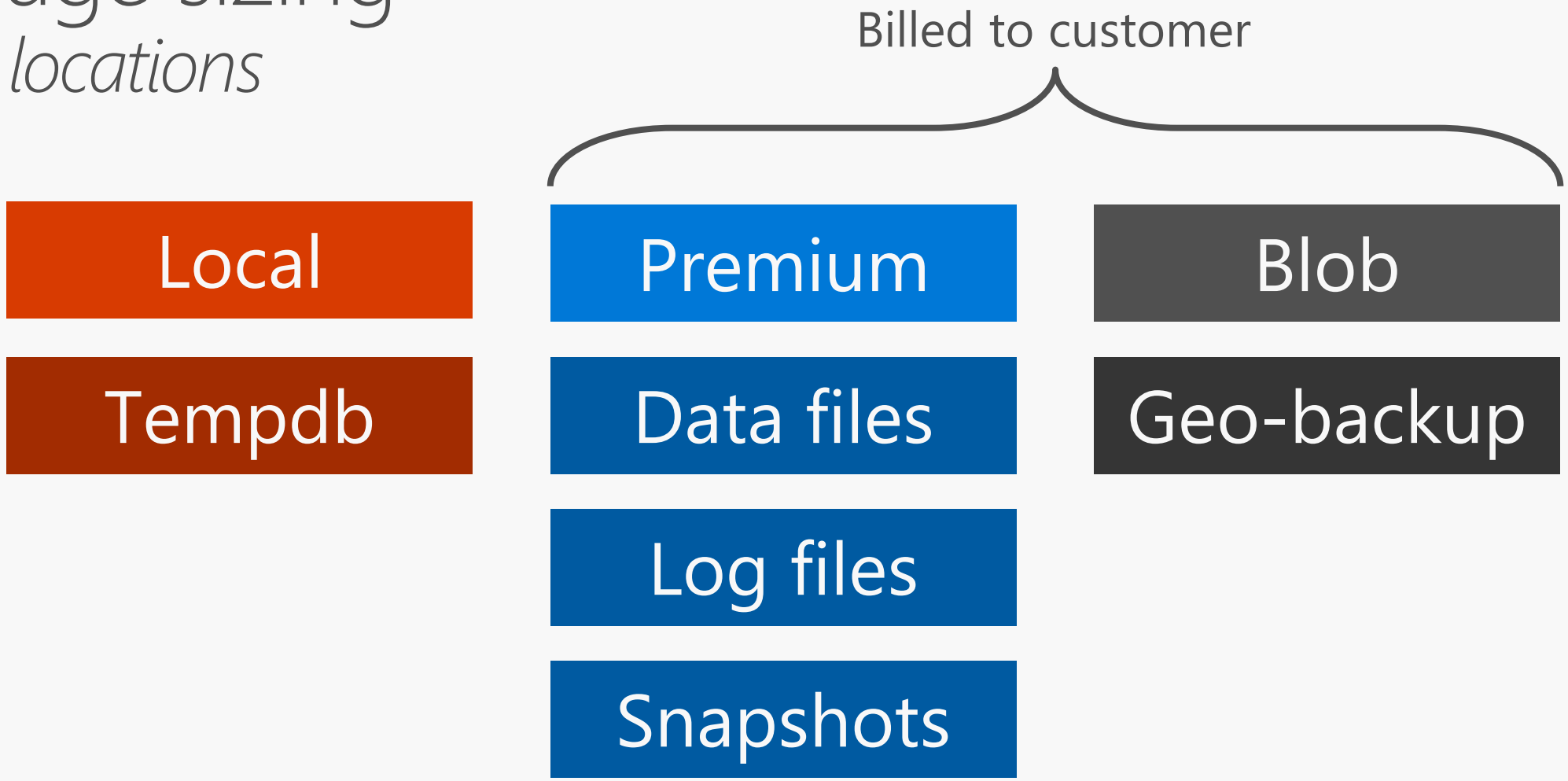
Compute sizing

Loading delimited text

A compressed text file cannot be read in parallel

Splitting data across multiple files maximises load performance

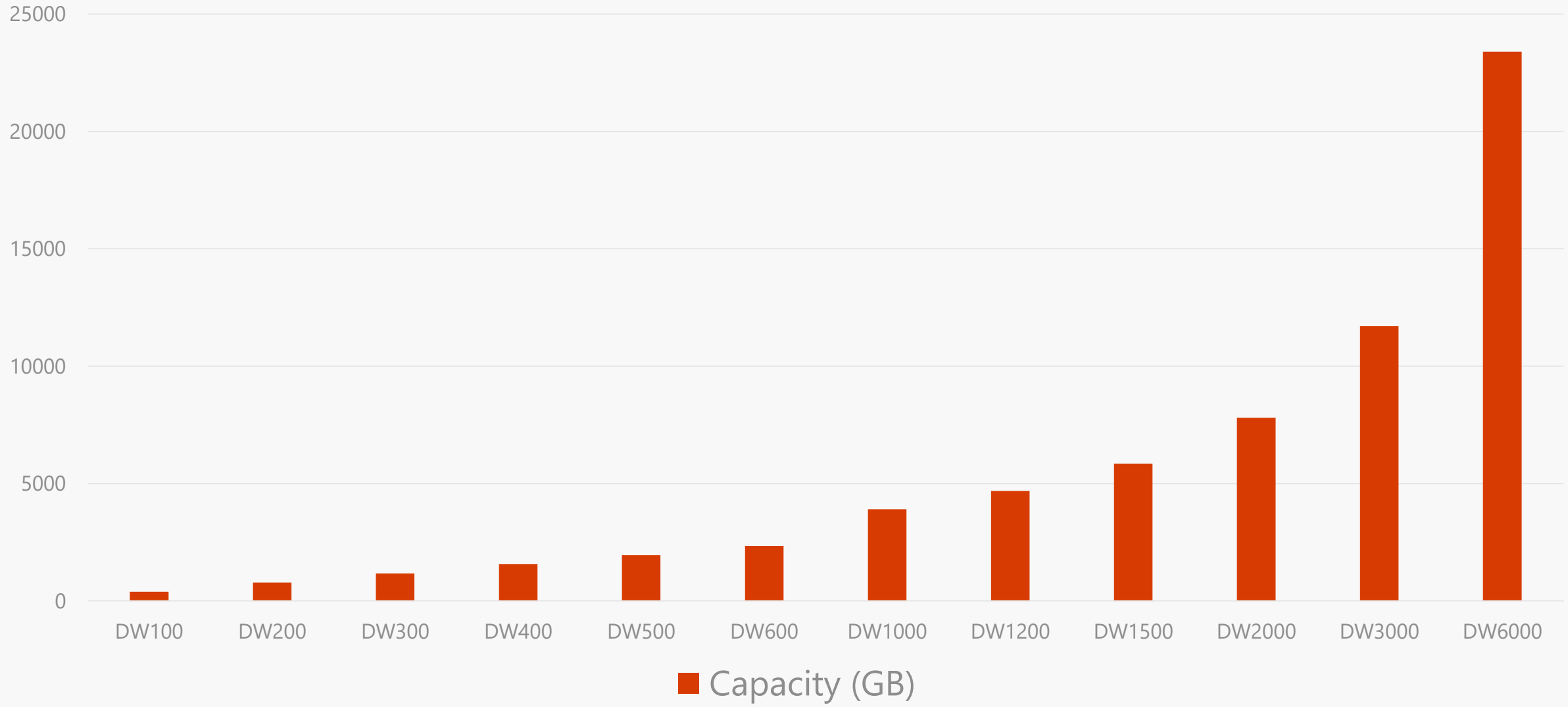
Storage sizing
Data locations



Storage sizing

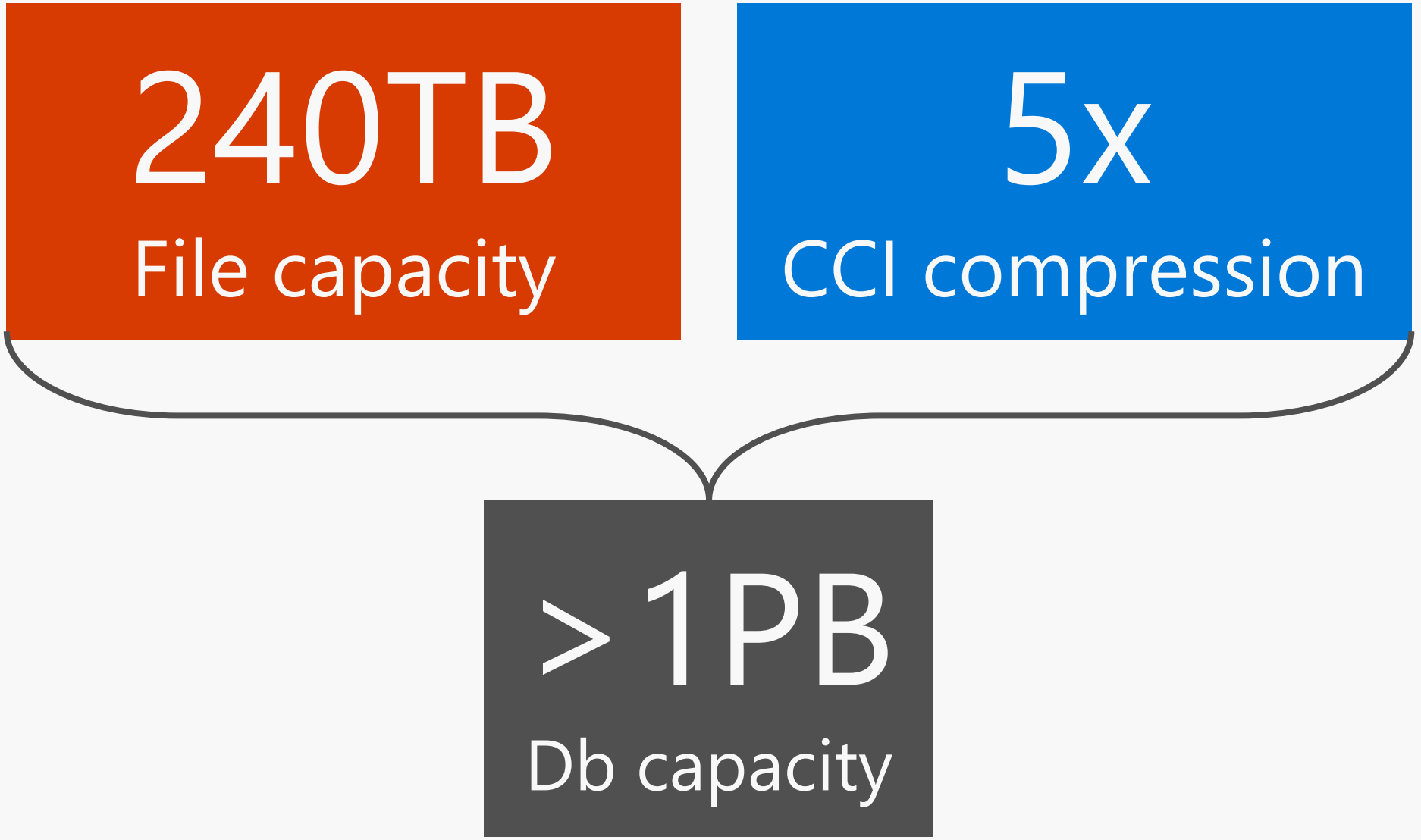
Local Storage: Temp db sizing

~399GB
per DW100



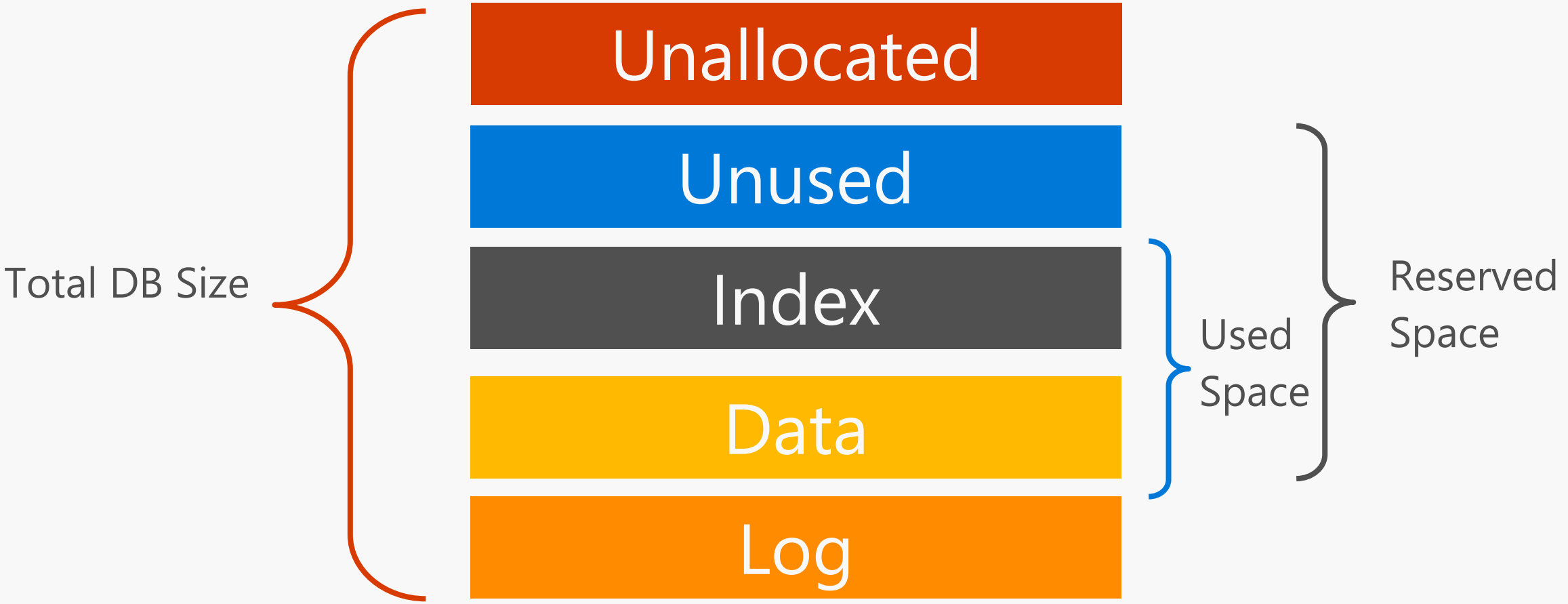
Storage sizing

Premium storage: Capacity limits



Storage sizing

Premium Storage: Database Size



Storage sizing

Premium Storage: Snapshots

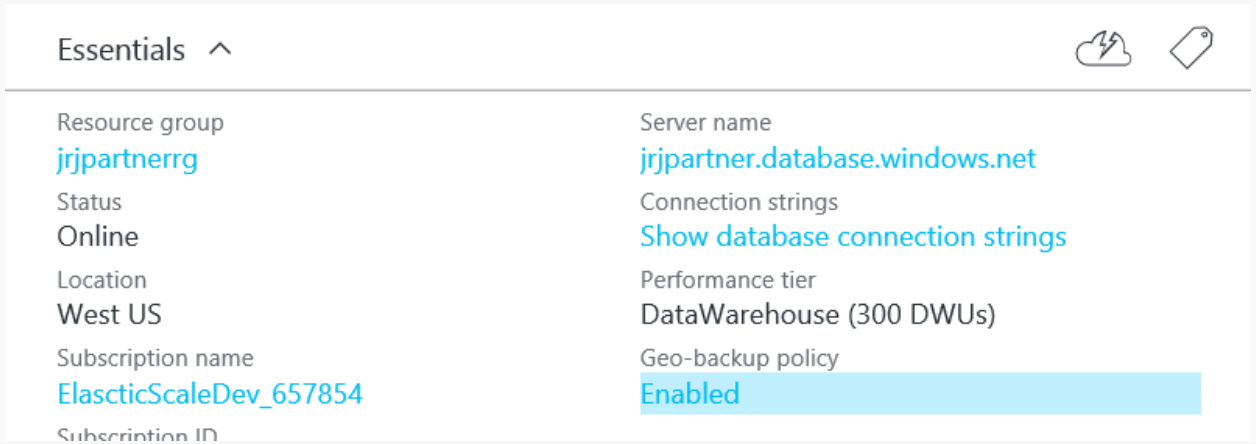
Storage sizing

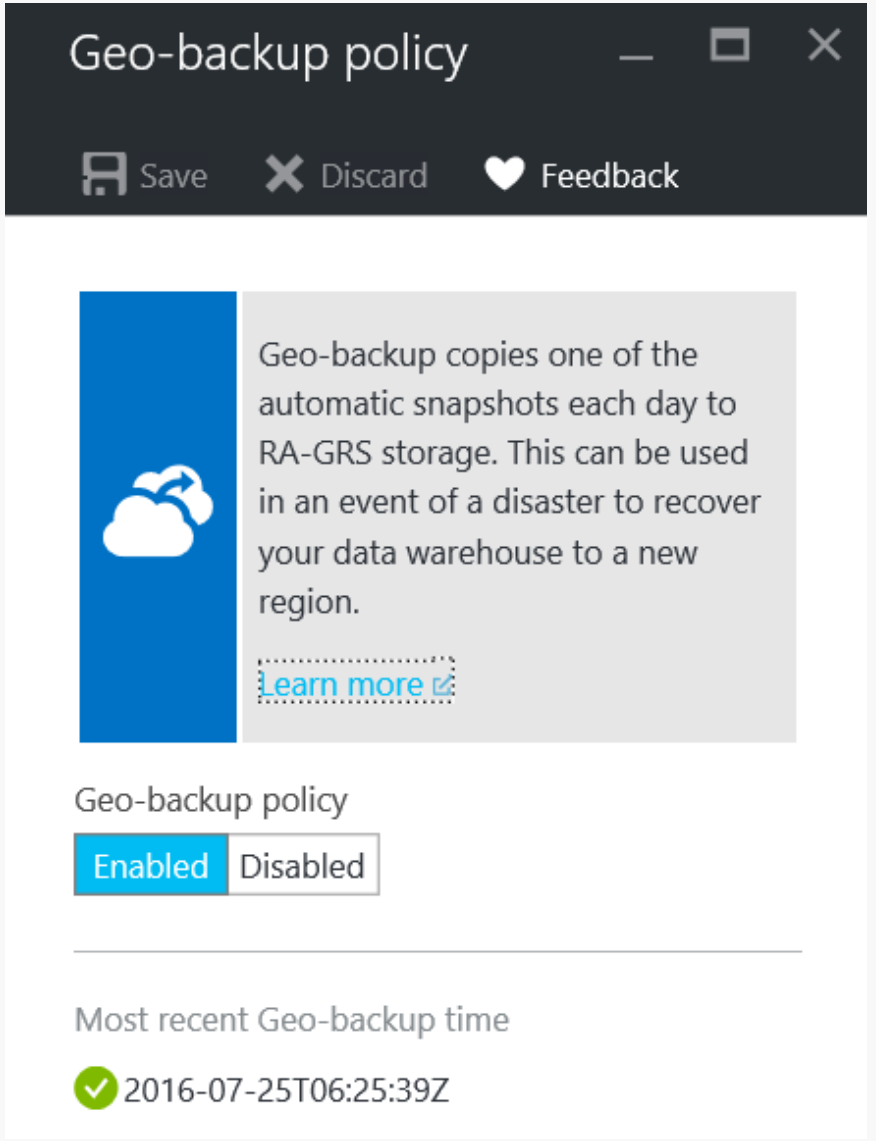
