KickStart

DP 203 certification

Data Engineering on Microsoft Azure

9h15 – 9h30 Revue contenu de la cerification DP203

9h30 Azure DataLake
Synapse SQLPool
Synapse DataFlow
Stream Analytics

12h30 Security SQL & Datalake

12h30-13h30 Pause Dej

13h30-14h Monitoring

14h-16h30 Dryrun

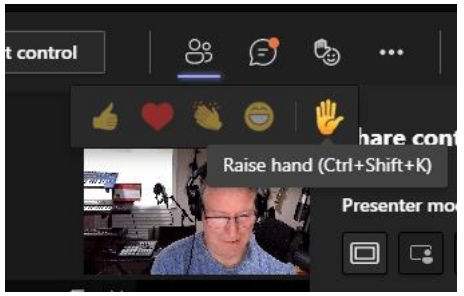
16h30-17h QnA

Microsoft

Questions during the session

In a regular teams meeting, please use the “Raise Your Hand” feature if you want to ask a question versus posting it in the chat window

Ctrl+Shift+K is a quick way to raise or lower your hand



Microsoft Confidential

All content is NDA unless otherwise stated.

Exam overview and objective domain
DP-203

Become Microsoft Certified Azure Data Engineer Associate

Start here
Decide if this is the right certification for you

This certification is a good fit if your responsibilities include:

- Building and maintaining secure and compliant data processing pipelines by using different tools and techniques.
- Using various Azure data services and languages to store and produce cleansed and enhanced datasets for analysis.

OR

If you already have or were working toward this certification:

- MCSA: Data Engineering with Azure

Brand new? First, master the basics.

New to the cloud or Azure?
Choose Azure Fundamentals training.

New to data solutions on Azure?
Choose Azure Data Fundamentals training.

Upskill with recommended training and experience

Skills outline guides

- DP-203

Self-paced online learning
Microsoft Learn

Instructor-led training
Course DP-203T00: Data Engineering on Microsoft Azure

Additional resources

- Microsoft Docs
- Azure Architecture Center

Pass required exams to earn your certification

Exam DP-203
Data Engineering on Microsoft Azure

Microsoft Certified:
Azure Data Engineer Associate

Azure Data Engineers integrate, transform, and consolidate data from various structured and unstructured data systems into structures that are suitable for building analytics solutions.

Continue to grow

Explore these resources next

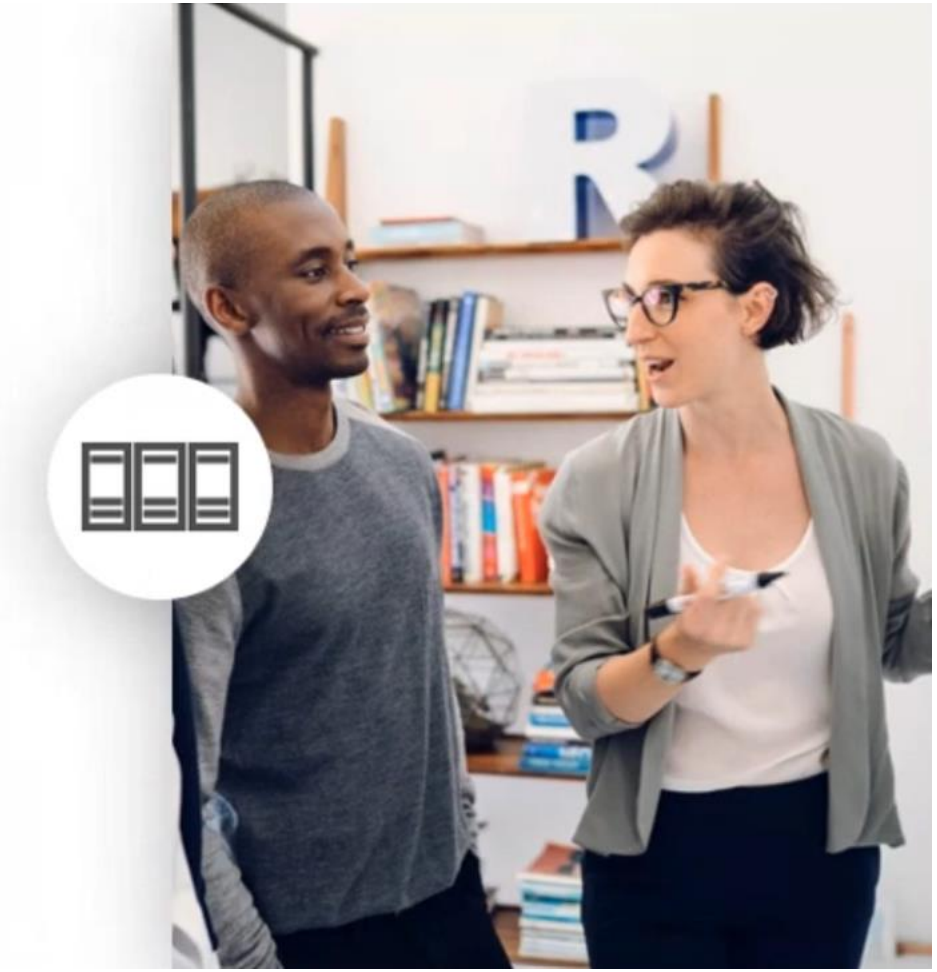
Self-paced online learning

- Migrate data to Azure

Instructor-led training

- Course DP-050T00: Migrate SQL workloads to Azure
- Course DP-060T00: Migrate NoSQL workloads to Azure Cosmos DB
- Course DP-070T00: Migrate Open Source Data Workloads to Azure

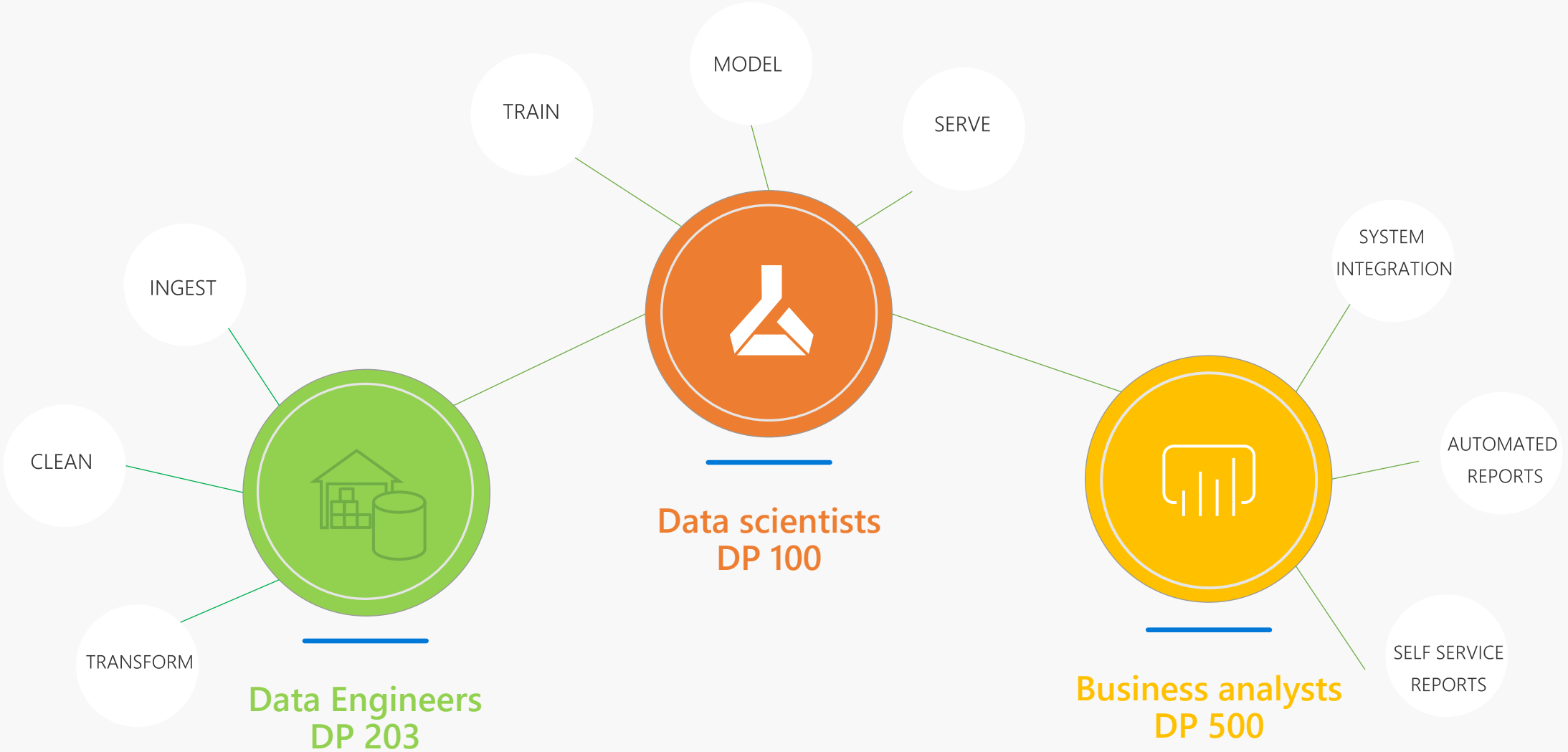
Azure Enterprise Data Analyst Associate Role



Data Engineer Role

Audience profile:
Responsibilities for this role include helping stakeholders understand the data through exploration, building and maintaining secure and compliant data processing pipelines by using different tools and techniques. This professional uses various Azure data services and languages to store and produce cleansed and enhanced datasets for analysis.
An Azure data engineer also helps ensure that data pipelines and data stores are high-performing, efficient, organized, and reliable, given a specific set of business requirements and constraints. This professional deals with unanticipated issues swiftly and minimizes data loss. An Azure data engineer also designs, implements, monitors, and optimizes data platforms to meet the data pipeline needs.
A candidate for this certification must have solid knowledge of data processing languages, such as SQL, Python, or Scala, and they need to understand parallel processing and data architecture patterns.

Azure Data workloads



Frictionless collaboration

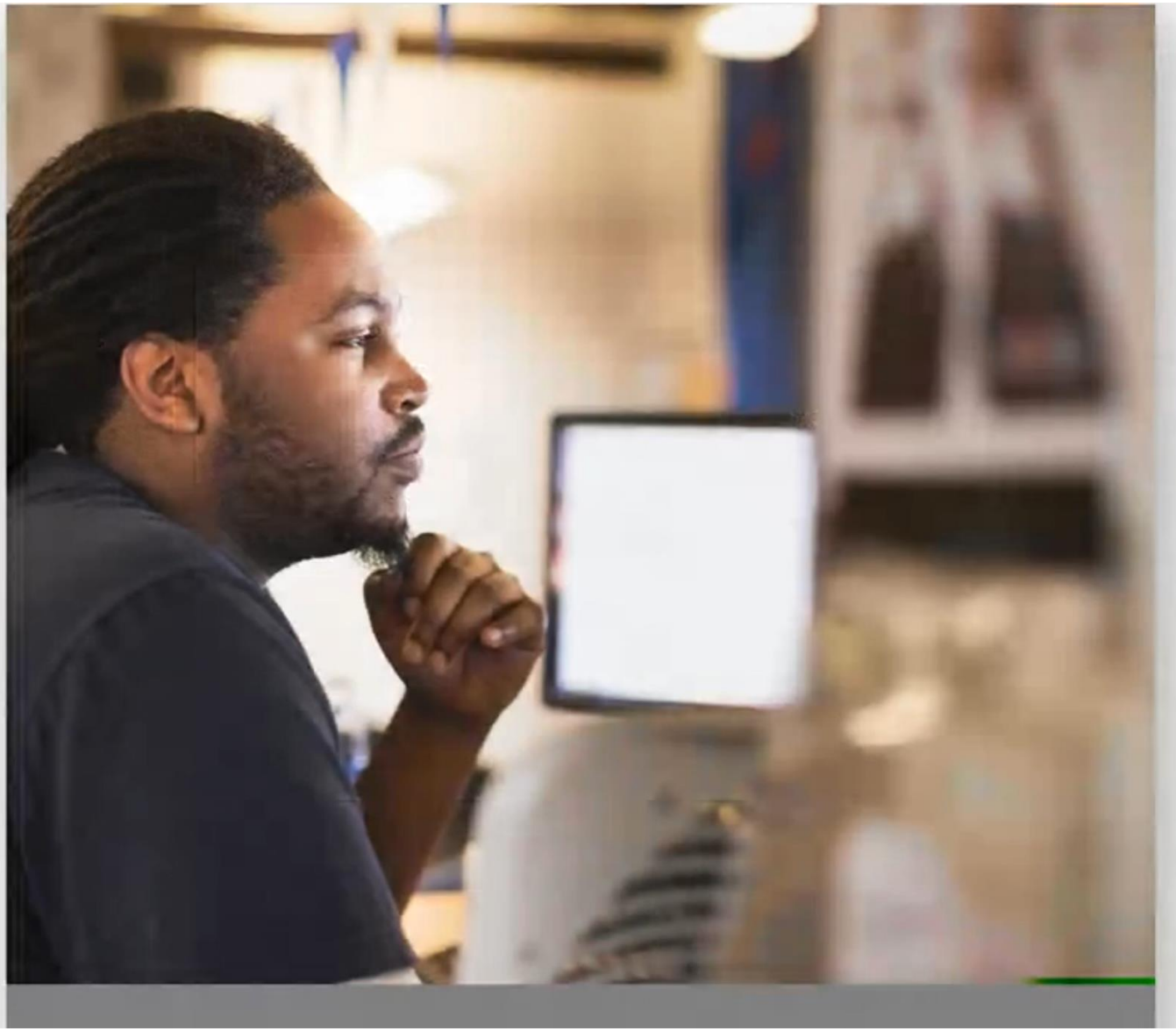
Exam objectives

Skills Measured	Weights
Design and implement data storage	40-45%
Design and develop data processing	25-30%
Design and implement data security	10-15%
Monitor and optimize data storage and data processing	10-15%

- Percentages indicate the relative weight of each area on the exam
- The higher the percentage, the more questions you are likely to see in that area

Passing score is 700

How to prepare



Training and certification

Microlearning

- Step-by-step, bite-sized tutorials by product, skill level, and job role
- Hands-on learning with interactive browser-based scripting environments
- Immediate assessment via knowledge quizzes

❑ Microsoft Learn

Online Courses

- Self-paced, structured courses with the ability to track progress and maintain a transcript
- Hands-on learning with hands-on labs
- Knowledge assessments

❑ Microsoft Learning Partners
❑ LinkedIn Learning
❑ Pluralsight

Instructor-led Training

- Blended learning, in-person, and online to suit learning needs
- Delivered by Microsoft Certified Trainers
- Deep technical training to give yo the technical expertise

❑ Microsoft Learning Partners

Certification

- New role-based certifications, including varying levels (Fundamentals, Associate, Expert)
- Industry-recognized technical certifications
- Share "credentials" with your professional network with an online badge

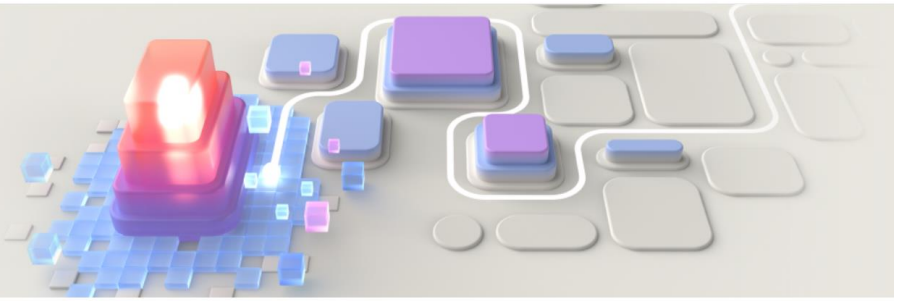
❑ aka.ms/MSCert
❑ Microsoft Learning Partners

https://learn.microsoft.com/en-us/certifications/browse/?roles=data-engineer

Not syncing

Browse Certifications and Exams

Learn new skills to boost your productivity and enable your organization to accomplish more with Microsoft Certifications.



Filter

Clear all

Products

Find a product

- Azure
- Dynamics 365
- Microsoft Power Platform
  - SQL Server

Types

- Certification
- Exam

Levels

- Advanced
- Beginner
- Intermediate

Roles

- Data Analyst
- Data Engineer
- Data Scientist
- Database Administrator
- Developer
- Functional Consultant
- Solution Architect
- Student

Certification Types

- Fundamentals
- Role-based
- Specialty

Search

Search

Data Engineer ×

14 results

CERTIFICATION
Microsoft Certified: Azure Cosmos DB Developer Specialty
ExamDP-420
Azure • Developer • Intermediate
Save

EXAM
Exam DP-420: Designing and Implementing Cloud-Native Applications Using Microsoft Azure...
Azure • Developer • Intermediate
Save

EXAM
Exam MB-260: Microsoft Customer Data Platform Specialist
Azure • Data Engineer • Intermediate
Save

CERTIFICATION
Microsoft Certified: Customer Data Platform Specialty
ExamMB-260
Azure • Data Engineer • Intermediate
Save

EXAM
Exam DP-203: Data Engineering on Microsoft Azure
Azure • Data Engineer • Intermediate
Save

CERTIFICATION
Microsoft Certified: Azure Data Engineer Associate
ExamDP-203
Azure • Data Engineer • Intermediate
Save

CERTIFICATION
Microsoft Certified: Azure Data Fundamentals
ExamDP-900
Azure • Database Administrator • Beginner
Save

EXAM
Exam DP-900: Microsoft Azure Data Fundamentals
Azure • Database Administrator • Beginner
Save

EXAM
Exam 70-467: Designing Business Intelligence Solutions with Microsoft SQL Server
SQL Server • Data Engineer • Advanced
Save

EXAM
Exam 70-464: Developing Microsoft SQL Server Databases
SQL Server • Data Engineer • Advanced
Save

EXAM
Exam DP-200: Implementing an Azure Data Solution
Azure • Data Engineer • Intermediate
Save

EXAM
Exam 70-465: Designing Database Solutions for Microsoft SQL Server
SQL Server • Data Engineer • Advanced
Save

EXAM
Exam DP-201: Designing an Azure Data Solution
Azure • Data Engineer • Intermediate
Save

EXAM
Exam 70-466: Implementing Data Models and Reports with Microsoft SQL Server
SQL Server • Data Engineer • Advanced
Save

Build enterprise-scale data analytics solutions

Maximize the value of data assets with advanced analytics

Two ways to prepare

Online - Free | Instructor-led - Paid

LEARNING PATH

Azure for the Data Engineer

0 of 3 modules completed

Beginner • Data Engineer • Azure

Start

Save

LEARNING PATH

Store data in Azure

0 of 5 modules completed

Beginner • Developer • Azure

Save

LEARNING PATH

Data integration at scale with Azure Data Factory or Azure Synapse Pipeline

0 of 7 modules completed

Intermediate • Data Engineer • Data Factory

Save

LEARNING PATH

Realize Integrated Analytical Solutions with Azure Synapse Analytics

0 of 4 modules completed

Beginner • Data Engineer • Synapse Analytics

Save

LEARNING PATH

Work with Data Warehouses using Azure Synapse Analytics

0 of 9 modules completed

Intermediate • Data Engineer • Synapse Analytics

Save

LEARNING PATH

Perform data engineering with Azure Synapse Apache Spark Pools

0 of 4 modules completed

Intermediate • Data Engineer • Synapse Analytics

Save

LEARNING PATH

Work with Hybrid Transactional and Analytical Processing Solutions using Azure Synapse Analytics

0 of 3 modules completed

Intermediate • Data Engineer • Synapse Analytics

Save

LEARNING PATH

Data engineering with Azure Databricks

0 of 5 modules completed

Intermediate • Data Engineer • Databricks

Save

LEARNING PATH

Large-Scale Data Processing with Azure Data Lake Storage Gen2

0 of 3 modules completed

Beginner • Data Engineer • Azure

Save

LEARNING PATH

Implement a Data Streaming Solution with Azure Streaming Analytics

0 of 3 modules completed

Beginner • Data Engineer • Azure

Save

https://www.starttest.com/ITDVersions/22.0.0.0/ITDStart.aspx?SVC=9b7e8f8d-2d1a-4d9f-a77c-cc0ca3a842bc

Microsoft

Exam AA-001_Sandbox

AA-001_Sandbox

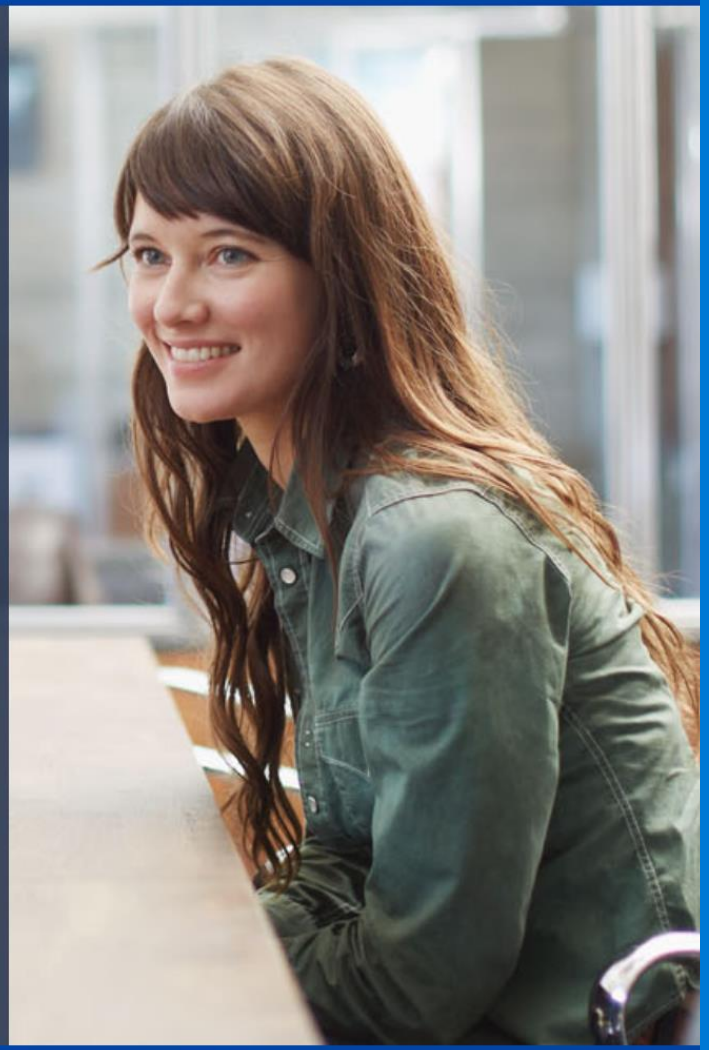
Welcome !

Maximum time for this session, including instructions, survey, and exam: 480 minutes

Number of exam questions: 10

Maximum time for exam: 480 minutes

Minimum score required to pass this exam: 700



Color scheme

Demo the exam experience by visiting our **Exam Sandbox**

Microsoft

Design and implement data storage (40-45%)

Design a data storage structure

• Design an Azure Data Lake solution • Recommend file types for storage • Recommend file types for analytical queries • Design for efficient querying • Design for data pruning • Design a folder structure that represents the levels of data transformation • Design a distribution strategy • Design a data archiving solution

Design a partition strategy

• Design a partition strategy for files • Design a partition strategy for analytical workloads • Design a partition strategy for efficiency/performance • Design a partition strategy for Azure Synapse Analytics • Identify when partitioning is needed in Azure Data Lake Storage Gen2

Design and develop data processing (25–30%)

Design the serving layer

• Design star schemas • Design slowly changing dimensions • Design a dimensional hierarchy • Design a solution for temporal data • Design for incremental loading • Design analytical stores • Design metastores in Azure Synapse Analytics and Azure Databricks

Implement physical data storage structures

• Implement compression • Implement partitioning • Implement sharding • Implement different table geometries with Azure Synapse Analytics pools • Implement data redundancy • Implement distributions • Implement data archiving

Implement logical data structures

• Build a temporal data solution • Build a slowly changing dimension • Build a logical folder structure • Build external tables • Implement file and folder structures for efficient querying and data pruning

Design and implement data security (10-15%)

Implement the serving layer

• Deliver data in a relational star • Deliver data in Parquet files • Maintain metadata • Implement a dimensional hierarchy

Monitor and optimize data storage and data processing (10-15%)

Microsoft

Design and implement data storage (40-45%)

Design and develop data processing (25–30%)

Design and implement data security (10-15%)

Monitor and optimize data storage and data processing (10-15%)

Ingest and transform data

• Transform data by using Apache Spark • Transform data by using Transact-SQL • Transform data by using Data Factory • Transform data by using Azure Synapse Pipelines • Transform data by using Stream Analytics • Cleanse data • Split data • Shred JSON • Encode and decode data • Configure error handling for the transformation • Normalize and denormalize values • Transform data by using Scala • Perform data exploratory analysis

Design and develop a batch processing solution

• Develop batch processing solutions by using Data Factory, Data Lake, Spark, Azure Synapse Pipelines, PolyBase, and Azure Databricks • Create data pipelines • Design and implement incremental data loads • Design and develop slowly changing dimensions • Handle security and compliance requirements • Scale resources • Configure the batch size • Design and create tests for data pipelines • Integrate Jupyter/Python notebooks into a data pipeline • Handle duplicate data • Handle missing data • Handle late-arriving data • Upsert data • Regress to a previous state • Design and configure exception handling • Configure batch retention • Design a batch processing solution • Debug Spark jobs by using the Spark UI

Design and develop a stream processing solution

• Develop a stream processing solution by using Stream Analytics, Azure Databricks, and Azure Event Hubs • Process data by using Spark structured streaming • Monitor for performance and functional regressions • Design and create windowed aggregates • Handle schema drift • Process time series data • Process across partitions • Process within one partition • Configure checkpoints/watermarking during processing • Scale resources • Design and create tests for data pipelines • Optimize pipelines for analytical or transactional purposes • Handle interruptions • Design and configure exception handling • Upsert data • Replay archived stream data • Design a stream processing solution

Manage batches and pipelines

• Trigger batches • Handle failed batch loads • Validate batch loads • Manage data pipelines in Data Factory/Synapse Pipelines • Schedule data pipelines in Data Factory/Synapse Pipelines • Implement version control for pipeline artifacts • Manage Spark jobs in a pipeline

Microsoft

Design and implement data storage (40-45%)

Design and develop data processing (25–30%)

Design and implement data security (10-15%)

Monitor and optimize data storage and data processing (10-15%)

Design security for data policies and standards

• Design data encryption for data at rest and in transit • Design a data auditing strategy • Design a data masking strategy • Design for data privacy • Design a data retention policy • Design to purge data based on business requirements • Design Azure role-based access control (Azure RBAC) and POSIX-like Access Control List (ACL) for Data Lake Storage Gen2 • Design row-level and column-level security

Implement data security

• Implement data masking • Encrypt data at rest and in motion • Implement row-level and column-level security • Implement Azure RBAC • Implement POSIX-like ACLs for Data Lake Storage Gen2 • Implement a data retention policy • Implement a data auditing strategy • Manage identities, keys, and secrets across different data platform technologies • Implement secure endpoints (private and public) • Implement resource tokens in Azure Databricks • Load a DataFrame with sensitive information • Write encrypted data to tables or Parquet files • Manage sensitive information

Microsoft

Design and implement data storage (40-45%)

Design and develop data processing (25–30%)

Design and implement data security (10-15%)

Monitor and optimize data storage and data processing (10-15%)

Monitor data storage and data processing

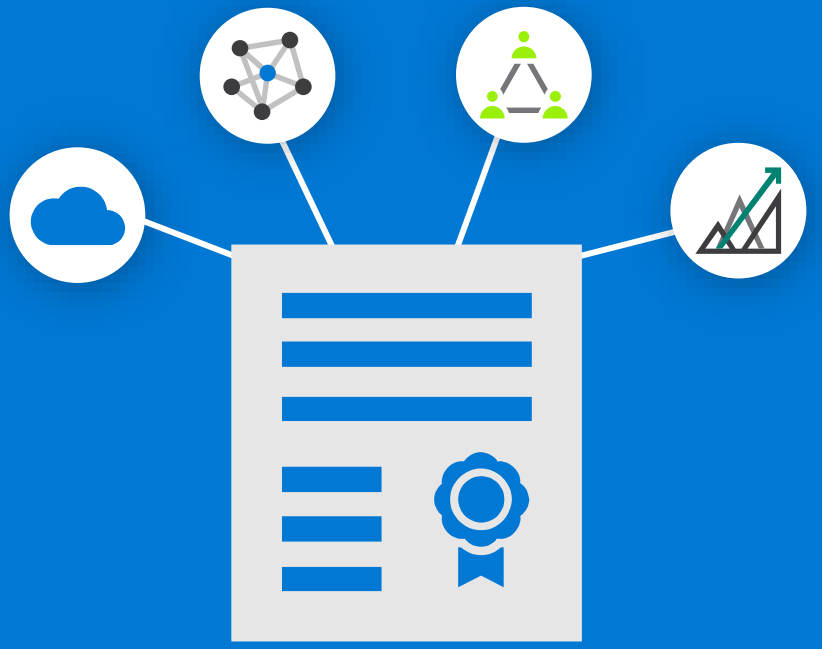
• Implement logging used by Azure Monitor • Configure monitoring services • Measure performance of data movement • Monitor and update statistics about data across a system • Monitor data pipeline performance • Measure query performance • Monitor cluster performance • Understand custom logging options • Schedule and monitor pipeline tests • Interpret Azure Monitor metrics and logs • Interpret a Spark directed acyclic graph (DAG)

Optimize and troubleshoot data storage and data processing

• Compact small files • Rewrite user-defined functions (UDFs) • Handle skew in data • Handle data spill • Tune shuffle partitions • Find shuffling in a pipeline • Optimize resource management • Tune queries by using indexers • Tune queries by using cache • Optimize pipelines for analytical or transactional purposes • Optimize pipeline for descriptive versus analytical workloads • Troubleshoot a failed spark job • Troubleshoot a failed pipeline run

Microsoft

Design and implement data storage

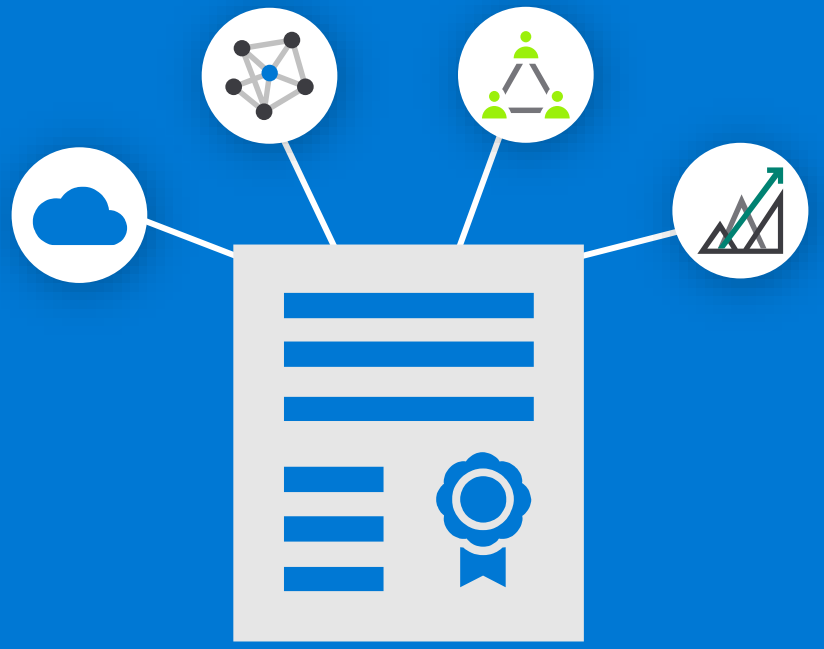


Implement and manage a data analytics environment	Query and transform data	Implement and manage data models	Explore and visualize data

Microsoft

Design and implement data storage

Design a data storage structure



- Design an Azure Data Lake solution
- Recommend file types for storage
- Recommend file types for analytical queries
- Design for efficient querying
- Design for data pruning
- Design a folder structure that represents the levels of data transformation
- Design a distribution strategy
- Design a data archiving solution

Azure Datalake

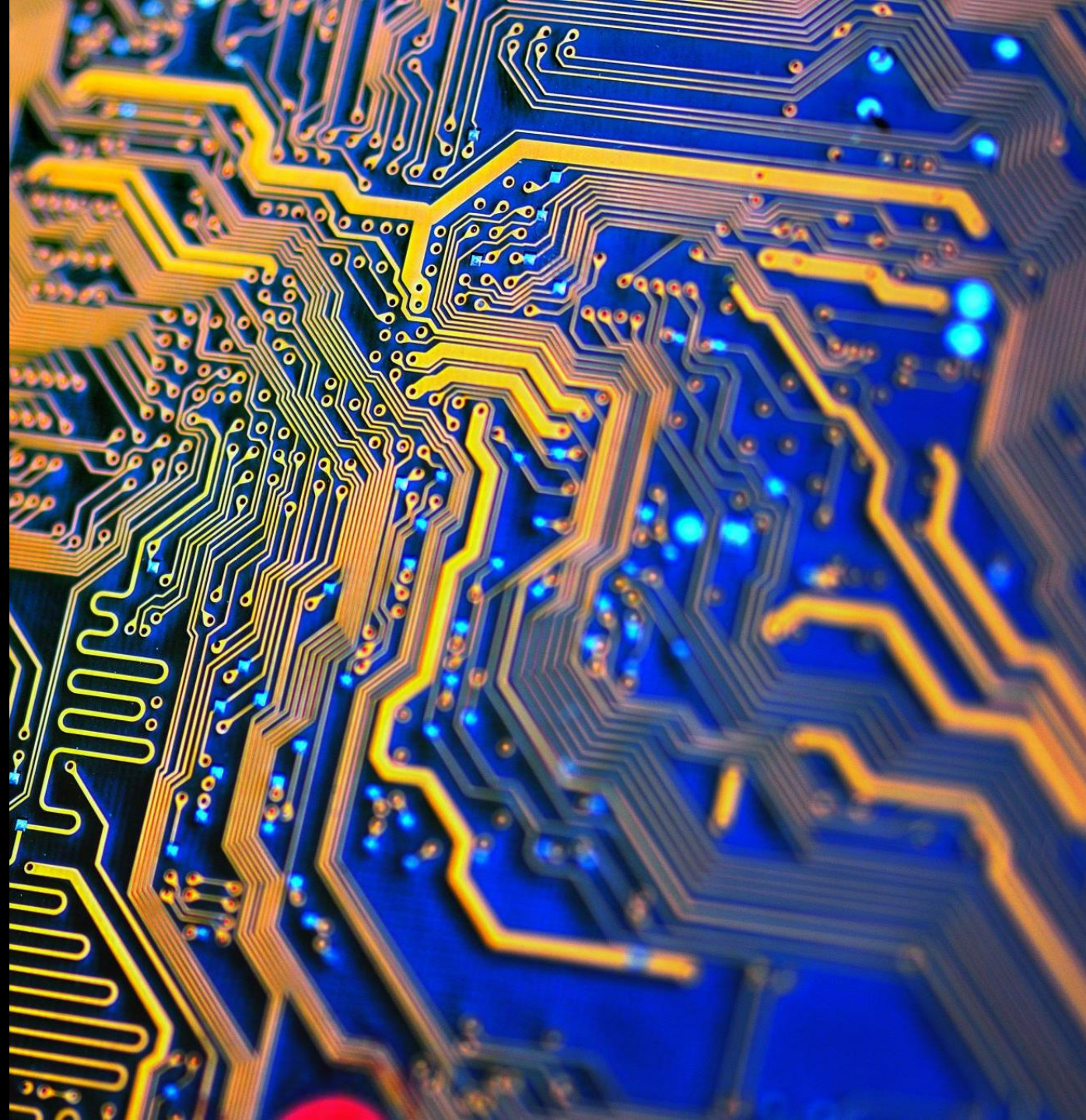
Main Considerations

- ✓ Ingestion Throughput & Data processing
- ✓ Single/multiple data lakes
- ✓ Data partitionning & File structure
- ✓ Security & Governance