Blob Storage: Geo-redundant backups

Frequency and Retention

1
Geo-backup

24hr
RPO





Storage sizing

Capping storage capacity

```
CREATE DATABASE MyDB COLLATE
SQL_Latin1_General_CP1_CI_AS (
    EDITION                 = 'DataWarehouse',
    SERVICE_OBJECTIVE       = 'DW400',
    MAXSIZE                 = 10240 GB );


ALTER DATABASE MyDB
MODIFY (MAXSIZE = 245760 GB);
```

Storage sizing

Summary

Database size:

sp_spaceused

Table sizing:

DMVs

Snapshot size:

Total storage size (portal) – database size (sp_spaceused)

Free space (unallocated):

sp_spaceused

Azure SQL Data Warehouse

Schema Design

Distributed Data = Buckets of Water

Evenly spreading the data leads to even use application resources

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
(Future improvement)

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

Selecting a Distribution Method

For large fact tables, best option is to Hash Distribute

- Distribute on column that is joined to other fact tables
- Primary or surrogate key

However, be mindful of …

- Hash column should have highly distinct values (Minimum 60 distinct values)
- Avoid distributing on a date column
- Avoid distributing on column with high frequency of NULLs and default values (e.g. -1)
- Distribution column is NOT updatable
- For compatible joins use the same data types for two distributed tables

If there are no distribution columns that make sense, then use Round Robin as last resort

Dimension Table

Small dimension table (< 60M rows)

- Clustered index
- Round Robin
- Future Improvement - Choose replicate option when it becomes available

Large dimension table

- Same design as fact table
- Clustered columnstore (by default) and distribute on join key

Optimizing with Indexes

Clustered Column Store
(SQL DW Default)

- Optimal choice for large tables
- Limits scans to columns in the query
- Optimal compression
- Slower to load than Heap
- Keep partitions large enough to compress (> 1 million rows)