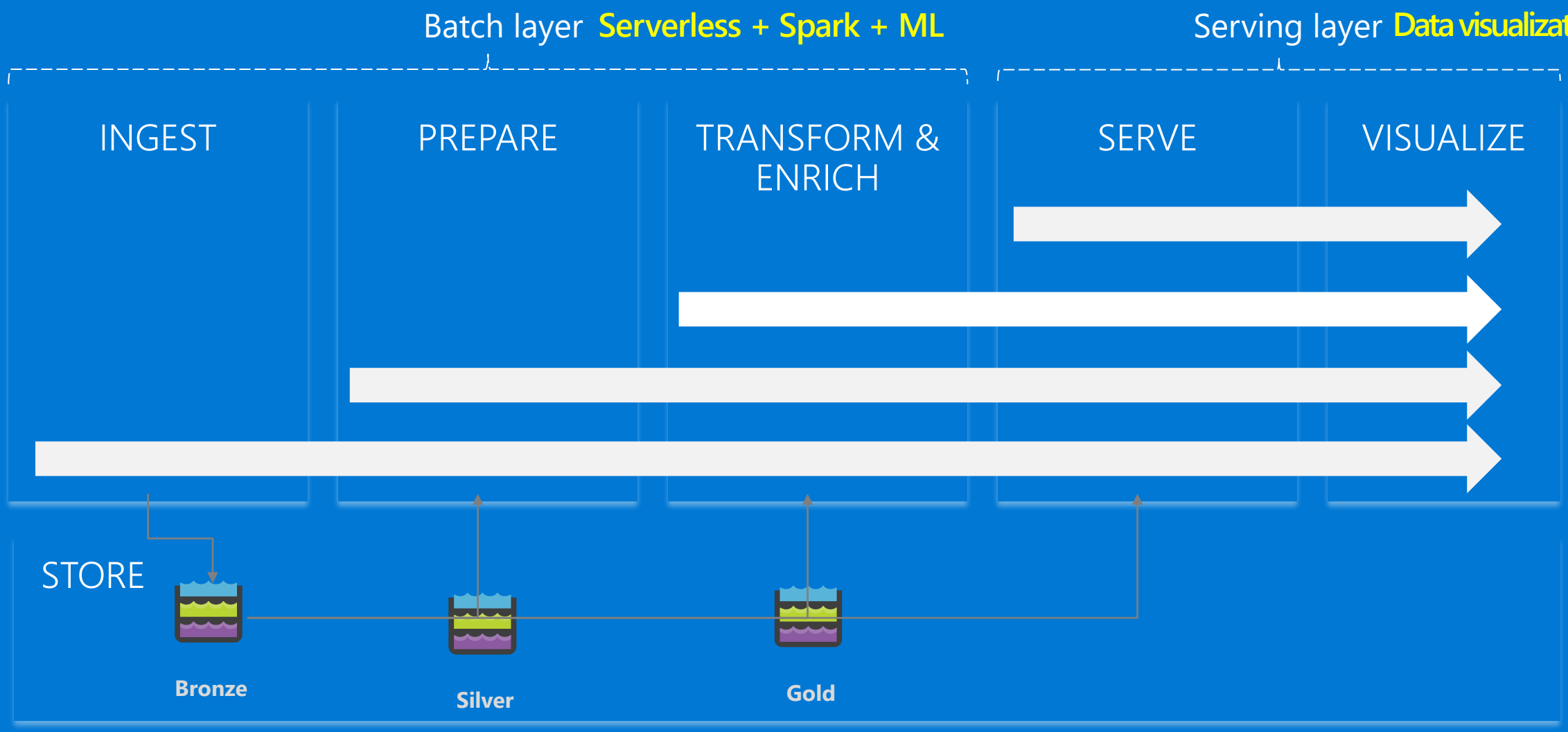
Optimize

Ingestion Throughput &

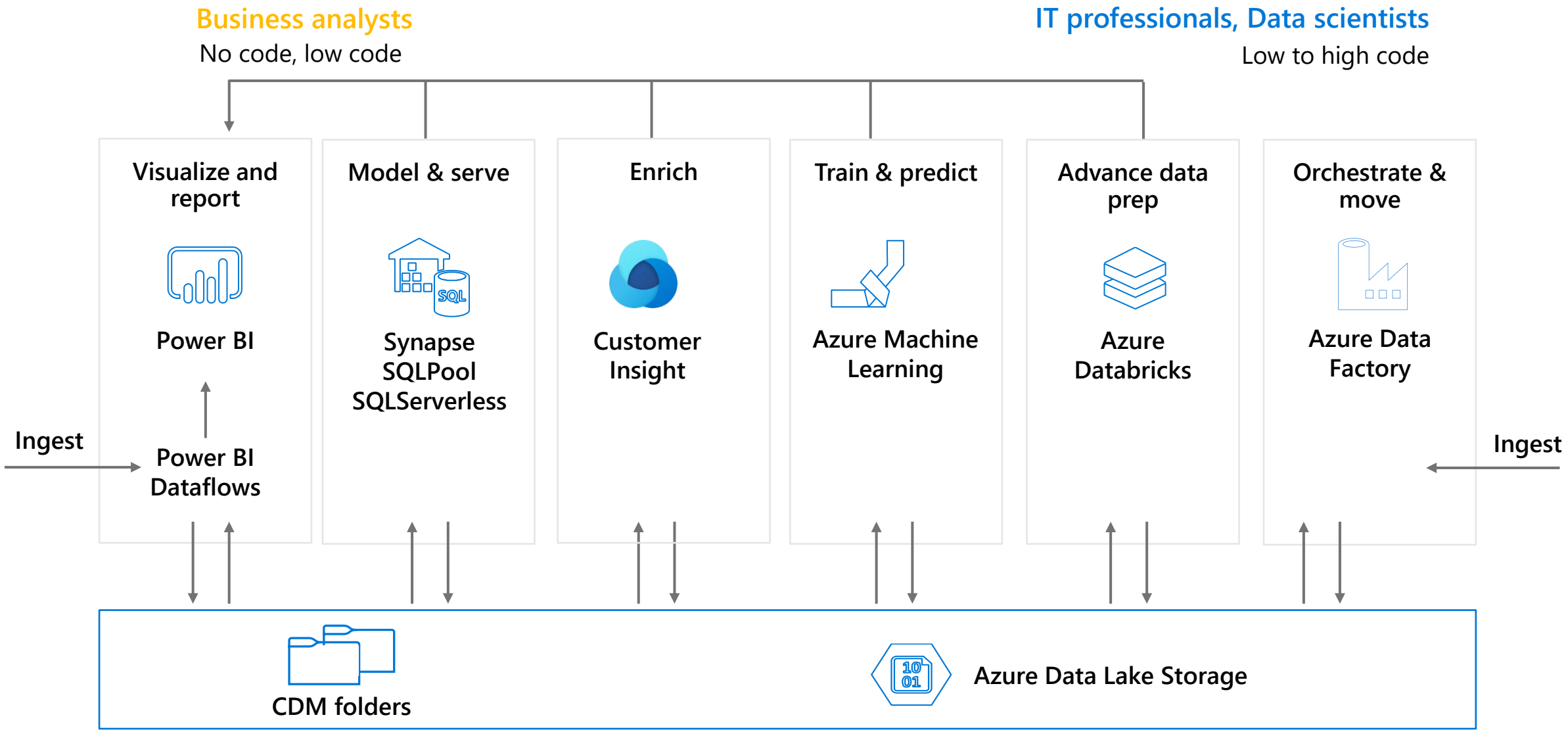
Data processing



Design an Azure Data Lake solution

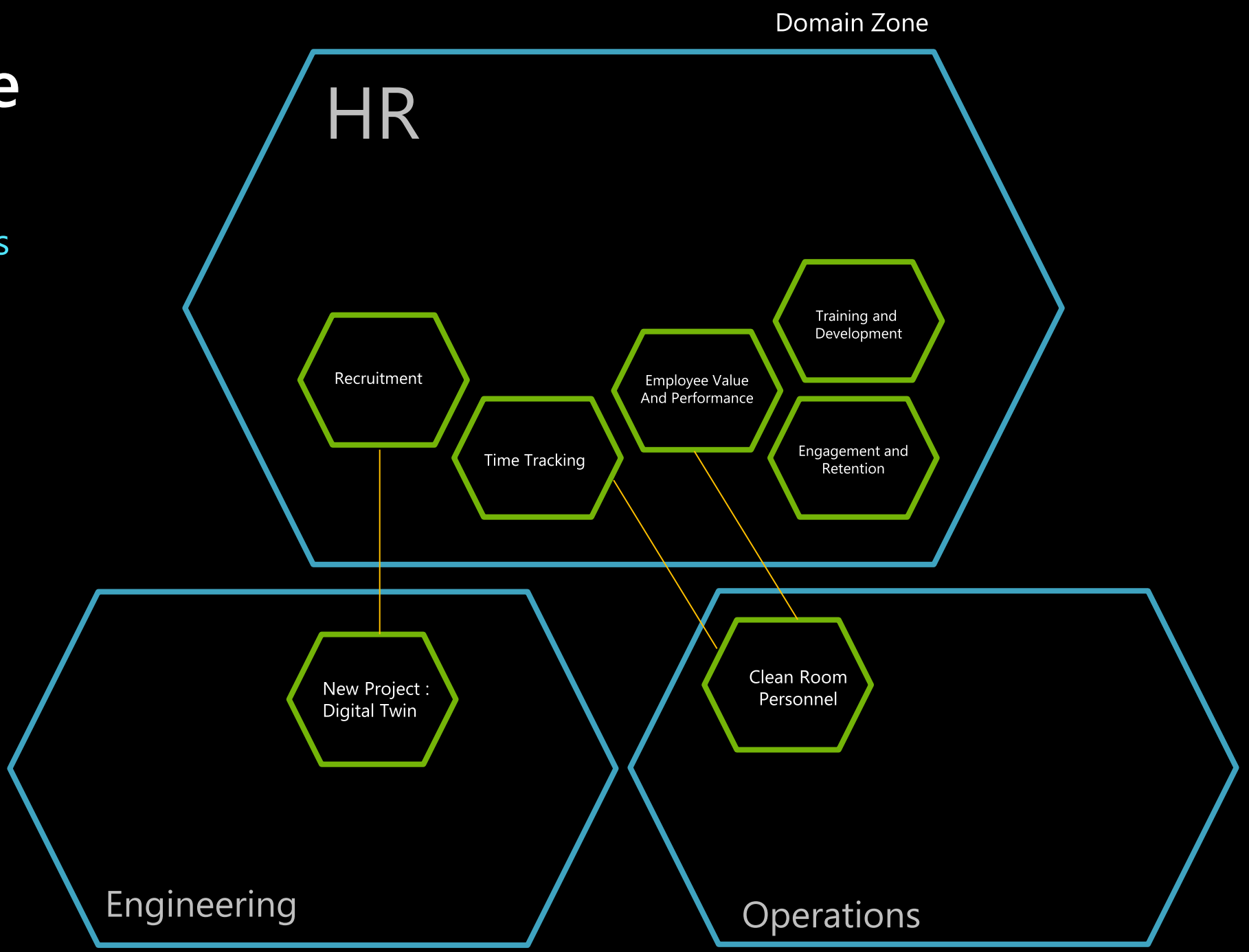


Data Producer / Data consumer

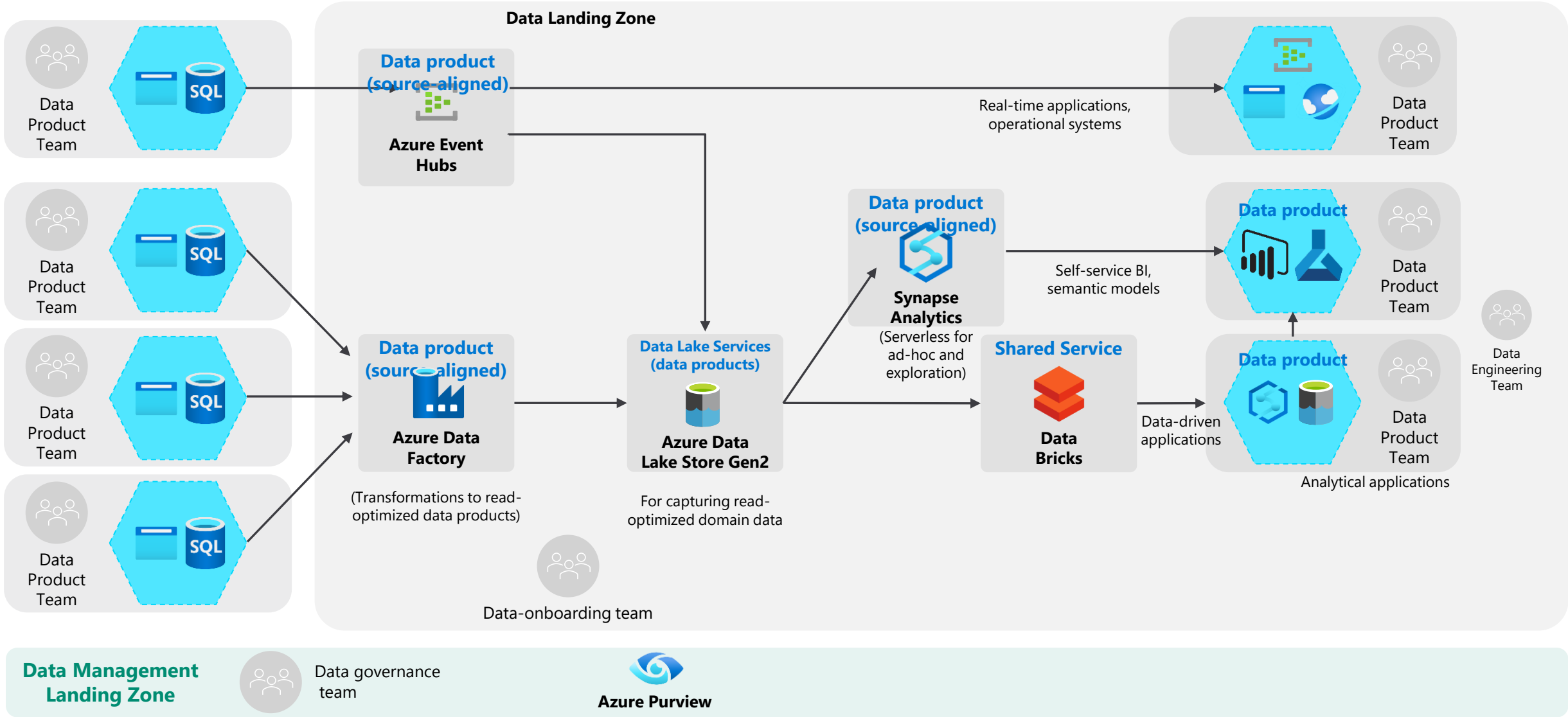


Single/multiple data lakes

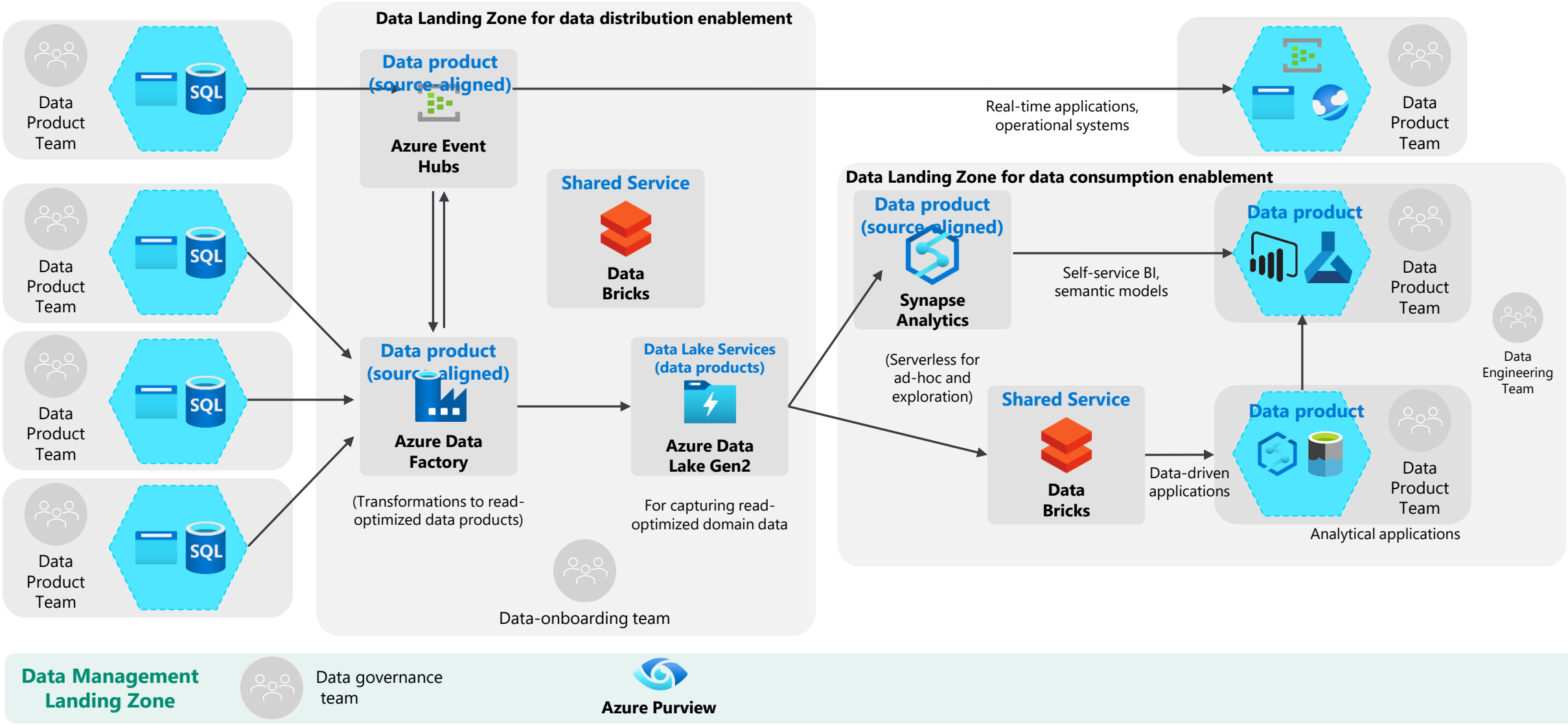
Data Mesh / Data Products



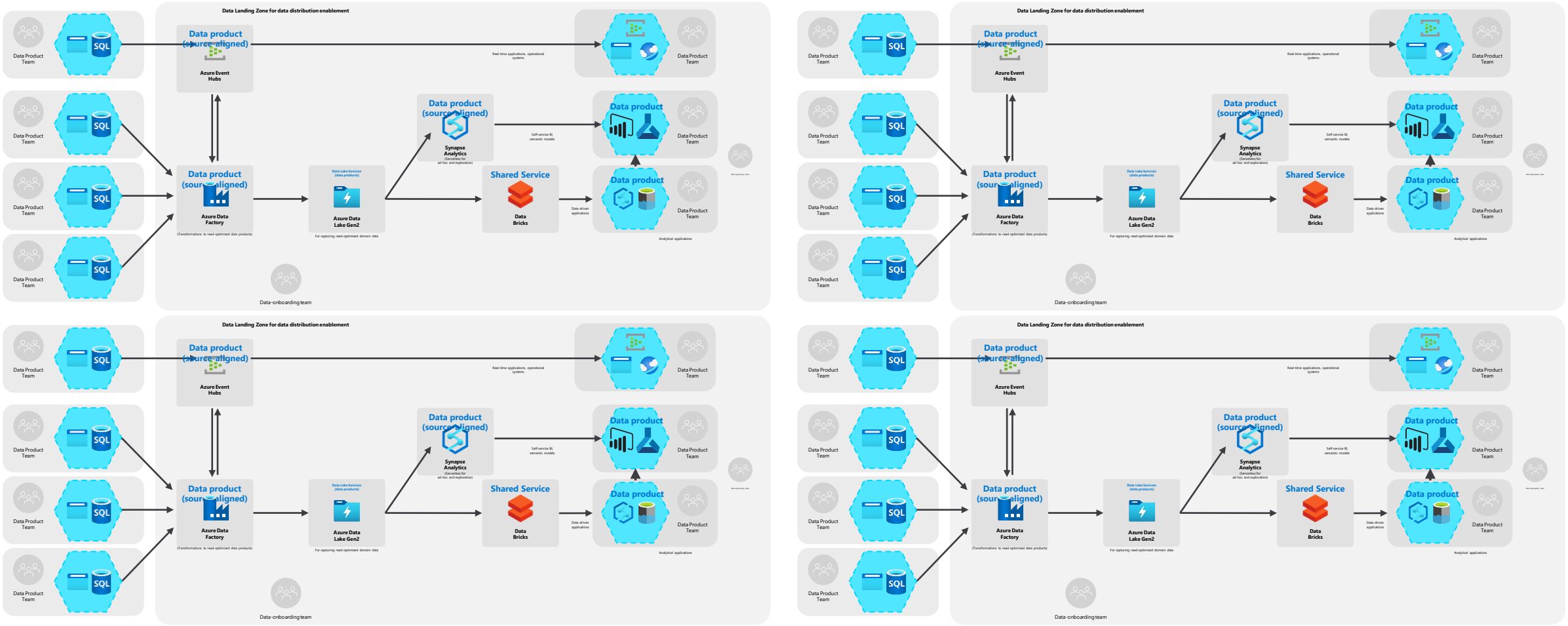
Example reference architecture for governed mesh; small-sized company



Example reference architecture for governed mesh; using landing zones to optimize distribution and consumption of data



Example reference architecture for harmonized mesh; using landing zones for larger domains

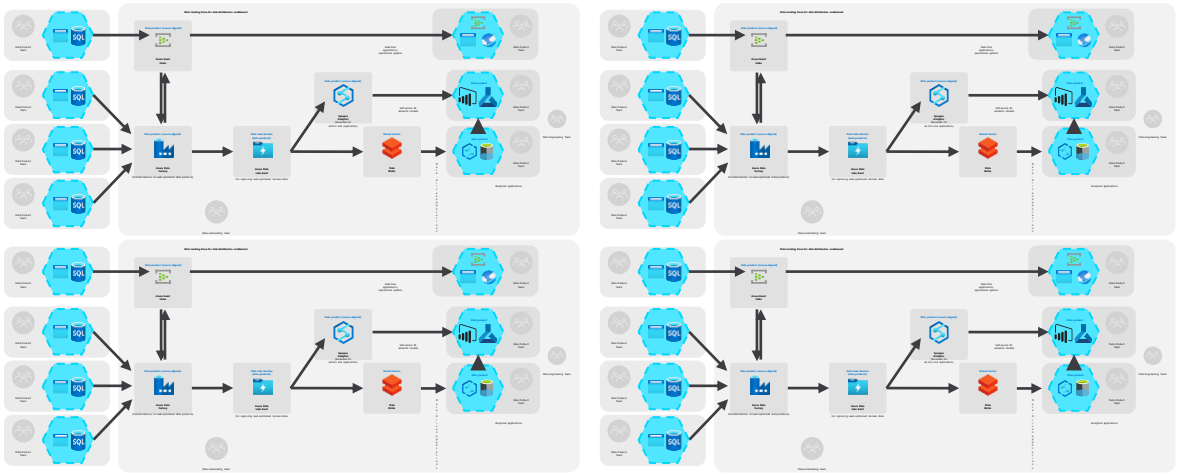


Data Management Landing Zone

Data governance team

Azure Purview

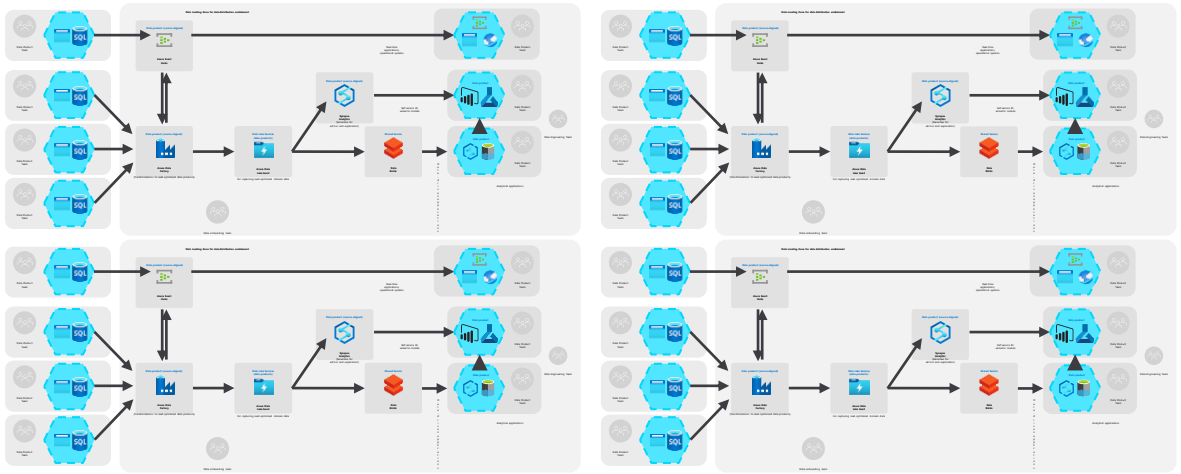
Example reference architecture for harmonized mesh; using multiple data management landing zones and many data landing zones to a world-wide distributed organization



Data Management Landing Zone

Data governance team

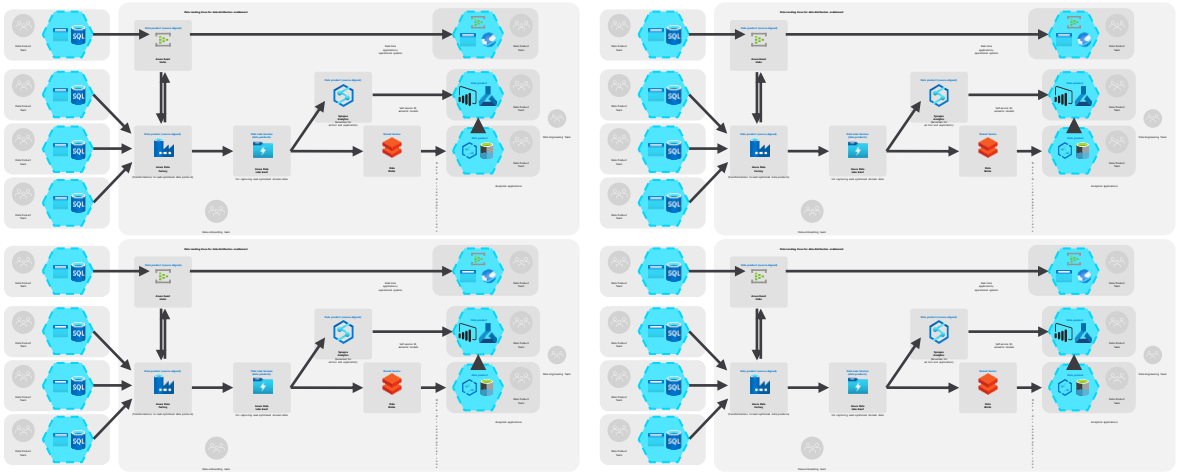
Azure Purview



Data Management Landing Zone

Data governance team

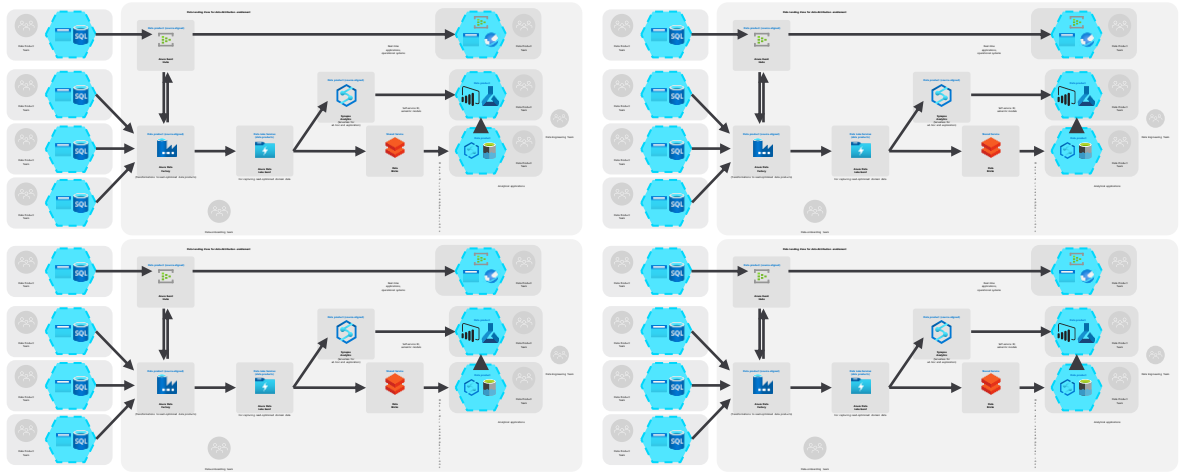
Azure Purview



Data Management Landing Zone

Data governance team

Azure Purview



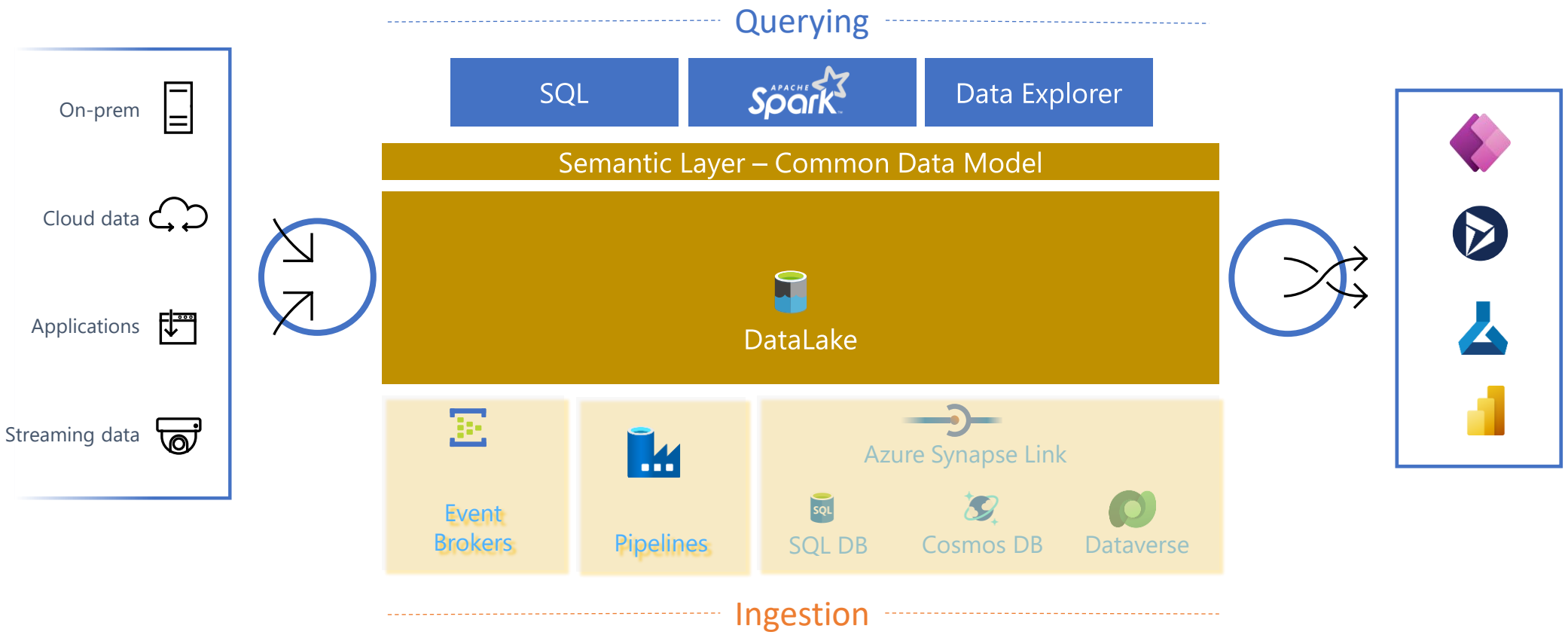
Data Management Landing Zone

Data governance team

Azure Purview

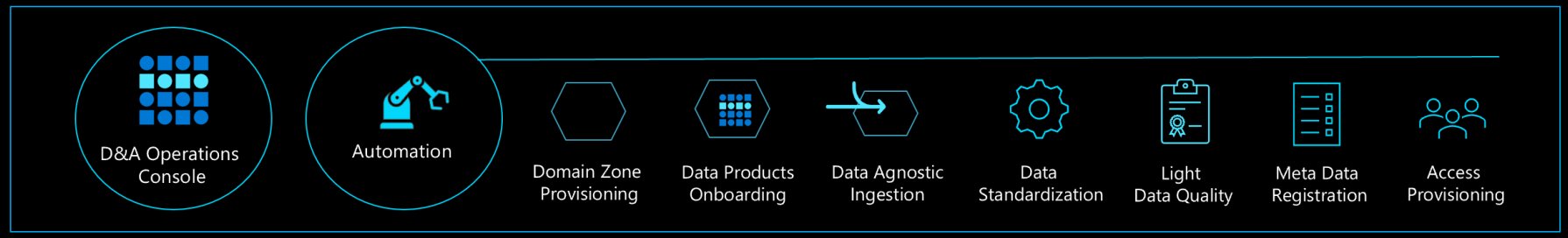
Data partitioning & File structure

Ingestion <> Query partitionning



Example: Automated Service

Based on MS Digital Implementation



On-premise or Other Cloud Provider

Data Sources

Data Pipeline

Azure

Data Agnostic Ingestion Service

Pull

Push

Landing

Standardized

Host

General

Sensitive

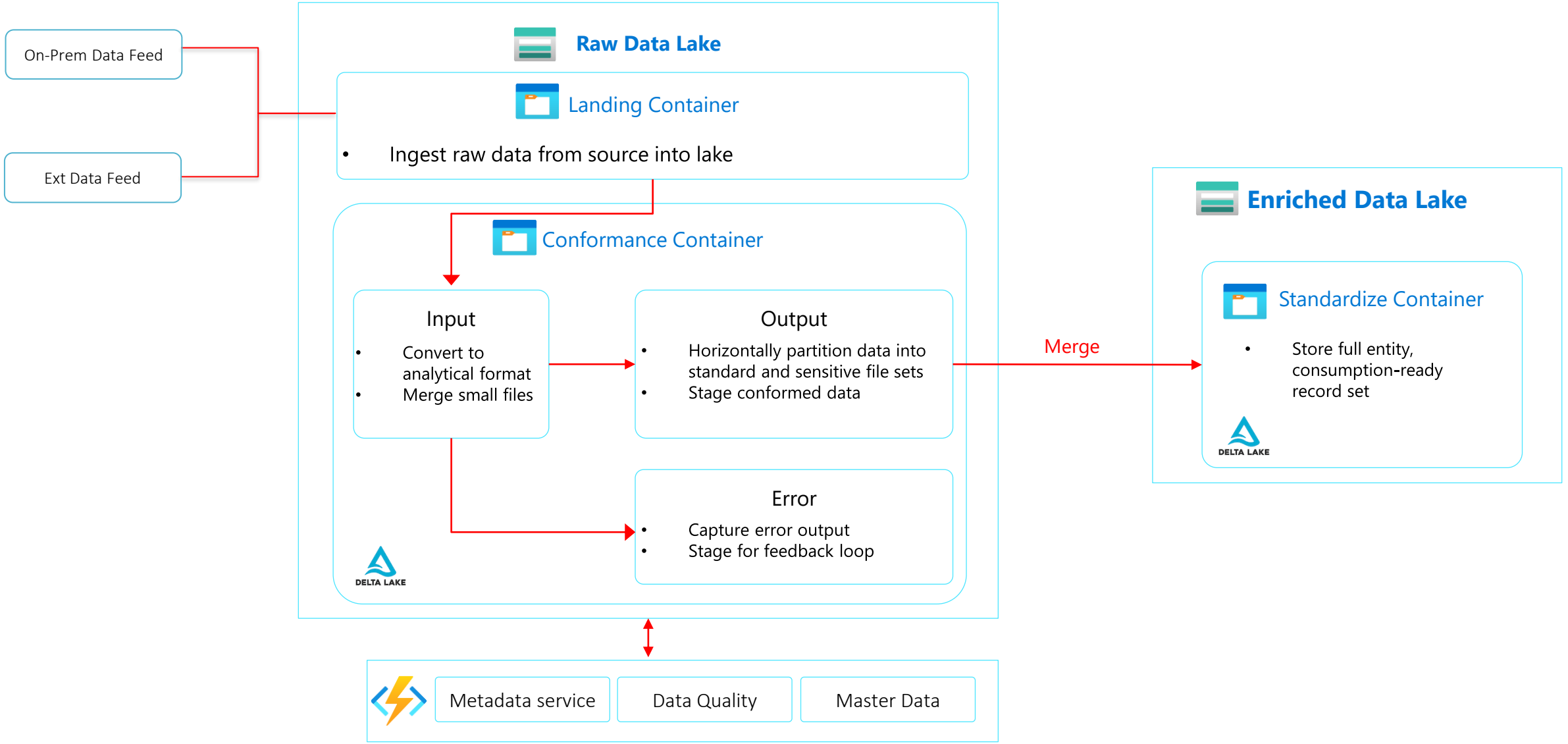
Drop Zone

Note: Containers above are for illustration only.
Detailed taxonomy for production data operational purposes will be implemented

- Data Agnostic Ingestion Service
  - Pull\Push
  - Format Agnostic
  - Data Agnostic
- Data Standardization Service
  - Analytics Format Conversion
  - Versioning
  - Merging
  - PII Handling
  - Data Quality
  - Common Data Model Transformation
  - Master Data Unification
  - Synchronous Processing
  - GPDR

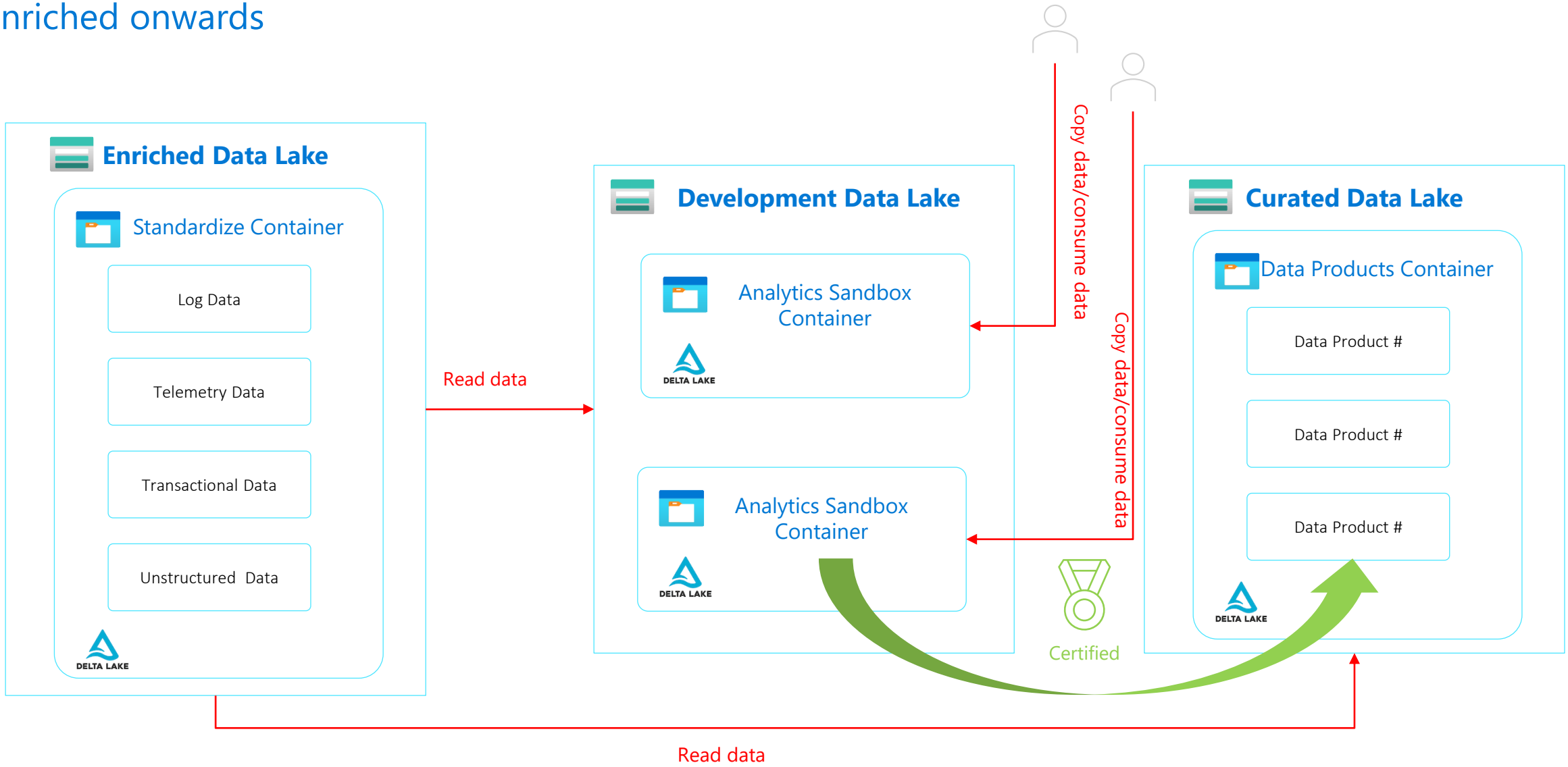
Data Standardization

Raw to Enriched



Data Standardization

Enriched onwards



Best Practice Throughput ingestion <> processing

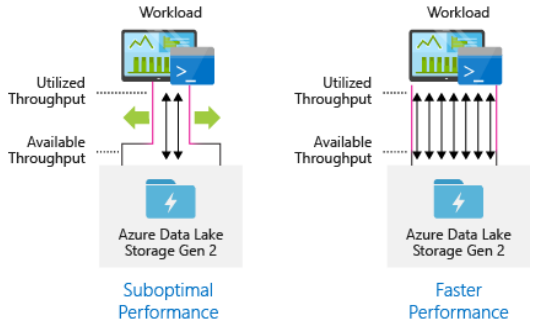
Premium tier for Azure Data Lake Storage

premium block blob storage account

+

Azure Data Lake Storage Gen2

Configure data ingestion tools for maximum parallelization



Tool	Settings
DistCp	-m (mapper)
Azure Data Factory	parallelCopies
Sqoop	fs.azure.block.size, -m (mapper)

Directory structure

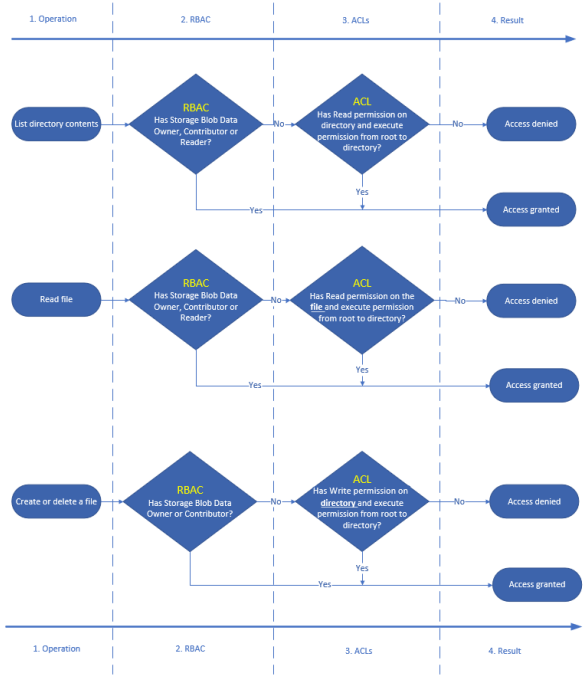
A general template to consider might be the following layout:

{Region}/{SubjectMatter(s)}/{yyyy}/{mm}/{dd}/{hh}/

by putting the date at the end of the directory structure, you can use ACLs to more easily secure regions and subject matters to specific users and groups.

partition pruning of time-series data can help some queries read only a subset of the data, which improves performance.

Security



File Format

Avro, Parquet, and Optimized Row Columnar (ORC) format.

Avro stores data in a row-based format
-> Consider using the Avro file format in cases where your I/O patterns are more write heavy, or the query patterns favor retrieving multiple rows of records in their entirety. For example, the Avro format works well with a message bus such as Event Hubs or Kafka that write multiple events/messages in succession

Parquet and ORC formats store data in a columnar format
Consider Parquet and ORC file formats when the I/O patterns are more read heavy or when the query patterns are focused on a subset of columns in the records. Read transactions can be optimized to retrieve specific columns instead of reading the entire record

File size

organize your data into larger sized files for better performance (256 MB to 100 GB in size)
Increasing file size can also reduce transaction costs. Read and write operations are billed in 4-megabyte increments so you're charged for operation whether or not the file contains 4 megabytes or only a few kilobytes

Data ingestion et querying partitioning

Synapse + Power BI

- Data Integration
- Data Engineering
- Data Warehousing
- Data Science
- Observational Analytics
- Business Intelligence
- Governance

Data Integration

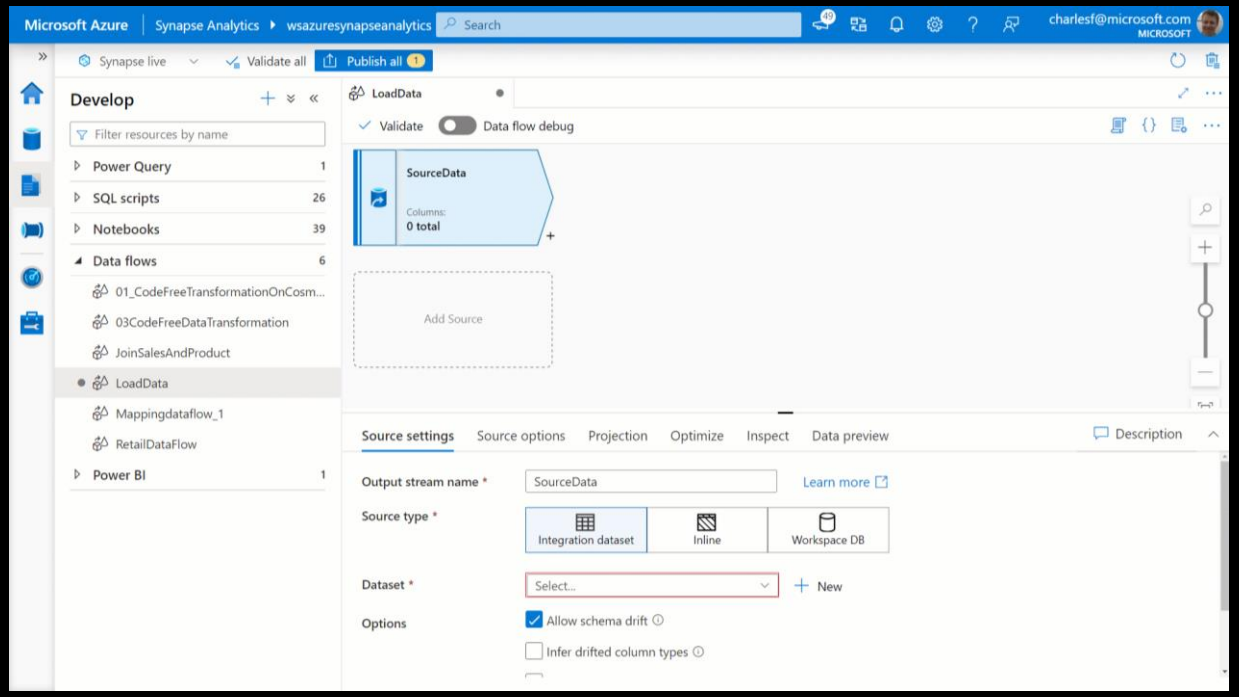
Over 100 connectors to ingest data from a variety of platforms

Integrate from On-Premise, PaaS, and SaaS

Batch and Real-time data integration

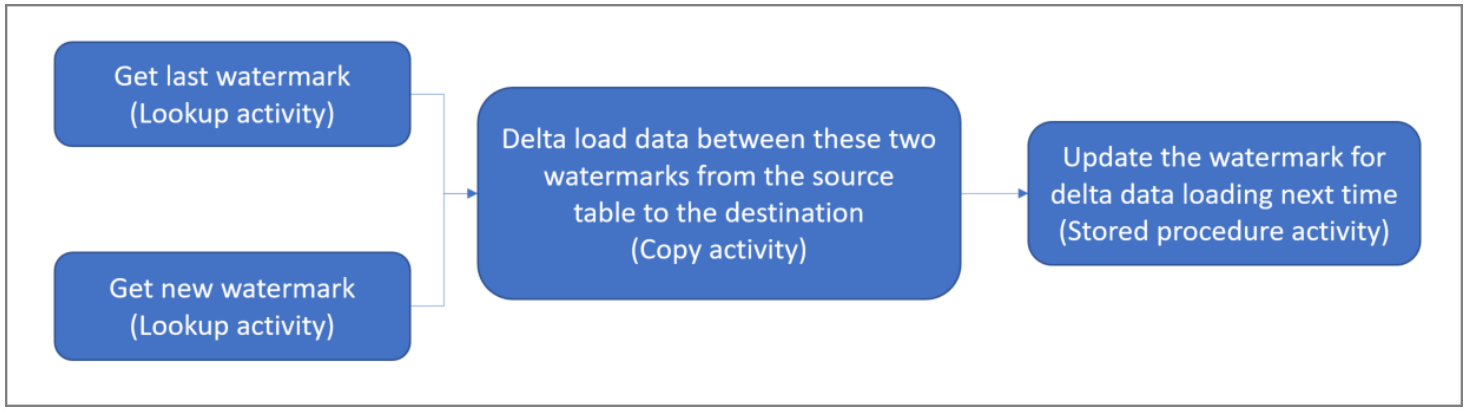
Secure hybrid connectivity

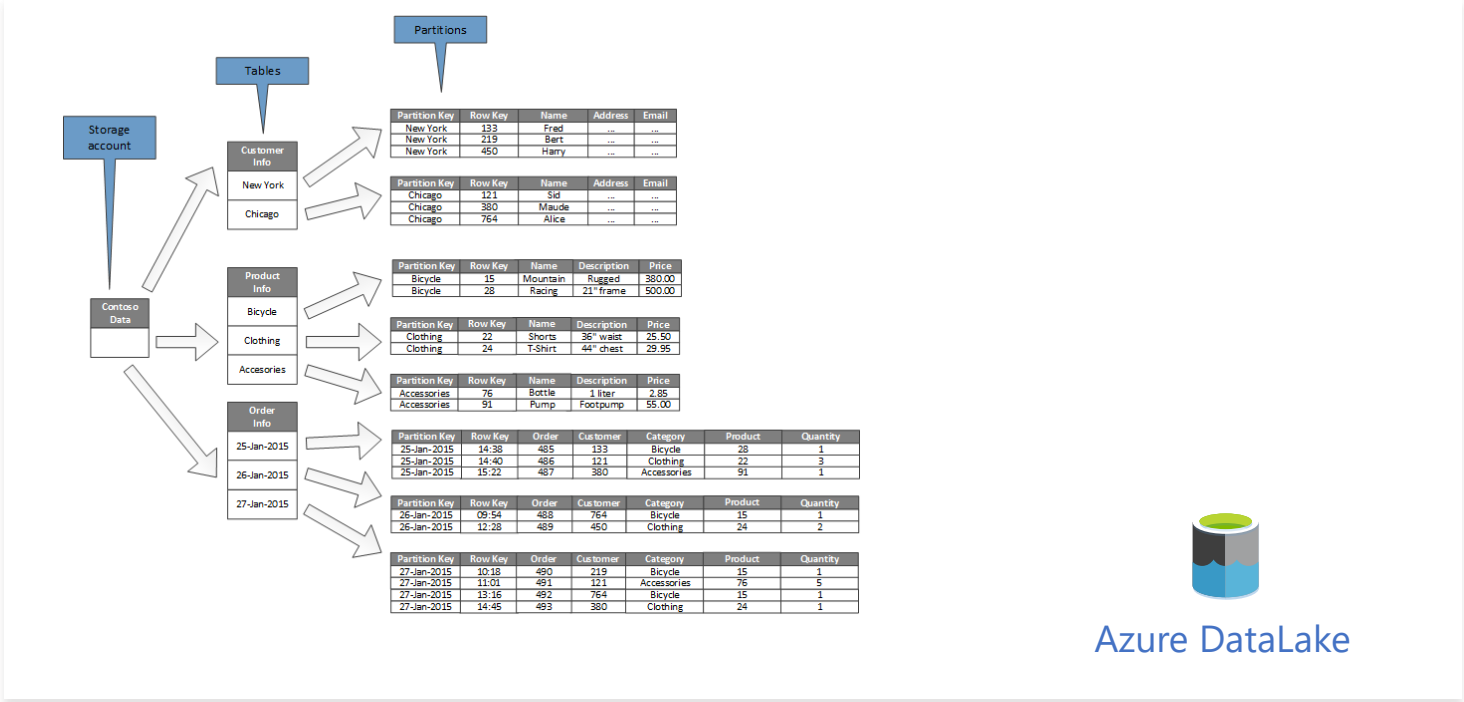
Code-free development environment



Incremental load & Datetime partitionning

Delta data loading from database by using a watermark



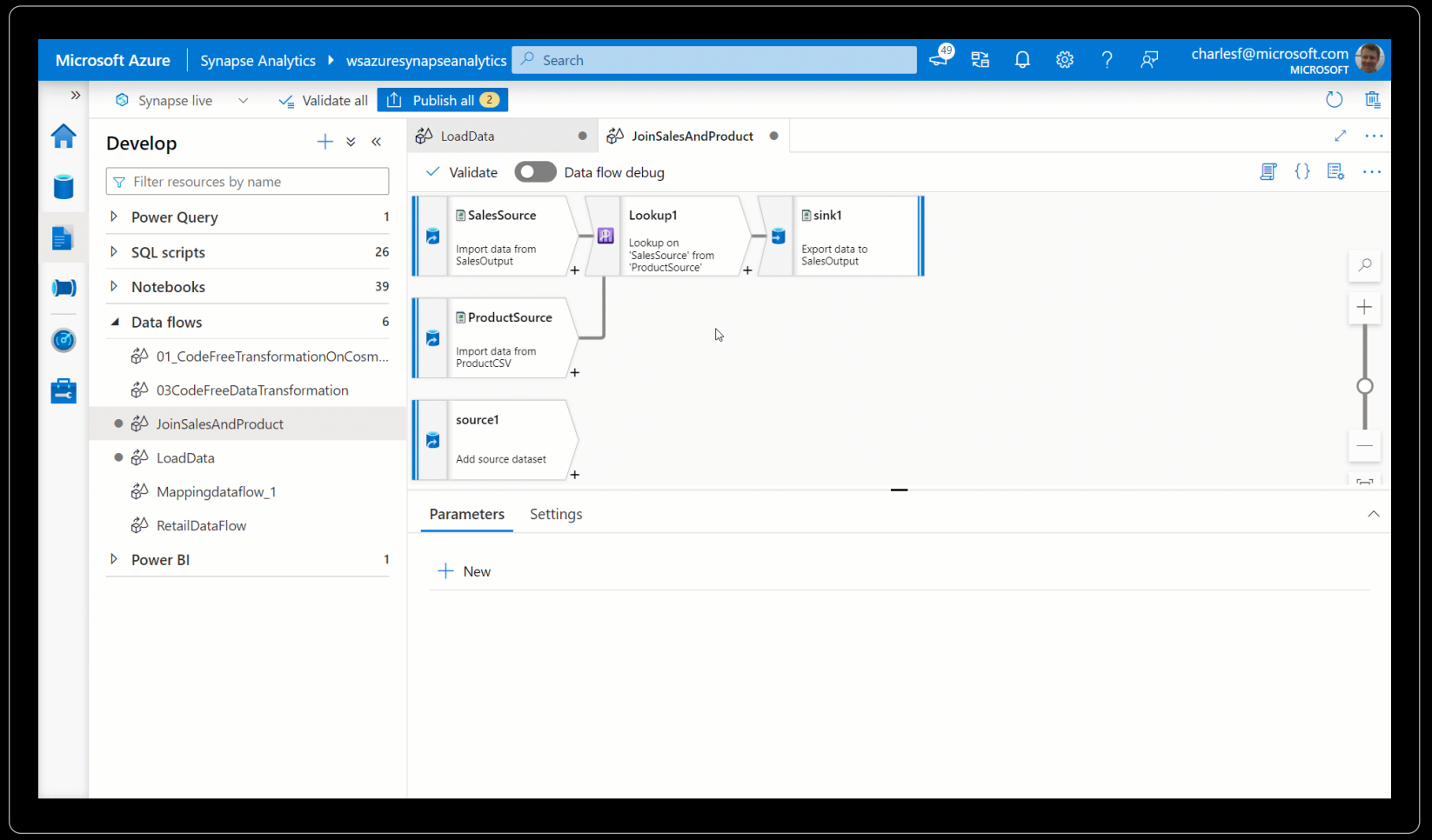


- Prepare the data store to store the watermark value.
- Create a data factory.
- Create linked services.
- Create source, sink, and watermark datasets.
- Create a pipeline.
- Run the pipeline.
- Monitor the pipeline run.

Code-free Data Flows

Enables developers to rapidly integrate data from a variety of sources

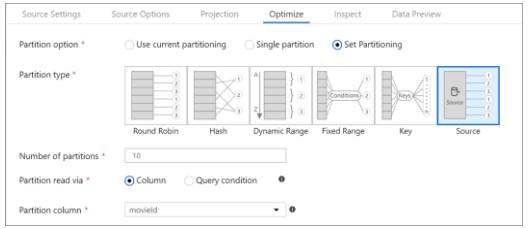
Execute on Spark for large scale processing



Data Flows – low code transformations and key partitioning

Handle upserts, updates, deletes on sql sinks

Add new partition methods



Add schema drift support

Add file handling (move files after read, write files to file names described in rows etc)

New inventory of functions (for e.g. Hash functions for row comparison)

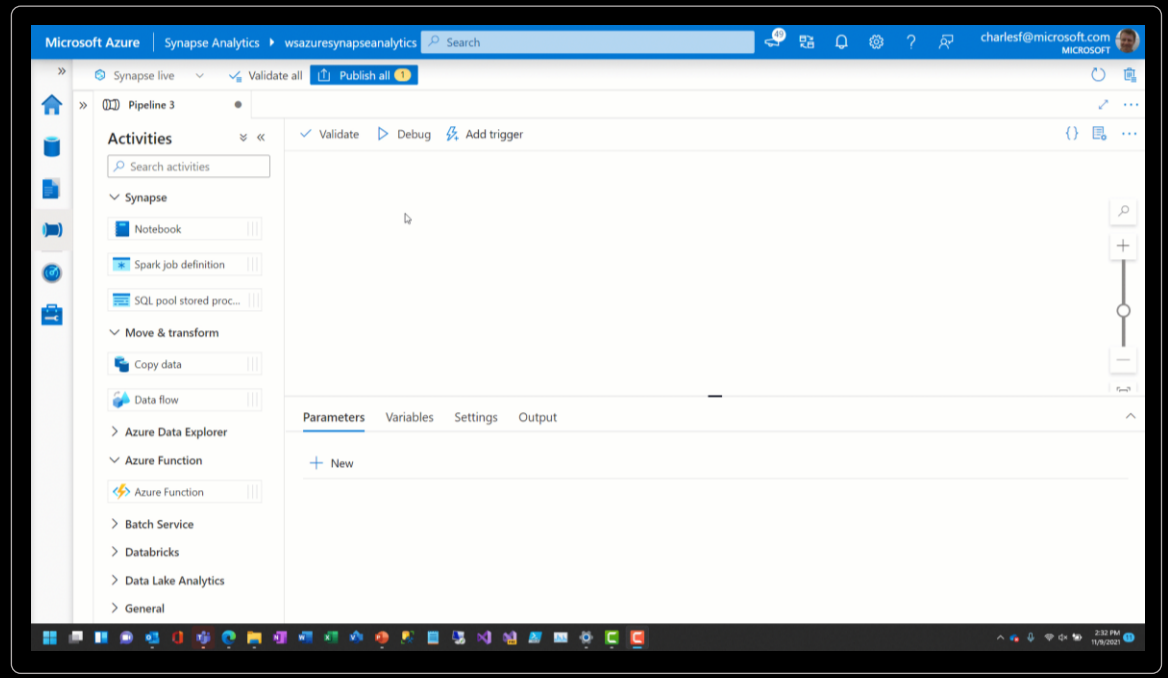
Commonly used ETL patterns(Sequence generator/Lookup transformation/SCD…)

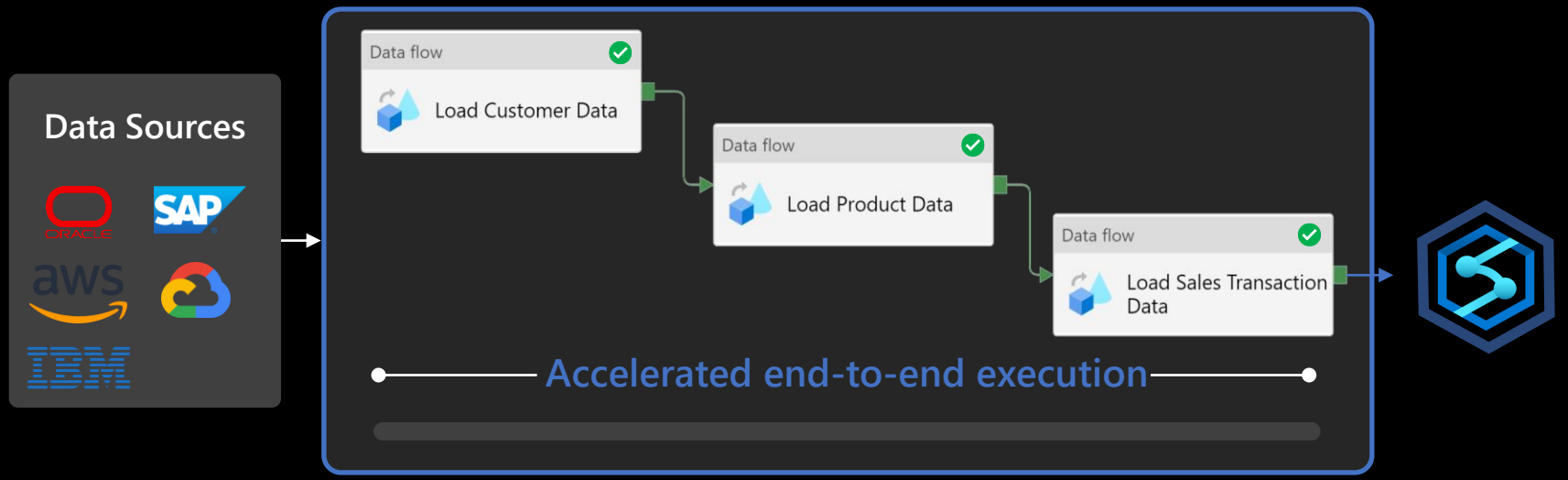
Data lineage – Capturing sink column lineage & impact analysis(invaluable if this is for enterprise deployment)

Implement commonly used ETL patterns as templates(SCD Type1, Type2, Data Vault)

Batch processing - Pipeline Orchestration

Code-free experience for orchestrating a sequence of data integration tasks

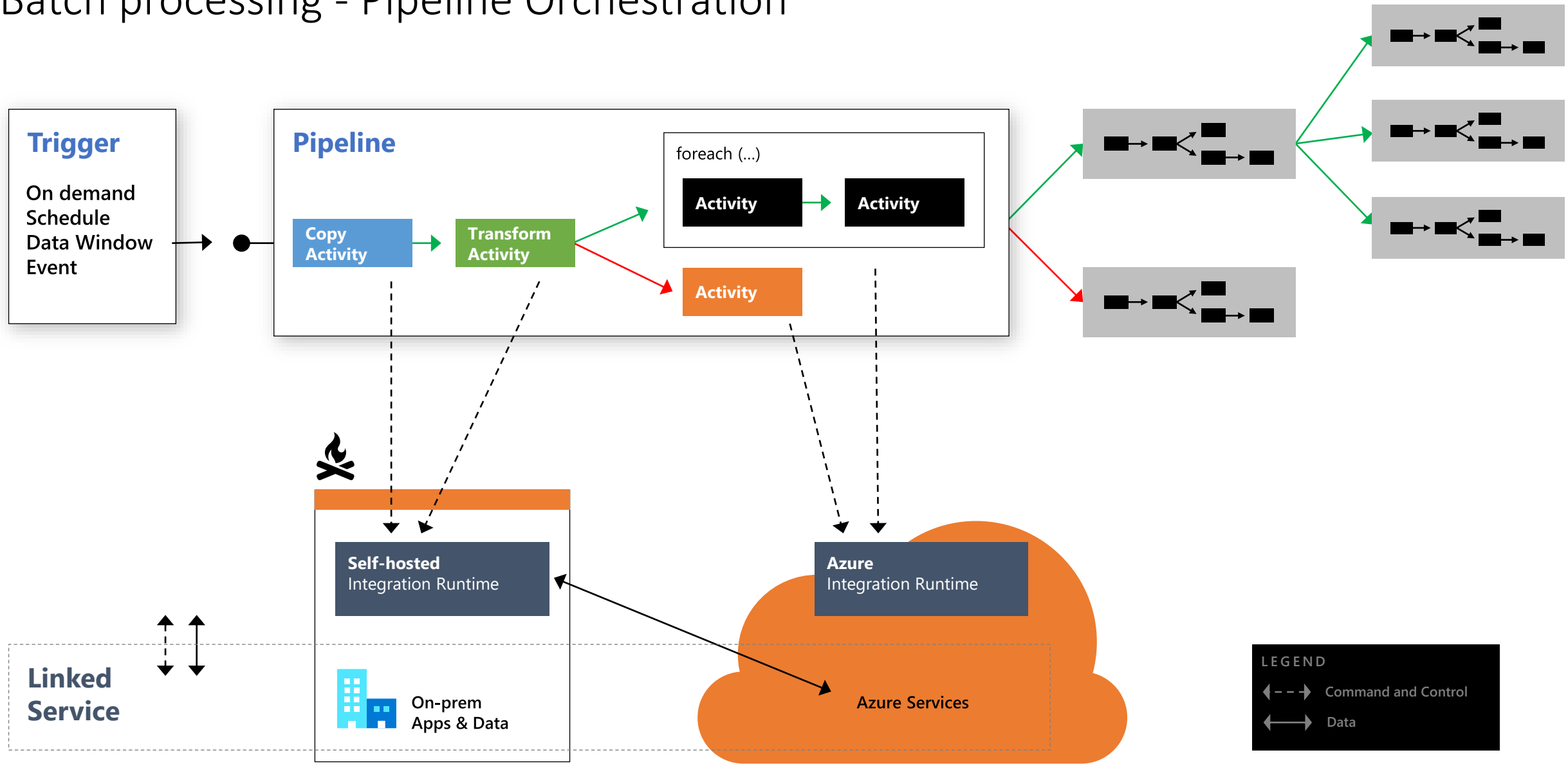




Cluster time-to-live enables near instant start of data flow pipelines for faster data integration

Data is available to the business faster to enable more timely decision making

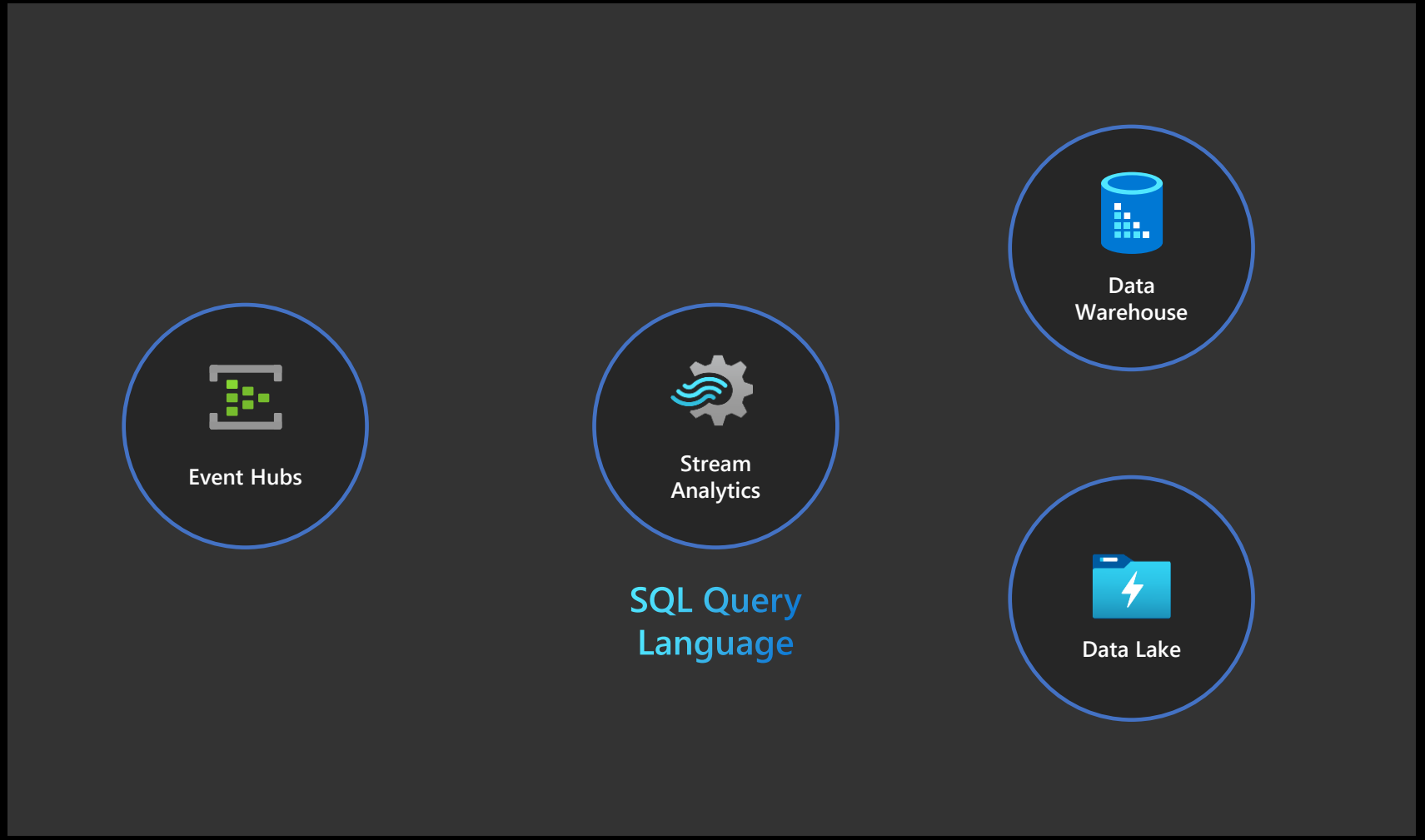
Batch processing - Pipeline Orchestration



Real-time Streaming Data processing

Enables IoT data streams from event brokers to load directly into the data warehouse or data lake

Analyze data in-flight with temporal T-SQL queries in Stream Analytics



Synapse Link for Dataverse

Parquet & Virtual Network Support

Parquet columnar file format optimizes query performance for user queries

Enables customers to apply Virtual Network security to Dataverse connection

Synapse Link for Microsoft SQL

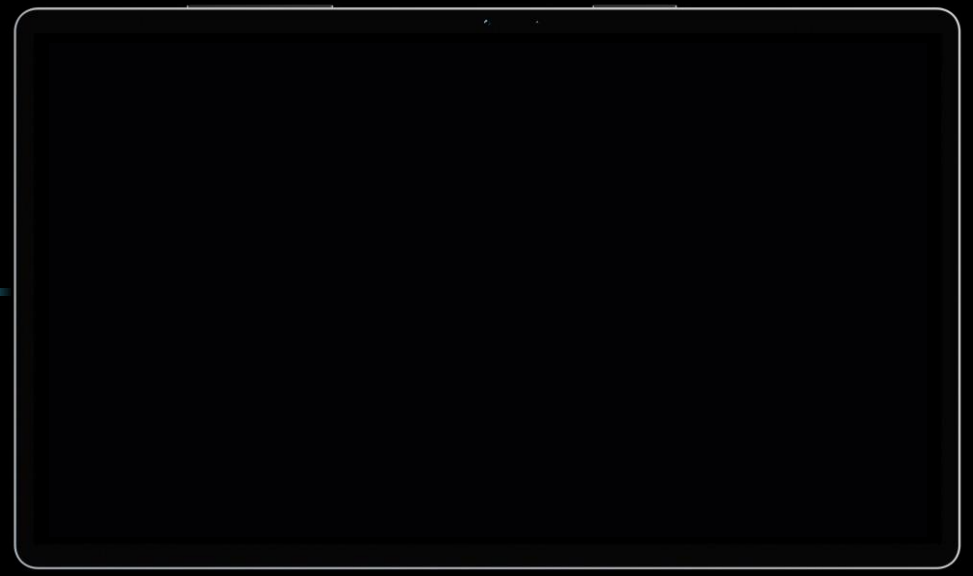
Near real-time operational analytics in Synapse

No data pipelines required

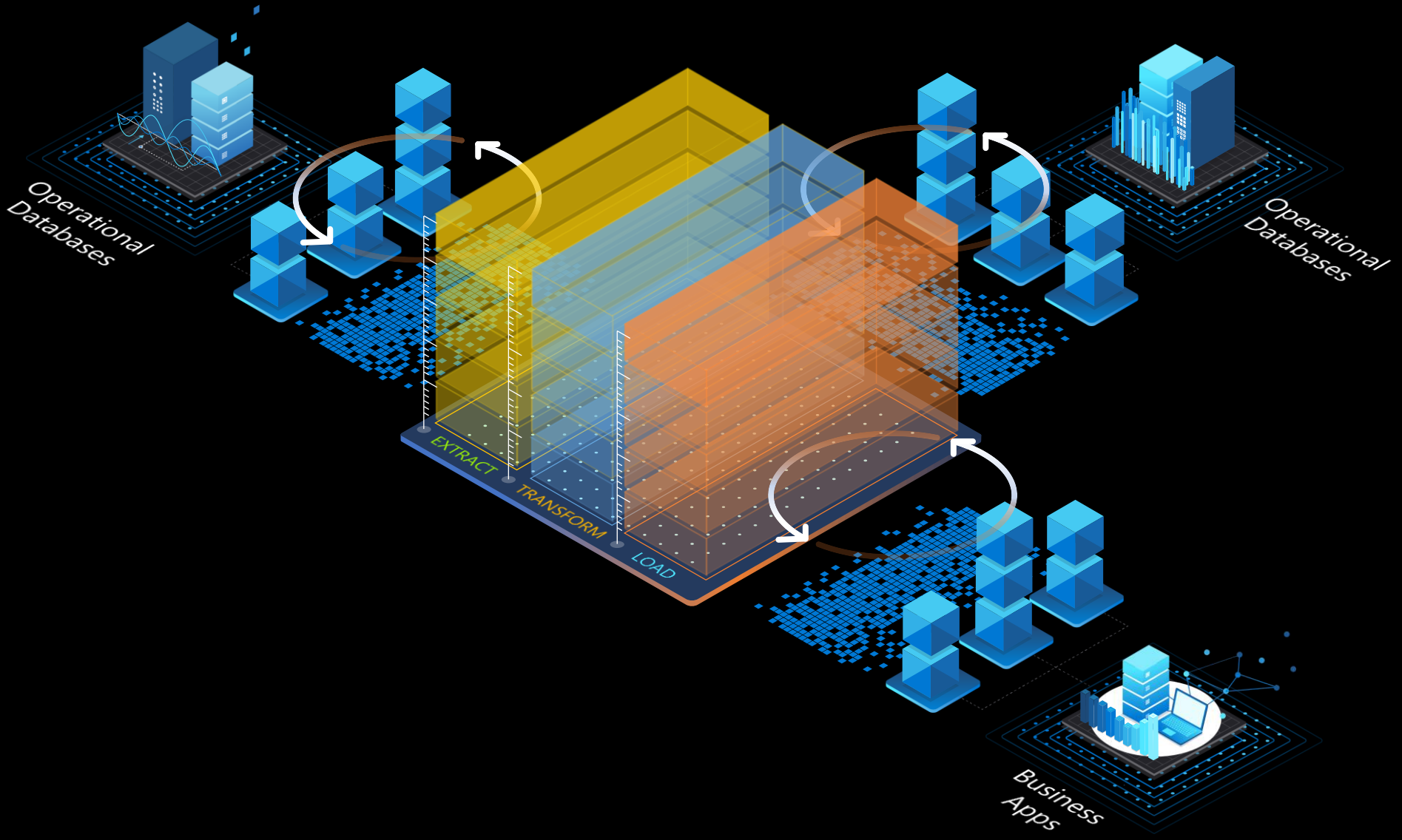
Hybrid integration for SQL Server running on-premise or in other clouds

SQL Server 2022
Public Preview Q2 2022

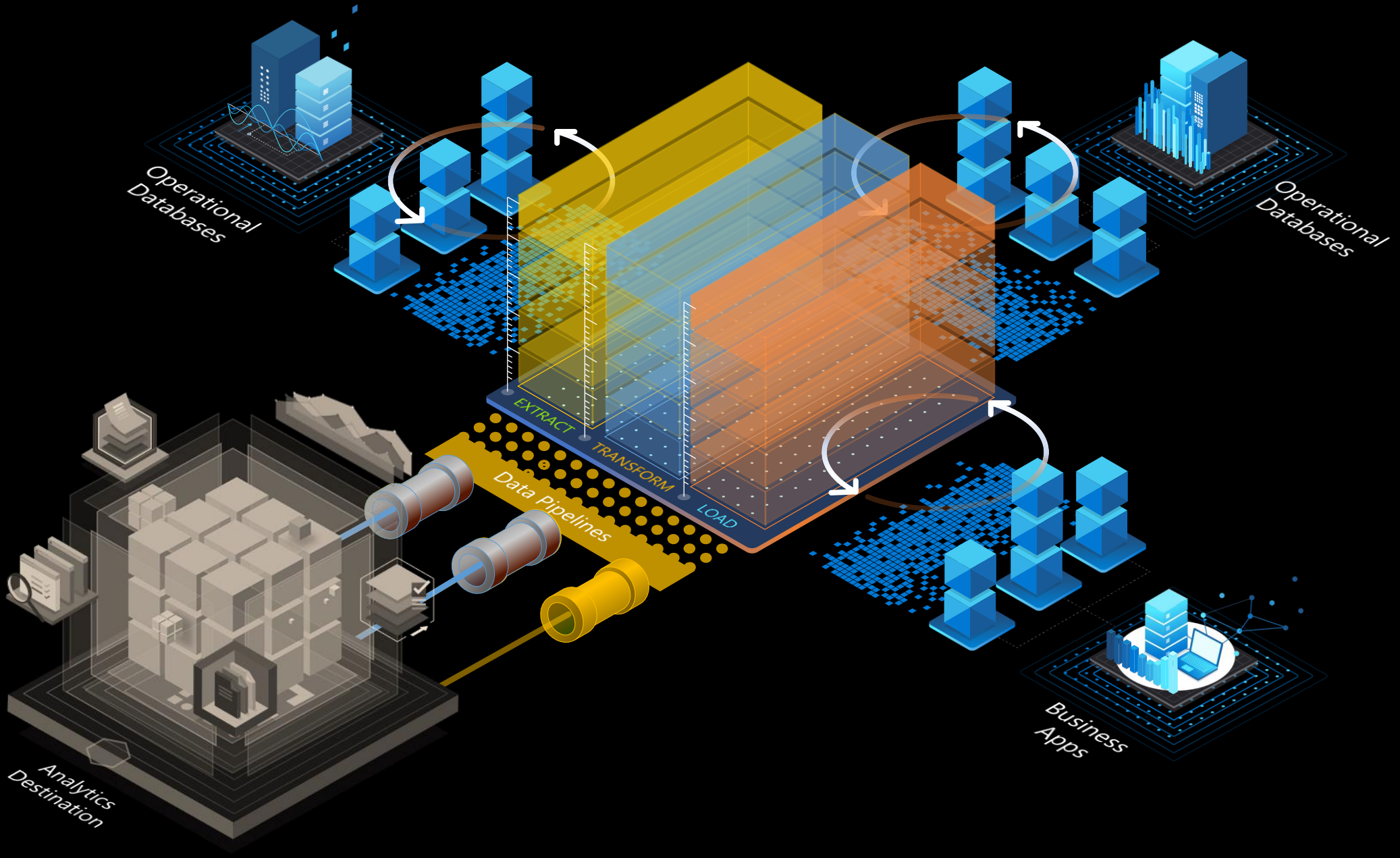
Azure SQL Database
Public Preview Q2 2022



Preparing data to be analytics-ready requires various processes



These processes result in additional issues that stall data insights



Azure Synapse Link

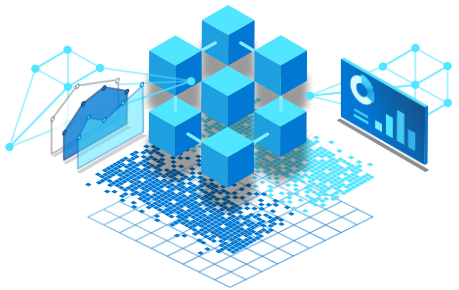
- No managed ETL
- Built for convergence
- Near real-time analytics

Azure Synapse Link

Go from after-the-fact analysis to near real-time insights by eliminating barriers between data and analytics. Automatically move data from both operational databases and business applications without time-consuming ETL. Deliver critical insights by easily connecting separate systems, bringing the power of analytics to every data-connected team.

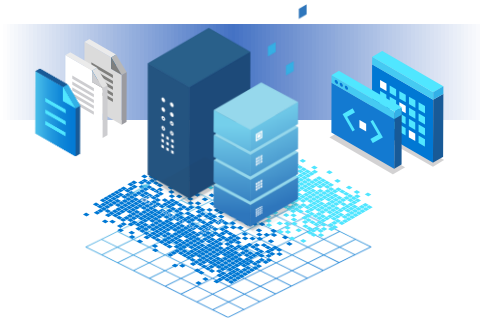
Azure Synapse Link

Azure Synapse Link for SQL



Simplified experience

With a streamlined setup, transfer data from operational systems seamlessly



Centralized data

Consolidate data from multiple sources into a single analytics solution

Near real-time insights

Maximize impact with automated systems, capturing and updating data in near real-time

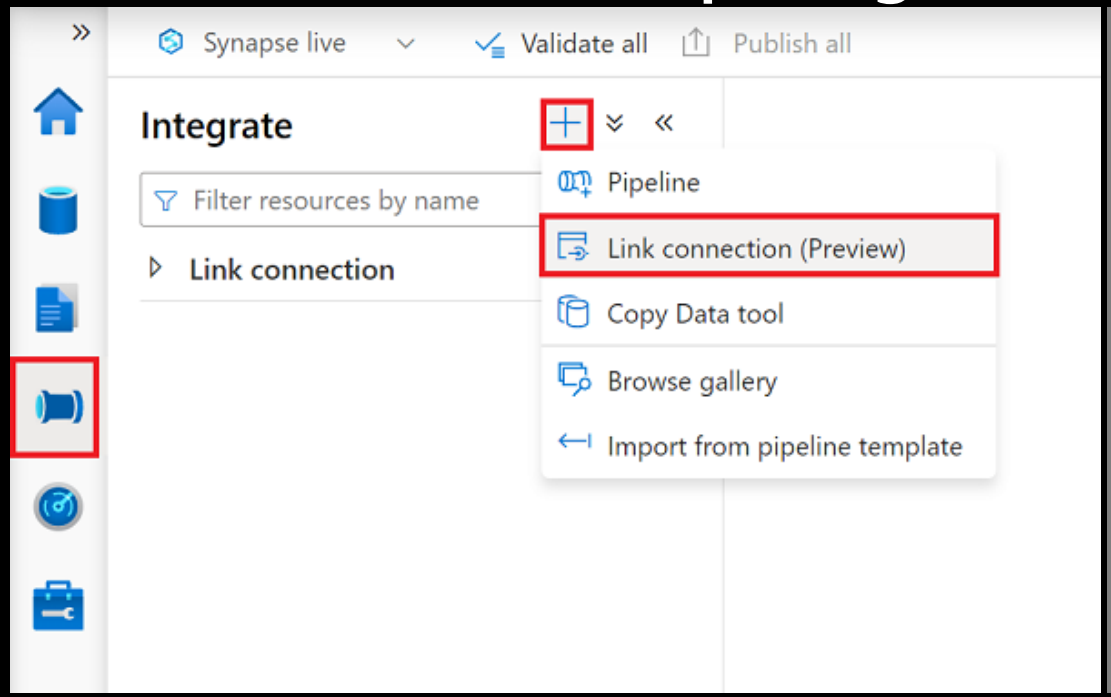
Minimized overhead

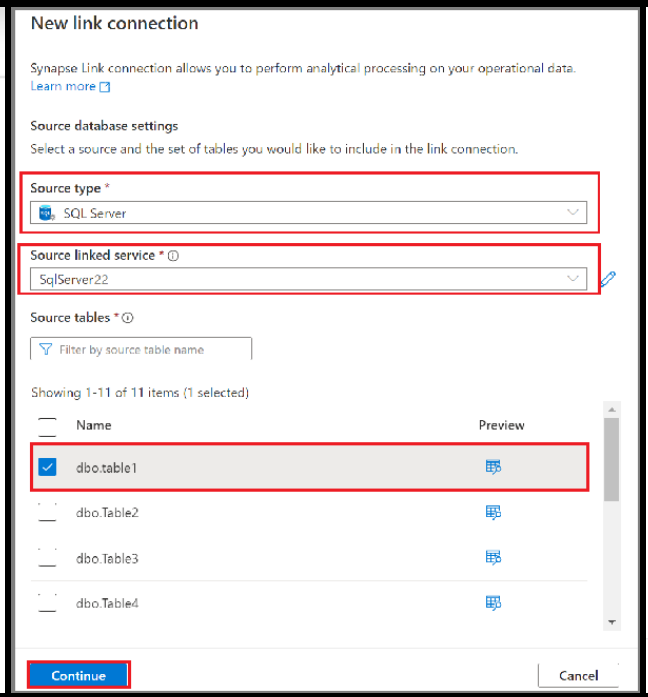
Record data changes without stretching source systems

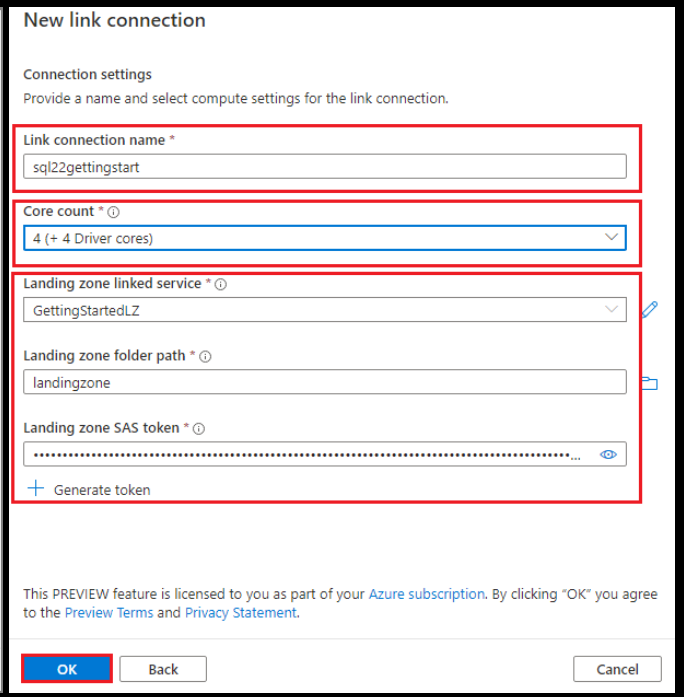
Azure Synapse Link for SQL

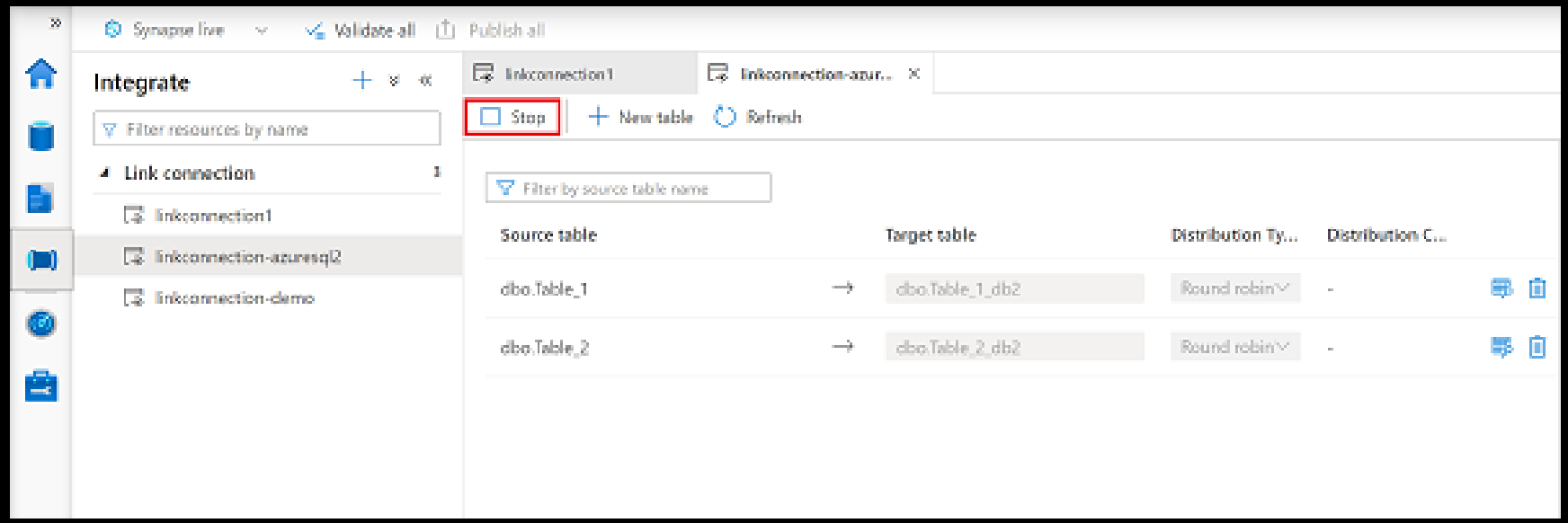
Automate data transfer and eliminate barriers with Azure Synapse Link for SQL

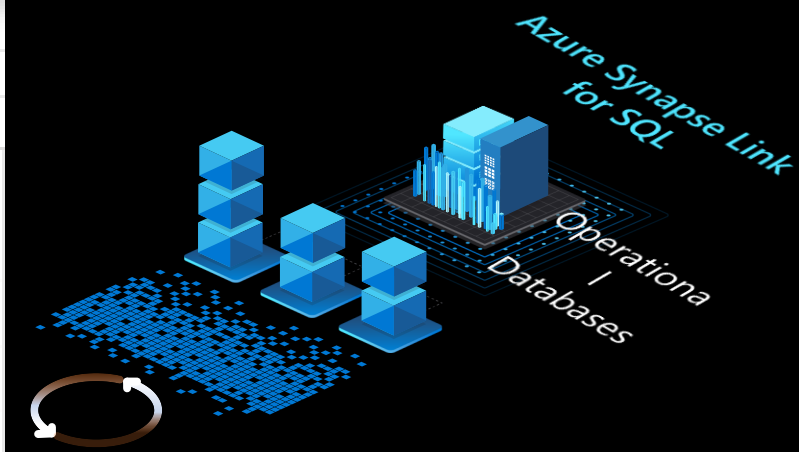
Preparing data to be near real-time insights







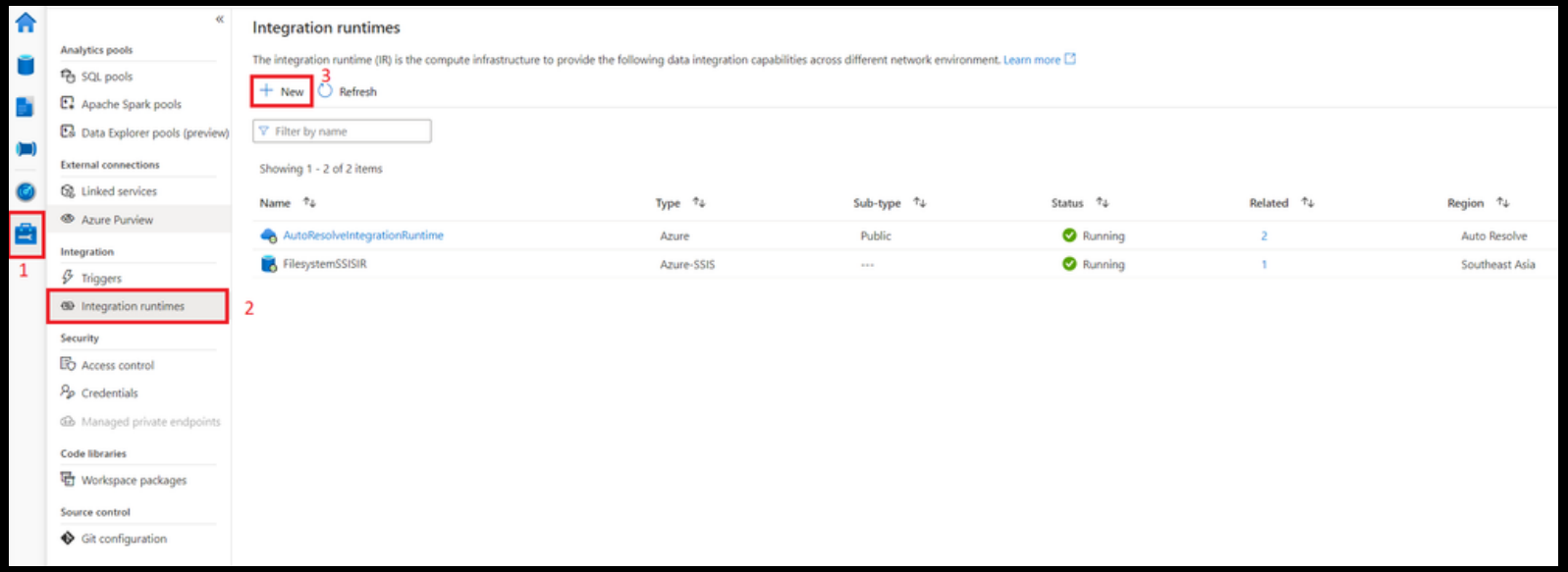




SSIS Integration Runtime for Azure Synapse

Running packages deployed into SSIS catalog (SSISDB) hosted by Azure SQL Database server/Managed Instance (Project Deployment Model)

Running packages deployed into Azure Files (Package Deployment Model)



Spark Processing

Public Preview

Q2 2022

Spark 3.2

Enables developers can leverage the latest innovations in the Spark ecosystem

Pandas (Koalas) integration

A highly popular and flexible library with broad industry adoption

Adaptive Query Execution (AQE) enabled by default

Significant improvements in query performance out-of-the-box

Small Query execution improvements

Small queries run faster due to reduced initialization overhead

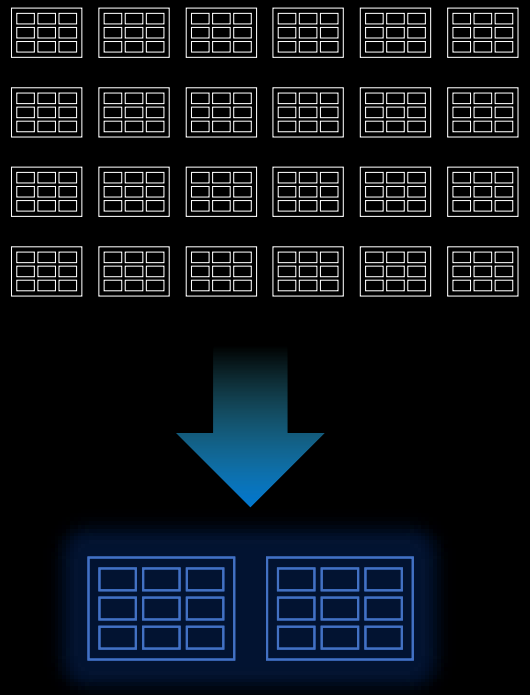
Public Preview

Q2 2022

Delta Lake Performance Enhancements

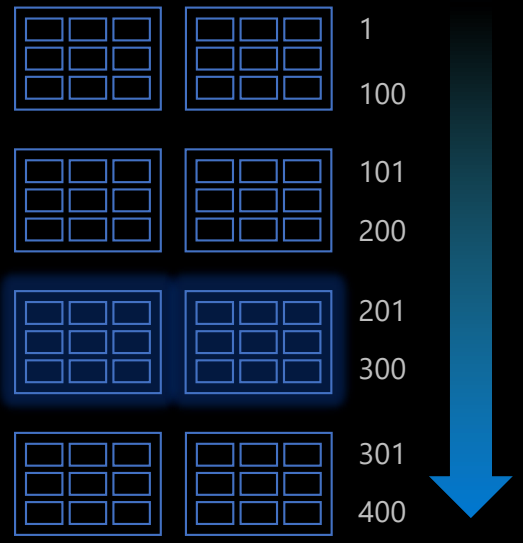
Improved performance and reduced cost with support for OPTIMIZE and Z-ORDER

OPTIMIZE



Improve query performance by coalescing small files into larger ones

Z-ORDER



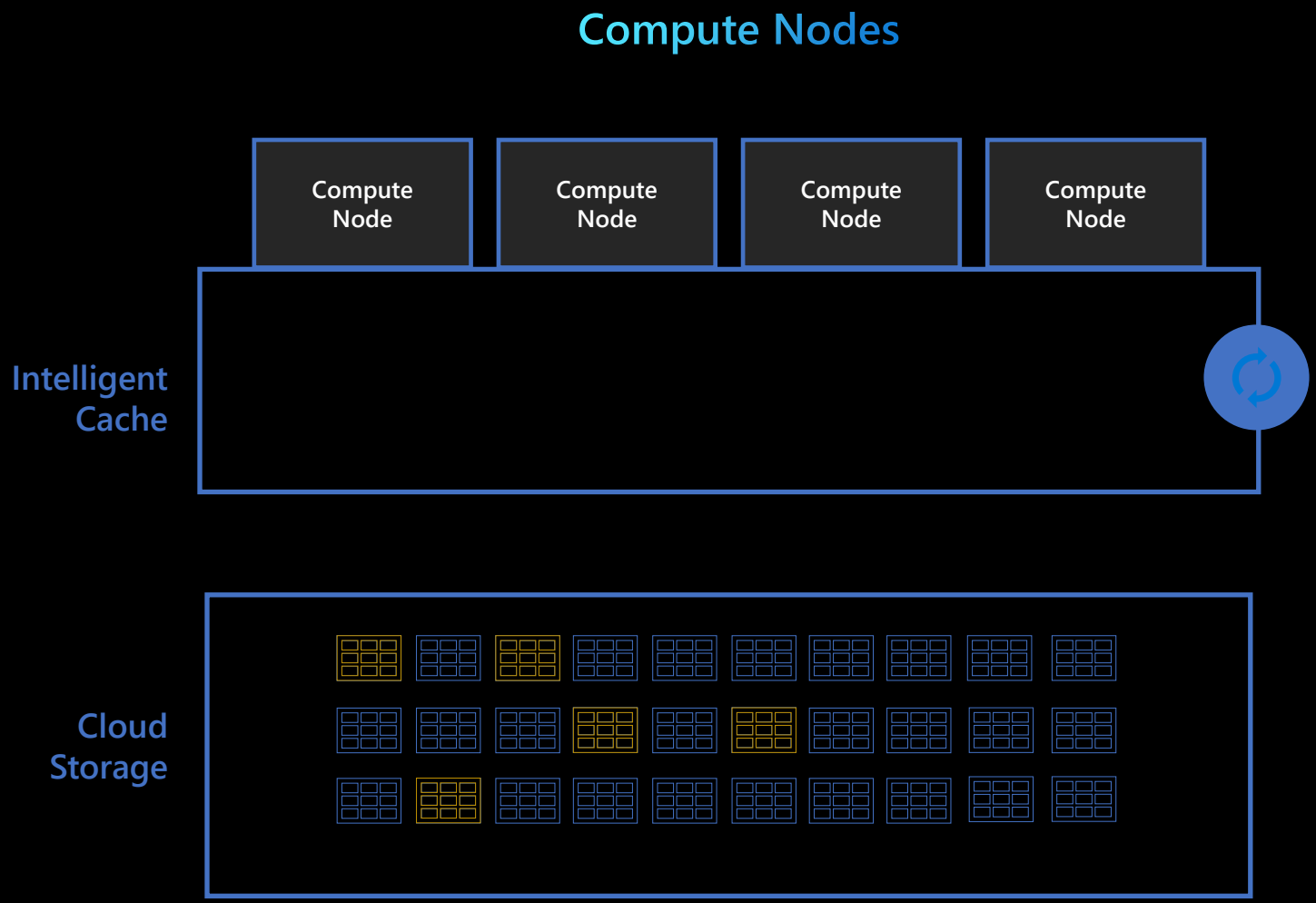
Improves filter query performance by ordering data for fast lookups on large datasets

Public Preview

Q1 2022

Spark Cashing Enhancements

Intelligent cache automatically detects changes in data to ensure data is fresh and results are accurate



SQLPool functionality

Advanced storage system

- Columnstore Indexes
- Table partitions
- Distributed tables
- Isolation modes
- Materialized Views
- Nonclustered Indexes
- Result-set caching

T-SQL Querying

- Windowing aggregates
- Approximate execution (Hyperloglog)
- JSON data support

Complete SQL object model

- Tables
- Views
- Stored procedures
- Functions

Question…

What is the primary factor for Azure Synapse winning a POC?

What about losing a POC?

Table design

Tables – Distributions

Round-robin distributed

Distributes table rows evenly across all distributions at random.

Hash distributed

Distributes table rows across the Compute nodes by using a deterministic hash function to assign each row to one distribution.

Replicated

Full copy of table accessible on each Compute node.

```
CREATE TABLE dbo.OrderTable
(
   OrderId  INT NOT NULL,
   Date     DATE NOT NULL,
   Name     VARCHAR(2),
   Country  VARCHAR(2)
)
WITH
(
 CLUSTERED COLUMNSTORE INDEX,
 DISTRIBUTION = HASH([OrderId]) |
                ROUND ROBIN |
                REPLICATED
);
```

Tables – Partitions

Overview

Table partitions divide data into smaller groups

In most cases, partitions are created on a date column

Supported on all table types

RANGE RIGHT – Used for time partitions

RANGE LEFT – Used for number partitions

Benefits

- Improves efficiency and performance of loading and querying by limiting the scope to subset of data.
- Offers significant query performance enhancements where filtering on the partition key can eliminate unnecessary scans and eliminate IO.

```
CREATE TABLE partitionedOrderTable
(
   OrderId  INT NOT NULL,
   Date     DATE NOT NULL,
   Name     VARCHAR(2),
   Country  VARCHAR(2)
)
WITH
(
   CLUSTERED COLUMNSTORE INDEX,
   DISTRIBUTION = HASH([OrderId]),
   PARTITION (
   [Date] RANGE RIGHT FOR VALUES (
   '2000-01-01', '2001-01-01', '2002-01-01',
   '2003-01-01', '2004-01-01', '2005-01-01'
   )
  )
);
```

Tables – Distributions & Partitions

Logical table structure

OrderId	Date	Name	Country
85016	11-2-2018	V	UK
85018	11-2-2018	Q	SP
85216	11-2-2018	Q	DE
85395	11-2-2018	V	NL
82147	11-2-2018	Q	FR
86881	11-2-2018	D	UK
93080	11-3-2018	R	UK
94156	11-3-2018	S	FR
96250	11-3-2018	Q	NL
98799	11-3-2018	R	NL
98015	11-3-2018	T	UK
98310	11-3-2018	D	DE
98979	11-3-2018	Z	DE
98137	11-3-2018	T	FR
…	…	…	…

Physical data distribution

(Hash distribution (OrderId), Date partitions)

Distribution1
(OrderId 80,000 – 100,000)

11-2-2018 partition

OrderId	Date	Name	Country
85016	11-2-2018	V	UK
85018	11-2-2018	Q	SP
85216	11-2-2018	Q	DE
85395	11-2-2018	V	NL
82147	11-2-2018	Q	FR
86881	11-2-2018	D	UK
…	…	…	…

11-3-2018 partition

OrderId	Date	Name	Country
93080	11-3-2018	R	UK
94156	11-3-2018	S	FR
96250	11-3-2018	Q	NL
98799	11-3-2018	R	NL
98015	11-3-2018	T	UK
98310	11-3-2018	D	DE
98979	11-3-2018	Z	DE
98137	11-3-2018	T	FR
…	…	…	…

… x 60 distributions (shards)

- Each shard is partitioned with the same date partitions
- A minimum of 1 million rows per distribution and partition is needed for optimal compression and performance of clustered Columnstore tables

Common table distribution methods

Table Category	Recommended Distribution Option
Fact	Use **hash-distribution with clustered columnstore index**. Performance improves because hashing enables the platform to localize certain operations within the node itself during query execution. Operations that benefit: COUNT(DISTINCT(<hashed_key>)) OVER PARTITION BY <hashed_key> most JOIN <table_name> ON <hashed_key> GROUP BY <hashed_key>
Dimension	Use **replicated** for smaller tables. If tables are too large to store on each Compute node, use hash-distributed.
Staging	Use **round-robin** for the staging table. The load with CTAS is faster. Once the data is in the staging table, use INSERT…SELECT to move the data to production tables.