Heap

- Optimal choice for temporary or staging tables
- Fastest load performance

Clustered Index

- Optimal for tables < 60M rows
- Sorting operation slows down load

Non-clustered Indexes

- **Use sparingly**
- Optimize single row lookups
- Will slow down load

Partitioning

Partition on date column for archiving purposes

- Improves performance by partition elimination

Partition granularity depends on your workload

- Reload, re-process
- Aim for min 100K, optimal 1 million rows per distribution/partition (remember the 60 databases)

Optimize load performance through partition switching

Considers different grain partitions if you have hot/cold data in different tables

- Example: Hot data daily, cold data monthly

Keep the number of partitions “reasonable” as there are overheads

Re-indexing by partition when needed

DDL Example

```
CREATE TABLE FactFinance
(
      FinanceKey int NOT NULL,
      DateKey int NOT NULL,
      OrganizationKey int NOT NULL,
      DepartmentGroupKey int NOT NULL,
      ScenarioKey int  NULL,
      AccountKey int  NULL,
      Amount float NOT NULL)
WITH (clustered columnstore index, DISTRIBUTION = HASH(FinanceKey),
       PARTITION (DateKey RANGE RIGHT FOR VALUES
            (20100101,20200101,20300101))
);
```

Optimizing with Statistics

Create and Update of Statistics are NOT yet automatic

- Cost Based Query Optimizer needs statistics
- Sampled stats are usually just fine
- Create statistics for all columns used in JOINs, GROUP BY, WHERE
- Update statistics after incremental load
- If needed, use multi-column statistics on join and group by

```
create statistics l_orderkey on [dbo.lineitem] (l_orderkey);
     select * from sys.stats where name = ‘l_orderkey’;
      dbcc show_statistics ("lineitem","l_orderkey");
```

Azure SQL Data Warehouse

Loading

Data Loading Options

1. PolyBase
2. BCP
3. Import/Export
4. SSIS
5. Azure Data Factory
6. Redgate Platform
7. …

	PolyBase	BCP	SqlBulkCopy	SSIS
Rate				
Rate increase as you increase DWU	Yes	No	No	No
Rate increase as you add concurrent load	No	Yes	Yes	Yes

Best Practices – Data Loading

Polybase is the fastest method

- Upload to BLOB via AZCOPY or Powershell library
- History – use CTAS
- Incremental – use INSERT INTO

Use the highest resource class without sacrificing concurrency
Increase DWU during load, decrease when done

Known Issues:

- Do not support extended ASCII, UTF-16.
- Do not support custom multi-date format. E.g. 2000-1-6
- No reject files/reason for rejected rows.

PolyBase Characteristics

- A single PolyBase load provides best performance for non-compressed text files.
- The load performance scales as you increase DWUs.
- Parallelizes the data load process, no need to explicitly break the input data into multiple files and issue concurrent loads.
- Each reader will slice 512 MB block from data files.
- Max throughput will depend on number of readers available on the SLO you use.
- Multiple readers will not work against a compressed text file (gzip). Only a single reader is used per compressed file since uncompressing the file in the buffer is single threaded. Alternatively, generate multiple compressed files.
- The number of files should be greater than or equal to the total number of readers of your service level objective (SLO).

Loading Options

PolyBase	BCP	Import/Export	SSIS	ADF
Fastest and preferred load option. Use CTAS for initial load. Use INSERT/INTO for incremental load or CTAS into stage table and partition switch into final table.	Use only for small files < 10 GB. Limited retry logic. Does not scale as you increase DWU (single thread, single CPU on client). Increase parallel threads to improve performance.	Recommended for > 10TB. Put data on encrypted Disks and ship to Microsoft. Microsoft loads the data into SQL DW.	Increase client timeout at least 10 min, default 30 sec. Increase parallel threads to improve performance. Slight performance improvement & greater reliability if run on VM.	A simpler way to use PolyBase. Quick to configure since you don't need to define the T-SQL objects.

Azure SQL Data Warehouse

Querying

Common Data Movement Types

DMS Operation	Description
ShuffleMoveOperation	Redistributes data for compatible join or aggregation
PartitionMoveOperation	Data moves from compute to control node
BroadcastMoveOperation	Table needs to become replicated for join compatibility

Best Practices – Query Performance

Check for SKEW (DBCC PDW_SHOWSPACEUSED)

Statistics

CETAS or CTAS large return operation

De-normalize if needed

DSQL Query Plan

- Minimize data movement operations
  - Distribution & aggregation compatible
- Minimize size of data movement
  - Check for predicate pushdown. Rewrite query if needed
- Use higher resource class for memory intensive queries
- Load large external tables rather than querying directly

Azure SQL Data Warehouse

High Availability

High Availability – Out of the Box

Storage

- Backed by premium LRS
- Three copies are maintained, intra-datacenter for fault tolerance
- Replicas are served in the event of failures by the framework automatically

Compute

- Service Fabric container backbone
- Compute nodes run on containers
- No replicas maintained, new container is served up from container pool
- Less than a minute to provision
- Framework seamlessly restores