Question…

Why is the best practice to load into a staging table, and then CTAS into the production table?

SQL Capabilities

Windowing functions

OVER clause

Defines a window or specified set of rows within a query result set

Computes a value for each row in the window

Aggregate functions

COUNT, MAX, AVG, SUM, APPROX_COUNT_DISTINCT, MIN, STDEV, STDEVP, STRING_AGG, VAR, VARP, GROUPING, GROUPING_ID, COUNT_BIG, CHECKSUM_AGG

Ranking functions

RANK, NTILE, DENSE_RANK, ROW_NUMBER

ROWS | RANGE

PRECEDING, UNBOUNDING PRECEDING, CURRENT ROW, BETWEEN, FOLLOWING, UNBOUNDED FOLLOWING

```
SELECT
  ROW_NUMBER() OVER(PARTITION BY PostalCode ORDER BY SalesYTD DESC) AS "Row Number",
  LastName,
  SalesYTD,
  PostalCode
FROM Sales
WHERE SalesYTD <> 0
ORDER BY PostalCode;
```

Row Number	LastName	SalesYTD	PostalCode
1	Mitchell	4251368.5497	98027
2	Blythe	3763178.1787	98027
3	Carson	3189418.3662	98027
4	Reiter	2315185.611	98027
5	Vargas	1453719.4653	98027
6	Ansman-Wolfe	1352577.1325	98027
1	Pak	4116870.2277	98055
2	Varkey Chudukaktil	3121616.3202	98055
3	Saraiva	2604540.7172	98055
4	Ito	2458535.6169	98055
5	Valdez	1827066.7118	98055
6	Mensa-Annan	1576562.1966	98055
7	Campbell	1573012.9383	98055
8	Tsoflias	1421810.9242	98055

Windowing Functions (continued)

Analytical functions

LAG, LEAD, FIRST_VALUE, LAST_VALUE, CUME_DIST, PERCENTILE_CONT, PERCENTILE_DISC, PERCENT_RANK

```
-- PERCENTILE_CONT, PERCENTILE_DISC
 SELECT DISTINCT Name AS DepartmentName
,PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY ph.Rate)
                OVER (PARTITION BY Name) AS MedianCont
,PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY ph.Rate)
                OVER (PARTITION BY Name) AS MedianDisc
FROM HumanResources.Department AS d
INNER JOIN HumanResources.EmployeeDepartmentHistory AS dh
   ON dh.DepartmentID = d.DepartmentID
INNER JOIN HumanResources.EmployeePayHistory AS ph
   ON ph.BusinessEntityID = dh.BusinessEntityID
WHERE dh.EndDate IS NULL;
```

DepartmentName	MedianCont	MedianDisc
Document Control	16.8269	16.8269
Engineering	34.375	32.6923
Executive	54.32695	48.5577
Human Resources	17.427850	16.5865

```
--LAG Function
SELECT BusinessEntityID,
YEAR(QuotaDate) AS SalesYear,
SalesQuota AS CurrentQuota,
LAG(SalesQuota, 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS PreviousQuota
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275 and YEAR(QuotaDate) IN ('2005','2006');
```

BusinessEntityID	SalesYear	CurrentQuota	PreviousQuota
275	2005	367000.00	0.00
275	2005	556000.00	367000.00
275	2006	502000.00	556000.00
275	2006	550000.00	502000.00
275	2006	1429000.00	550000.00
275	2006	1324000.00	1429000.00

Windowing Functions (continued)

ROWS | RANGE

PRECEDING, UNBOUNDING PRECEDING, CURRENT ROW, BETWEEN, FOLLOWING, UNBOUNDED FOLLOWING

```
-- First_Value
SELECT JobTitle, LastName, VacationHours AS VacHours,
FIRST_VALUE(LastName) OVER (PARTITION BY JobTitle
ORDER BY VacationHours ASC ROWS UNBOUNDED PRECEDING ) AS FewestVacHours
FROM HumanResources.Employee AS e
INNER JOIN Person.Person AS p
ON e.BusinessEntityID = p.BusinessEntityID
ORDER BY JobTitle;
```

JobTitle	LastName	VacHours	FewestVacHours
Accountant	Moreland	58	Moreland
Accountant	Seamans	59	Moreland
Accounts Manager	Liu	57	Liu
Accounts Payable Specialist	Tomic	63	Tomic
Accounts Payable Specialist	Sheperdigian	64	Tomic
Accounts Receivable Specialist	Poe	60	Poe
Accounts Receivable Specialist	Spoon	61	Poe
Accounts Receivable Specialist	Walton	62	Poe

Approximate execution

HyperLogLog accuracy

Reduce query latency in exchange for a small reduction in accuracy.

Returns a result with a 2% accuracy of true cardinality on average.

e.g. COUNT (DISTINCT) returns 1,000,000, HyperLogLog will return a value in the range of 999,736 to 1,016,234.

APPROX_COUNT_DISTINCT

Returns the approximate number of unique non-null values in a group.

Use Case: Approximating web usage trend behavior

```
-- Syntax
APPROX_COUNT_DISTINCT ( expression )

-- The approximate number of different order keys by order status from the orders table.
SELECT O_OrderStatus, APPROX_COUNT_DISTINCT(O_OrderKey) AS Approx_Distinct_OrderKey
FROM dbo.Orders
GROUP BY O_OrderStatus
ORDER BY O_OrderStatus;
```

Approximate execution

APPROX_COUNT_DISTINCT

```
SELECT APPROX_COUNT_DISTINCT([SalesOrderDetailID]) AS Approx_Distinct_OrderKey
FROM [SalesLT].[SalesOrderDetail]
```

100 %

Results | Messages

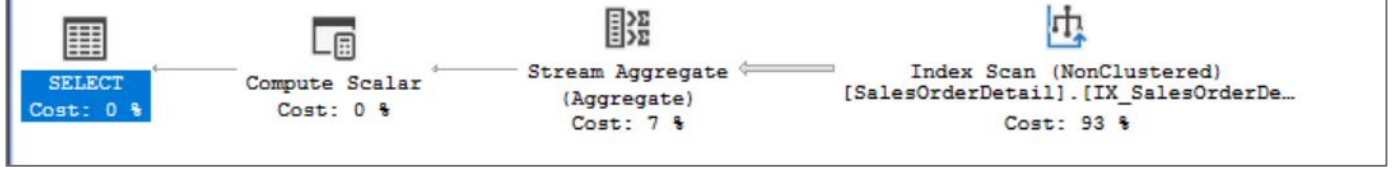
	Approx_Distinct_OrderKey
1	540

```
SELECT APPROX_COUNT_DISTINCT([SalesOrderDetailID]) AS Approx_Distinct_OrderKey
FROM [SalesLT].[SalesOrderDetail]
```

100 %

Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT APPROX_COUNT_DISTINCT([SalesOrderDetailID]) AS Approx_Distinct_OrderKey FROM [S
```



COUNT DISTINCT

```
SELECT COUNT (DISTINCT [SalesOrderDetailID]) AS Distinct_OrderKey
FROM [SalesLT].[SalesOrderDetail]
```

100 %

Results | Messages

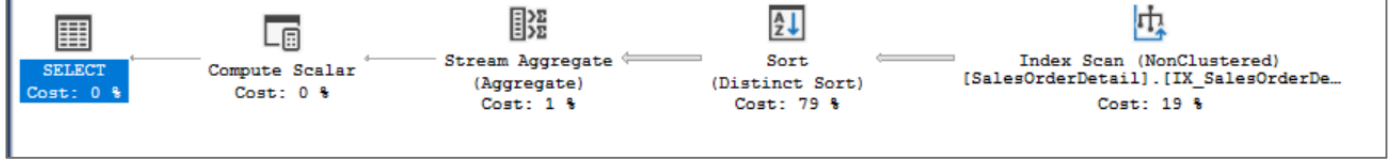
	Distinct_OrderKey
1	542

```
SELECT COUNT (DISTINCT [SalesOrderDetailID]) AS Distinct_OrderKey
FROM [SalesLT].[SalesOrderDetail]
```

100 %

Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT COUNT (DISTINCT [SalesOrderDetailID]) AS Distinct_OrderKey FROM [SalesLT].[SalesOrderDetail]
```



Group by options

Group by with rollup

Creates a group for each combination of column expressions.

Rolls up the results into subtotals and grand totals

Calculate the aggregates of hierarchical data

Grouping sets

Combine multiple GROUP BY clauses into one GROUP BY CLAUSE.

Equivalent of UNION ALL of specified groups.

```
-- GROUP BY SETS Example --
SELECT Country,
SUM(Sales) AS TotalSales
FROM Sales
GROUP BY GROUPING SETS ( Country, () );
```

```
-- GROUP BY ROLLUP Example --
SELECT Country,
Region,
SUM(Sales) AS TotalSales
FROM Sales
GROUP BY ROLLUP (Country, Region);
-- Results --
```

Country	Region	TotalSales
Canada	Alberta	100
Canada	British Columbia	500
Canada	NULL	600
United States	Montana	100
United States	NULL	100
NULL	NULL	700

Snapshot isolation

Overview

Specifies that statements cannot read data that has been modified but not committed by other transactions.

This prevents dirty reads.

```
ALTER DATABASE MyDatabase
SET ALLOW_SNAPSHOT_ISOLATION ON
```

```
ALTER DATABASE MyDatabase SET
READ_COMMITTED_SNAPSHOT ON
```

Isolation level

- READ COMMITTED
- REPEATABLE READ
- SNAPSHOT
- READ UNCOMMITTED
- SERIALIZABLE

READ_COMMITTED_SNAPSHOT

OFF (Default) – Uses shared locks to prevent other transactions from modifying rows while running a read operation

ON – Uses row versioning to present each statement with a transactionally consistent snapshot of the data as it existed at the start of the statement. Locks are not used to protect the data from updates.

JSON data support – insert JSON data

Overview

The JSON format enables representation of complex or hierarchical data structures in tables.

JSON data is stored using standard NVARCHAR table columns.

Benefits

Transform arrays of JSON objects into table format

Performance optimization using clustered columnstore indexes and memory optimized tables

```
-- Create Table with column for JSON string
CREATE TABLE CustomerOrders
(
 CustomerId   BIGINT NOT NULL,
 Country      NVARCHAR(150) NOT NULL,
 OrderDetails NVARCHAR(3000) NOT NULL –- NVARCHAR column for JSON
) WITH (DISTRIBUTION = ROUND_ROBIN)

-- Populate table with semi-structured data
INSERT INTO CustomerOrders
VALUES
( 101, -- CustomerId
 'Bahrain', -- Country
 N'[{ StoreId": "AW73565",
     "Order": { "Number":"SO43659",
                "Date":"2011-05-31T00:00:00"
              },
     "Item":  { "Price":2024.40, "Quantity":1 }
   }]' -- OrderDetails
)
```

JSON data support – read JSON data

Overview

Read JSON data stored in a string column with the following:

- ISJSON – verify if text is valid JSON
- JSON_VALUE – extract a scalar value from a JSON string
- JSON_QUERY – extract a JSON object or array from a JSON string

Benefits

- Ability to get standard columns as well as JSON column
- Perform aggregation and filter on JSON values

```
-- Return all rows with valid JSON data
SELECT CustomerId, OrderDetails
FROM   CustomerOrders
WHERE  ISJSON(OrderDetails) > 0;
```

CustomerId	OrderDetails
101	N'[{ StoreId": "AW73565", "Order": { "Number":"SO43659", "Date":"2011-05-31T00:00:00“ }, "Item": { "Price":2024.40, "Quantity":1 }}]'

```
-- Extract values from JSON string
SELECT CustomerId,
      Country,
      JSON_VALUE(OrderDetails,'$.StoreId') AS StoreId,
      JSON_QUERY(OrderDetails,'$.Item') AS ItemDetails
FROM   CustomerOrders;
```

CustomerId	Country	StoreId	ItemDetails
101	Bahrain	AW73565	{ "Price":2024.40, "Quantity":1 }

JSON data support – modify and operate on JSON data

Overview

Use standard table columns and values from JSON text in the same analytical query.

Modify JSON data with the following:

- JSON_MODIFY – modifies a value in a JSON string
- OPENJSON – convert JSON collection to a set of rows and columns

Benefits

- Flexibility to update JSON string using T-SQL
- Convert hierarchical data into flat tabular structure

```
-- Modify Item Quantity value
UPDATE CustomerOrders SET OrderDetails =
JSON_MODIFY(OrderDetails, '$.OrderDetails.Item.Quantity',2)
```

OrderDetails

```
N'[{ StoreId": "AW73565",  "Order": { "Number":"SO43659",
"Date":"2011-05-31T00:00:00“ }, "Item":  { "Price":2024.40, "Quantity": 2}}]'
```

```
-- Convert JSON collection to rows and columns
SELECT CustomerId,
      StoreId,
      OrderDetails.OrderDate,
      OrderDetails.OrderPrice
FROM   CustomerOrders
CROSS APPLY OPENJSON (CustomerOrders.OrderDetails)
WITH ( StoreId      VARCHAR(50)  '$.StoreId',
      OrderNumber   VARCHAR(100) '$.Order.Date',
      OrderDate     DATETIME     '$.Order.Date',
      OrderPrice    DECIMAL      ‘$.Item.Price',
      OrderQuantity INT          '$.Item.Quantity'
     ) AS OrderDetails
```

CustomerId	StoreId	OrderDate	OrderPrice
101	AW73565	2011-05-31T00:00:00	2024.40

Stored Procedures

Overview

It is a group of one or more SQL statements or a reference to a Microsoft .NET Framework common language runtime (CLR) method.

Promotes flexibility and modularity.

Supports parameters and nesting.

Benefits

- Reduced server/client network traffic, improved performance
- Stronger security
- Easy maintenance

```
CREATE PROCEDURE HumanResources.uspGetAllEmployees
AS
   SET NOCOUNT ON;
   SELECT LastName, FirstName, JobTitle, Department
   FROM HumanResources.vEmployeeDepartment;
GO

-- Execute a stored procedures
EXECUTE HumanResources.uspGetAllEmployees;
GO
-- Or
EXEC HumanResources.uspGetAllEmployees;
GO
-- Or, if this procedure is the first statement within a batch:
HumanResources.uspGetAllEmployees;
```

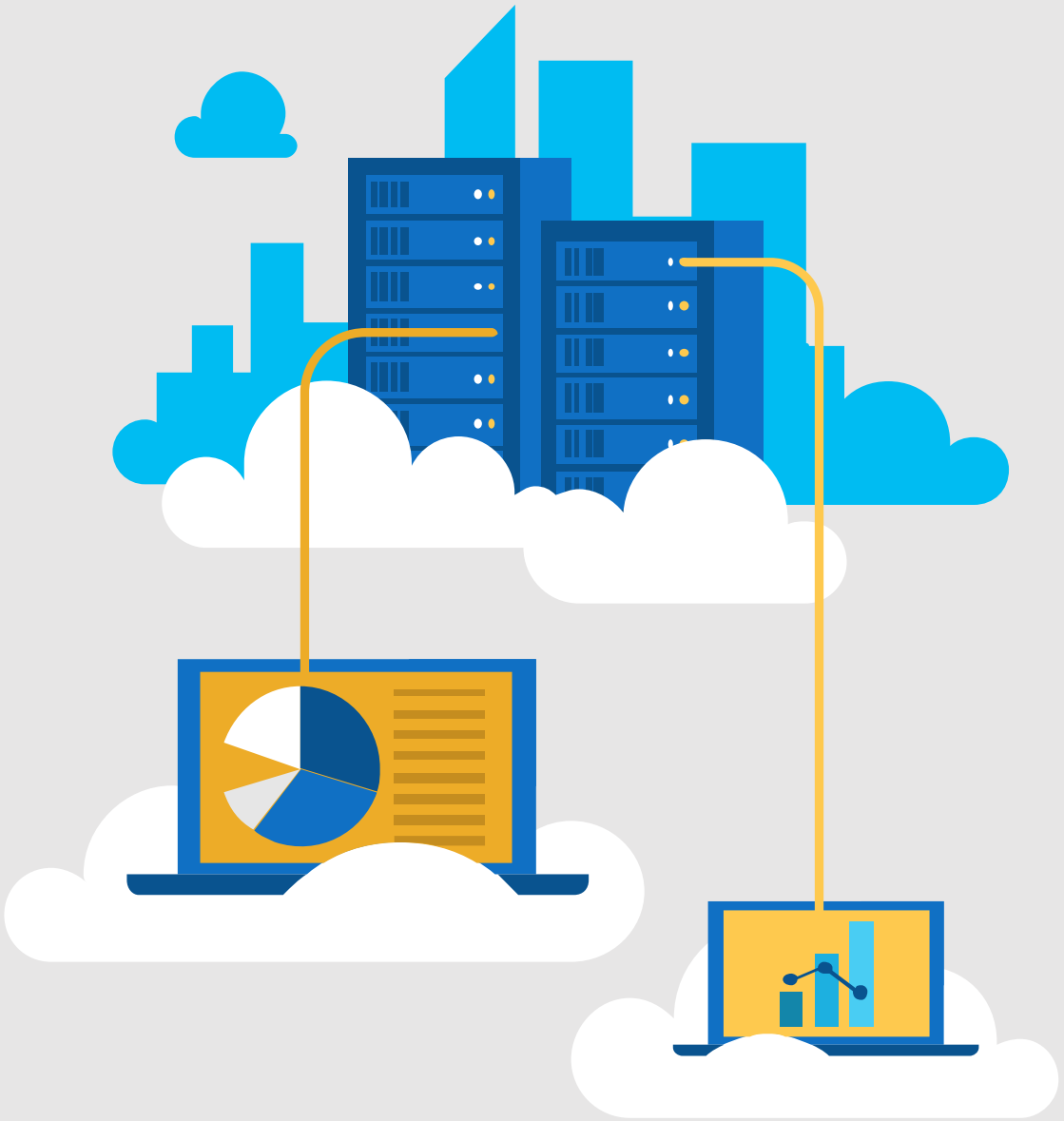
Pop Quiz 1

What reduction in accuracy does APPROXIMATE_COUNT_DISTINCT (Hyper Log Log) show on average?

A) 2%

B) 0.1%

C) 0.5%



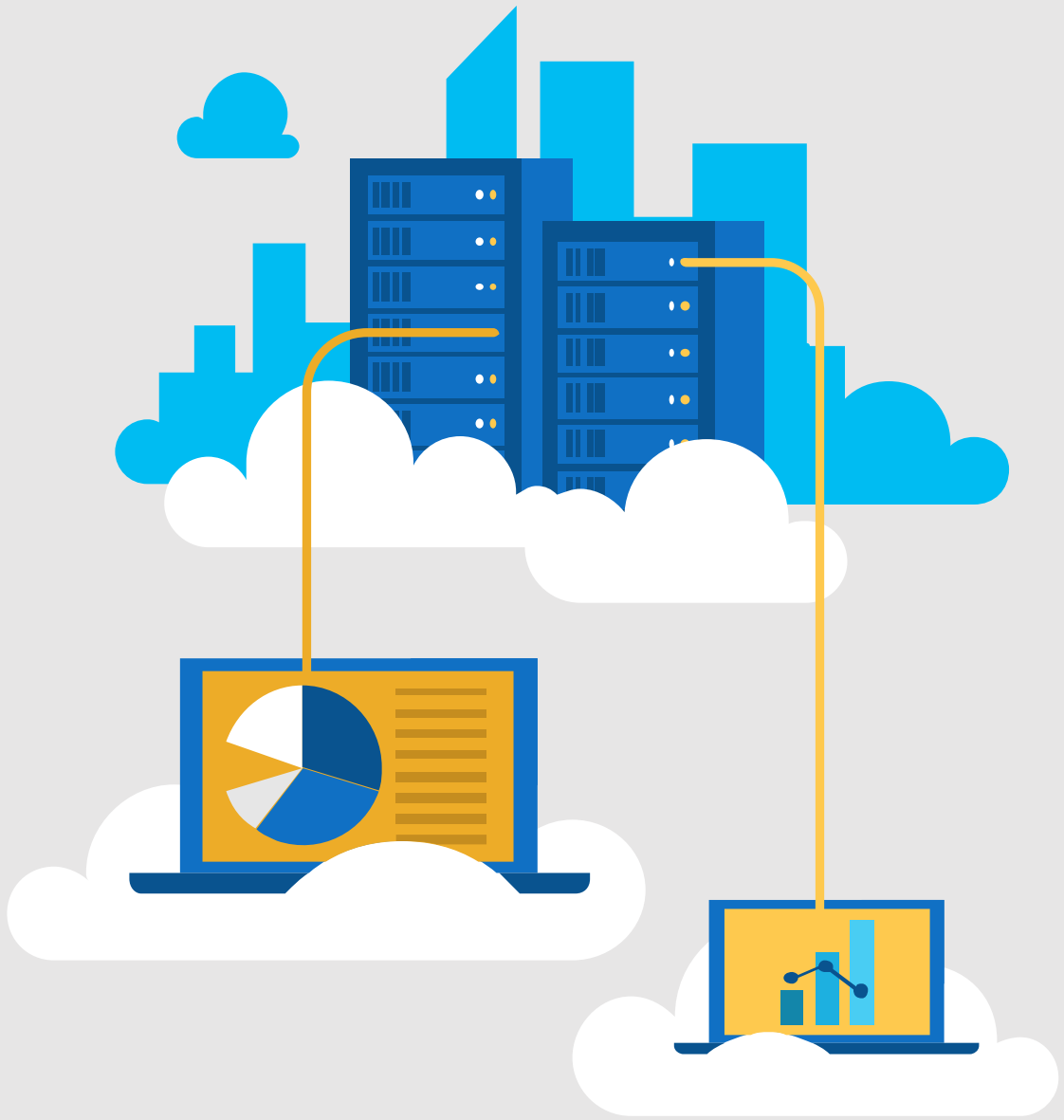
Pop Quiz 1

What reduction in accuracy does APPROXIMATE_COUNT_DISTINCT (Hyper Log Log) show on average?

A) 2%

B) 0.1%

C) 0.5%



SQL Recap

- Query JSON using Azure Synapse SQL in conjunction with T-SQL OPENJSON, JSON_VALUE, and JSON_Query statements.
- Modify JSON data during UPDATE using the JSON_MODIFY statement.
- Use APPROXIMATE_COUNT_DISTINCT for better count query performance- results within average 2% accuracy of the true count.

Performance Patterns

Distributed table design recommendations

Hash Distribution:

- Small fact tables exceeding several GBs with frequent inserts should use a hash distribution.

Round Robin Distribution:

- Potentially useful tables created from raw input.
- Temporary staging tables used in data preparation.

Replicated Tables:

- Lookup tables that range in size from 100’s MBs to 1.5 GBs should be replicated. Works best when table size is less than 2 GB compressed.

Automatic statistics management

Overview

Statistics are automatically created and maintained for SQL pool. Incoming queries are analyzed, and individual column statistics are generated on the columns that improve cardinality estimates to enhance query performance.

Statistics are automatically updated as data modifications occur in underlying tables. By default, these updates are synchronous but can be configured to be asynchronous.

Statistics are considered out of date when:

- There was a data change on an empty table
- The number of rows in the table at time of statistics creation was 500 or less, and more than 500 rows have been updated
- The number of rows in the table at time of statistics creation was more than 500, and more than 500 + 20% of rows have been updated

```
-- Turn on/off auto-create statistics settings
ALTER DATABASE {database_name}
SET AUTO_CREATE_STATISTICS { ON | OFF }

-- Turn on/off auto-update statistics settings
ALTER DATABASE {database_name}
SET AUTO_UPDATE_STATISTICS { ON | OFF }

-- Configure synchronous/asynchronous update
ALTER DATABASE {database_name}
SET AUTO_UPDATE_STATISTICS_ASYNC { ON | OFF }

-- Check statistics settings for a database
SELECT      is_auto_create_stats_on,
            is_auto_update_stats_on,
            is_auto_update_stats_async_on
FROM        sys.databases
```

Performance Anti-Patterns

Too many partitions

- Partitions can be useful when maintaining current rows in very large fact tables. Partition switching is a good alternative to full CTAS
- Partitioning CCIs is only useful when the row count is greater than 60million * #partitions
- In general, avoid partitions, particularly in POCs

Views on Views

- Views on Views will not support performance optimization using Materialized Views (more later)

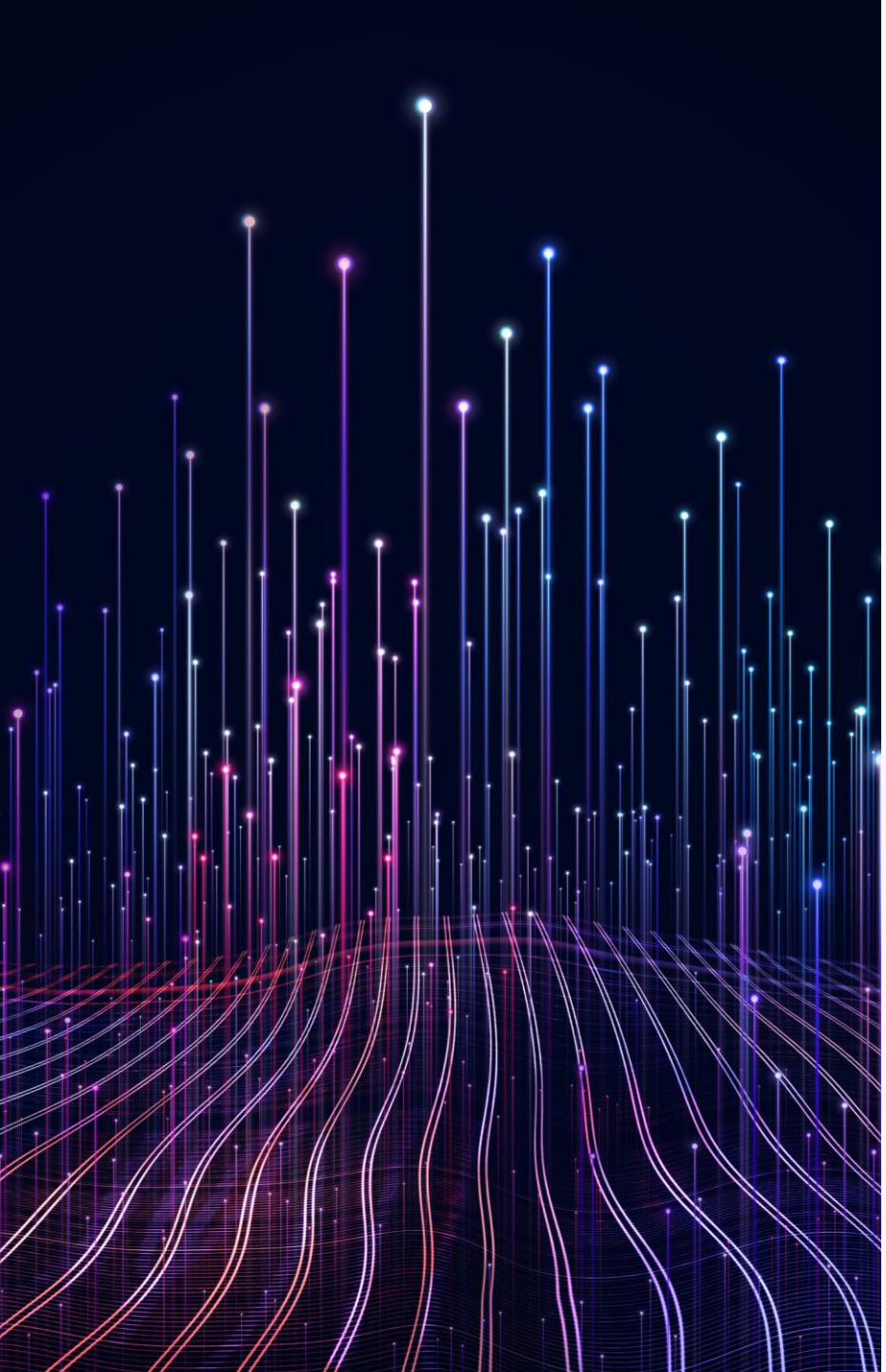
- Views cannot be distributed


Microsoft Azure Solutions Business
DW Optimization Part 2

Question…

How many indexing methods are there for a Synapse table?

Can you name them all?



Agenda

1. **Performance Patterns**
   Result set caching and materialized views.

2. **Index design**
   Clustered columnstore index and ordered variant, clustered index, heap and non-clustered index.

3. **Perf Anti-Patterns**
   Approaches the impede or limit performance.

Performance Patterns

Result set caching motivated

- Use result-set caching to improve query performance when the same queries are executed repeatedly against mainly static data.
- Result-set cache is invalidated and refreshed when underlying table data changes or the query code changes
- Result-set cache persists when SQL Pool is paused and resumed.

Result-set caching

Overview

Cache the results of a query in SQL pool storage. This enables interactive response times for repetitive queries against tables with infrequent data changes.

The result-set cache persists even if SQL pool is paused and resumed later.

Query cache is invalidated and refreshed when underlying table data or query code changes.

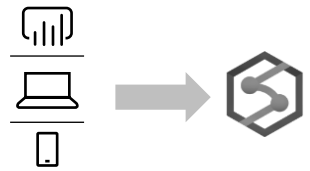
Result cache is evicted regularly based on a time-aware least recently used algorithm (TLRU).

Benefits

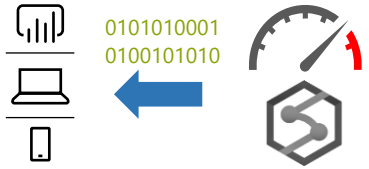
- Enhances performance when same result is requested repetitively
- Reduced load on server for repeated queries
- Offers monitoring of query execution with a result cache hit or miss

```
-- Turn on/off result-set caching for a database
-- Must be run on the MASTER database
ALTER DATABASE {database_name}
SET RESULT_SET_CACHING { ON | OFF }

-- Turn on/off result-set caching for a client session
-- Run on target Azure Synapse Analytics
SET RESULT_SET_CACHING {ON | OFF}

-- Check result-set caching setting for a database
-- Run on target Azure Synapse Analytics
SELECT is_result_set_caching_on
FROM   sys.databases
WHERE  name = {database_name}

-- Return all query requests with cache hits
-- Run on target data warehouse
SELECT *
FROM   sys.dm_pdw_request_steps
WHERE  command like '%DWResultCacheDb%'
     AND step_index = 0
```

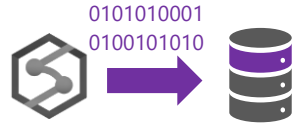
Result-set caching flow



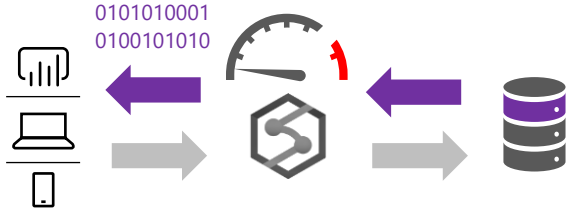
1 Client sends query to SQL pool



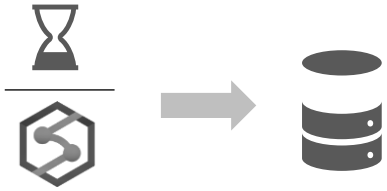
2 Query is processed using compute nodes which pull data from remote storage, process query and output back to client app



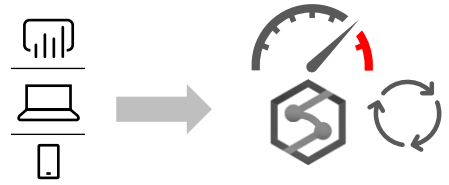
Query results are cached in remote storage so subsequent requests can be served immediately



3 Subsequent executions for the same query bypass compute nodes and can be fetched instantly from persistent cache in remote storage



4 Remote storage cache is evicted regularly based on time, cache usage, and any modifications to underlying table data.



5 Cache will need to be regenerated if query results have been evicted from cache

Materialized views

Overview

A materialized view pre-computes, stores, and maintains its data like a table.

Materialized views are automatically updated when data in underlying tables are changed. This is a synchronous operation that occurs as soon as the data is changed.

The auto caching functionality allows Azure Synapse Analytics Query Optimizer to consider using indexed view even if the view is not referenced in the query.

Supported aggregations: MAX, MIN, AVG, COUNT, COUNT_BIG, SUM, VAR, STDEV

Benefits

- Automatic and synchronous data refresh with data changes in base tables. No user action is required.
- High availability and resiliency as regular tables

```
-- Create indexed view
CREATE MATERIALIZED VIEW Sales.vw_Orders
WITH
(
  DISTRIBUTION = ROUND_ROBIN |
  HASH(ProductID)
)
AS
   SELECT SUM(UnitPrice*OrderQty) AS Revenue,
         OrderDate,
         ProductID,
         COUNT_BIG(*) AS OrderCount
   FROM   Sales.SalesOrderDetail
   GROUP  BY OrderDate, ProductID;
GO

-- Disable index view and put it in suspended mode
ALTER INDEX ALL ON Sales.vw_Orders DISABLE;

-- Re-enable index view by rebuilding it
ALTER INDEX ALL ON Sales.vw_Orders REBUILD;
```

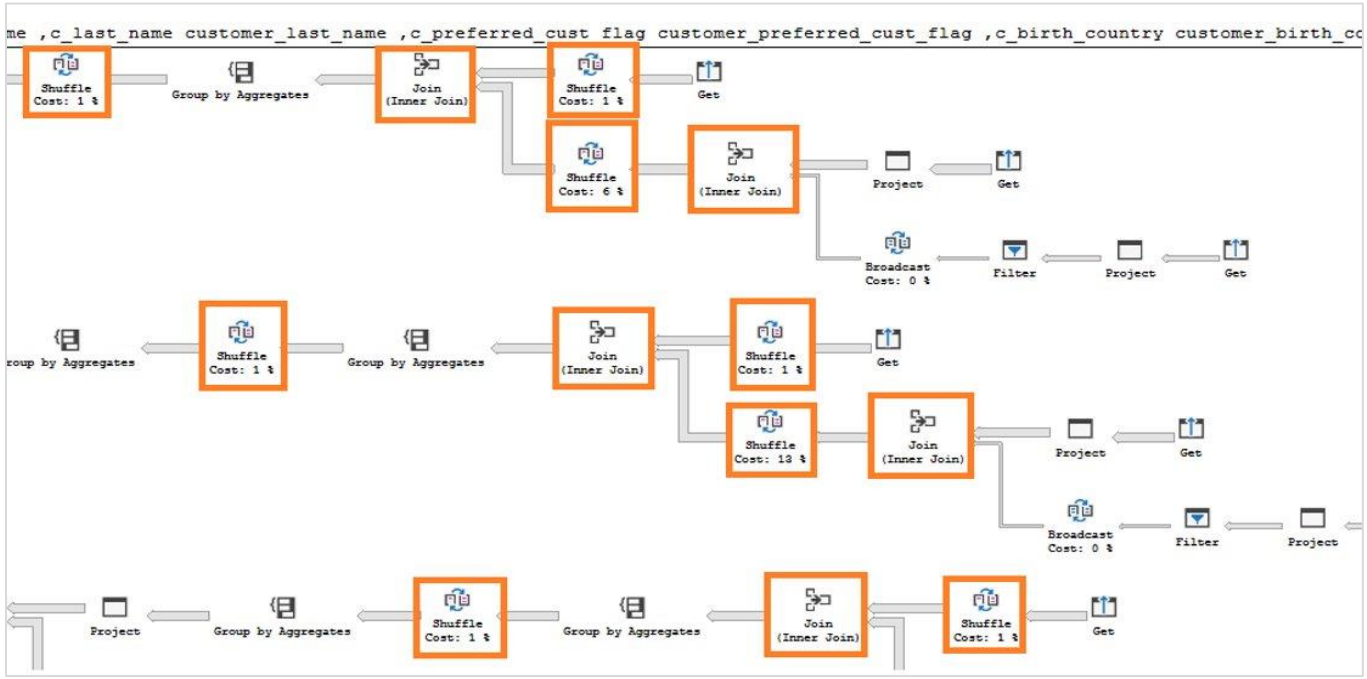
Materialized views - example

In this example, a query to get the year total sales per customer is shown to have a lot of data shuffles and joins that contribute to slow performance:

No relevant indexed views created on the data warehouse

```
-- Get year total sales per customer
(WITH year_total AS
        SELECT customer_id,
              first_name,
              last_name,
              birth_country,
              login,
              email_address,
              d_year,
              SUM(ISNULL(list_price – wholesale_cost –
              discount_amt + sales_price, 0)/2)year_total
        FROM   customer cust
        JOIN   catalog_sales sales ON cust.sk = sales.sk
        JOIN   date_dim ON sales.sold_date = date_dim.date
        GROUP  BY customer_id, first_name,
             last_name,birth_country,
             login,email_address ,d_year
)
SELECT TOP 100 …
FROM   year_total …
WHERE  …
ORDER BY …
```

Execution time: 103 seconds
Lots of data shuffles and joins needed to complete query



Materialized views - example

Now, we add an indexed view to the data warehouse to increase the performance of the previous query.
This view can be leveraged by the query even though it is not directly referenced.

Original query – get year total sales per customer

```
-- Get year total sales per customer
(WITH year_total AS
        SELECT customer_id,
            first_name,
            last_name,
            birth_country,
            login,
            email_address,
            d_year,
            SUM(ISNULL(list_price – wholesale_cost –
            discount_amt + sales_price, 0)/2)year_total
        FROM   customer cust
        JOIN   catalog_sales sales ON cust.sk = sales.sk
        JOIN   date_dim ON sales.sold_date = date_dim.date
        GROUP  BY customer_id, first_name,
            last_name,birth_country,
            login,email_address ,d_year
)
SELECT TOP 100 …
FROM   year_total …
WHERE  …
ORDER BY …
```

Create indexed view with hash distribution on customer_id column

```
-- Create indexed view for query
CREATE INDEXED VIEW nbViewCS WITH (DISTRIBUTION=HASH(customer_id)) AS
SELECT customer_id,
        first_name,
        last_name,
        birth_country,
        login,
        email_address,
        d_year,
        SUM(ISNULL(list_price – wholesale_cost – discount_amt +
        sales_price, 0)/2) AS year_total
FROM   customer cust
JOIN   catalog_sales sales ON cust.sk = sales.sk
JOIN   date_dim ON sales.sold_date = date_dim.date
GROUP  BY customer_id, first_name,
    last_name,birth_country,
    login, email_address, d_year
```

Indexed (materialized) views - example

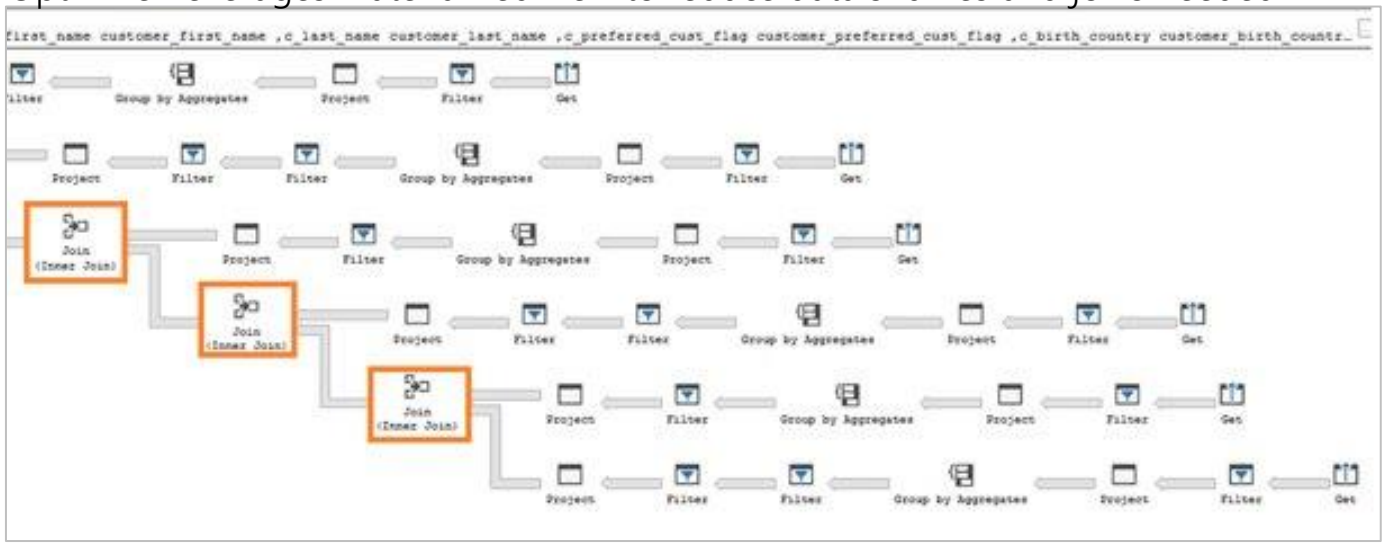
SQL pool query optimizer automatically leverages the indexed view to speed up the same query. Notice that the query does not need to reference the view directly

Original query – no changes have been made to query

```
-- Get year total sales per customer
(WITH year_total AS
        SELECT customer_id,
             first_name,
             last_name,
             birth_country,
             login,
             email_address,
             d_year,
             SUM(ISNULL(list_price – wholesale_cost –
             discount_amt + sales_price, 0)/2)year_total
        FROM   customer cust
        JOIN   catalog_sales sales ON cust.sk = sales.sk
        JOIN   date_dim ON sales.sold_date = date_dim.date
        GROUP  BY customer_id, first_name,
            last_name,birth_country,
            login,email_address ,d_year
)
SELECT TOP 100 …
FROM   year_total …
WHERE  …
ORDER BY …
```

Execution time: 6 seconds

Optimizer leverages materialized view to reduce data shuffles and joins needed



Materialized views- Recommendations

EXPLAIN - provides query plan for SQL statement without running the statement; view estimated cost of the query operations.

EXPLAIN WITH_RECOMMENDATIONS - provides query plan with recommendations to optimize the SQL statement performance.

```
EXPLAIN WITH_RECOMMENDATIONS
select count(*)
from (
        (select distinct c_last_name, c_first_name, d_date
        from store_sales, date_dim, customer
        where store_sales.ss_sold_date_sk = date_dim.d_date_sk
        and store_sales.ss_customer_sk = customer.c_customer_sk
        and d_month_seq between 1194 and 1194+11)
except
        (select distinct c_last_name, c_first_name, d_date
        from catalog_sales, date_dim, customer
        where catalog_sales.cs_sold_date_sk = date_dim.d_date_sk
        and catalog_sales.cs_bill_customer_sk = customer.c_customer_sk
        and d_month_seq between 1194 and 1194+11)
) top_customers
```

Materialized Views

- Indexed views cache the schema and data for a view in DW remote storage. They are useful for improving the performance of 'SELECT' statement queries that include aggregations

- Indexed views are automatically updated when data in underlying tables are changed. This is a synchronous operation that occurs as soon as the data is changed.

- The auto caching functionality allows SQL DW Query Optimizer to consider using indexed view even if the view is not referenced in the query

- Supported aggregations: MAX, MIN, AVG, COUNT, COUNT_BIG, SUM, VAR, STDEV

Index design

Tables – Indexes

Clustered Columnstore index (Default Primary)

Highest level of data compression

Best overall query performance

Clustered index (Primary)

Performant for looking up a single to few rows

Heap (Primary)

Faster loading and landing temporary data

Best for small lookup tables

Nonclustered indexes (Secondary)

Enable ordering of multiple columns in a table

Allows multiple nonclustered on a single table

Can be created on any of the above primary indexes

More performant lookup queries

```
-- Create table with index
CREATE TABLE orderTable
(
   OrderId  INT NOT NULL,
   Date     DATE NOT NULL,
   Name     VARCHAR(2),
   Country  VARCHAR(2)
)
WITH
(
CLUSTERED COLUMNSTORE INDEX |
   HEAP |
   CLUSTERED INDEX (OrderId)
);

-- Add non-clustered index to table
CREATE INDEX NameIndex ON orderTable (Name);
```

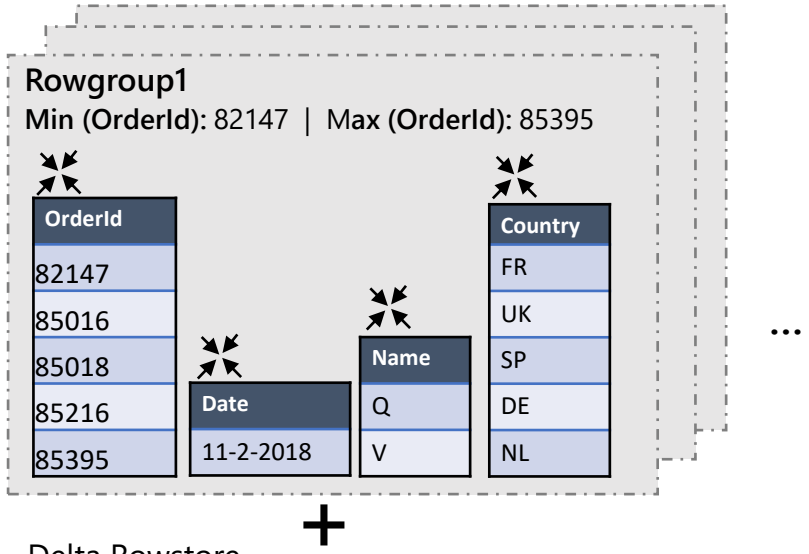
SQL Analytics Columnstore Tables

Logical table structure

OrderId	Date	Name	Country
85016	11-2-2018	V	UK
85018	11-2-2018	Q	SP
85216	11-2-2018	Q	DE
85395	11-2-2018	V	NL
82147	11-2-2018	Q	FR
86881	11-2-2018	D	UK
93080	11-3-2018	R	UK
94156	11-3-2018	S	FR
96250	11-3-2018	Q	NL
98799	11-3-2018	R	NL
98015	11-3-2018	T	UK
98310	11-3-2018	D	DE
98979	11-3-2018	Z	DE
98137	11-3-2018	T	FR
…	…	…	…

Clustered columnstore index

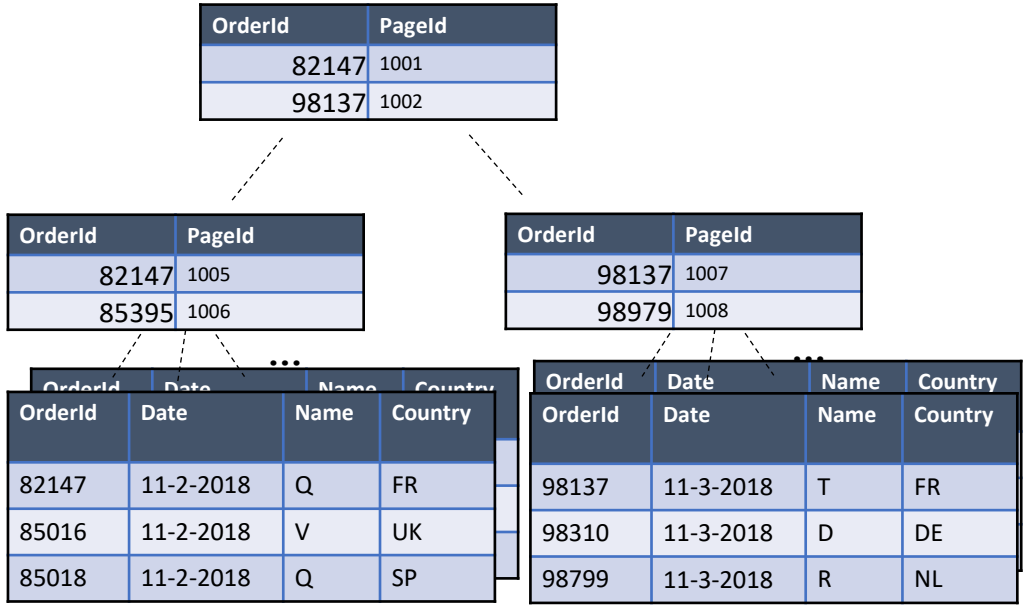
(OrderId)



- Data stored in compressed columnstore segments after being sliced into groups of rows (rowgroups/micro-partitions) for maximum compression
- Rows are stored in the delta rowstore until the number of rows is large enough to be compressed into a columnstore

Clustered/Non-clustered rowstore index

(OrderId)



- Data is stored in a B-tree index structure for performant lookup queries for particular rows.
- Clustered rowstore index: The leaf nodes in the structure store the data values in a row (as pictured above)
- Non-clustered (secondary) rowstore index: The leaf nodes store pointers to the data values, not the values themselves

Ordered Clustered Columnstore Indexes

Overview

Queries against tables with ordered columnstore segments can take advantage of improved segment elimination to drastically reduce the time needed to service a query.

```
-- Create Table with Ordered Columnstore Index
CREATE TABLE sortedOrderTable
(
   OrderId  INT NOT NULL,
   Date     DATE NOT NULL,
   Name     VARCHAR(2),
   Country  VARCHAR(2)
)
WITH
(
 CLUSTERED COLUMNSTORE INDEX ORDER (OrderId)
)
-- Create Clustered Columnstore Index on existing table
CREATE CLUSTERED COLUMNSTORE INDEX cciOrderId
ON dbo.OrderTable ORDER (OrderId)
```

```
-- Insert data into table with ordered columnstore index
INSERT INTO sortedOrderTable
VALUES (1, '01-01-2019','Dave', 'UK')
```

Choosing the right index

- Clustered Columnstore indexes (CCI) are best for fact tables.
- CCI offer the highest level of data compression and best query performance for tables with over 100 million rows.
- Heap tables are best for small lookup tables and recommended for tables with less than 100 million rows.
- Clustered Indexes may outperform CCI when very few rows need to be retrieved quickly.
  - Add non-clustered indexes to improve performance for less selective queries.
  - Each additional index added to a table increases storage space required and processing time during data loads.
- Speed load performance by staging data in heap tables and temporary tables prior to running transformations.

Ordered CCI

- Queries against tables with ordered columnstore segments can take advantage of improved segment elimination to drastically reduce the time needed to service a query.

- Columnstore Segments are automatically updated as data is inserted, updated, or deleted in data warehouse tables.

Stream processing at scale, in SQL
Azure Stream Analytics

Agenda

- Near-real time analytics and stream processing
- Azure Stream Analytics (ASA) overview
- Concepts
- Streaming SQL
- Developer experience
- Operations

Near-real time analytics

What qualifies as near-real time

- Latency : milliseconds to minutes
- Time to insight, including transportation (network), ingestion and processing time

	Real time	Near-real time	Batch processing
!	Local / edge	Local to cloud	Cloud
	Strict guarantees	Late arrival policies	Scheduled processes
	Synchronous	Milliseconds – Minutes	Minutes – Months

Why go near-real time?

- Insights with a short shelf life (alerts, notifications, anomaly and fraud detections…)
- Most business processes are now happening in real time
- In automated workflows, the user of the software is another software

Near-real time analytics : Stream Processing

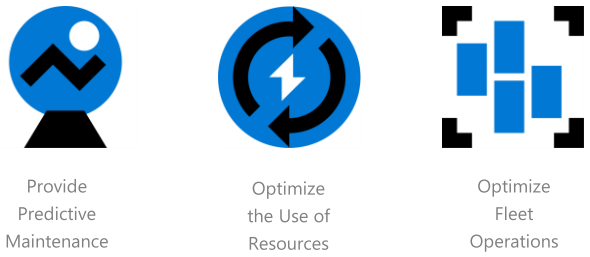
Data streams : continuous flow of events

- Sequence of data, ever-growing, often with limited retention
- Events are facts (“something happened”, broadcasted), and not intents (commands or jobs)

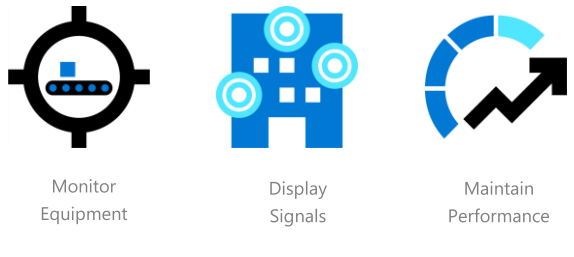
Stream processing : extracting value from streams

Stream(s) to App(s)
- Rules engine
- Derived events
- Alerting

Stream(s) to Data store(s)
- Streaming ETL to database, data lake, data mart, data-warehouse



Stream(s) to the Human eye
- Real-time reports
- Real-time dashboards
- Notifications



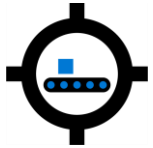
Azure Stream Analytics : Overview

The **native** stream processing service in Azure, in **PaaS**, since **2014**

Based on Trill, high-performance in-memory analytics library from Microsoft Research

Our mission is to democratize stream processing

- Easy to use: serverless, low code with SQL for business logic, well integrated in Azure, portal or local developer experience, CI/CD capabilities…
- Easy to run: at least once delivery guarantee, easy to understand pricing, logs and alerts with Azure Monitor, 99.9% availability, 40+ Azure regions…

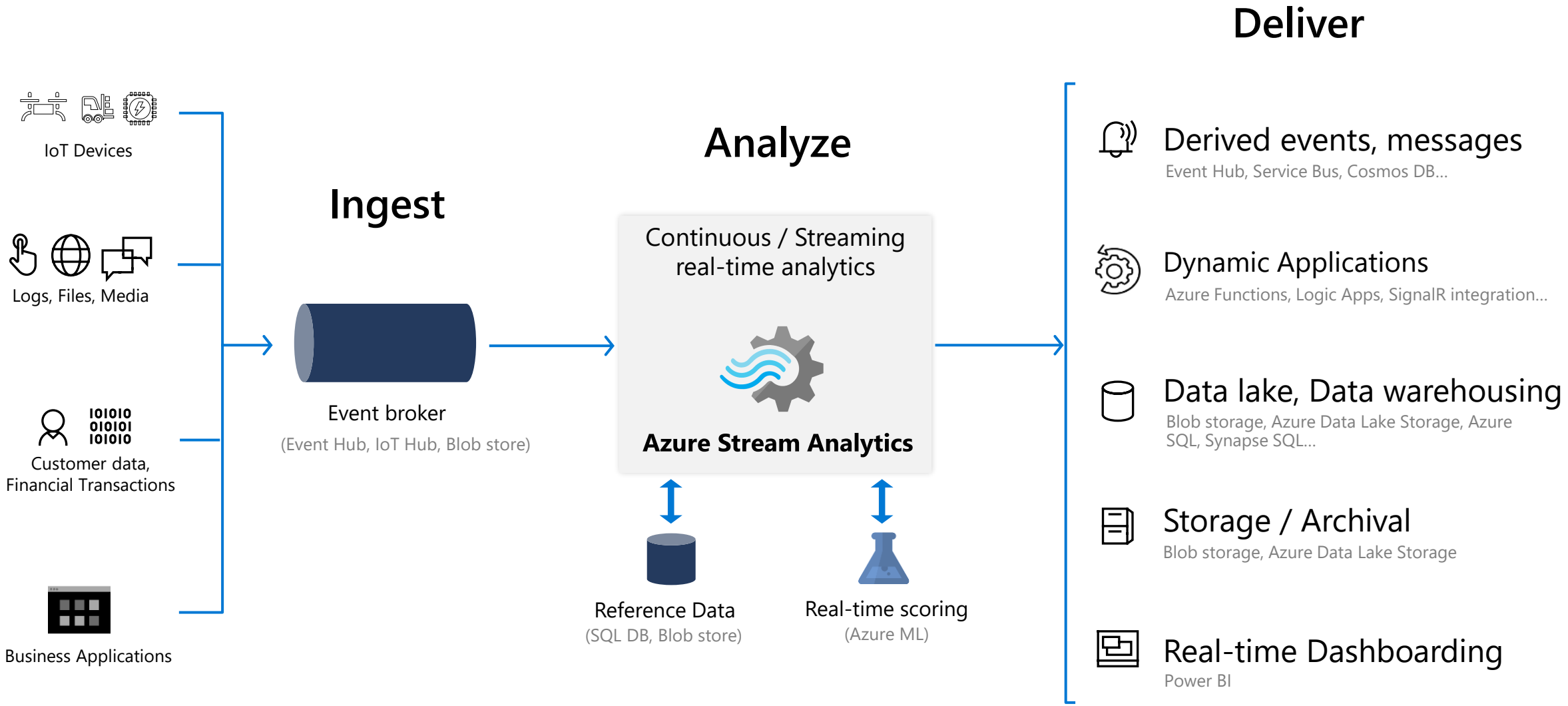


Monitor Equipment

I want to know when an equipment has its average temperature per minute rising above 75.0 degrees

```
SELECT
    EquipementId,
    AVG(Temperature) AS AvgTemperature
INTO NotificationBus
FROM FactoryFloorStream
GROUP BY EquipementId, Tumbling(minute,1)
HAVING AvgTemperature >= 75.0
```

Azure Stream Analytics : Canonical architecture



Concepts

- **Service** level objects
- **Job** level objects
- **Configuration** and options

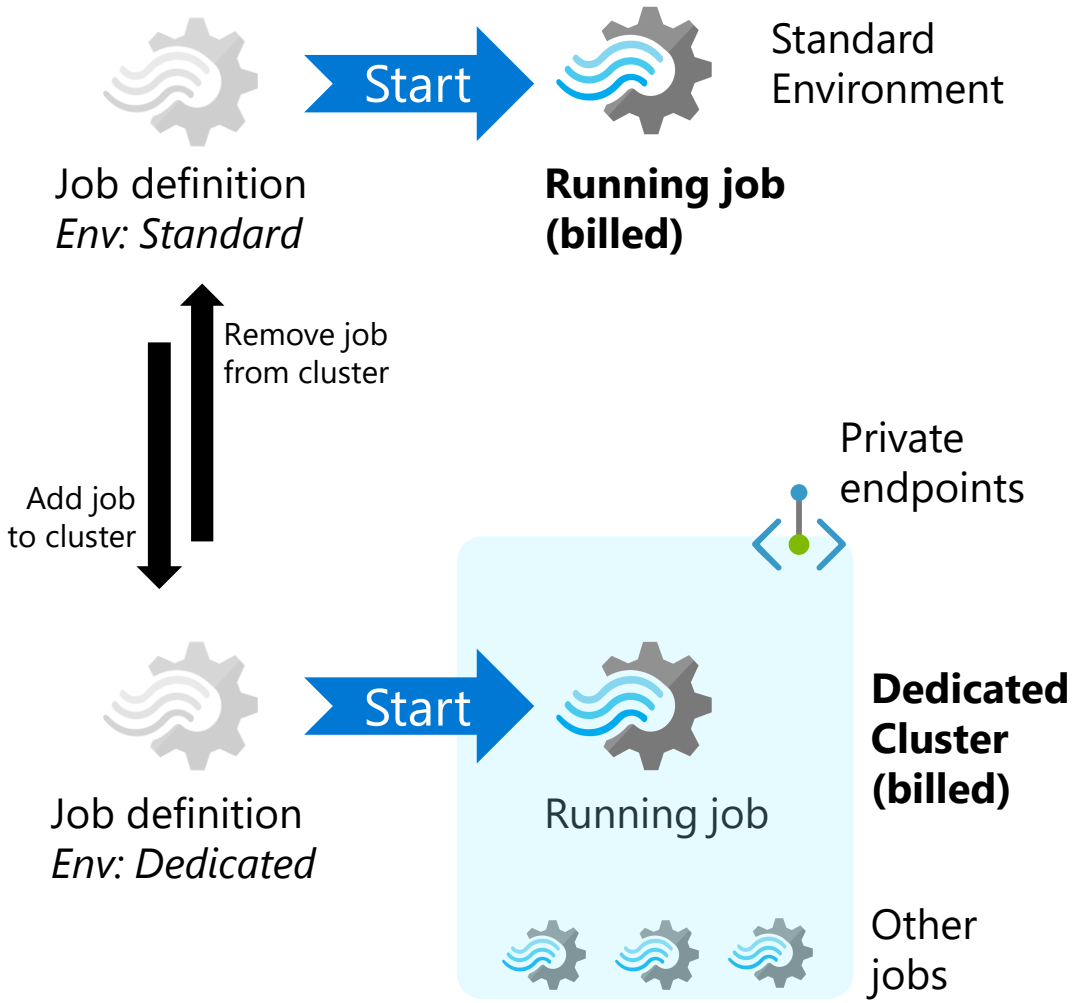
Service level objects

Job : Single streaming query

- Either a single data flow for performance, or multiple data flows for cost optimization (via query steps)
- Can be sized from 1 to 396 SU (scalable up and down), billed only when the job is running
- Streaming solutions can consist of one to multiple ASA jobs interconnected via event hubs. These jobs will be completely independent from the perspective of the ASA service

Cluster : Dedicated hosting environment for jobs

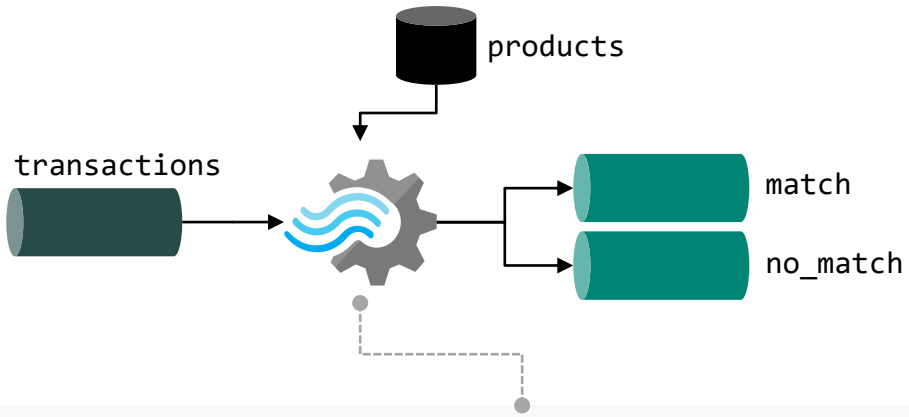
- Provides VNet support via private endpoints, custom deserializers in any region, and more
- By default, a job will be created in the standard (multi-tenant) environment. When a cluster is provisioned, the job configuration can be updated to target it instead (easy migration)
- Can be sized from 36 to 396 SU (scalable up and down), billed even when not used by jobs



Job level objects – high level

An ASA job consists of :

- A single streaming SQL query
- At least one input
- At least one output
- Optionally some functions




```
WITH joined_data AS (
  SELECT t.transactionId, t.productId, p.productName
  FROM transactions AS t
  LEFT JOIN products AS p ON t.productId = p.productId
)
SELECT * INTO match    FROM joined_data WHERE p.productName IS NOT NULL
SELECT * INTO no_match FROM joined_data WHERE p.productName IS NULL
```

Illustration

Scenario

- Incoming transactions are checked against a reference product table. If the product id is not found, transactions are isolated for further downstream processing

Job definition

- Two **inputs** are defined: `transactions` (streaming input) and `products` (reference)
- Two **outputs** are defined : `match` and `no_match`
- The **query** implements the logic : a left join from transactions to products, and two output statements (INTO) with filters

Job level objects - details

Inputs

Each input defines a **connection** to an existing data source

- Streaming inputs read data in real time, from Event Hubs, IoT Hub, and Blob Storage / Data Lake Storage Gen2
- Reference inputs read data from static or slowly changing sources, from Blob Storage / Data Lake Storage Gen2 or SQL Database

In addition to connection information, an input needs to be configured with a **serialization format**, describing how the data is transmitted over the network

- JSON, CSV and Avro are supported natively
- Other formats are supported via custom deserializers (only supported in every region via clusters)

Outputs

Each output defines a connection to an existing data sink

- ASA supports a long list of services including Event Hubs, Blob Storage / Data Lake Storage Gen2, SQL Database, Cosmos DB, Data Explorer, Power BI…

Functions (custom code)

Additional functionality can be provided via custom code, accessed directly from the query

- UDF: User Defined Functions, written in JavaScript or C# (C# are in preview, and only supported in every region via clusters)
- UDA: User Defined Aggregates, written in JavaScript
- Azure ML functions, that wrap and expose existing Azure ML endpoints in the query

Configuration and options : job level

- **Environment** : standard (multi-tenant) or cluster
- **Storage account settings** : To unlock some features (faster Synapse output, custom code…), a job needs to be linked to a storage account (blob / ADLS Gen2)
- **Scale** : Streaming units (from 1 to 396, raised by increments of 6 over 6)
  - 6 SU represent one full compute node
    - From 1 to 6 SU, a job will run on a fraction to an entire single node
    - From 12 SU (2 nodes), a job will be running on multiple nodes, distributing the workload either by data partition (best performance, automatically selected when possible) or query step (lower maximum scale)
  - When a job is distributed by data partition (parallel topology), its scale can be changed when running. If not, it will need to be restarted
- **Event ordering** : time skew policies
  - Only applied when using event time (TIMESTAMP BY clause), ignored when using arrival time (default behavior)
  - Defines how late and out-of-order events will be ingested
- **Error policy** : output data error handling action (default is retry)
- **Compatibility level** : runtime version for backward compatibility (newer is better)
- **Managed identity** :
  - User-defined or system-assigned managed identity definition for the job
  - Central location to check authentication modes to all input and output

Streaming SQL

Streaming SQL : standing query

Subset of T-SQL, extended for time processing and complex types handling, in a **continuously running query**

Classic database query, runs once and returns a result

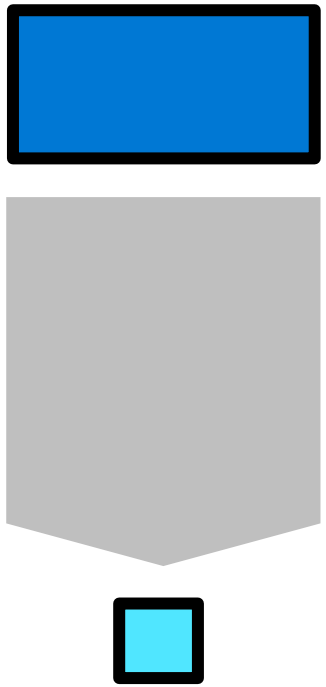
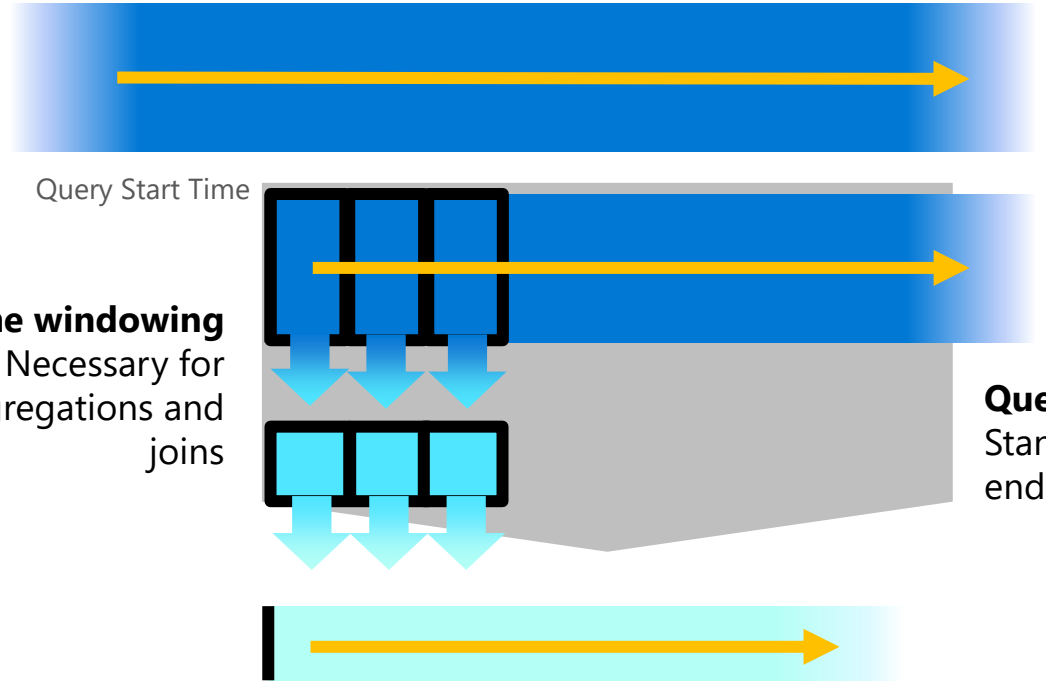


Table
Bounded (isolation)
Unordered

Query
Transient, expected to yield

Result set
Bounded

Streaming query, runs continuously, returns output results as data is arriving in input, event per event or via time windows



Input stream
Unbounded
Ordered (time)

Time windowing
Necessary for aggregations and joins

Query
Standing, never ends

Output stream
Unbounded, or collapsed into a table

Streaming SQL : A subset of T-SQL

Language Elements

SELECT, INTO, **FROM**, JOIN, CROSS APPLY, WHERE, **GROUP BY**, HAVING

WITH, UNION

CASE, COALESCE, NULLIF

MATCH_RECOGNIZE

Typical query flow

1. Reading and transformation steps via Common Table Expressions (WITH) – optional
2. Output via SELECT INTOs statements

```
WITH
step1 AS (
    SELECT ...
    FROM input1
    ...
),
step2 AS (
    SELECT ...
    FROM step1
    WHERE ...
),
...

-- Core business logic
SELECT ... INTO SQLOutput  FROM step5

-- Rejected output, for further
downstream processing
SELECT *   INTO BlobOutput FROM step6
```

Streaming SQL : Functions

Aggregate

Require GROUP BY + Time Window

AVG, COUNT, MAX, MIN, SUM

Collect, CollectTop, TopOne

STEDEV, STEDVP, VAR, VARP

Analytic

Require OVER clause, most aggregates are available as analytic function too

ISFIRST, LAG, LAST

AnomalyDetection_SpikeAndDip, AnomalyDetection_ChangePoint

Conversion

TRY_CAST is recommended for resilience

CAST, TRY_CAST, GetType

Date and time

DATEADD, DATEDIFF

DATENAME, DATEPART, DATETIMEFROMPARTS

DAY, MONTH, YEAR

Mathematics

ABS, CEILING, EXP, FLOOR, POWER, ROUND, SIGN, SQUARE, SQRT

Geospatial

CreateLineString, CreatePoint, CreatePolygon

ST_DISTANCE, ST_OVERLAPS, ST_INTERSECTS, ST_WITHIN

String

CHARINDEX, PATINDEX, REGEXMATCH, REPLACE, TRANSLATE

LEFT, RIGHT, SUBSTRING, LTRIM, RTRIM, TRIM, CONCAT, CONCAT_WS

LEN, REPLICATE, REVERSE

LOWER, UPPER, SPACE, STUFF

NCHAR, UNICODE

Bitwise Operators

& (AND), | (OR), ^ (XOR), ~ (NOT)

And more (next slides)

Array and Record Functions

Input Metadata Functions

Windowing Functions

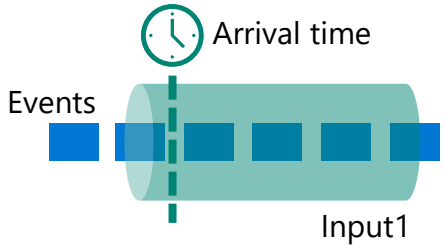
Specific to ASA : Time management

Arrival time VS Application time, defined with TIMESTAMP BY

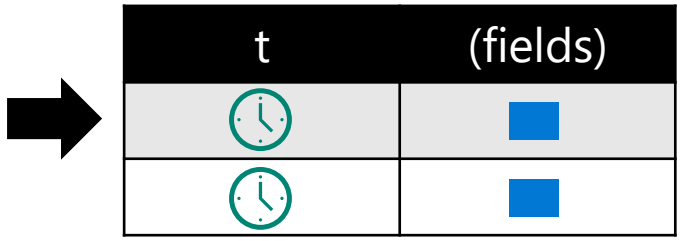
- System.Timestamp() is used to project the current timestamp

Arrival time

- The time the event reached the input broker
- Ignores time skew policies
- Default behavior

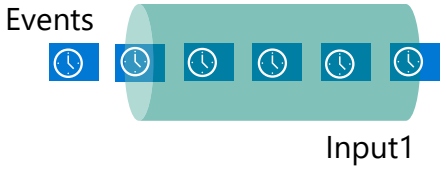


```
SELECT
    System.Timestamp() AS t,
    *
FROM input1
```

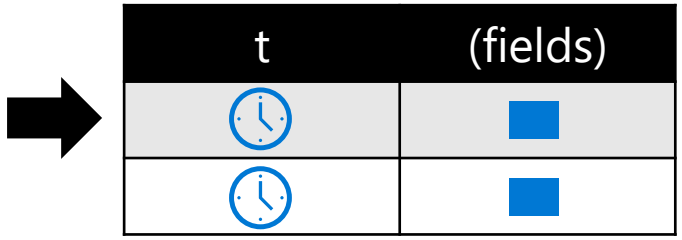


Application time

- As defined via a field in the payload
- Leverages time skew policies to handle late (adjust/discard) and out-of-order (buffer to sort) events



```
SELECT
    System.Timestamp() AS t,
    *
FROM input1
TIMESTAMP BY mytimestamp
```

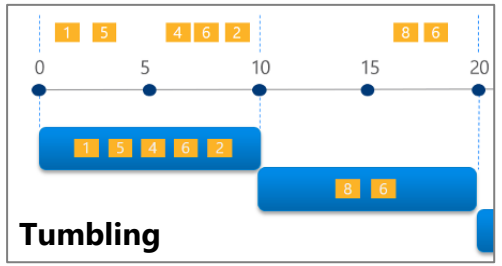


Specific to ASA : Time management (windows)

Windowed functions : bounding data for grouping operations

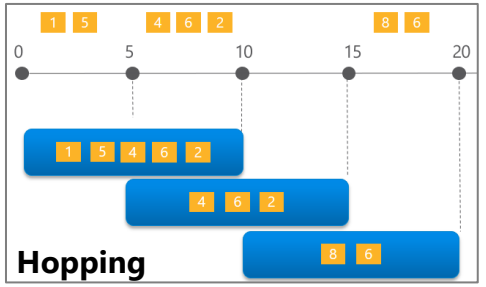
GROUP BY **Tumbling**(second,10)

10 second window, every 10 seconds



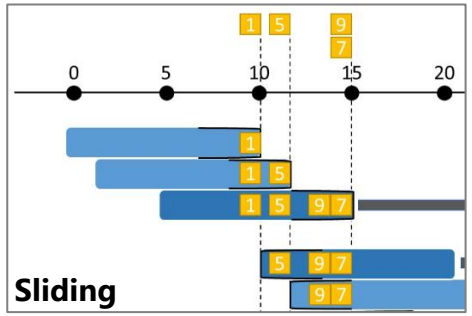
GROUP BY **Hopping**(second,10,5)

10 second window, every 5 seconds



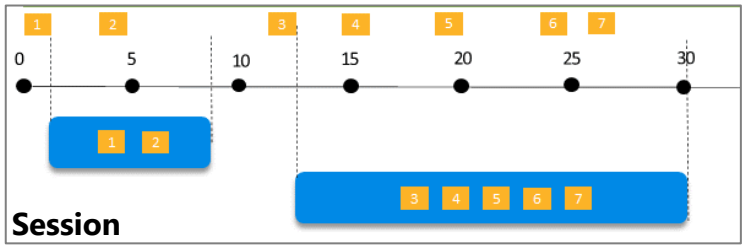
GROUP BY **Sliding**(second,10)

10 second window, every time a record enter or leave the window



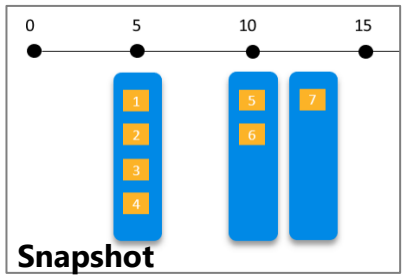
GROUP BY **Session**(second,5,10)

max 10 second window, max 5 seconds timeout between records



GROUP BY **Snapshot**()

At every timestamp where at least a record exist



Specific to ASA : Time management (example)

Sum and average by key over a 1-minute tumbling window

Step1

- Timestamping by myTimestamp

Step2

- Aggregation
- WindowStart and WindowEnd calculation

INTO

- Output to destination

myString	WindowStart	WindowEnd	mySum	myCount
A	00:00	00:01	120.75	4721
B	00:00	00:01	2316.64	89451
A	00:01	00:02	119.01	4614
B	00:01	00:02	2577.12	93104
C	00:01	00:02	1245.54	266

```
WITH
Step1 AS (
    SELECT
        myString,
        myNumeric
    FROM myInput TIMESTAMP BY myTimestamp
),

Step2 AS (
    SELECT
        DATEADD(minute,-1,System.Timestamp()) AS WindowStart,
        System.Timestamp() AS WindowEnd,
        myString,
        SUM(myNumeric) AS mySum,
        COUNT(*) AS myCount
    FROM Step1
    GROUP BY
        Tumbling(minute,1),
        myString
)

SELECT
    myString,
    WindowStart,
    WindowEnd,
    mySum,
    myCount
INTO myOutput
FROM Step2
```

Specific to ASA : Type management

ASA has its own type system, and provides implicit casting in most cases

- Supported types : `bit`, `bigint`, `float`, `nvarchar(max)`, `datetime`, `record`, `array`
- Explicit casting (TRY_CAST) is highly recommended for resilience

Five factors will impact data types from input to output :

ASA **input type mapping**

Query : implicit and explicit conversions

ASA **output type mapping**

Input format
Schema on read with a type system inherited from the serialization format: JSON, AVRO, CSV…

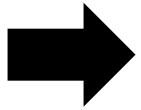
Output format
Depending on the service, schema on read (eh, Cosmos DB, blob/adls…) or schema on write (SQL databases…)
Error policy applies here : discard/retry

Specific to ASA : Complex types support (records, arrays)

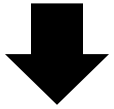
Dotted notation, record (SensorReadings) and array (sensor03) projection

Input event

```
{
  "DeviceId" : "12345",
  "SensorReadings" : {
    "Temperature" : 80,
    "Humidity" : 70,
    "Sensor01" : 5,
    "Sensor02" : 99,
    "Sensor03": [12,-5,0]
  },
  "SensorMetadata": [
    { "smKey":"Manufacturer", "smValue":"ABC" },
    { "smKey":"Version", "smValue":"1.2.45" }
  ]
}
```



```
SELECT
  DeviceId,
  SensorReadings,
  SensorReadings.Sensor03 AS Sensor03,
  SensorMetadata
 FROM Input1
```



DeviceId	SensorReadings	Sensor03	SensorMetadata
"12345"	{ "Temperature" : 80, "Humidity" : 70, "Sensor01" : 5, "Sensor02" : 99, "Sensor03": [12,-5,0] }	[12,-5,0]	[{ "smKey":"Manufacturer", "smValue":"ABC" }, { "smKey":"Version", "smValue":"1.2.45" }]
String	Record	Array (of integers)	Array (of records)

Specific to ASA : Complex types support (functions)

Array Functions

- GetArrayLength : returns the array length
- GetArrayElement : returns a single value at a given position
- GetArrayElements : expands an array into rows (with CROSS APPLY, example next slide)

Record Functions

- GetRecordProperties : splits a record into rows (with CROSS APPLY, example next slide)
- GetRecordPropertyValue : returns a single value from a property name (dynamic or not)

Input Metadata Function

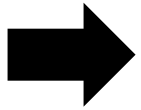
- GetMetadataPropertyValue : surfaces input metadata properties, system and user

Specific to ASA : Complex types support (array expansion)

CROSS APPLY + GetArrayElements to expand an array into rows

Input event

```
{
  "DeviceId" : "12345",
  "SensorReadings" : {
    "Temperature" : 80,
    "Humidity" : 70,
    "Sensor01" : 5,
    "Sensor02" : 99,
    "Sensor03": [12,-5,0]
  },
  "SensorMetadata": [
    { "smKey":"Manufacturer", "smValue":"ABC" },
    { "smKey":"Version", "smValue":"1.2.45" }
  ]
}
```



```
SELECT
  I.DeviceId,
  'Sensor03' AS Sensor,
  S03.ArrayValue AS Value
FROM Input1 AS I
CROSS APPLY GetArrayElements(I.SensorReadings.Sensor03 AS S03
```

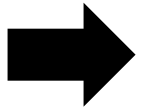
DeviceId	Sensor	Value
"12345"	Sensor03	12
"12345"	Sensor03	-5
"12345"	Sensor03	0

Specific to ASA : Complex types support (record splitting)

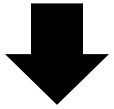
CROSS APPLY + GetRecordProperties to split a record into rows

Input event

```
{
  "DeviceId" : "12345",
  "SensorReadings" : {
    "Temperature" : 80,
    "Humidity" : 70,
    "Sensor01" : 5,
    "Sensor02" : 99,
    "Sensor03": [12,-5,0]
  },
  "SensorMetadata": [
    { "smKey":"Manufacturer", "smValue":"ABC" },
    { "smKey":"Version", "smValue":"1.2.45" }
  ]
}
```



```
SELECT
  I.DeviceId,
  SR.PropertyName AS Sensor,
  SR.PropertyValue AS Value
FROM Input1 AS I
CROSS APPLY GetRecordProperties(I.SensorReadings) AS SR
```

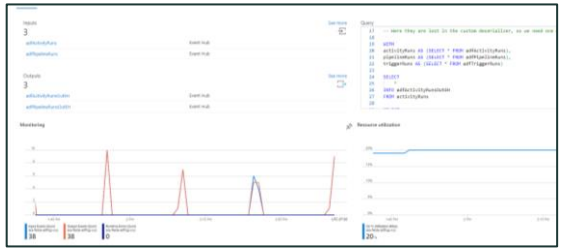


DeviceId	Sensor	Value
"12345"	Temperature	80
"12345"	Humidity	70
"12345"	Sensor01	5
"12345"	Sensor02	99
"12345"	Sensor03	"[12,-5,0]"

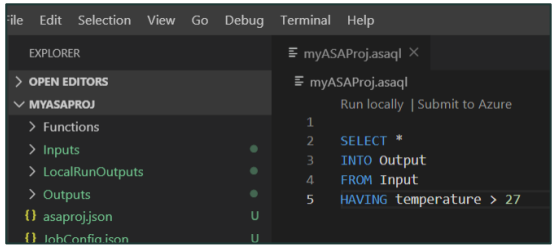
Developer Experience

Development tools

Azure portal



Visual Studio Code



No code editor (preview)

Full comparison

	Azure Portal	Visual Studio Code	No code editor (Preview)
Authoring	Write and test query with samples Syntax highlighting	Write and test query with samples or live data Syntax highlighting, code completion, error marker	Visual designer with live data
Job level configuration	All settings	Most settings	Not applicable
Source control	No	Yes	No
CI/CD support	Partial	Yes (npm package)	No
Unit testing	No	Yes (npm package)	No
C# Custom Code	Read-only	Yes	No
JavaScript Custom Code	Yes	Yes	No

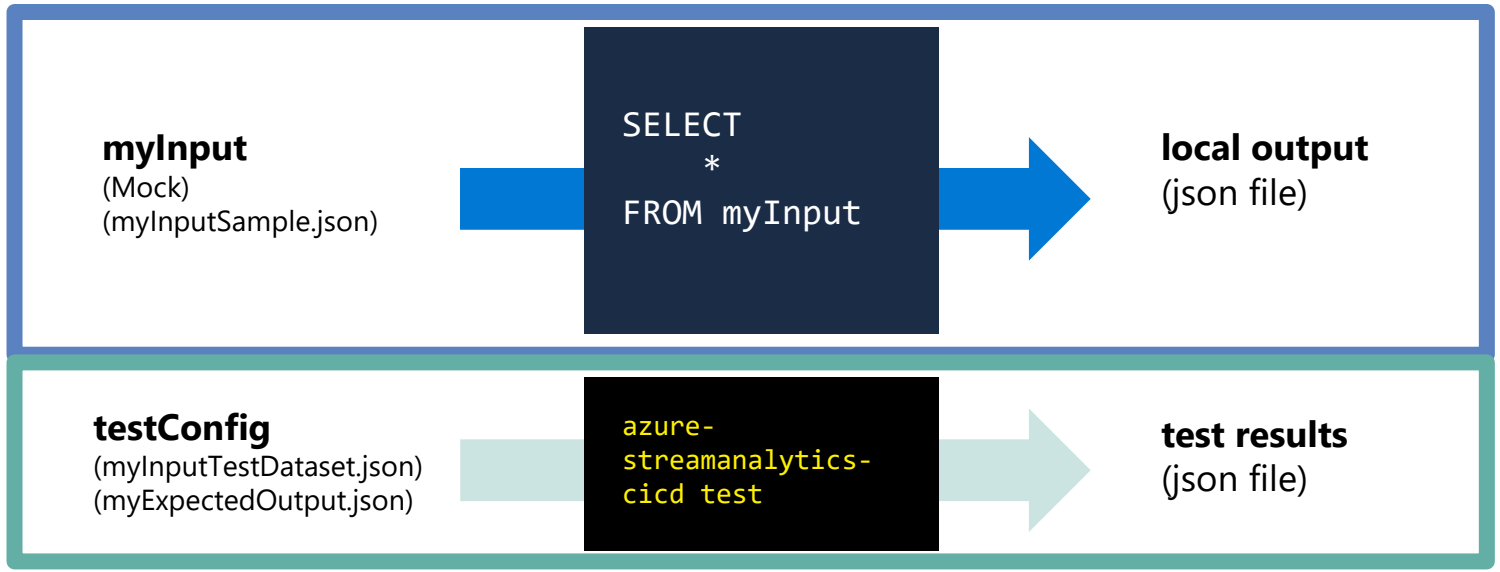
Local developer experience with VS Code

Local run modes

- **Local** input to **local** output: best for offline development at no cost, unit testing with the npm package…
- **Live** input to **local** output: best for input configuration, de-serialization, and partitioning debugging…
- **Live** input to **live** output: best for output configuration, serialization, and conversion errors debugging…

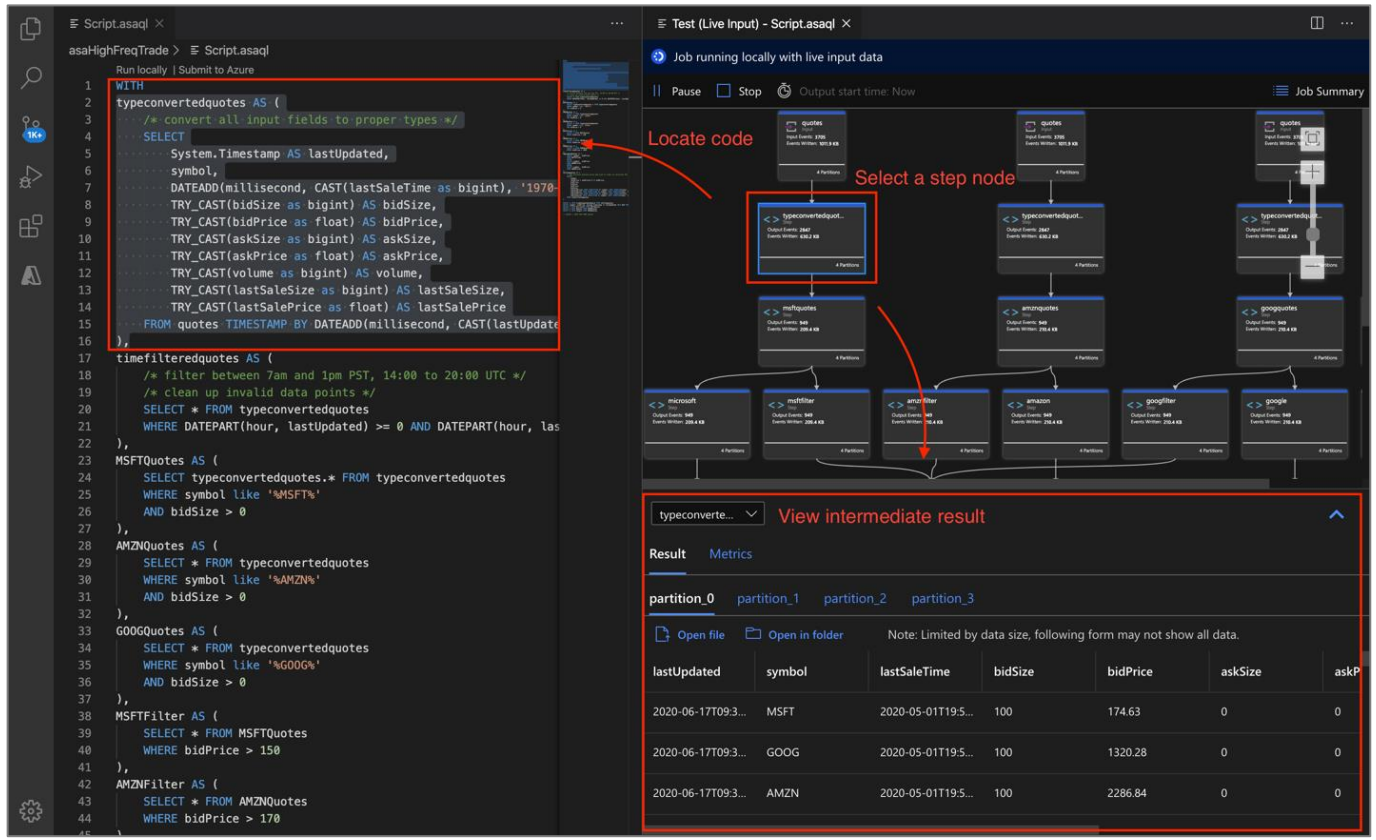
Environment	Mode	Input	Output
VS Code	Local input to local output	JSON/CSV/AVRO files	JSON files (the corresponding live output format isn't used even if it exists)
VS Code	Live input to local output	All input adapters	JSON files (the corresponding live output format isn't used even if it exists)
VS Code	Live input to live output	All input adapters	Event Hub, Storage Account, Azure SQL
Azure	N/A	All input adapters	All output adapters

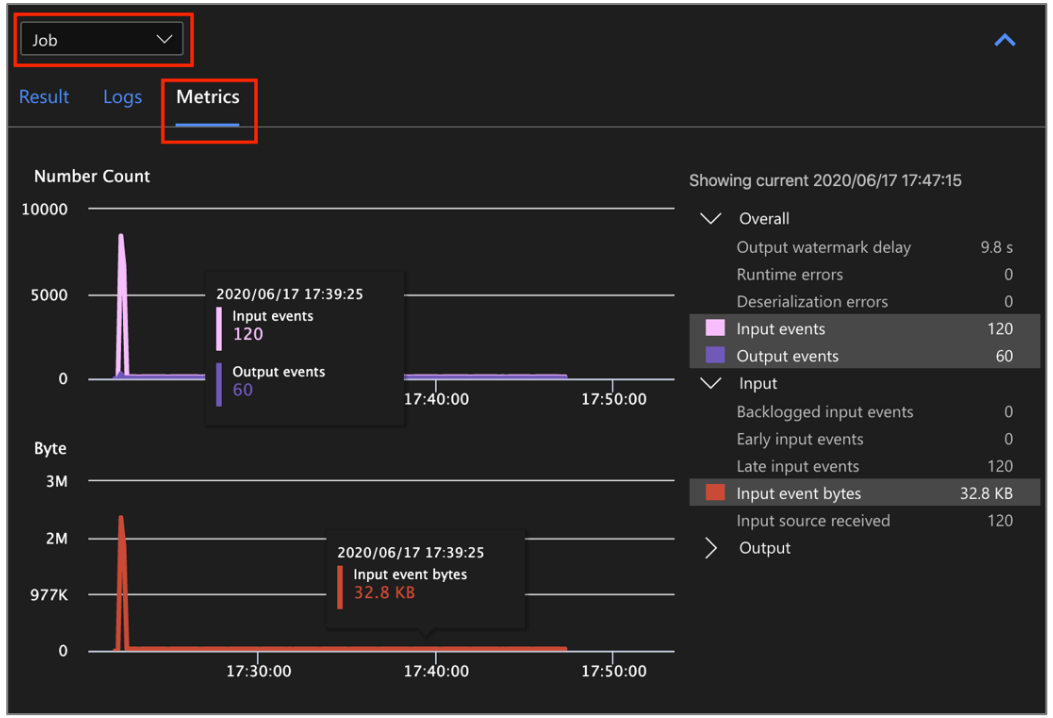
Unit testing via a companion npm package



Troubleshooting

- Reference : input, output, query, logs, metrics, data quality
- Tooling : Job diagram in VS Code





Management APIs and SDKs

5 options, intended to be aligned

- .NET SDK : Reference, Sample
- REST API : Reference
- Azure CLI : Reference
- Azure PowerShell : Reference, Sample: auto-pause
- Terraform : azurerm provider