Azure SQL Data Warehouse

Backup and Disaster Recovery

Snapshots – out of the box

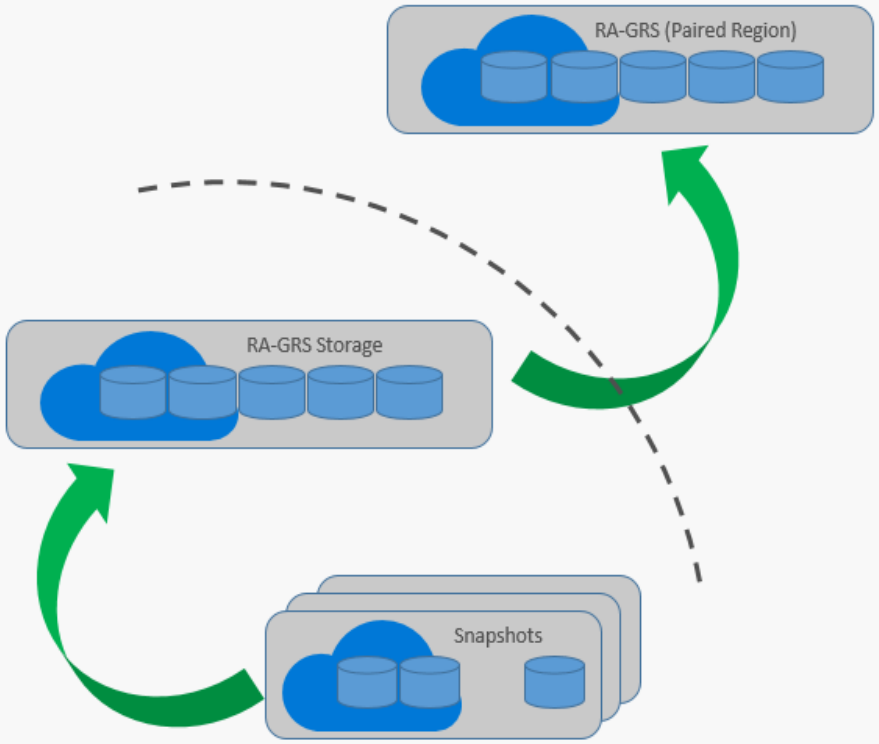
- Automatically scheduled to run every 4 – 8 hours
- Snapshot is incremental
- Saved to premium LRS (Locally Redundant Storage)
- Retention period is 7 days
- Point-in-time restore configurable
- Snapshots are taken only when NOT paused
- Snapshots cannot be invoked programmatically
- RPO is 8 hours
- Workarounds available to keep snapshots older than 7 days
- .Net to programmatically create snapshot
- Query to see when last snapshot started →

```
select top 1 *
from sys.pdw_loader_backup_runs
order by run_id desc;
```

Back up to DR datacenter – out of the box

Features:

- Scheduled to run every 24 hours
- On by default, can opt-out
- RPO is 24 hours
- Full warehouse backup – only 1 backup copy
- Access your geo-backup from either the primary or the secondary location