Operations

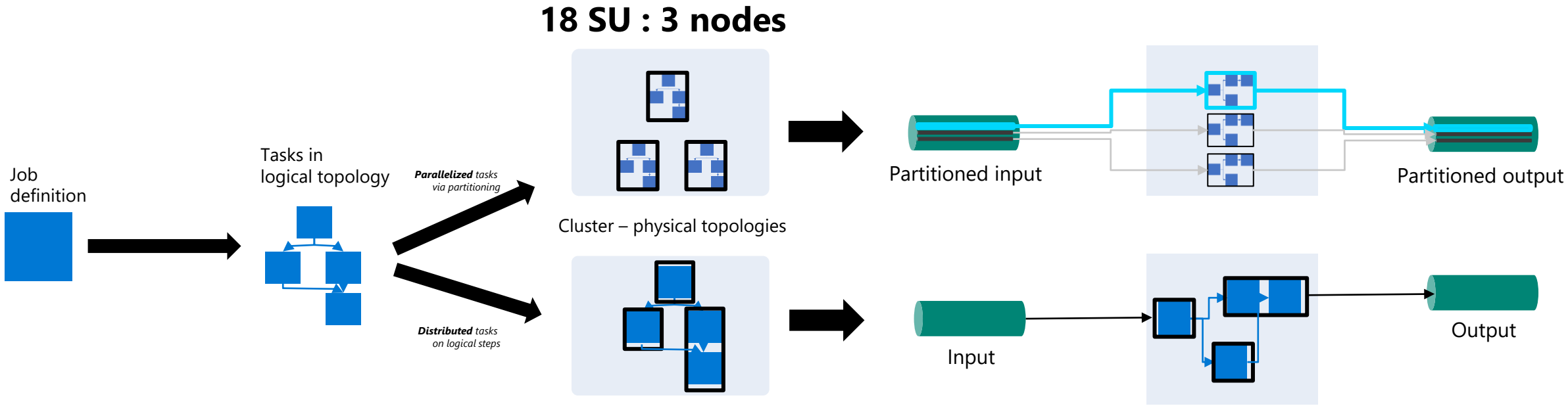
Scaling : streaming units and distribution modes

Streaming units : 6 SU for one full compute node

- From 1 to 6 SU, a job will run on a fraction to an entire single node
- From 12 SU (starting at 2 nodes), a job will be running on multiple nodes, distributing the workload either by data partition (sharding) or query steps

Distribution modes

- Partitioning gives the best performances and is automatically selected when possible. It requires the input, query and output to be aligned (same key). Its scale can be changed live, enabling auto-scale
- Query step offers less scalability. It also requires the job to be restarted to scale up and down (topology change)

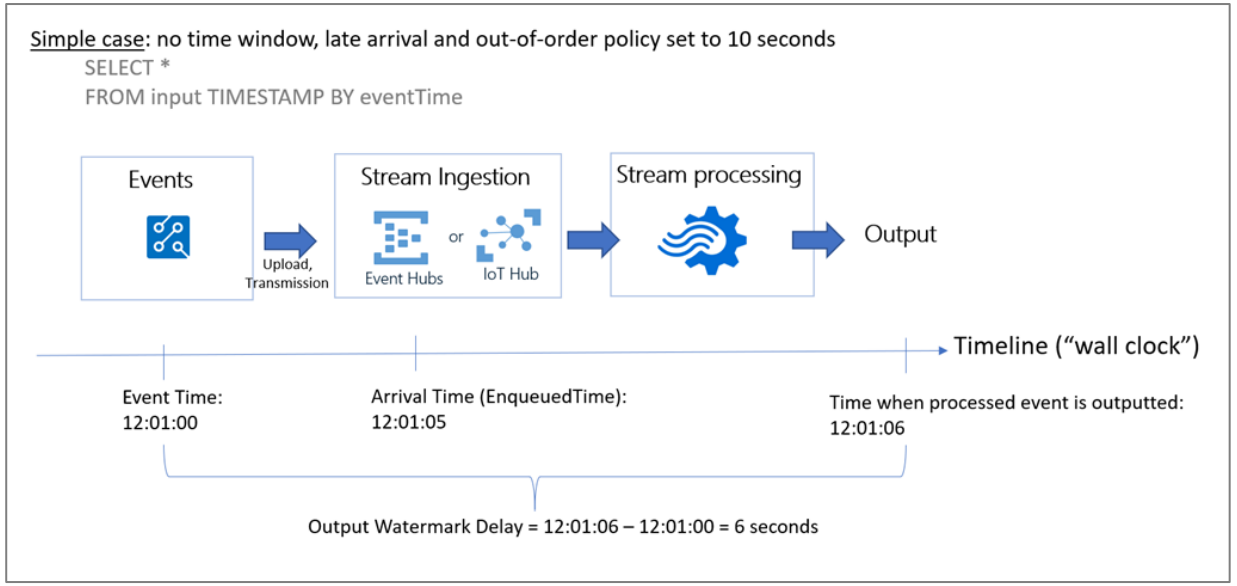


Monitoring

Both logs and metrics are available in the Azure portal and VS Code

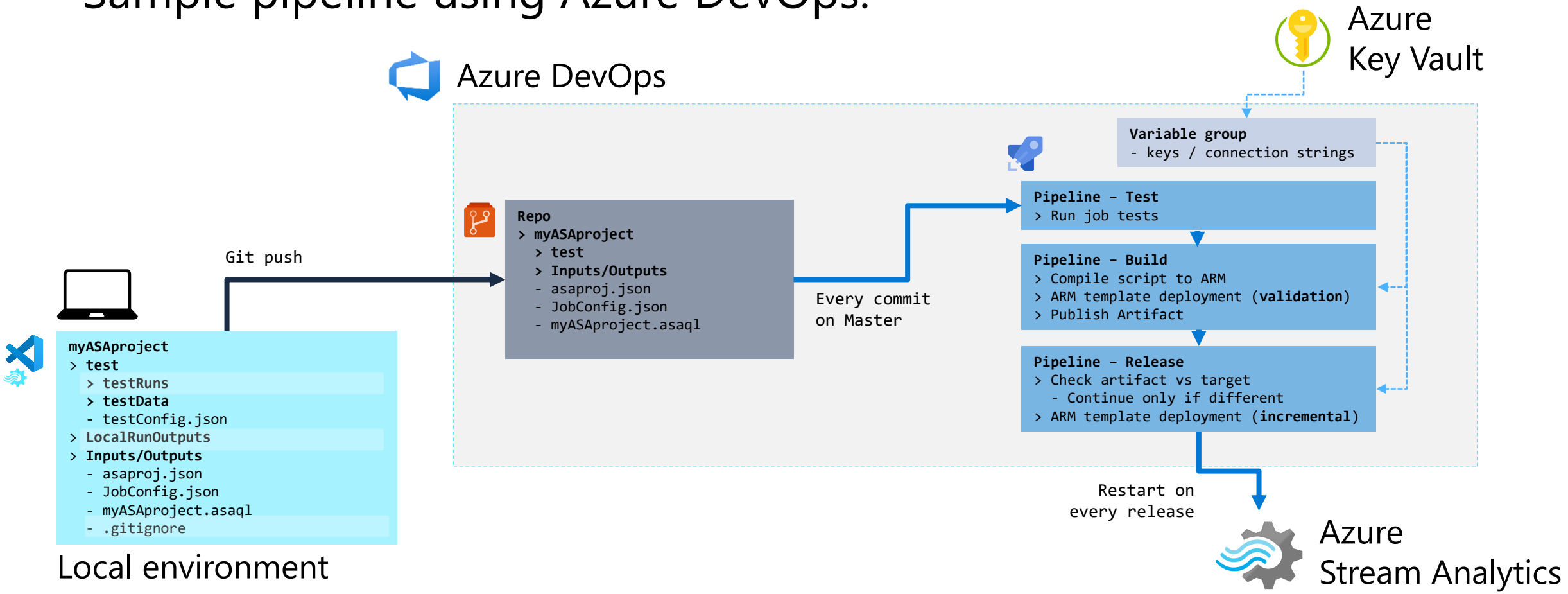
- Logs : enable diagnostics logging to a Log Analytics workspace for a better logging experience (default is off)
- Key metrics
  - **Backlogged Input Events** : Number of input events that are backlogged. A non-zero value for this metric implies that your job isn't able to keep up with the number of incoming events…
  - **CPU % Utilization** : The percentage of CPU utilized by your job. Even if this value is very high (90% or above), you should not increase number of SUs based on this metric alone…
  - **SU % Utilization** : The percentage of memory utilized by your job. If SU % utilization is consistently over 80%, the watermark delay is rising…
  - **Watermark Delay** : The maximum watermark delay across all partitions of all outputs in the job.

Understanding watermark delay



CI/CD – Continuous Integration / Continuous Deployment

- Reference : overview, setting up a pipeline, build tool
- Sample pipeline using Azure DevOps:

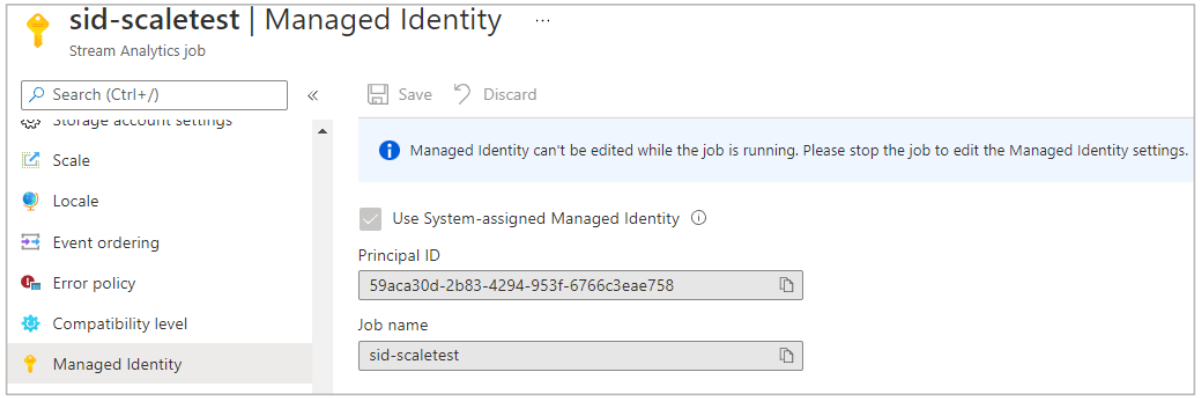


Security

Reference : security baseline, data protection (CMK), policies

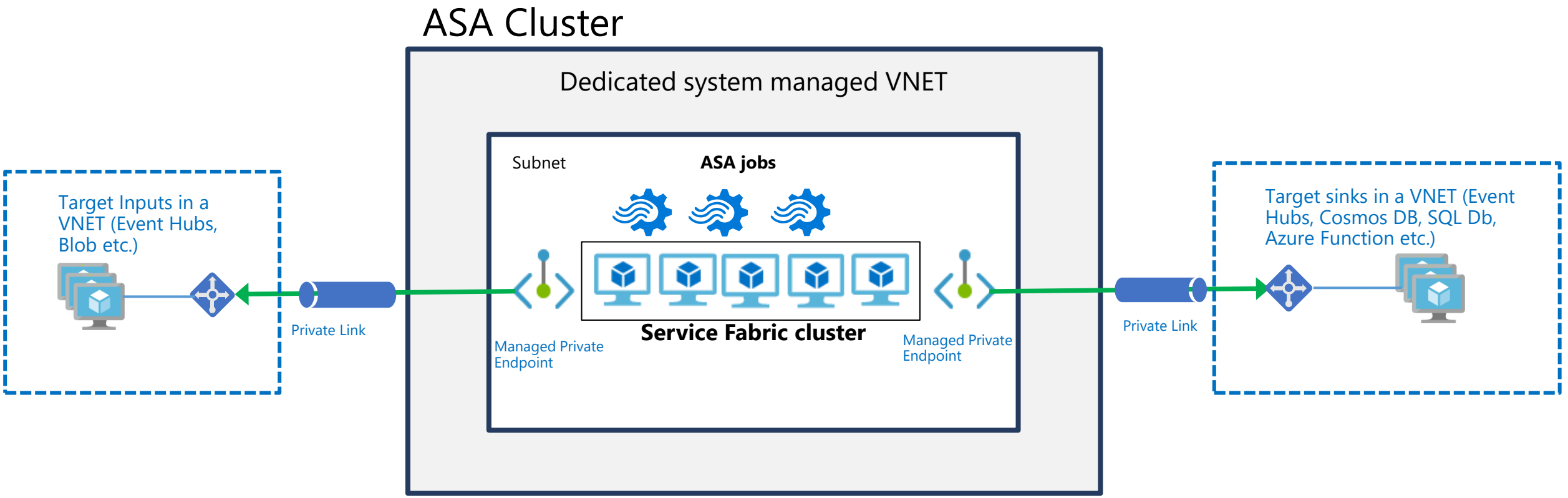
Managed Service Identity (MSI) authentication

- Key Steps
  1. Create a System-Assigned Managed Identity for your ASA job. Or configure job to use User-Assigned Managed Identity.
  2. Grant your job's managed identity permissions to access input or output resources (like Event Hubs, ADLS etc.).
  3. Your job will be able to connect to inputs/outputs using this identity.
- Reference
  - Overview of managed identities for ASA
  - User-assigned managed identities for ASA



Networking

- Standard SKU : networking via public cloud (trusted service via MSI)
- Dedicated SKU (cluster) : managed private endpoints



Guarantees and resilience

Guarantees

- Repeatability (deterministic re-playability) : same inputs will always generate the same outputs
- Only once processing : no data loss, potential duplication with at-least-once delivery
  - Exactly once delivery : Azure Table and Cosmos DB with upsert, SQL with unique key (duplicates will be rejected)
- 3 x 9s SLA
- In-region data residency

Resilience

- Query level
  - Leverage input validation to prevent data quality issue related crashes
- Job level
  - Set up job monitoring and create alarms on the scenarios to monitor (memory pressure, watermark delays…)
  - Pair jobs for reliability and/or geo-redundancy (not yet supported at the service level)

Getting started with Azure Stream Analytics

Getting hands-on

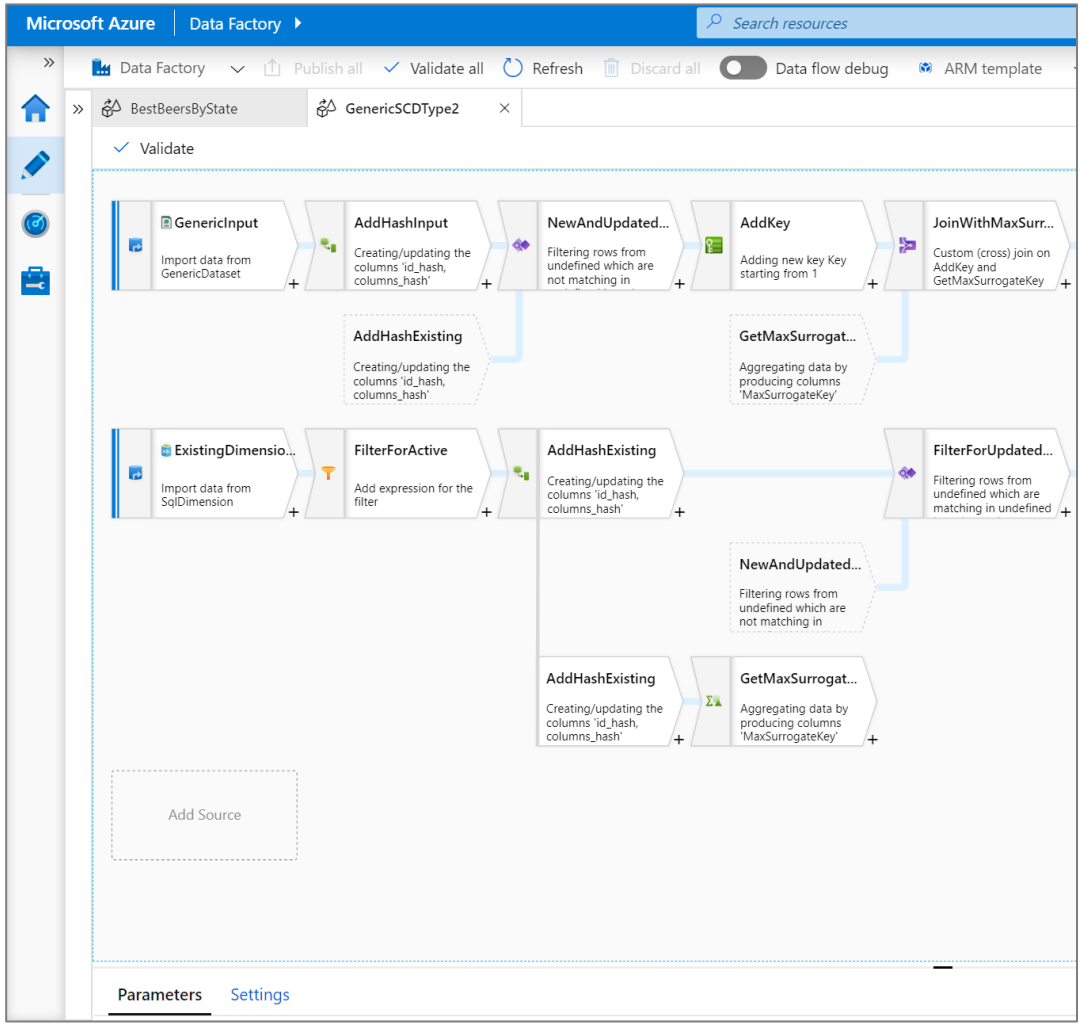
- Create a job in the portal : Quickstart - Azure portal
- Create a job in VS Code : Quickstart - VS Code
  - Using local files to bypass the need of an event generator

Learn more

- Introduction to Azure Stream Analytics : Microsoft Docs
- (External) Streaming 101: The world beyond batch – O'Reilly (oreilly.com)
- Understanding time handling in ASA : Microsoft Docs

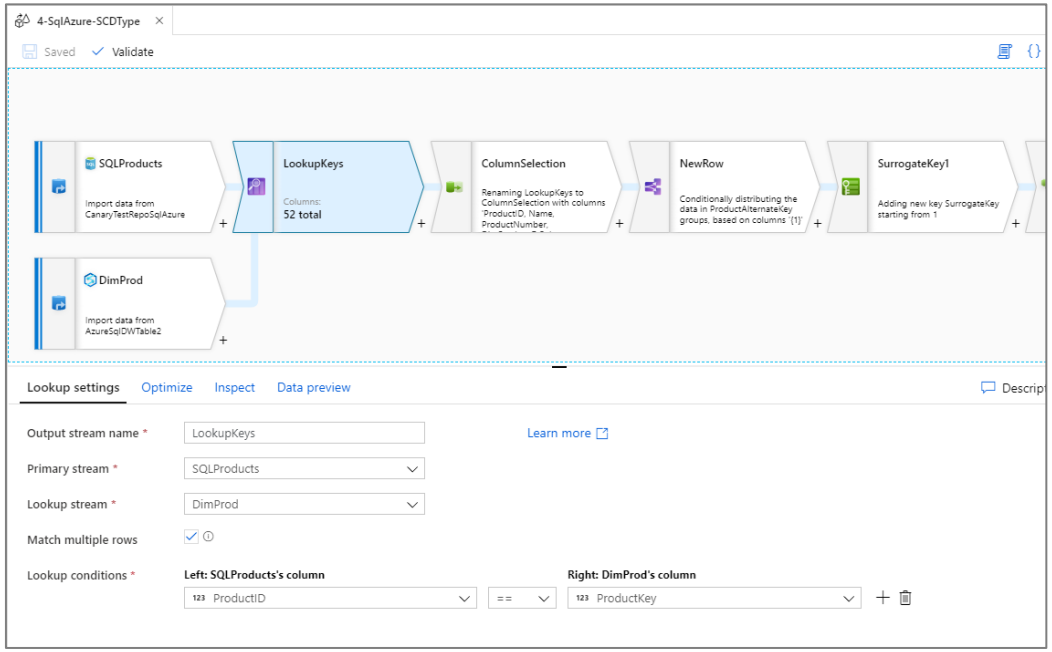
What are mapping data flows?

- Code-free data transformation at scale
- Serverless, scaled-out, ADF/Synapse-managed Apache Spark™ engine
- Resilient flows handle structured and unstructured data
- Operationalized as a data pipeline activity

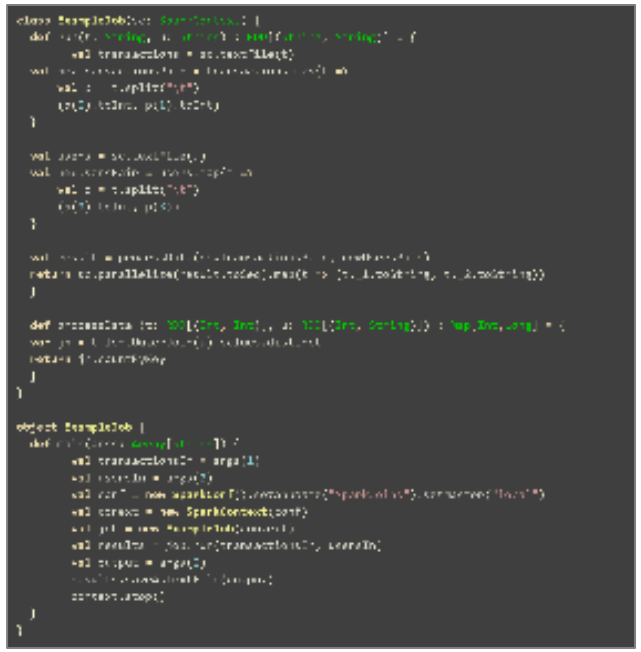


Code-free data transformation at scale

- Intuitive UX lets you focus on building transformation logic
  - Data cleansing
  - Data validation
  - Data aggregation
- No requirement of knowing Spark, cluster management, Scala, Python, etc



vs



ADF Data Flows Service Architecture

ADF Data Flow UI Designer

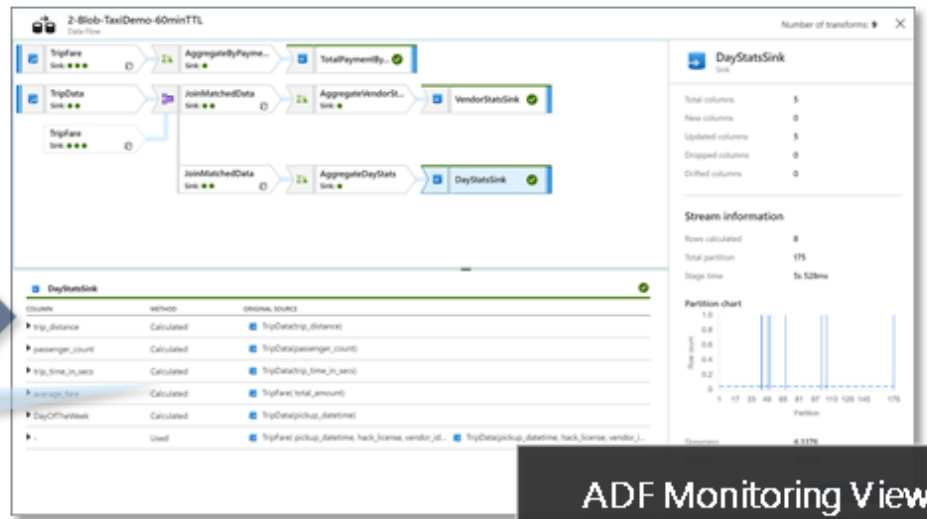
Design, debug, manage data transform logic in browser UI

```
Data Flow Name  metadataChecker

1  parameters{
2    filename as string ('SampleData/moviesDB.csv'),
3    columns as string ('movie,title')
4  }
5  source(allowSchemaDrift: true,
6    validateSchema: false,
7    limit: 1,
8    wildcardPaths:[($filename)] ~> source1
9  DerivedColumn1 foldDown(unroll(cols),
10   mapColumn(
11     cols
12   ),
13   skipDuplicateMapInputs: false,
14   skipDuplicateMapOutputs: false) ~> Flatten1
15 Flatten1 filter(instr($columns,cols) > 0) ~> Filter1
16 source1 derive(cols = columnNames()) ~> DerivedColumn1
17 Filter1 sink(allowSchemaDrift: true,
18   validateSchema: false,
```

Data Flow Script

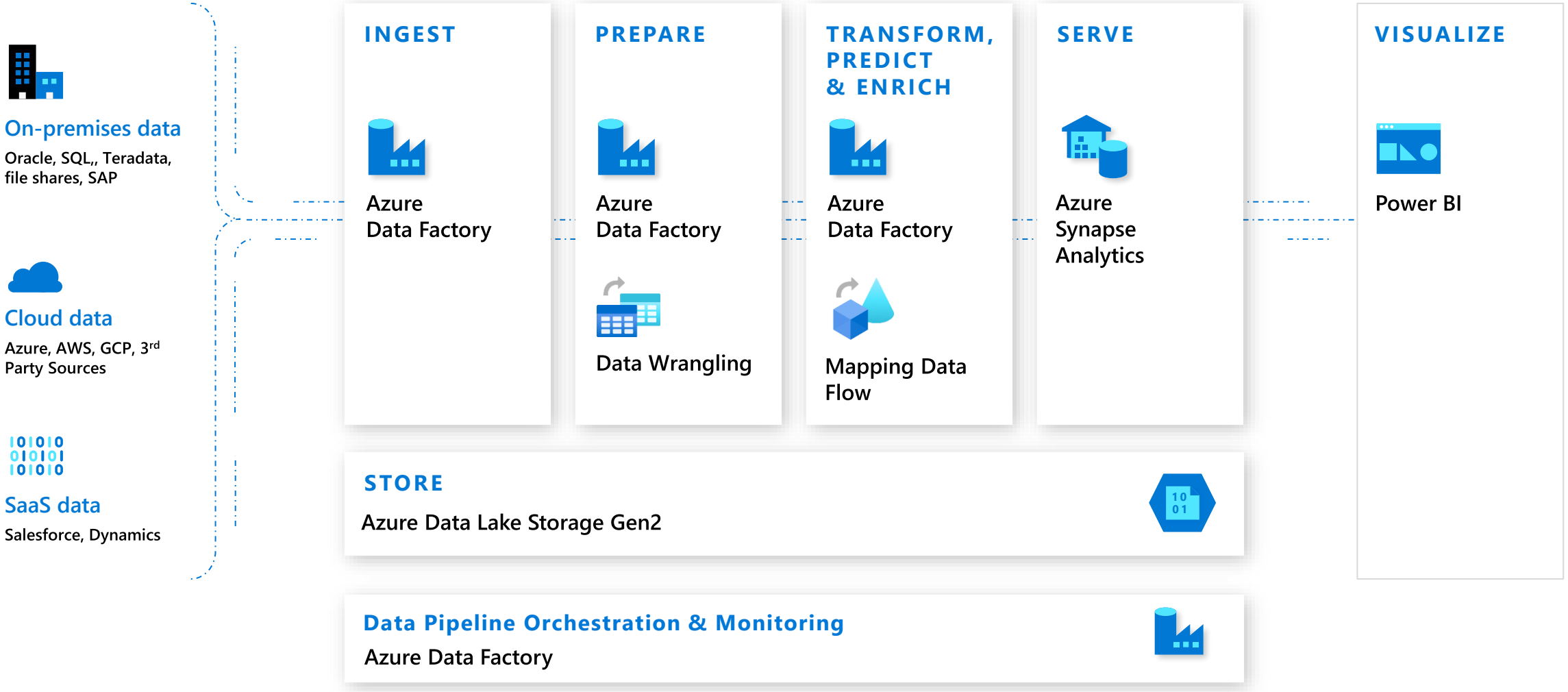
UI builds data transformation scripts that contain metadata defining your logic. Script payload is combined with the ADF JSON definition of your pipeline activities. ADF spins-up a JIT on-demand Spark cluster and builds an execution plan for your data flow in Spark.

Using your Azure IR configs, ADF will spin-up Spark clusters and an executor job will execute your data transformation logic. The results of the data ingest, transformation, data partitioning, data egress, and timings will all appear in the monitoring view.



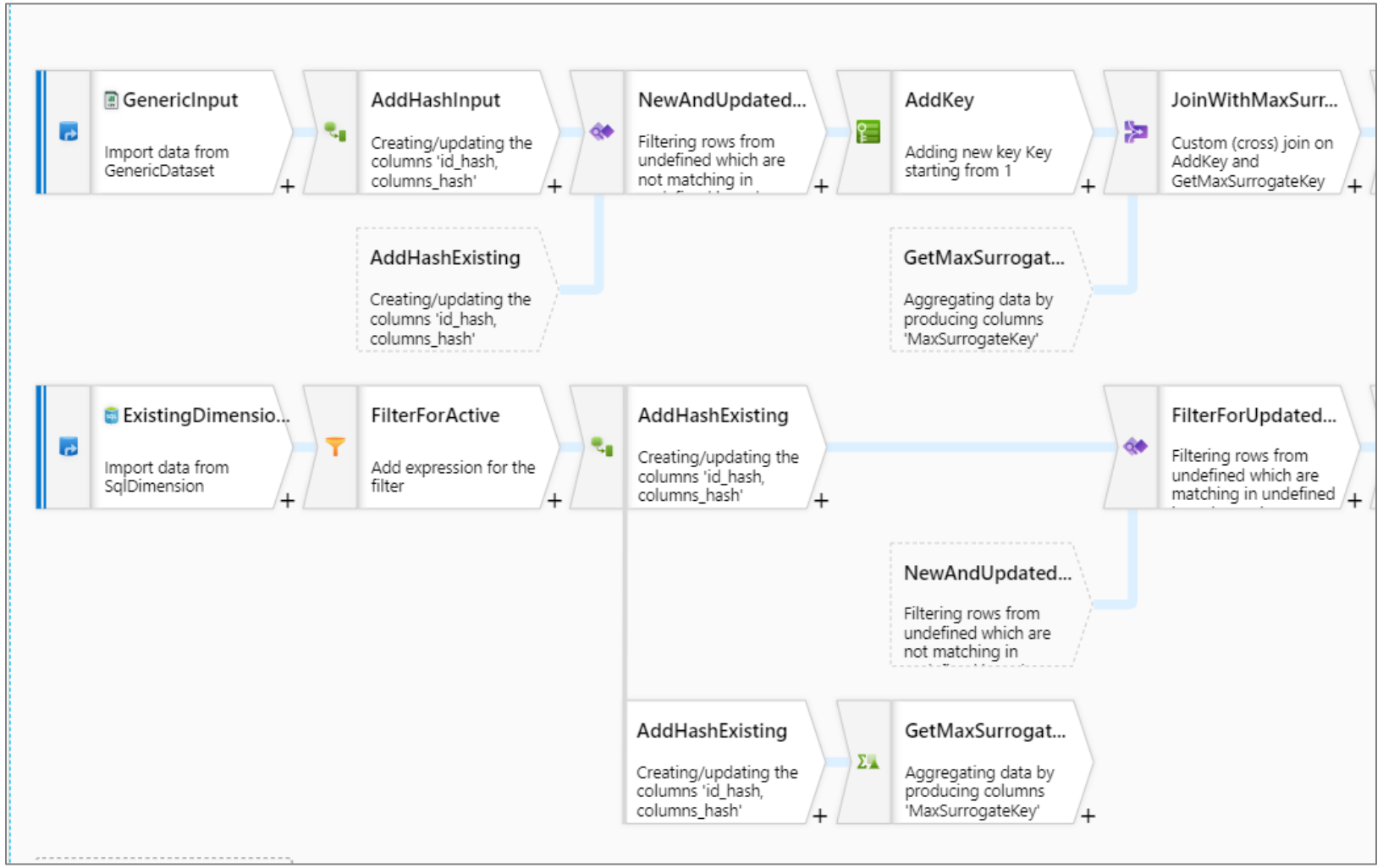
ADF Monitoring View

Modern Data Warehouse (MDW)

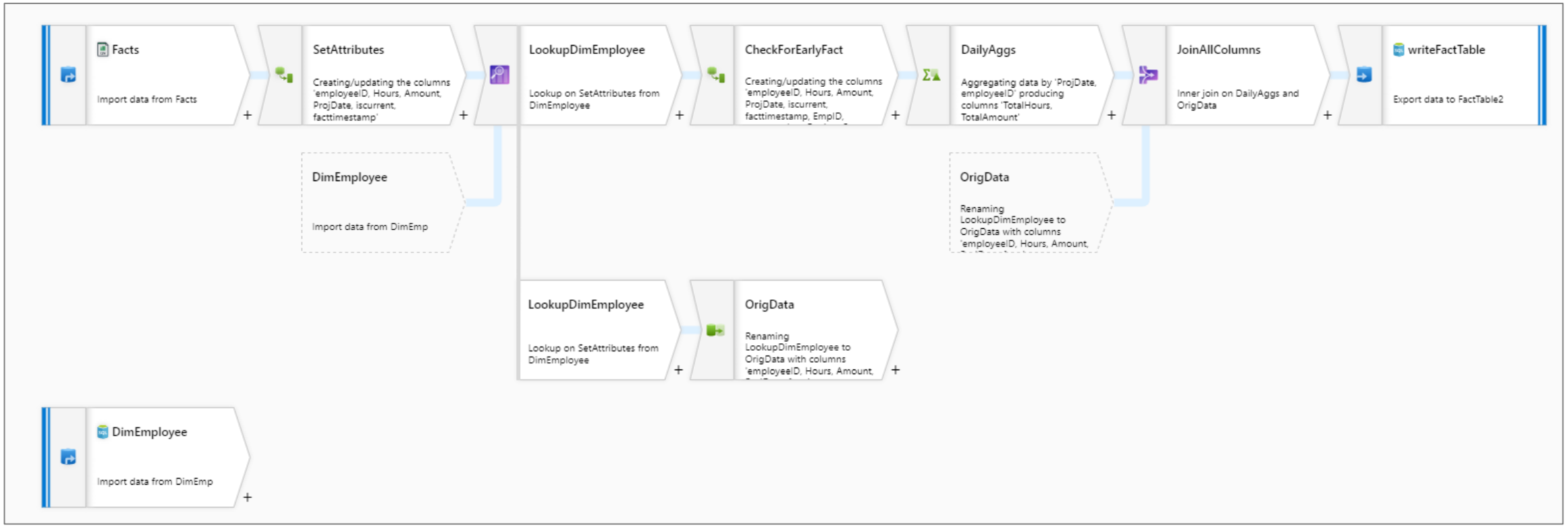


Common data flow scenarios

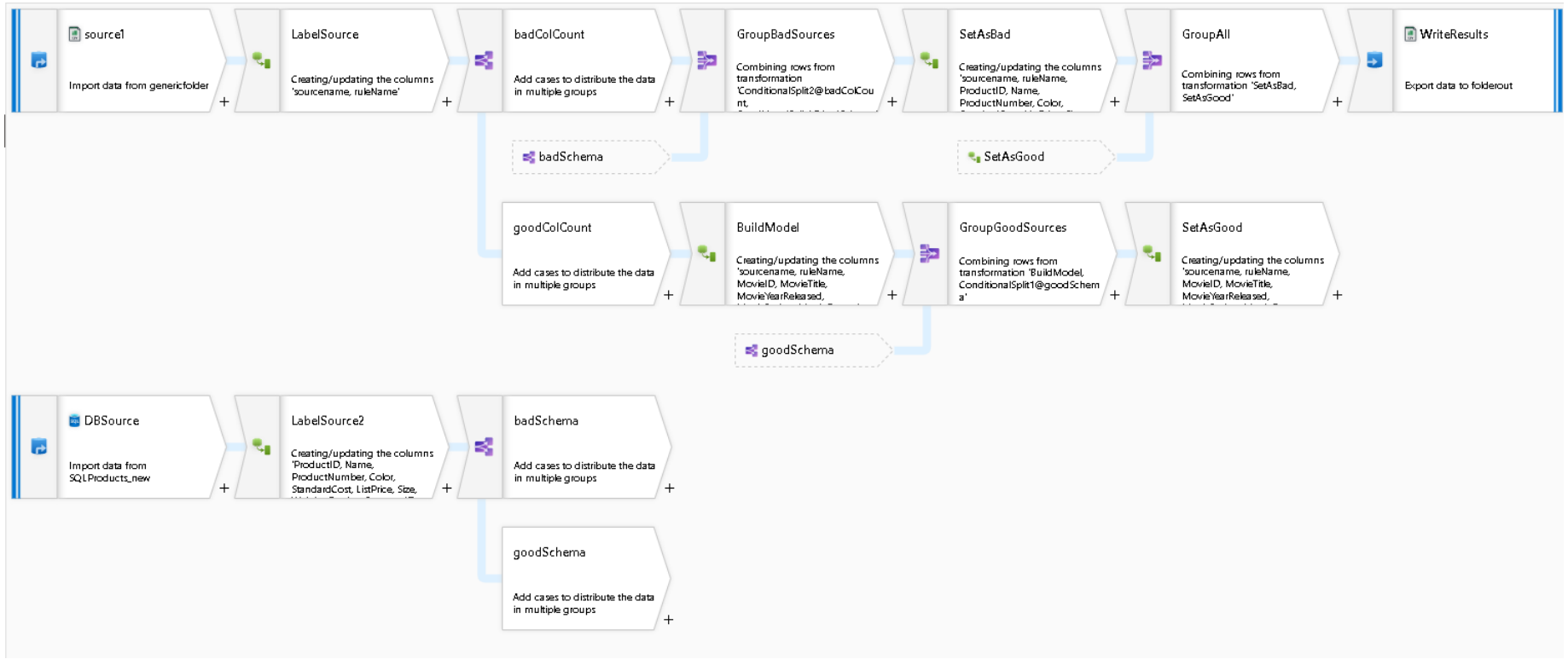
Slowly changing dimensions



Fact loading into a data warehouse

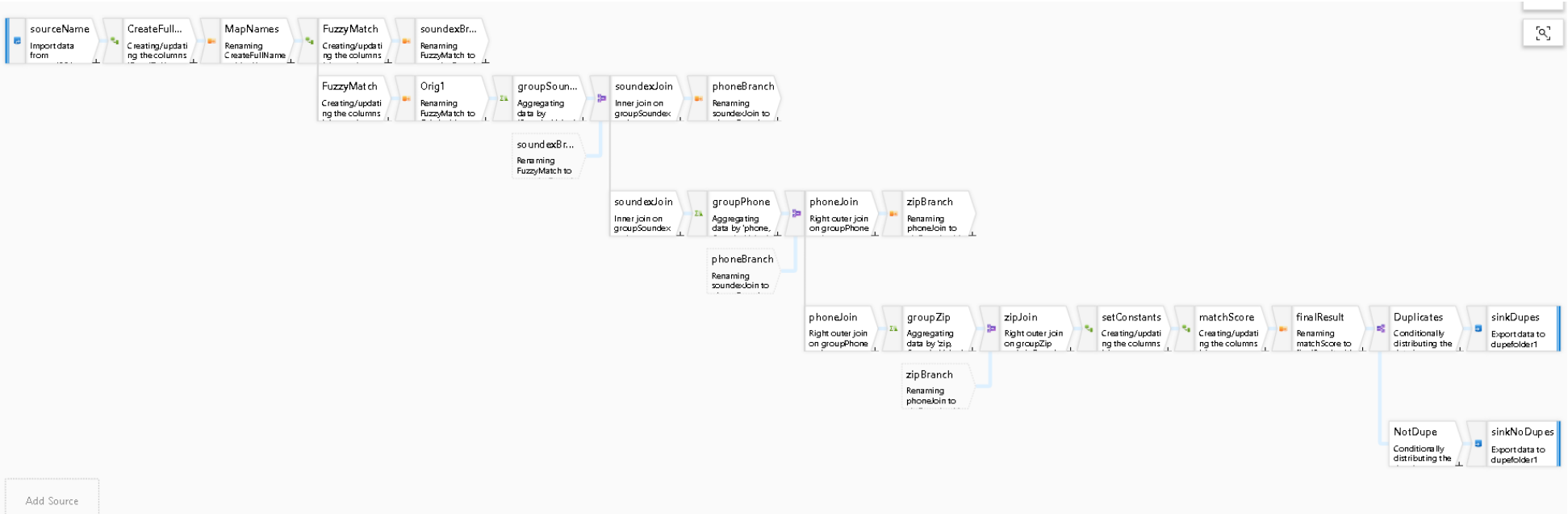


Metadata Validation Rules

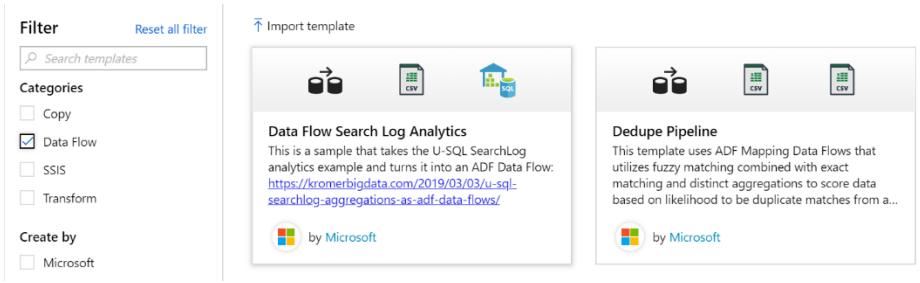


- Transform data conditionally based on metadata traits
- Create and manage metadata quality rules
- Manipulate column properties of source data
- https://www.youtube.com/watch?v=E_UD3R-VpYE

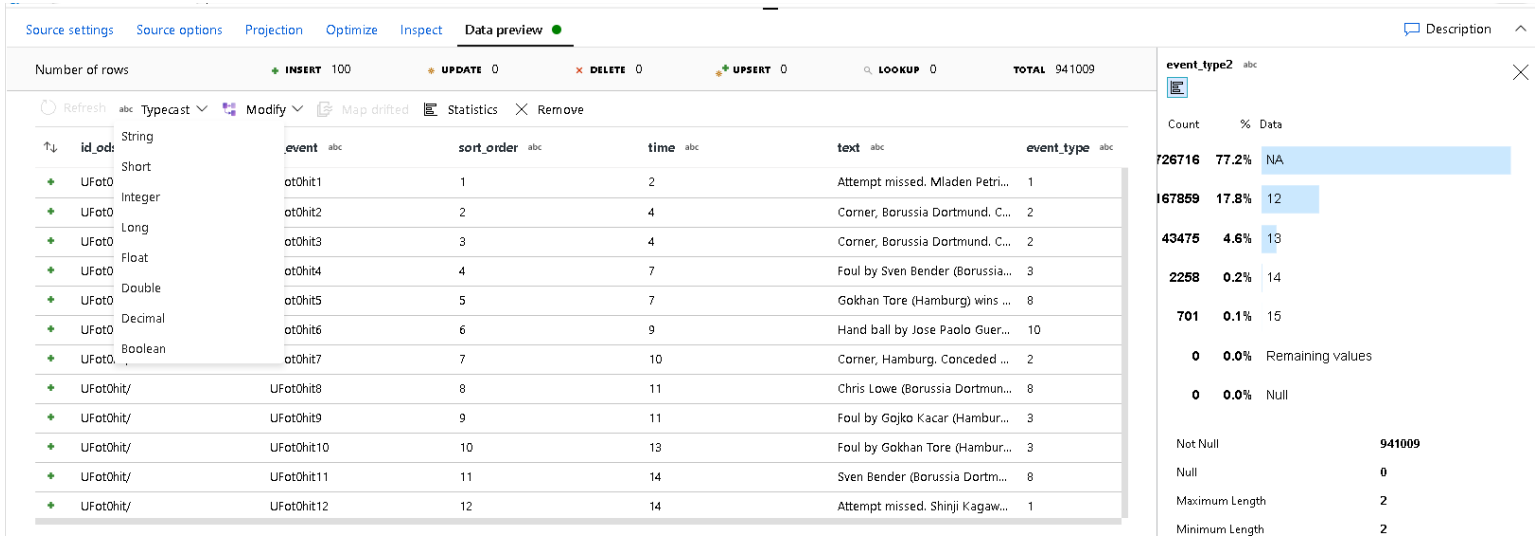
Data De-Duplication and Distinct Rows

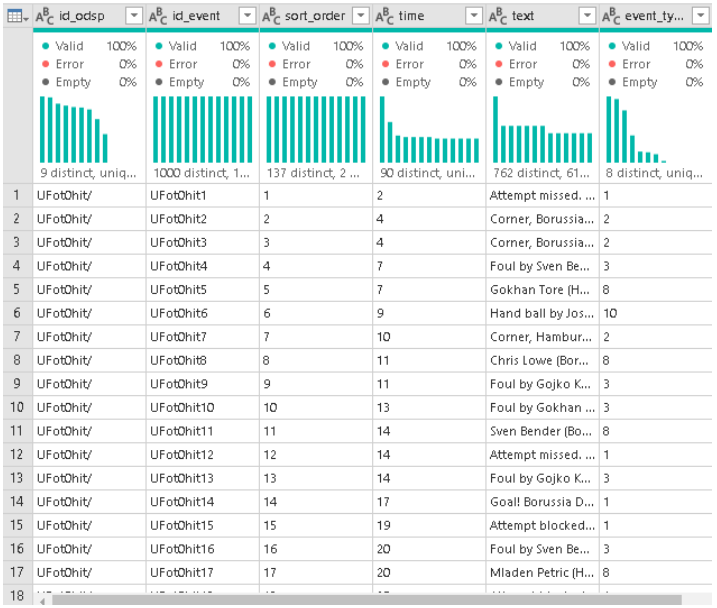


- Use this pattern to eliminate common rows from your data
- You pick a heuristic to use during duplicate matching
- You can tag rows and/or remove duplicate rows
- Use exact matching and/or fuzzy matching
- Available as pipeline template *Dedupe Pipeline*



Data Profiling

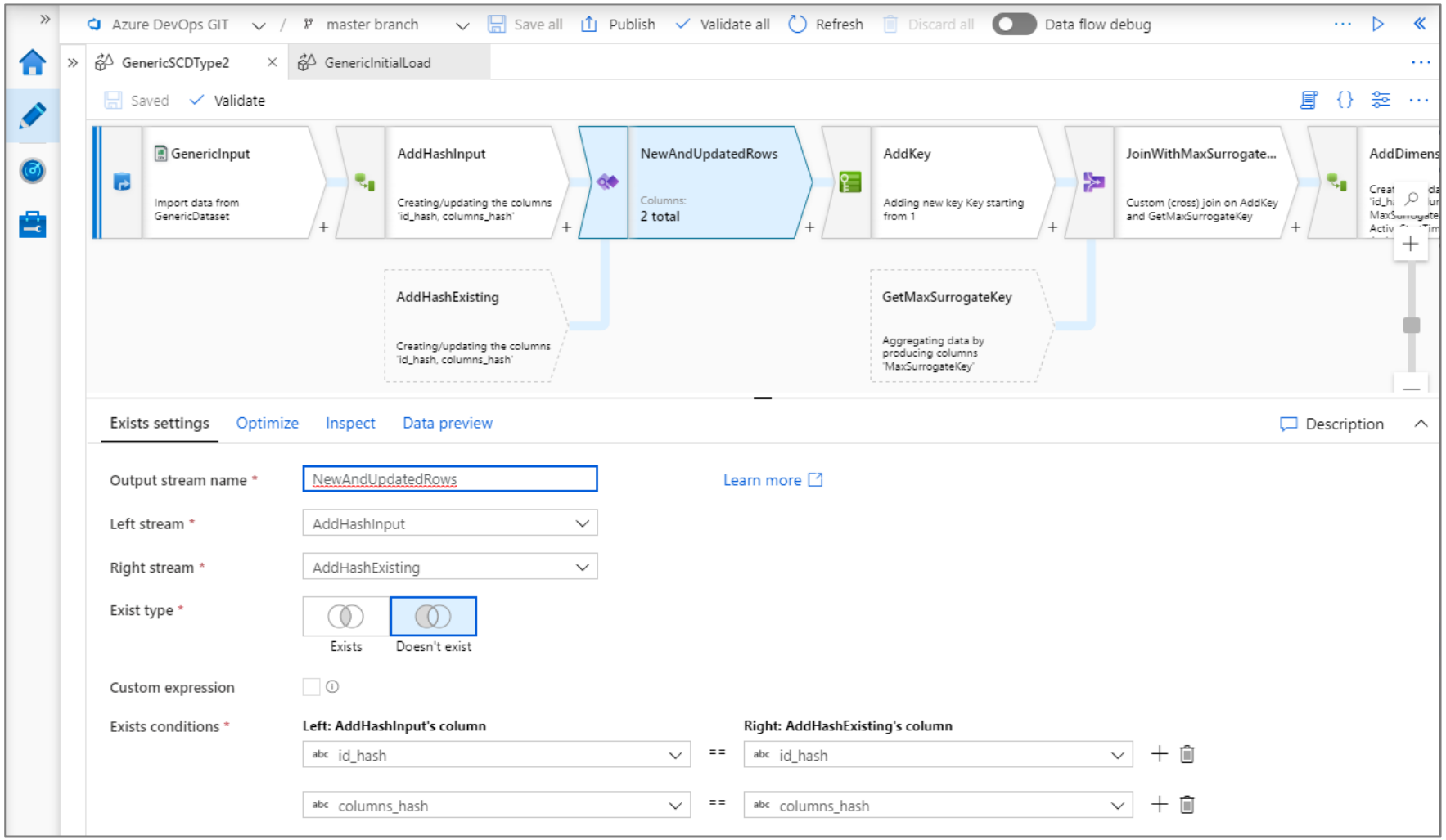




- Summary statistics describing the shape, size, content of your data
- Helps you to understand what is inside your data
- As a Data Engineer, you have the responsibility to provide proper data for analytics and models
- For big data sets, use sampling / good enough approach
- View data statistics at any step in your data transformation
- Here is how to persist your data profile stats: https://techcommunity.microsoft.com/t5/azure-data-factory/how-to-save-your-data-profiler-summary-stats-in-adf-data-flows/ba-p/1243251

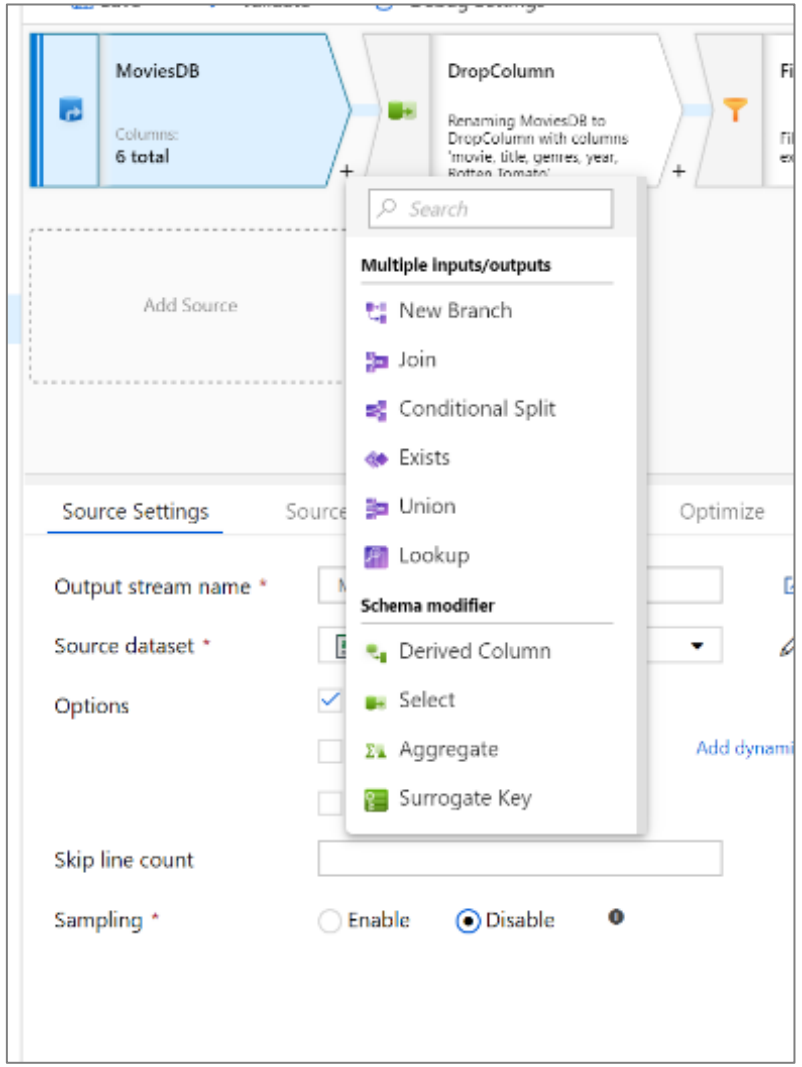
Authoring mapping data flows

Dedicated development canvas



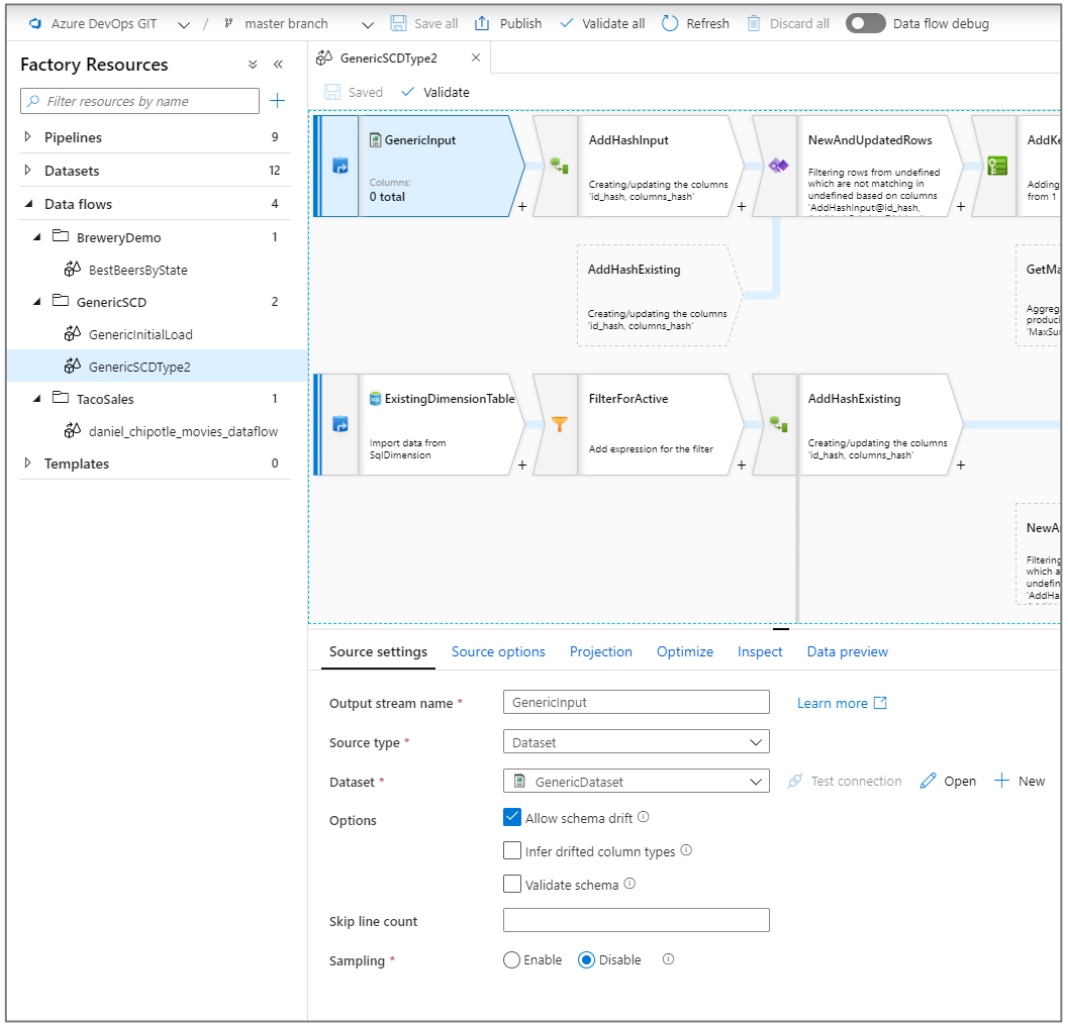
Building transformation logic

- Transformations: A ‘step’ in the data flow
  - Engine intelligently groups them at runtime
  - 19 currently available
- Core logic of data flow
  - Add/Remove/Alter Columns
  - Join or lookup data from datasets
  - Change number or order of rows
  - Aggregate data
  - Hierarchal to relational



Source transformation

- Define the data read by your data flow
  - Import projection vs generic
  - Schema drift
  - Connector specific properties and optimizations
- Min: 1, Max: ∞
- Define in-line or use dataset



Source: In-line vs dataset

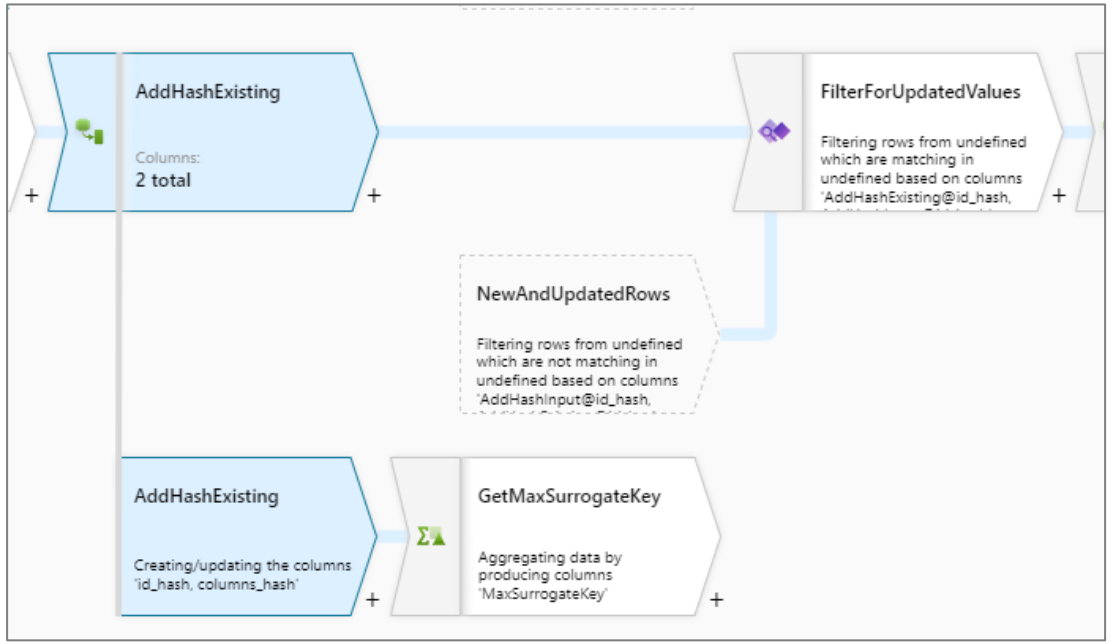
- Define all source properties within a data flow or use a separate entity to store them
- Dataset:
  - Reusable in other ADF activities such as Copy
  - Not based in Spark -> some settings overridden
- In-line
  - Useful when using flexible schemas, one-off source instances or parameterized sources
    - Do not need “dummy” dataset object
  - Based in Spark, properties native to data flow
- Most connectors only in available in one

Supported connectors

- File-based data stores (ADLS Gen1/Gen2, Azure Blob Storage)
  - Parquet, JSON, DelimitedText, Excel, Avro, XML
  - In-line only: Common Data Model, Delta Lake
- SQL tables
  - Azure SQL Database
  - Azure Synapse Analytics (formerly SQL DW)
- Cosmos DB
- Coming soon: Snowflake
- **If not supported, ingest to staging area via Copy activity**
  - 90+ connectors supported natively

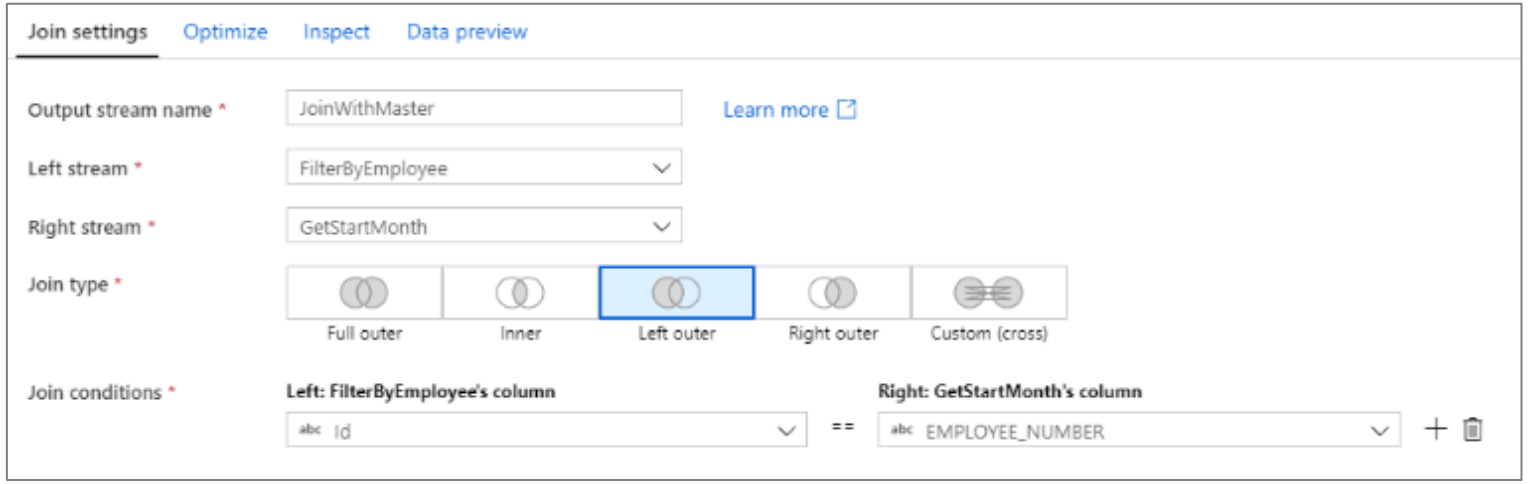
Duplicating data streams

- Duplicate data stream from any stage of your data flow
  - Select ‘New branch’
- Operate on same data with different transformation requirements
  - Self-joins
  - Writing to different sinks
  - Aggregating in one branch



Joining two data streams together

- Use *Join transformation* to append columns from incoming stream to any stream in your data flow
  - Join types: full outer, inner, left outer, right outer, cross
  - SQL Join equivalent
- Match on computed columns or use non-equality conditions
- Broadcast small data streams to cache data and improve performance



Lookup transformation

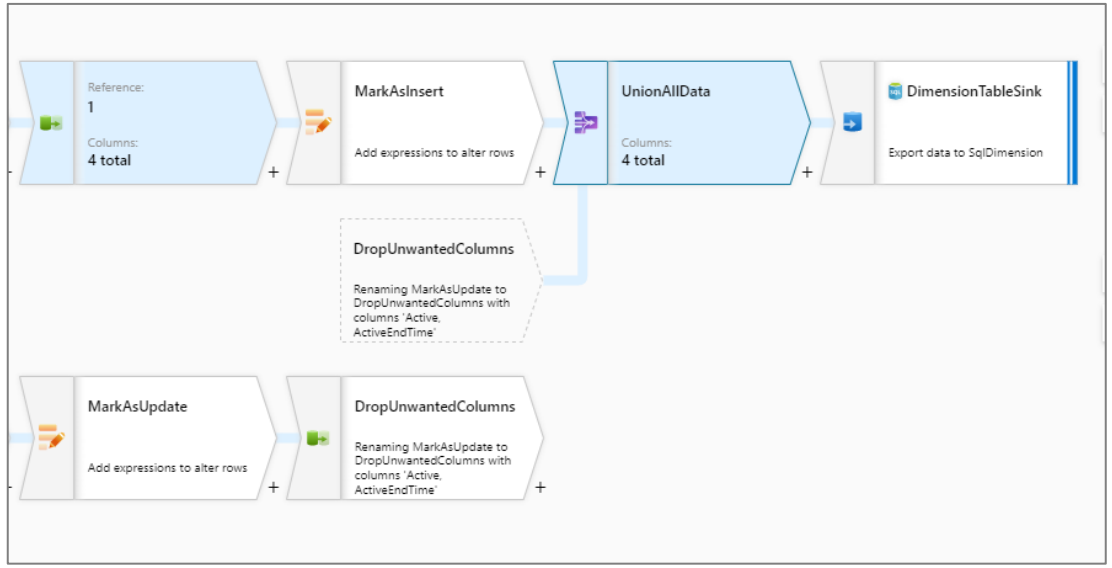
- Similar to left outer join, but with more functionality
  - All incoming rows are passed through regardless of match
- Matching conditions same as a join
- Multi or single row lookup
  - Match on all, first, last, or any row that meets join conditions
- *isMatch()* function can be used in downstream transformations to verify output

Exists transformation

- Check for existence of a value in another stream
  - SQL Exists equivalent
  - See if any row matches in a subquery, just like SQL
- Filter based on join matching conditions
- Choose **Exist** or **Not Exist** for your filter conditions
- Can specify a custom expressoin

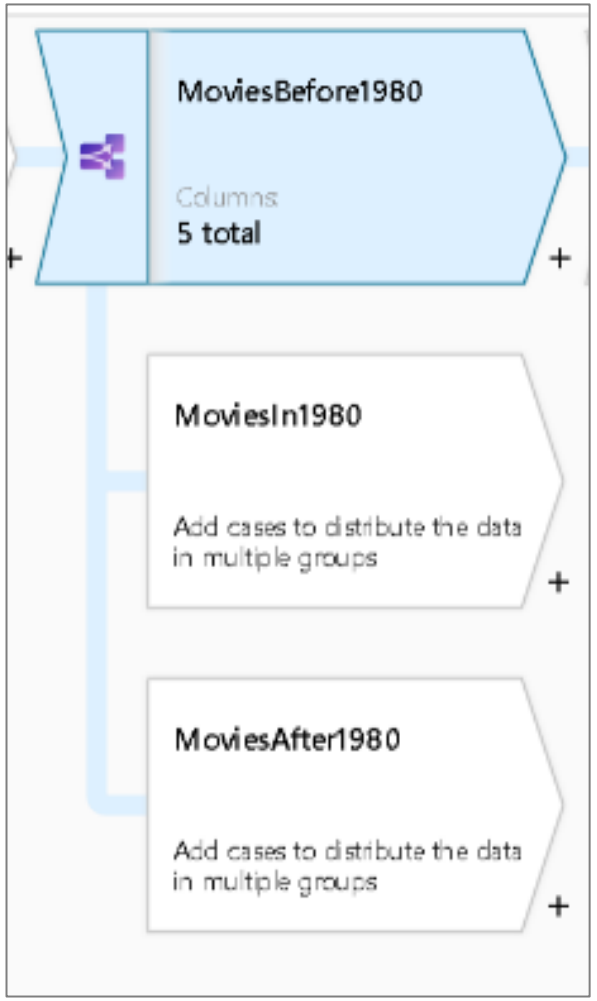
Union transformation

- Combine rows from multiple streams
- Add as many streams as needed
- Combine data based upon column name or ordinal column position
- Use cases:
  - Similar data from different connection that undergo same transformations
  - Writing multiple data streams into the same sink



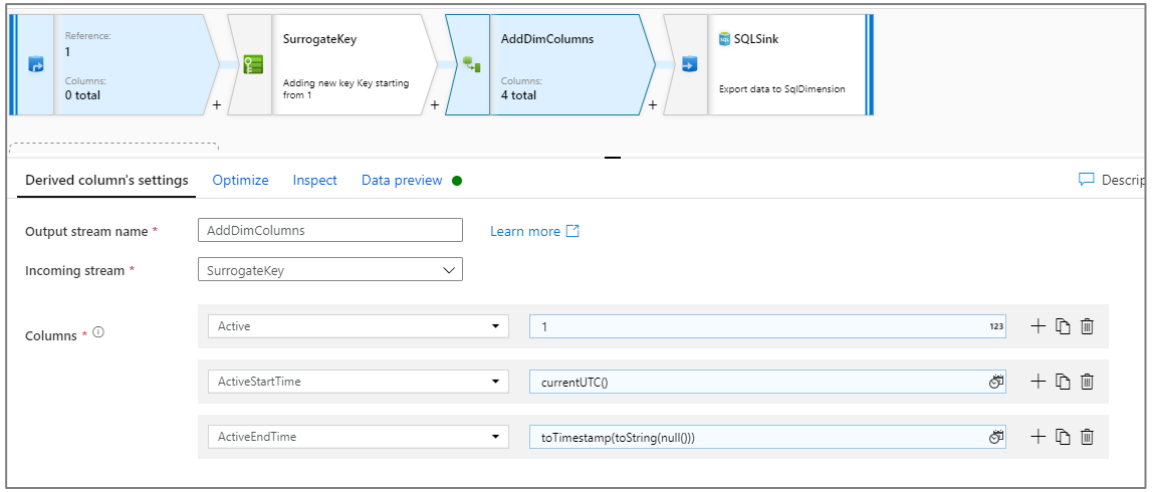
Conditional split

- Split data into separate streams based upon conditions
  - Use data flow expression language to evaluate boolean
- Use cases:
  - Sinking subset of data to different locations
  - Perform different calculations on data depending on a set of values



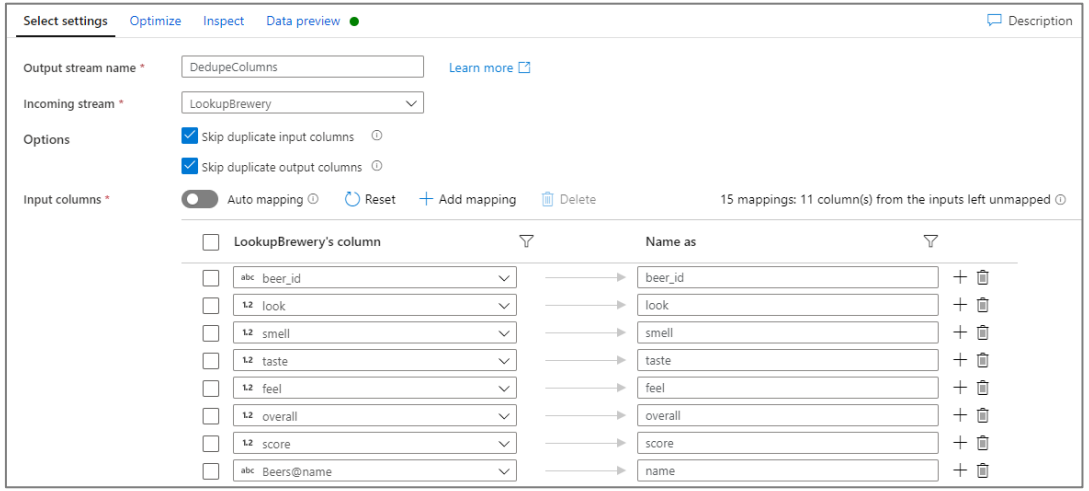
Derived column

- Transform data at row and column level using expression language
  - Generate new or modify existing columns
  - Build expressions using the expression builder
- Handle structured or unstructured data
  - Use column patterns to match on rules and regular expressions
  - Can be used to transform multiple columns in bulk
- Most heavily used transformation



Select transformation

- Metadata and column maintenance
  - SQL Select statement
- Alias or renames data stream and columns
- Prune unwanted or duplicate columns
  - Common after joins and lookups
- Rule-based mapping for flexible schemas, bulk mapping
  - Map hierarchal columns to flat structure

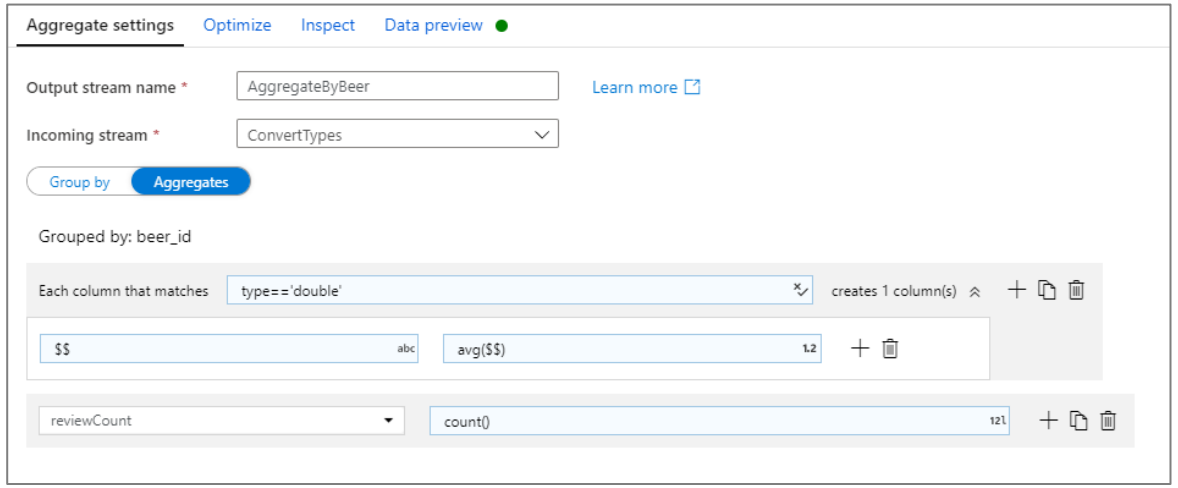


Surrogate key transformation

- Generate incrementing key to use as a non-business key in your data
- To seed the starting value of your surrogate key, use derived column and a lookup from an existing table
  - Examples are in documentation
- Useful for generating keys for star schema dimension tables

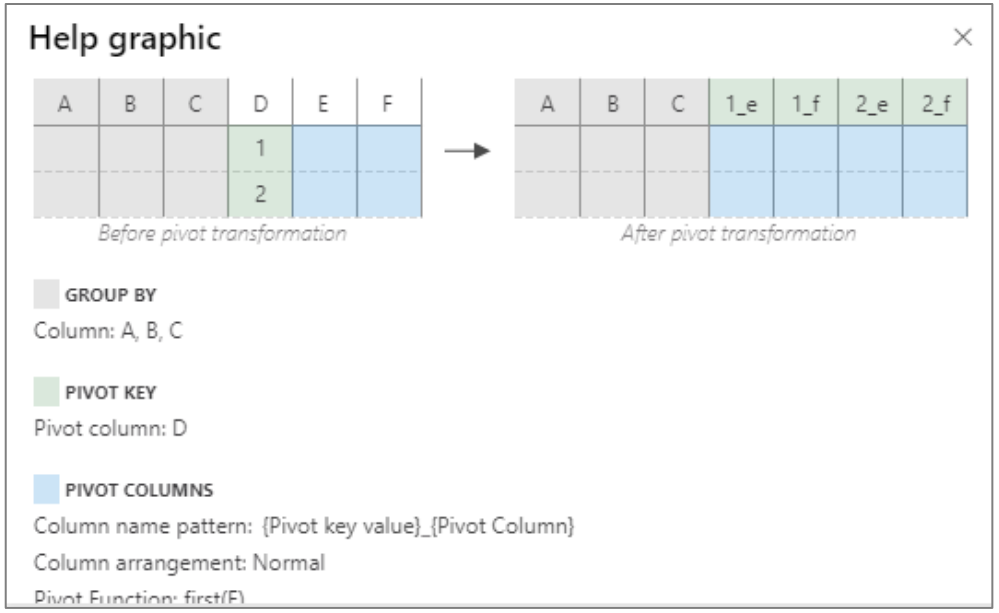
Aggregate transformation

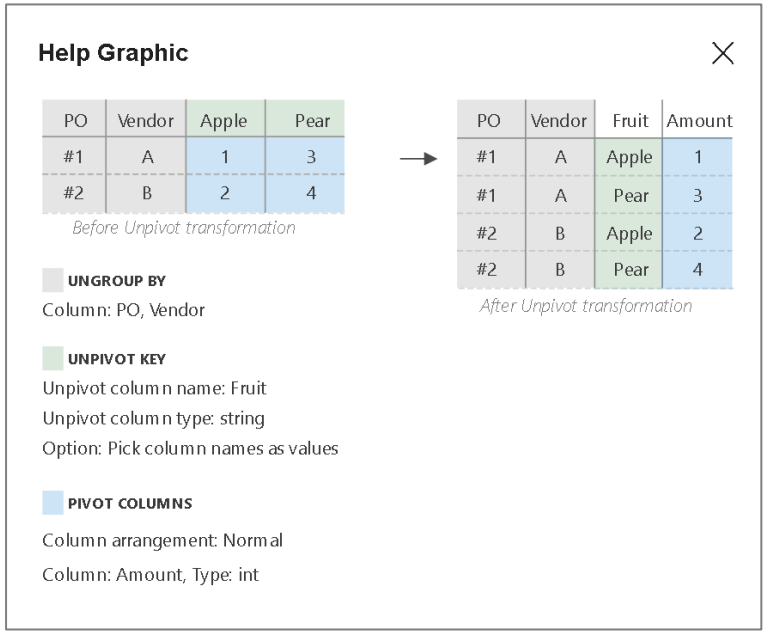
- Aggregate data into groups using aggregate function
  - Like SQL GROUP BY clause in a Select statement
  - Aggregate functions include *sum()*, *max()*, *avg()*, *first()*, *collect()*
- Choose columns to group by
  - One row for each unique group by column value
- Only columns used in transformation are in output data stream
  - Use self-join to append to existing data
- Supports pattern matching



Pivot and unpivot transformations

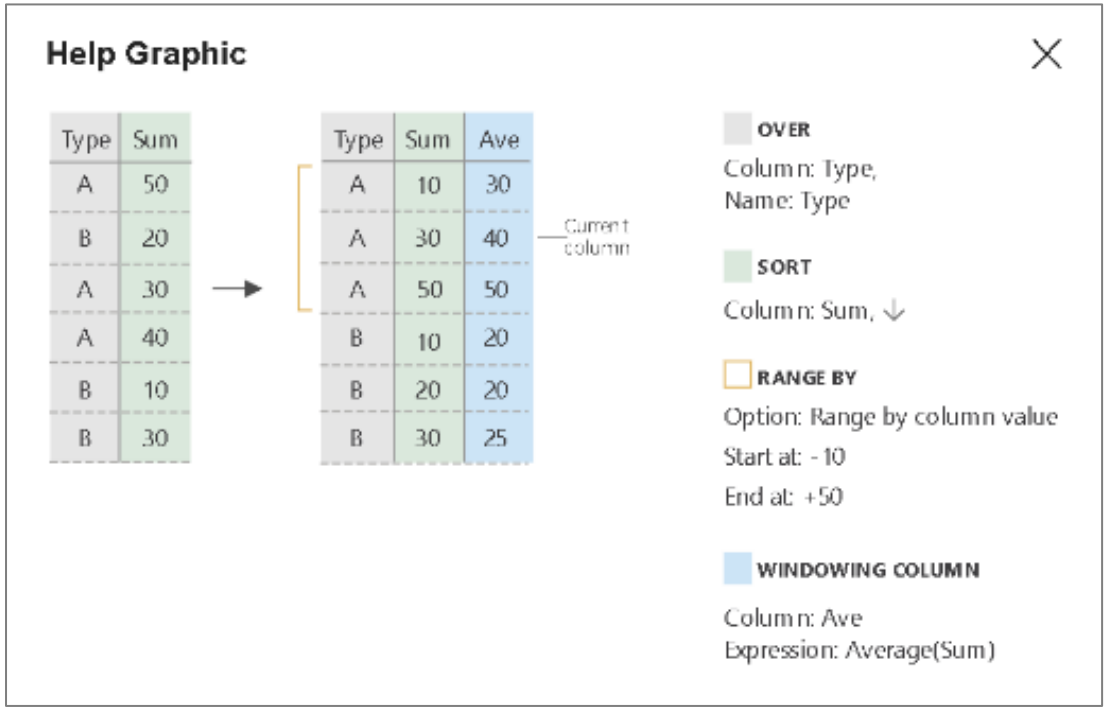
- Pivot row values into new columns and vice-versa
- Both are aggregate transformations that require aggregate functions
- If pivot key values not specified, all columns become drifted
  - Use map drifted quick action to add to schema quickly





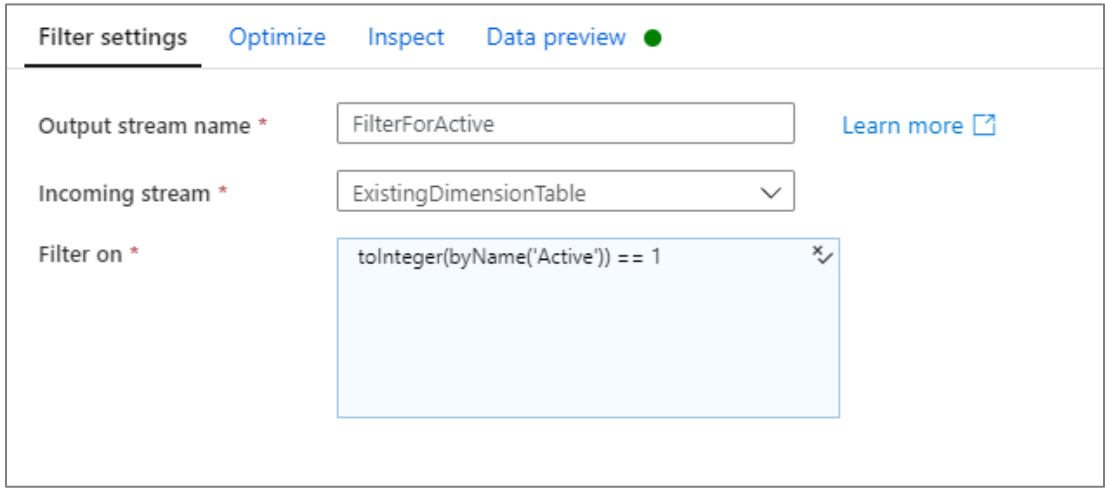
Window transformation

- Aggregate data across “windows” of data partitions
  - Used to compare a row of data against others in its ‘group’
- Group determined by group by columns, sorting conditions and range bounds
- Used for ranking rows in a group and getting lead/lag
- Sorting causes reshuffling of data
  - “Expensive” operation



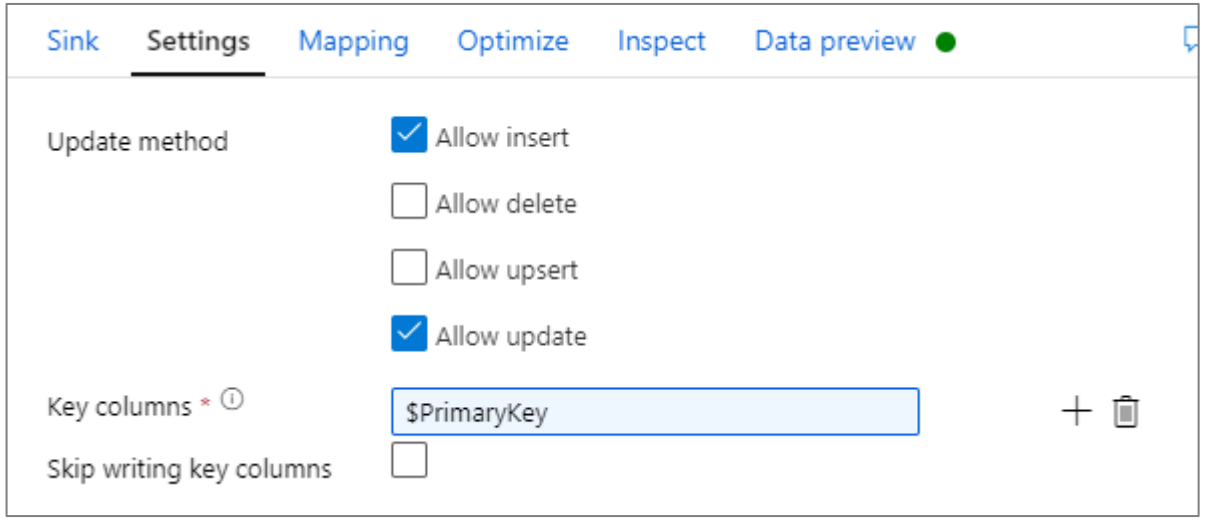
Filter transformation

- Filter rows based upon an expression
  - Like SQL WHERE clause
- Expressions return true or false



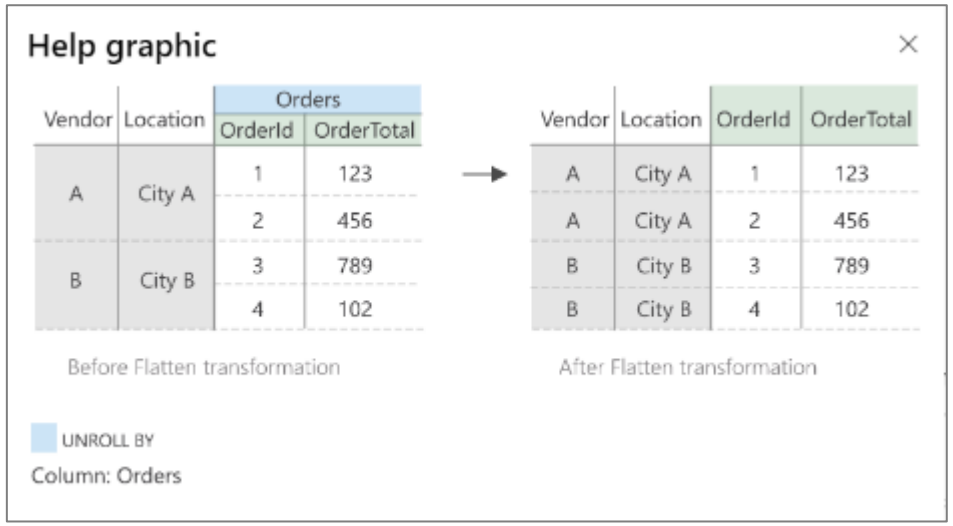
Alter row transformation

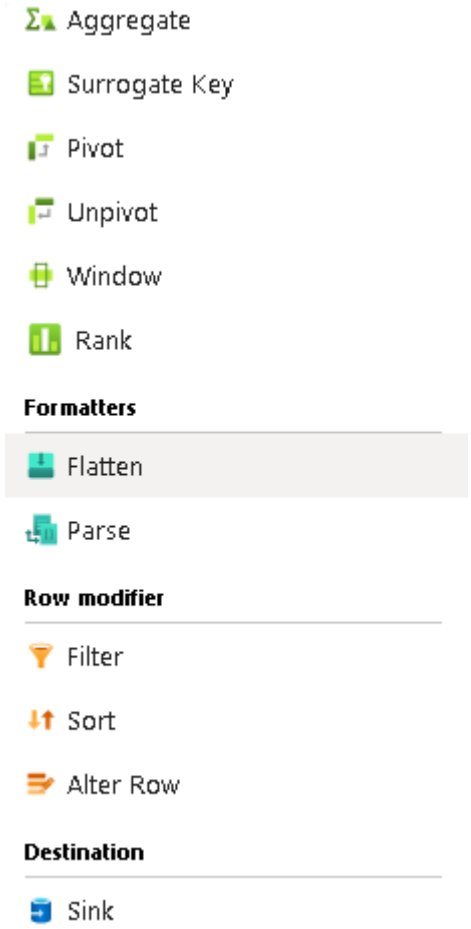
- Mark rows as Insert, Update, Delete, or Upsert
  - Like SQL MERGE statement
  - Insert by default
- Define policies to update your database
  - Works with SQL DB, Synapse, Cosmos DB, and Delta Lake
- Specify allowed update methods in each sink



Flatten transformation

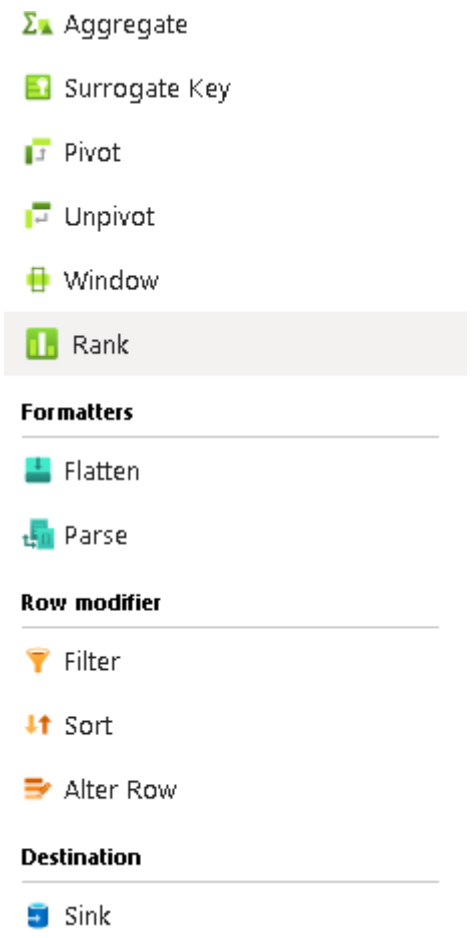
- Unroll array values into individual rows
  - One row per value
- Used to convert hierarchies to flat structures
- Opposite of collect() aggregate function





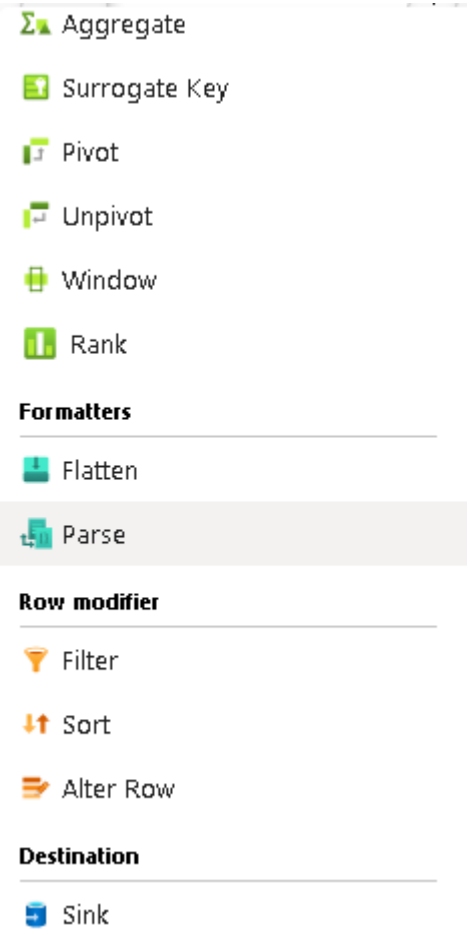
Rank transformation

- Rank data across an entire dataset
- Same as Rank() function in Window transformation,but scales better for a ranking an entire dataset
- For ranking of data partitions, use Window rank()
- For ranking entire dataset, use Rank transformation



Parse transformation

- Parses string data from columns that are formatted text
- Currently supported formats: JSON, text delimited
- Ex: Turn plain text JSON strings from a source file in a single column into formatted JSON output

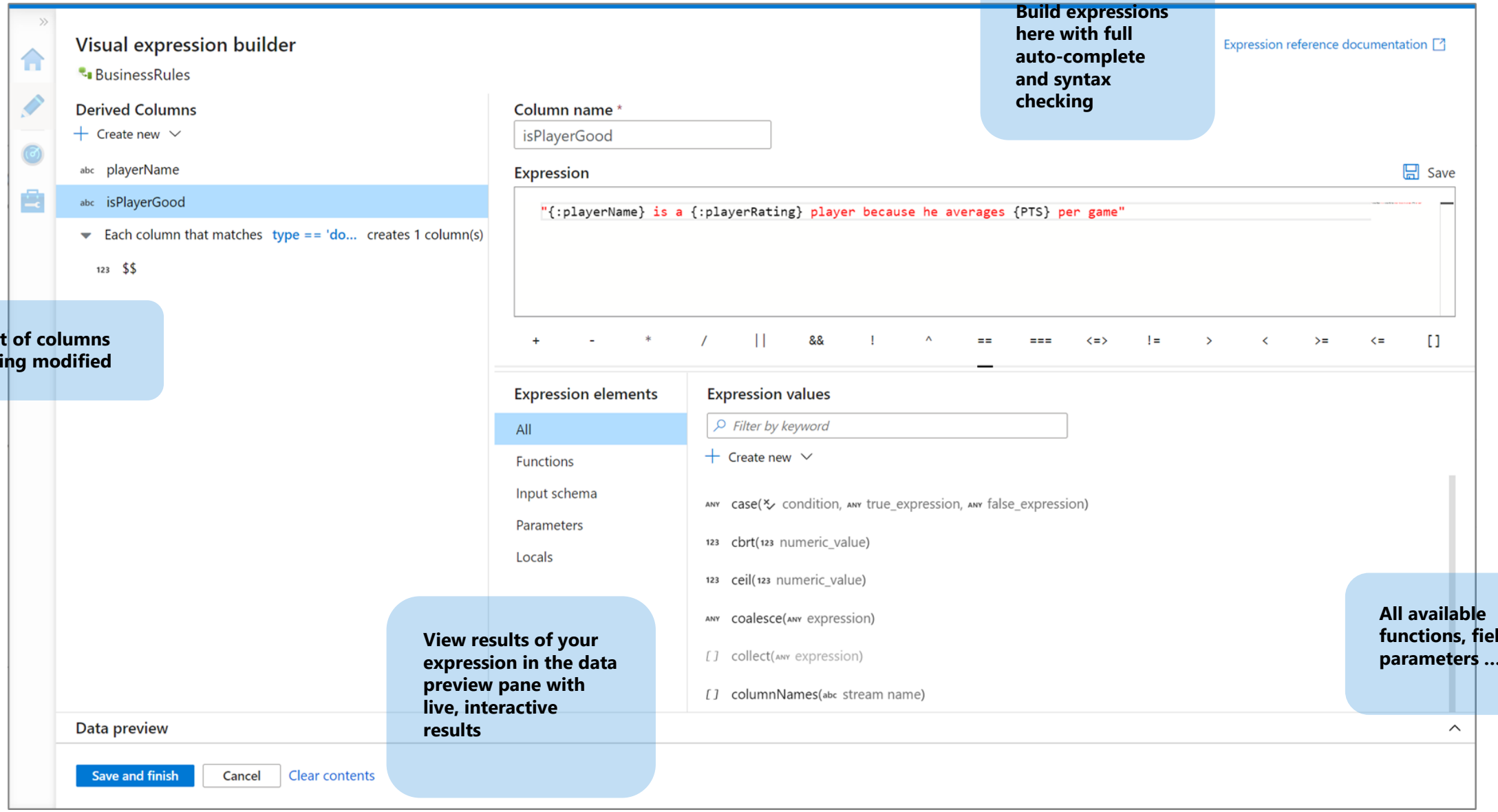


Sink transformation

- Define the properties for landing your data in your destination target data store
  - Define using dataset or in-line
- Can map columns similar to select transformation
  - Import schema definition from destination
- Set actions on destinations
  - Truncate table or clear folder, SQL pre/post actions, database update methods
- Choose how the written data is partitioned
  - Use current partitioning is almost always fastest
- **Note: Writing to single file can be very slow with large amounts of data**

Mapping data flow expression language

Visual expression builder



Build expressions here with full auto-complete and syntax checking

List of columns being modified

View results of your expression in the data preview pane with live, interactive results

All available functions, fields, parameters ...