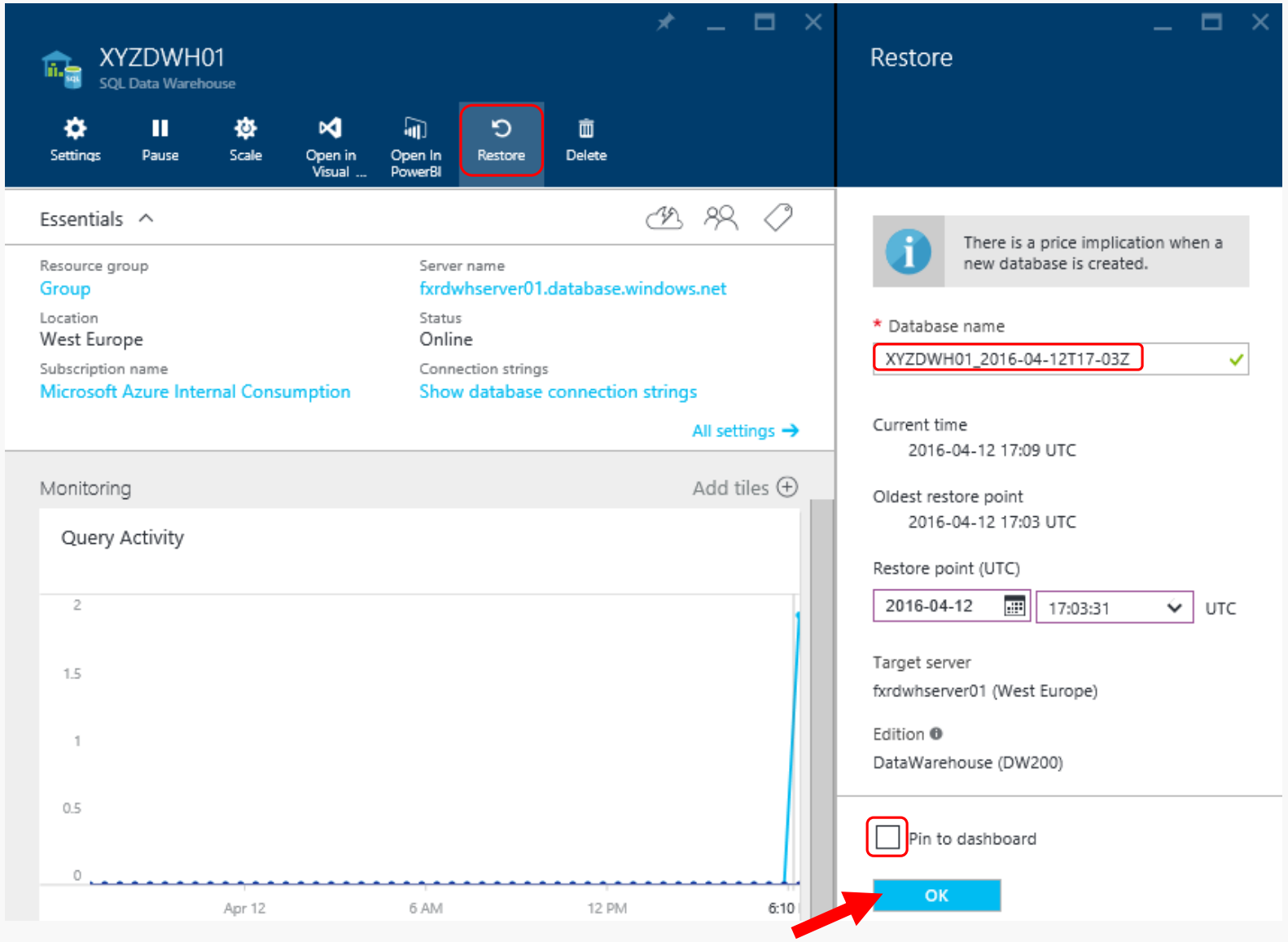


Backup Process:

- Copies latest snapshot to a RA-GRS storage account (standard) in primary data center
- Data in this RA-GRS account gets replicated to paired Geo-DR center automatically

SQL DW – Restore (3 Methods)

Azure Portal
(Method 1)



SQL DW – Restore (3 Methods)

Rest API (Method 2)

1. List all of your restorable deleted databases by using the List restorable dropped databases operation.
2. Get the details for the deleted database you want to restore by using the Get restorable dropped database operation.
3. Begin your restore by using the Create database restore request operation.
4. Track the status of your restore by using the Database operation status operation.

SQL DW – Restore (3 Methods)

Powershell (Method 3)

1. Select the subscription under your account that contains the database to be restored.
2. List restore points for the database (requires Azure Resource Management mode)
3. Pick the desired restore point using the RestorePointCreationDate.
4. Restore the database to the desired restore point.
5. Monitor the progress of the restore.

Azure SQL Data Warehouse

Security

Security

Authentication

- SQL Server authentication
- Azure Active Directory
- Create administrator user at provision time, then create users for SQL Server authentication

Authorization

- Same model as SQL Server – role, role memberships, permissions

Data Protection

- Transparent data encryption at rest – AES 256 bit
- Microsoft managed certificates and periodic rotation (90 days) by Microsoft
- Encrypted configuration, automatically results in encrypted Geo-DR backup, local snapshots, metadata and log files
- Encrypt using portal or T-SQL

Connection Security

- Firewall rules used by both the server and the database to reject connection attempts from IP addresses that have not been explicitly whitelisted
- Set up server-level firewall rule to enable client access

Auditing & Threat Detection

Auditing

- Audit log persisted in storage account
- Retain an audit trail of selected events by defining categories
- Report on database activity including some pre-configured reports.
- Excel dashboard template available
- Find suspicious events, unusual activity and trends

Threat Detection

- A new layer of security
- Detect and respond to potential threats as they occur by providing alerts on anomalous activities
- Makes it simple to address potential threats without a need for a security expert

Azure SQL Data Warehouse

Best Practices

Best practice

Cost management:

Reduce cost with pause and scale down

Pausing/scaling:

Drain transactions prior to pausing/scaling

Statistics:

Maintain statistics for performance

External tables:

Save locally, then query (versus PolyBase query)

Best practice

Scaling:

Check DMV sys.dm_pdw_exec_requests before "Change SLO" to make sure no running requests

Loading:

Use PolyBase

Scaling provides additional readers/IO

CTAS to a partition and switch partition

Use appropriate resource class for bulk insert

No singleton inserts - Batch single row inserts into larger loads

Export:

Use PolyBase

Best practice

Querying:

Keep up to date statistics
Choose distributions carefully
Size resource classes appropriately
Avoid querying external tables, load to local and then query
Use DMVs to monitor and optimize queries

Upgrading:

Use minimally logged operations for fast recovery
Build retries into applications/processes

Availability:

Build retries into applications/processes

Best practice

Partitioning:

Avoid over-partitioning – leads to poor performance

*If you issue - create 100 partitions, it translates to 100*60, 6000 partitions*

Plan for 1M rows per partition to leverage clustered columnstore

Transactions:

- Minimize transaction sizes

- Leverage special minimal logging cases, like CTAS, TRUNCATE, DROP TABLE or INSERT to empty tables, to reduce rollback risk

- For unpartitioned tables consider using a CTAS to write the data you want to keep in a table rather than using DELETE

- For partitioned tables, use Metadata Only operations like partition switching for data management

Best practice

Column-size:

Define to smallest possible size for performance

*If you issue - create 100 partitions, it translates to 100*60, 6000 partitions*

Plan for 1M rows per partition

Optimize clustered columnstore tables:

*- Partitioned tables: leverage ONLY when row count > 60 * number of partitions * 1 million rows*

- Segment quality is important (measured by number of rows in a compressed Row Group); Use User IDs in medium or large resource classes to load data; Fewer DWUs – assign higher resource class of loading user

- Columnstore tables do not yet support secondary indexes

- Queries will run faster if you select only the columns you need

Best practice

Transient data:

E.g. Stage tables

Leverage heap tables (session-access constrained) for best performance (remember to create statistics)

Use resource classes wisely:

- Resource classes are used to allocate memory to queries

- Default: all users are assigned to the small resource class which grants 100 MB of memory per distribution

- Larger resource classes, allocate more memory and therefore impact concurrency

- Certain queries, like large joins or loads to clustered columnstore tables, will benefit from larger memory allocations. Some queries, like pure scans, will see no benefit.

Best practice

Real-time ingestion and BI queries:

- *Avoid real time data ingestion (ASA, Spark Streaming, Storm)*
- *Avoid PowerBI direct query against SQL DW*
    - *Use Azure Analysis Services or SSAS in a VM*

Data marts:

- *Use data marts to enable high concurrency workloads*

Tune concurrency:

- *Larger resource classes consume a higher number of concurrency slots causing other queries to queue up*

Best practice

Security:

- *Restrict IP ranges when configuring server-level firewall rule*
- *Set up individual users (versus server admin)*
- *Use principle of least privilege*

Azure SQL Data Warehouse

Management & Monitoring

Management tools

Azure portal

- *Create, update, delete database*
- *Monitor database resources*

SQL Server Data Tools in Visual Studio

- *Connect, manage, develop*

Command-line tools

- *Automation; PowerShell and sqlcmd*

Dynamic Management Views

- *Bread and butter of managing SQL datawarehouse*

Monitor workloads

Dynamic Management Views:

- *Monitor connections*
- *Monitor query execution*
- *Monitor queued queries*
- *UserID needs VIEW DATABASE STATE or CONTROL permission to query DMVs*

https://azure.microsoft.com/en-us/documentation/articles/sql-data-warehouse-manage-monitor/

Azure SQL Data Warehouse

Migration

Why does Migration efficiency matter?

The amount of data to be migrated is large

- Usually several TB in DW to be moved from on-prem to cloud

You want to cut down the time to migrate to cloud

- Reduce downtime window
- Reduce risk of in-flight changes during migration

You want to reduce to cost of migration

- Reduce network cost
- Reduce compute cost for migration operations

Understanding the steps and tips/techniques in each helps realize an efficient migration

Data Preparation and Metadata Migration

- Filter essential objects to migrate
- Create performant local storage to receive exported data
- Establish standard or dedicated connectivity to cloud
- Chose region nearest to you with Azure SQL DW
- PolyBase: One folder per table in storage container

Compatibility check must be local and as the first step

Must complete all compatibility corrections before move

Compatibility check automation

SQL Data Warehouse Migration utility

TSQL Queries

Use the SQLCAT guidance for choosing the type of distributed table

Data Migration Best Practices

Use Migration Tool - convert DDL, generate T-SQL compat report, data migration

Understand current T-SQL surface area and workarounds

Avoid Singleton DML operations (INSERT, UPDATE, DELETE)

- Batch DML if possible
- If unavoidable, wrap in transaction (BEGIN TRAN…COMMIT)

Use heap table, or temp table for “staging” data

Avoid large fully logged operations

- Considers CTAS as this is minimal logged operation
- Use LOJ as alternative for DELETE
- Process by partition to leverage parallelism and partition switching

Design retry logic to address service disruption

Data Migration Best Practices

Data Format Conversion

- Date Format, Field delimiters, escaping, field order, encoding

Compression

- Must use Gzip
- 7-Zip utility, .NET/JAVA libraries

Export

- BCP for fast export
- Multiple files per large table, one folder per table

Copy

- AZCopy
- Data Movement Library

Tips

- Incorrect format means migration needs to be entirely repeated
- Exploit bcp options, hints, parallelism
- Multiple compressed files, Split files
  - Parallel import, reliable transfer
- Don’t use multiple files in the same zipfiles
- Efficient Copy
  - Parallel, Async, Resumable
  - Limit concurrent copies if low bandwidth
- Only One AzCopy per machine
- Very Large Data transfer
  - Express Route, Import/Export Service

Data Import and Transformations

Import is fastest using PolyBase

- CREATE TABLE AS…
- INSERT INTO.. SELECT FROM

Import options and Tradeoffs

- Importing Compressed
  - Faster export and transfer, slower load
- Importing Un-compressed
  - Slower export and transfer, faster load

Data Transformation

- Prefer ELT vs ETL
- Create statistics

Tips

- Increase DWUs allocated for Azure SQL DW for faster import
- Use Higher resource classes for data loading
- Refer to SQLCAT guidance on Data Loading Strategies
- Distinguish between compatibility conversions and data transformations

ETL Best Practices

Generating Identities in ETL

- Fastest done in pulling data from source
- Next best option: ETL tool generates identities

Merging into SQL DW

- Incremental load: Generate hash on columns and compare hash in DELETE, INSERT or UPDATE with SQL DW HashBytes

Sometimes ETL Tool may score over PolyBase

- Small Tables: Simpler dev experience, Error routing etc.
  - Perf Impact not noticeable
- Tables with Large Rows (>32K): No support in PolyBase yet

Tips

- Exploit SQL Server/Oracle features like ROW_NUMBER
- Use the SQL DW built in HashBytes
- Table with large sized rows and a large number of rows
- Consider vertically splitting the tables for fast loading with PolyBase

Azure SQL Data Warehouse

Pricing

<Documentation>

Azure SQL DW - Appendix

- # Demo Steps
  - Create a Logical Server
  - Create a SQL DW (1000 DWU)
  - Load data (10GB) from TPCH Blob
  - Run simple queries
  - Pause and run simple queries again (Error)
  - Scale Down to 400
  - Run sample queries again
  - Walk through the code as needed for the above