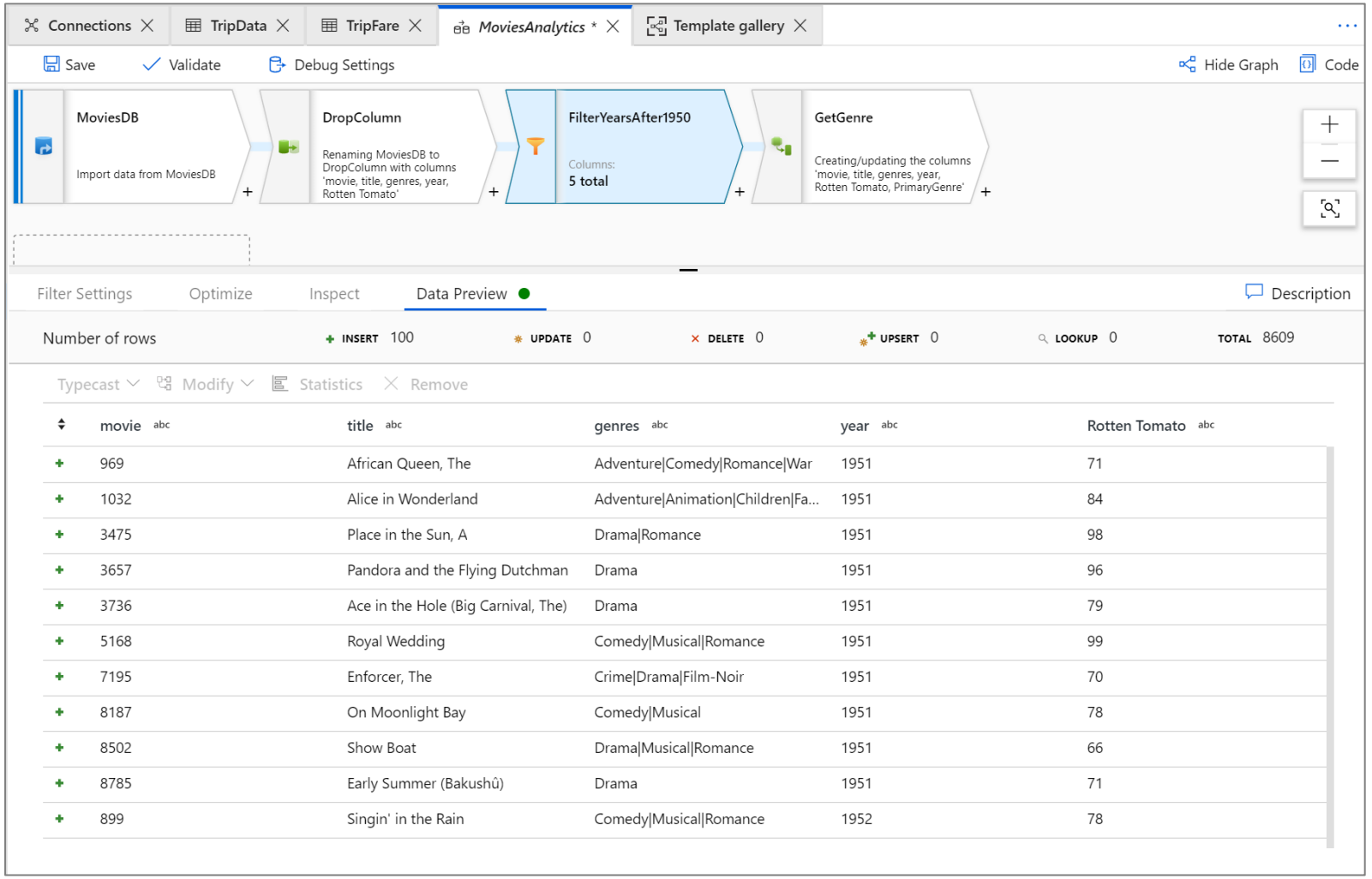
Expression language

- Expressions are built using the data flow expression language
- Expressions can reference:
  - Built-in expression functions
  - Defined input schema columns
  - Data flow parameters
  - Literals
- Certain transformations have unique functions
  - Count(), sum() in Aggregate, denseRank() in Window, etc
- Evaluates to spark data types
- 100s of transformation, analytical, array, metadata, string, and mathematical functions available

Debug mode

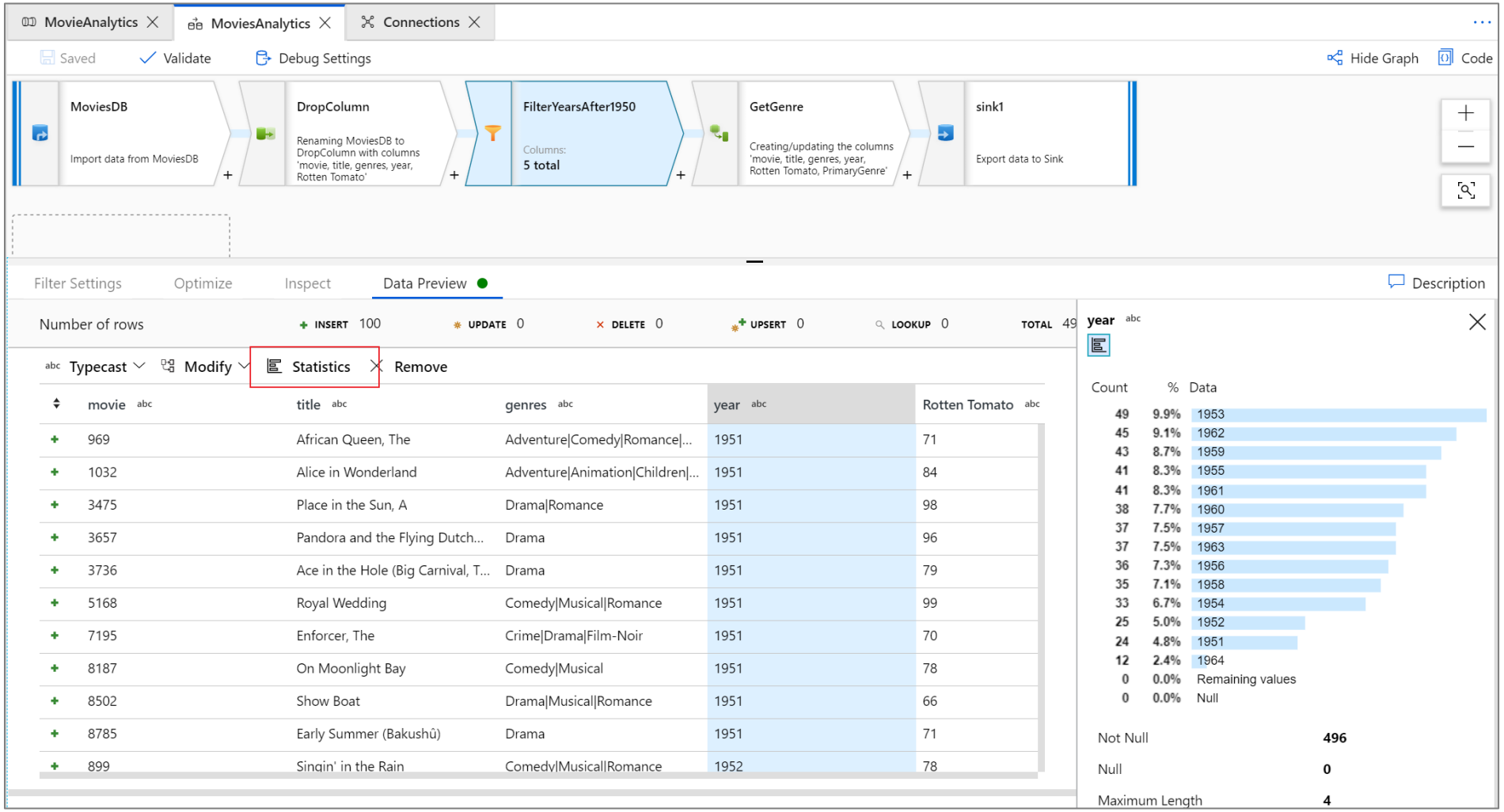
- Quickly verify logic during development on small interactive cluster
  - 4 core, 60-minute time to live
- Enables the following:
  - Get data preview snapshot at each transformation
  - Preview output of expression in expression builder
  - Run debug pipeline with no spin up
  - Import Spark projection of source schema
- **Rule of thumb**: If developing Data Flows, turn on right away
  - Initial 3-5-minute start up time

Debug mode: data preview

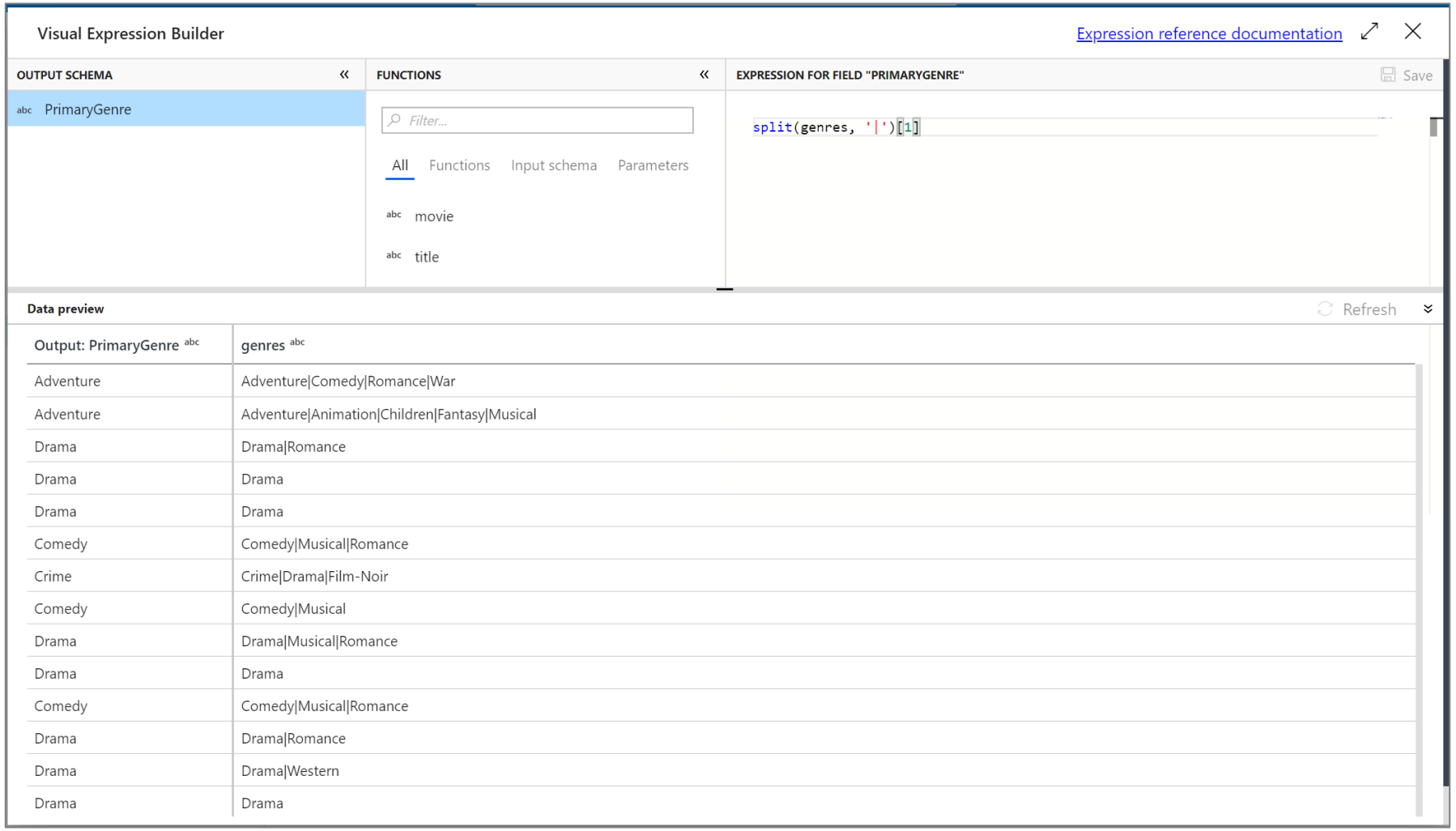


movie	title	genres	year	Rotten Tomato
969	African Queen, The	Adventure|Comedy|Romance|War	1951	71
1032	Alice in Wonderland	Adventure|Animation|Children|Fa...	1951	84
3475	Place in the Sun, A	Drama|Romance	1951	98
3657	Pandora and the Flying Dutchman	Drama	1951	96
3736	Ace in the Hole (Big Carnival, The)	Drama	1951	79
5168	Royal Wedding	Comedy|Musical|Romance	1951	99
7195	Enforcer, The	Crime|Drama|Film-Noir	1951	70
8187	On Moonlight Bay	Comedy|Musical	1951	78
8502	Show Boat	Drama|Musical|Romance	1951	66
8785	Early Summer (Bakushû)	Drama	1951	71
899	Singin' in the Rain	Comedy|Musical|Romance	1952	78

Debug mode: data profiling



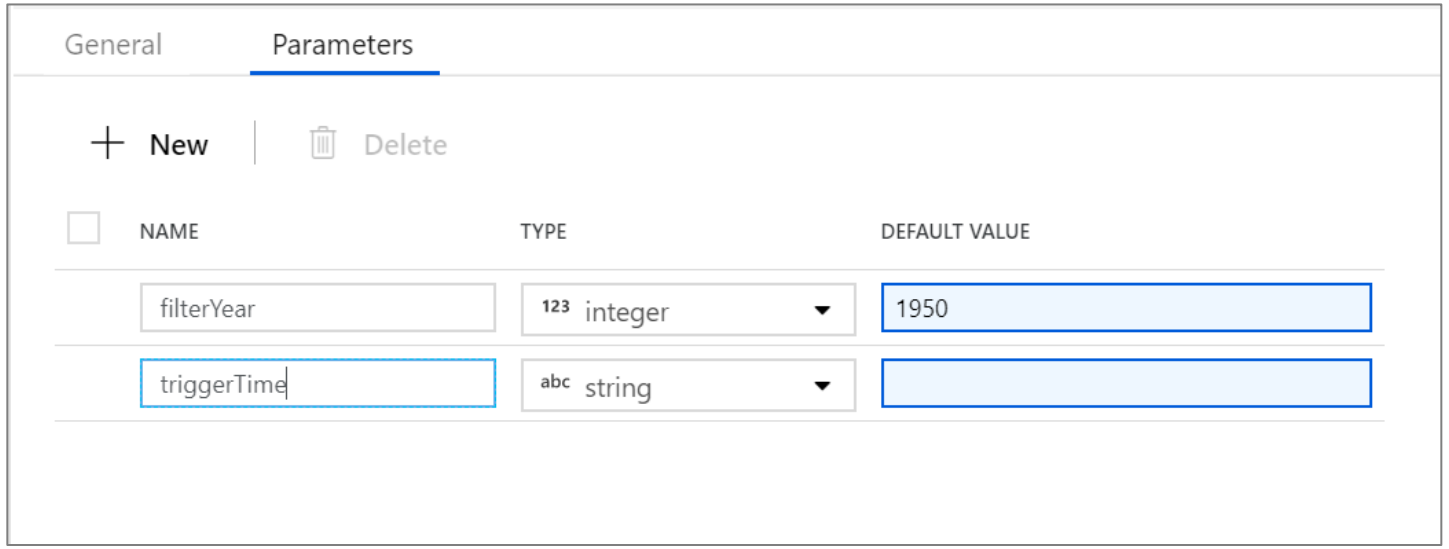
Debug mode: expression output



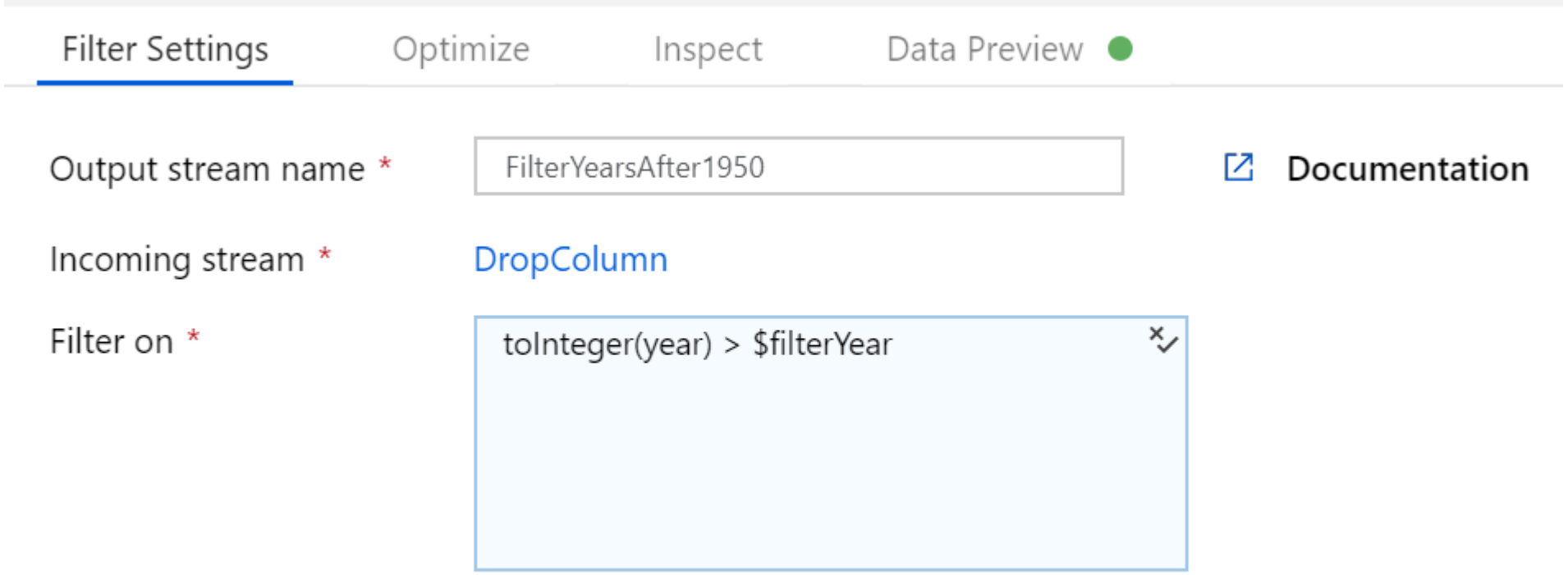
Output: PrimaryGenre abc	genres abc
Adventure	Adventure|Comedy|Romance|War
Adventure	Adventure|Animation|Children|Fantasy|Musical
Drama	Drama|Romance
Drama	Drama
Drama	Drama
Comedy	Comedy|Musical|Romance
Crime	Crime|Drama|Film-Noir
Comedy	Comedy|Musical
Drama	Drama|Musical|Romance
Drama	Drama
Comedy	Comedy|Musical|Romance
Drama	Drama|Romance
Drama	Drama|Western
Drama	Drama

Parameterizing data flows

- Both dataset properties and data-flow expressions can be parameterized
  - Passed in via data flow activity
    - Can use data flow or pipeline expression language
  - Expressions can reference *$parameterName*
- Can be literal values or column references



Referencing data flow parameters



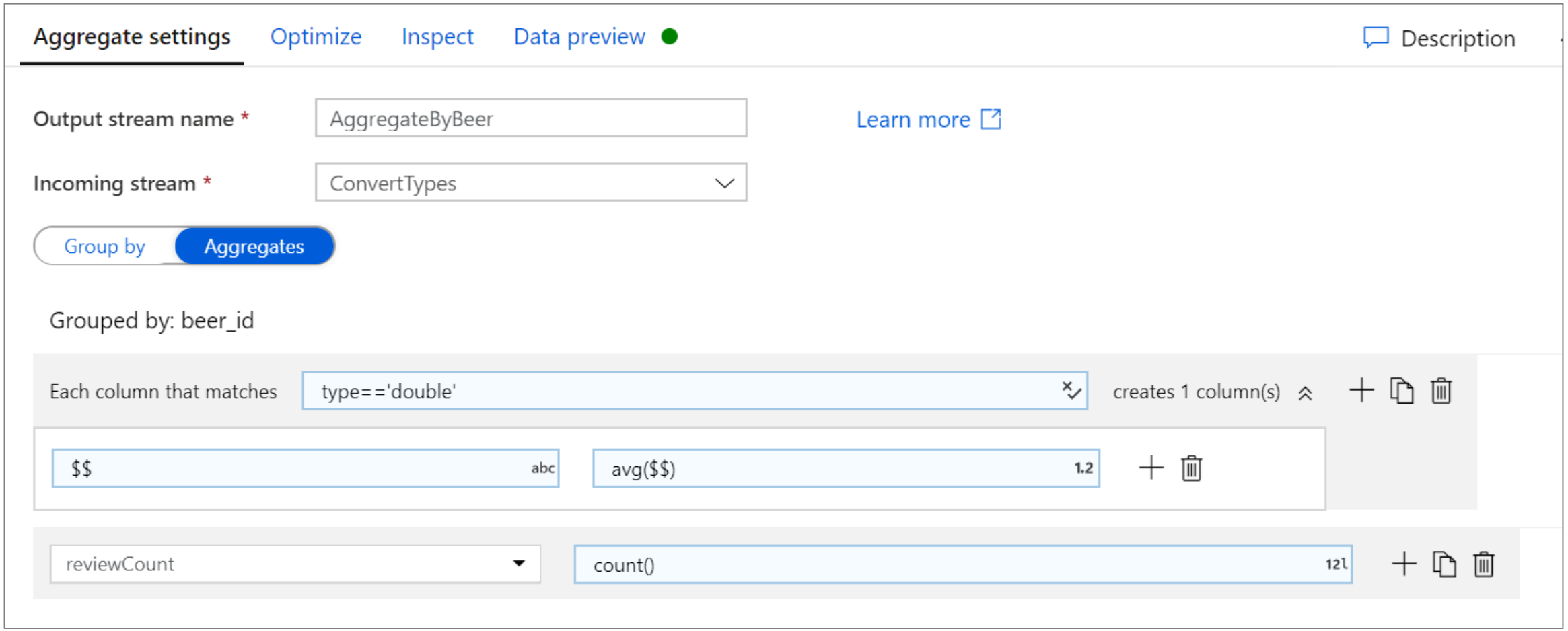
Working with flexible schemas

Schema drift

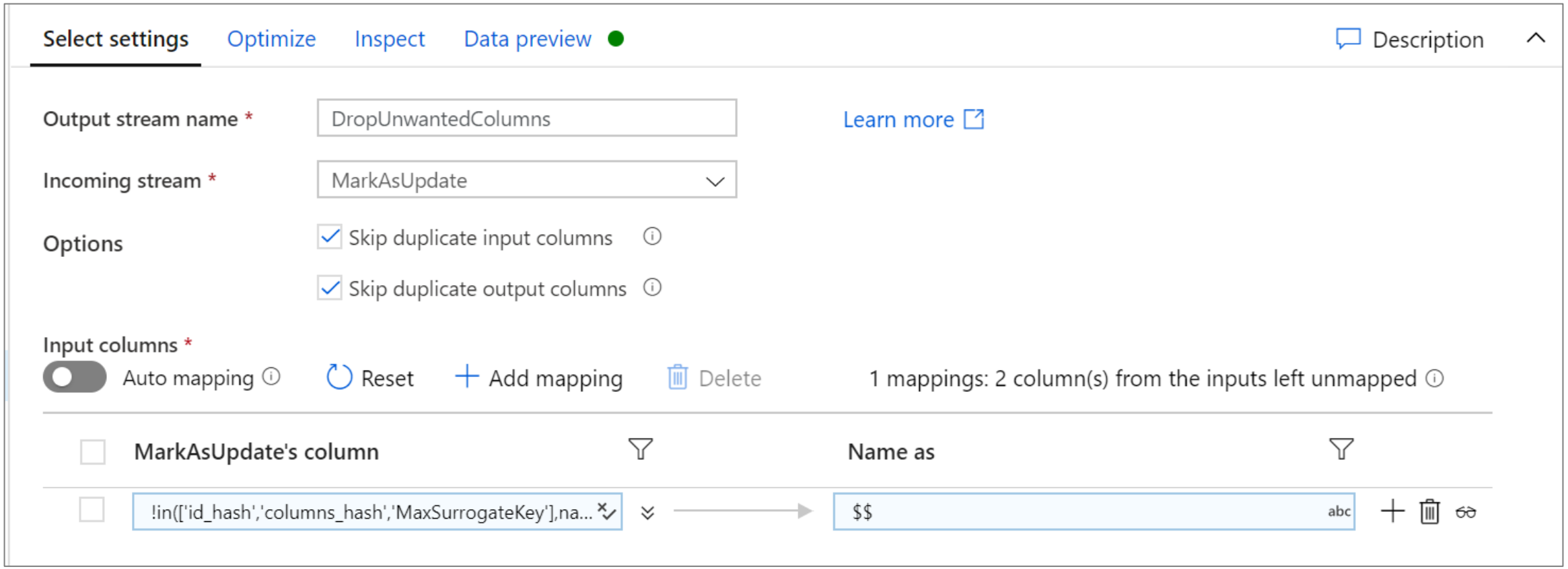
- In real-world data integration solutions, source/target data stores change shape
  - Source data fields can change names
  - Number of columns can change over time
- Traditional ETL processes break when schemas drift
- Mapping data flow has built-in handling for flexible schemas
  - Patterns, rule-based mappings, byName(s) function, etc
  - Source: Read additional columns on top of what is defined in the source schema
  - Sink: Write additional columns on top of what is defined in the sink schema

Column pattern matching

- Match by name, type, stream, position



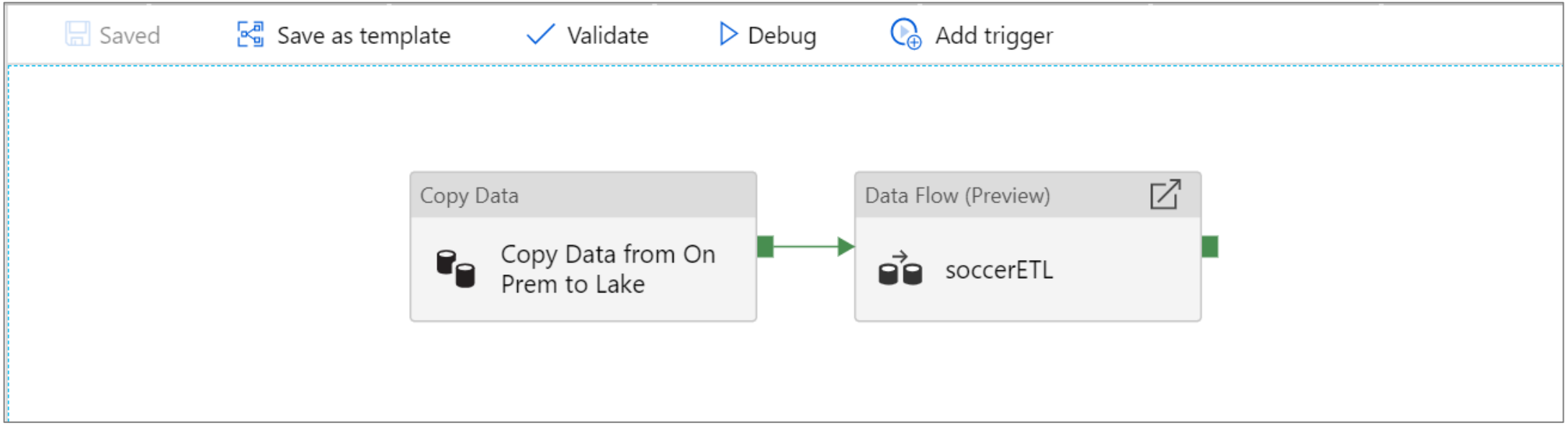
Rule-based mapping



Operationalizing and monitoring data flows

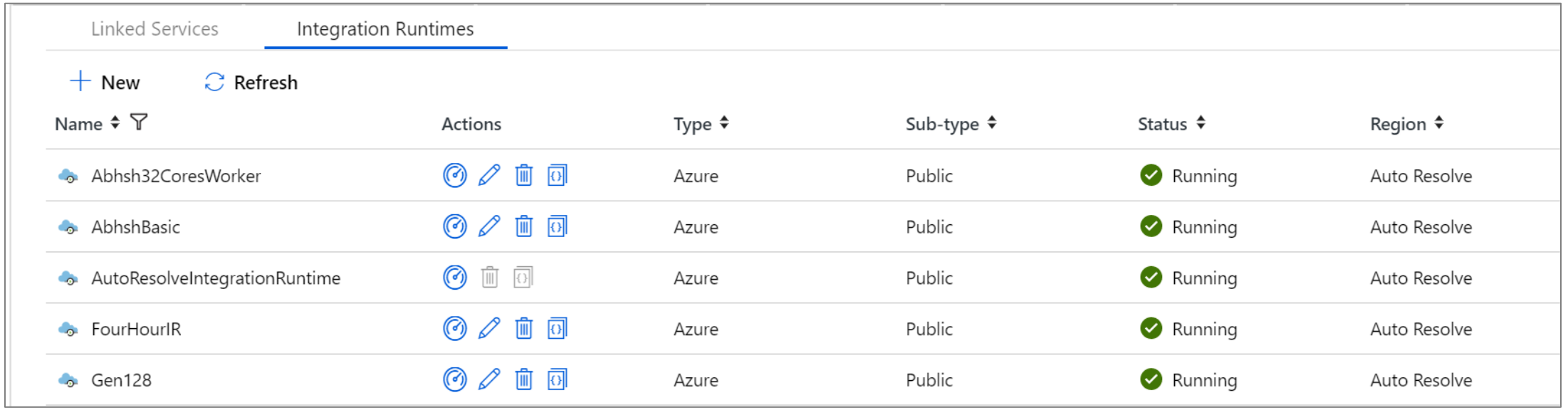
Data flow activity

- Run as activity in pipeline
  - Integrated with existing ADF control flow, scheduling, orchestration, montoring, C/ICD
- Choose which integration runtime (IR) to run on
  - # of cores, compute type, cluster time to live
- Assign parameters

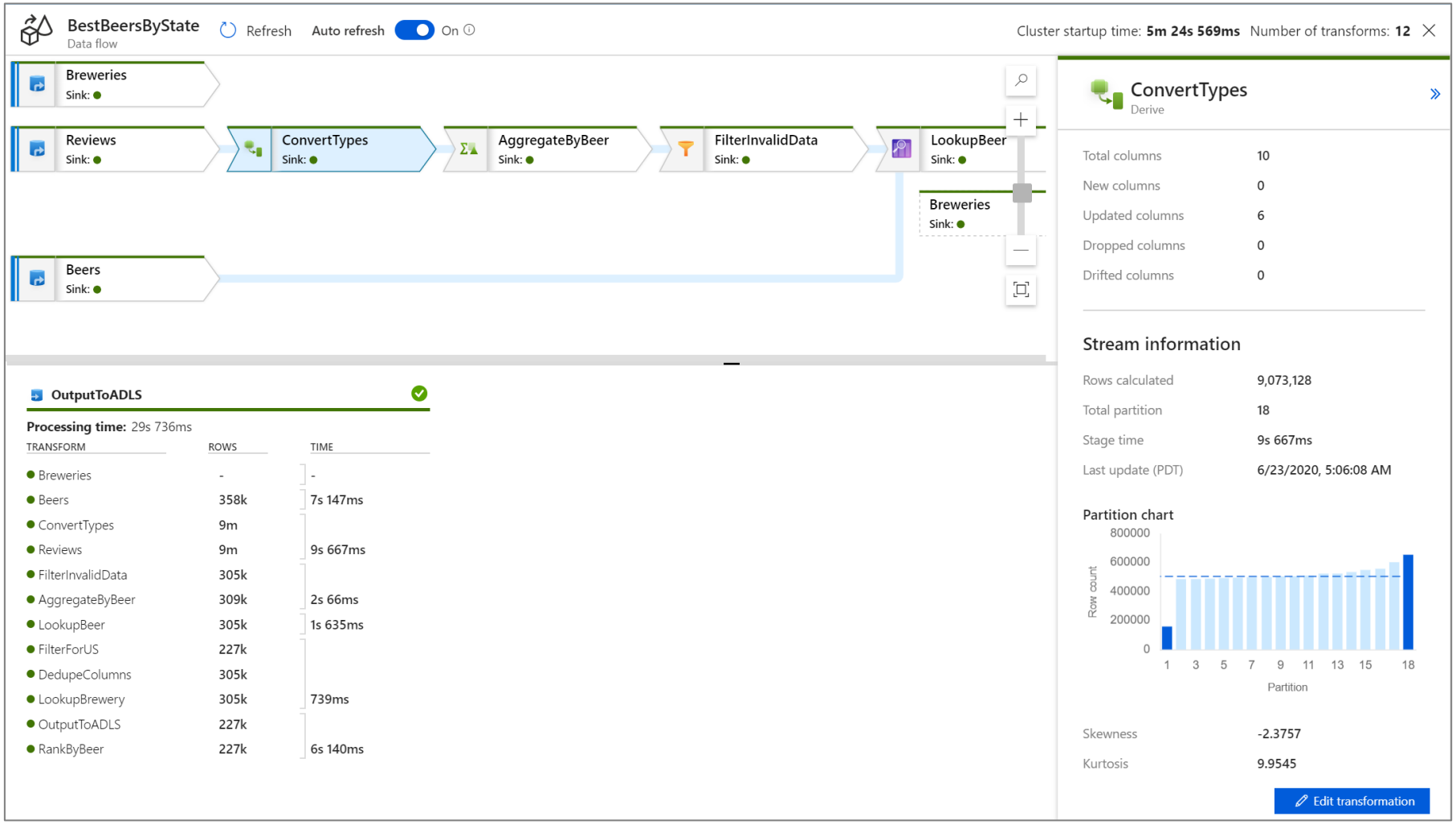


Data flow integration runtime

- Integrated with existing Azure IR
  - Choose compute type, # of cores, time to live
- **Time to live:** time a cluster is alive after last execution concludes
  - Minimal start up time for sequential data flows
- Parameterize compute type, # of cores if using *Auto Resolve*



Monitoring data flows



Data flow security considerations

- All data stays inside VMs that run the Spark cluster which are spun up JIT for each job
  - Azure Spark attaches storage to the VMs for logging and spill-over from in-memory data frames during job operation. These storage accounts are fully encrypted and within the Microsoft tenant.
  - Each cluster is single-tenant and specific to your data and job. This cluster is not shared with any other tenant
- Data flow processes are completely ephemeral. Once a job is completed, all associated resources are destroyed
  - Both cluster and storage account are deleted
- Data transfers in data flows are protected using certificates
- Active telemetry is logged and maintained for 45 days for troubleshooting by the Azure Data Factory team

Data flow best practices and optimizations

Best practices – Lifecycle

1. Test your transformation logic using debug mode and data preview
   - Limited source size or use sample files
2. Test end-to-end pipeline logic using pipeline debug
   - Verify data is read/written correctly
   - Used as smoke test before merging your changes
3. Publish and trigger your pipelines within a Dev Factory
   - Test performance and cluster size
4. Promote pipelines to higher environments such as UAT and PROD using CI/CD
   - Increase size and scope of data as you get to higher environments

Best practices – Debug (Data Preview)

- Data Preview
  - Data preview is inside the data flow designer transformation properties
  - Uses row limits and sampling techniques to preview data from a small size of data
  - Allows you to build and validate units of logic with samples of data in real time
  - You have control over the size of the data limits under Debug Settings
  - If you wish to test with larger datasets, set a larger compute size in the Azure IR when switching on “Debug Mode”
  - Data Preview is only a snapshot of data in memory from Spark data frames. This feature does not write any data, so the sink drivers are not utilized and not tested in this mode.

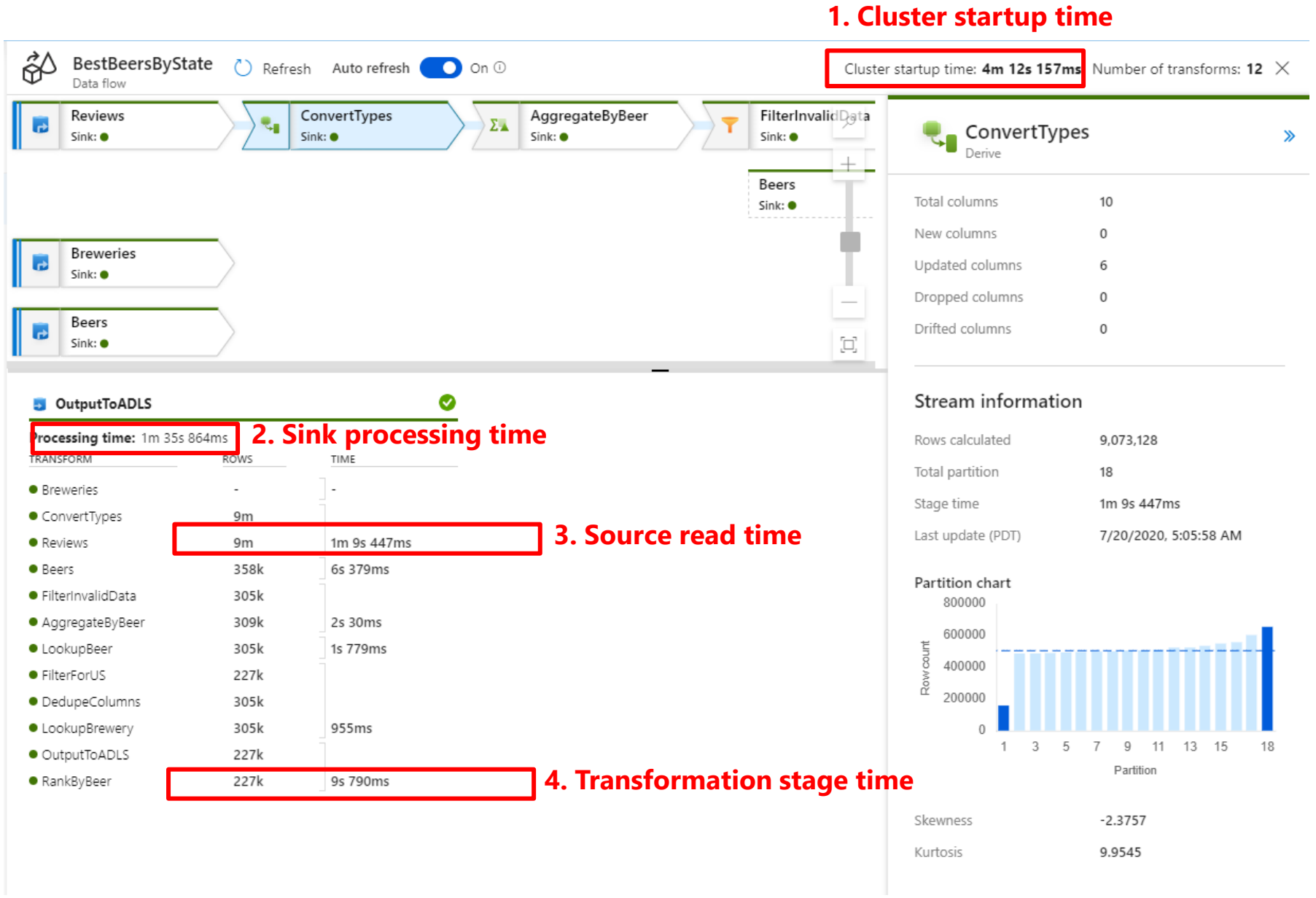
Best practices – Debug (Pipeline Debug)

- Pipeline Debug
  - Click debug button to test your data flow inside of a pipeline
  - Default debug limits the execution runtime so you will want to limit data sizes
  - Sampling can be applied here as well by using the “Enable Sampling” option in each Source
  - Use the debug button option of “use activity IR” when you wish to use a job execution compute environment
    - This option is good for debugging with larger datasets. It will not have the same execution timeout limit as the default debug setting

Optimizing data flows

- Transformation order generally does not matter
  - Data flows have a Spark optimizer that reorders logic to perform as best as it can
  - Repartitioning and reshuffling data negates optimizer
- Each transformation has ‘Optimize’ tab to control partitioning strategies
  - Generally do not need to alter
- Altering cluster size and type has performance impact
- Four components
  1. Cluster startup time
  2. Reading from sources
  3. Transformation time
  4. Writing to sinks

Identifying bottlenecks



1. Sequential executions can lower the cluster startup time by setting a TTL in Azure IR
2. Total time to process the stream from source to sink. There is also a post-processing time when you click on the Sink that will show you how much time Spark had to spend with partition and job clean-up. Write to single file and slow database connections will increase this time
3. Shows you how long it took to read data from source. Optimize with different source partition strategies
4. This will show you bottlenecks in your transformation logic. With larger general purpose and mem optimized IRs, most of these operations occur in memory in data frames and are usually the fastest operations in your data flow

Best practices - Sources

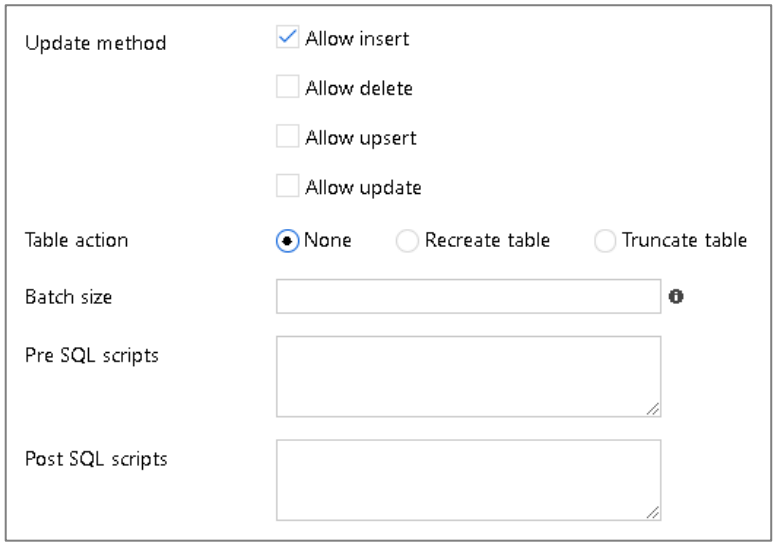
- When reading from file-based sources, data flow automatically partitions the data based on size
  - ~128 MB per partition, evenly distributed
  - Use current partitioning will be fastest for file-based and Synapse using PolyBase
    - Enable staging for Synapse
- For Azure SQL DB, use Source partitioning on column with high cardinality
  - Improves performance, but can saturate your source database
- Reading can be limited by the I/O of your source

Optimizing transformations

- Each transformation has its own optimize tab
  - Generally better to not alter -> reshuffling is a relatively slow process
- Reshuffling can be useful if data is very skewed
  - One node has a disproportionate amount of data
- For Joins, Exists and Lookups:
  - If you have a lot, memory optimized greatly increases performance
  - Can ‘Broadcast’ if the data on one side is small
    - Rule of thumb: Less than 50k rows
- Increasing integration runtime can speed up transformations
- Transformations that require reshuffling like Sort negatively impact performance

Best practices - Sinks

- SQL:
  - Disable indexes on target with pre/post SQL scripts
  - Increase SQL capacity during pipeline execution
  - Enable staging when using Synapse
- File-based sinks:
  - Use current partitioning allows Spark to create output
  - Output to single file is a very slow operation
    - Combines data into single partition
    - Often unnecessary by whoever is consuming data
  - Can set naming patterns or use data in column
  - Any reshuffling of data is slow
- Cosmos DB
  - Set throughput and batch size to meet performance requirements



Azure Integration Runtime

- Data Flows use JIT compute to minimize running expensive clusters when they are mostly idle
  - Generally more economical, but each cluster takes ~4 minutes to spin up
  - IR specifies what cluster type and core-count to use
    - Memory optimized is best, compute optimized doesn't generally work for production workloads
- When running Sequential jobs utilize *Time to Live* to reuse cluster between executions
  - Keeps cluster alive for TTL minutes after execution for new job to use
  - Maximum one job per cluster
- **Rule of thumb**: start small and scale up

Data flow script

Data flow script (DFS)

- DFS defines the logical intent of your data transformations
- Script is bundled and marshalled to Spark cluster as a job for execution
- DFS can be auto-generated and used for programmatic creation of data flows
- Access script behind UI via “Script” button

- Click “Copy as Single Line” to save version of script that is ready for JSON

- https://docs.microsoft.com/en-us/azure/data-factory/data-flow-script

Data flow script (DFS)

Movie Analytics: Split and aggregate genres , then unpivot column values to rows


```
source(output(
                movie as string,
                title as string,
                genres as string,
                year as string,
                Rating as integer,
                {Rotton Tomato} as string
        ),
        allowSchemaDrift: true,
        validateSchema: false) ~> source1
Aggregate1 unpivot(output(
                avg_genre as string,
                avgrating as double
        ),
        ungroupBy(year),
        lateral: false,
        ignoreNullPivots: true) ~> Unpivot1
source1 aggregate(groupBy(year),
        avg_comedy = round(avgIf(split(lower(genres),'|')[1]=='comedy',Rating),1,2),
                avg_drama = round(avgIf(split(lower(genres),'|')[1]=='drama',Rating),1,2),
                avg_action = round(avgIf(split(lower(genres),'|')[1]=='action',Rating),1,2)) ~> Aggregate1
Unpivot1 sort(desc(year, true)) ~> Sort1
Sort1 sink(allowSchemaDrift: true,
        validateSchema: false) ~> sink1
```

Source projection (1)

Source properties

Unpivot transformation (3)

Aggregate Transformation (2)

Sort (4)

Sink (5)

- Syntax: input_name transform_type(properties) ~> stream_name

Data flow script (DFS)

Eliminate duplicate rows and only keep distinct values

```
source(output(
                movie as integer,
                title as string,
                genres as string,
                year as string,
                Rating as string,
                {Rotton Tomato} as string
        ),
        allowSchemaDrift: true,
        validateSchema: false) ~> MoviesCSV
MoviesCSV aggregate(groupBy(movie),
        each(match(name!='movie'), $$ = first($$))) ~> DistinctRows
DistinctRows aggregate(rowcount_agg = count(1)) ~> RowCountDistinct
MoviesCSV select(mapColumn(
                movie,
                title,
                genres,
                year,
                Rating,
                {Rotton Tomato}
        ),
        skipDuplicateMapInputs: false,
        skipDuplicateMapOutputs: false) ~> OriginalData
OriginalData aggregate(rowcount_orig = count(1)) ~> RowCountOrig
DistinctRows sink(allowSchemaDrift: true,
        validateSchema: false,
        partitionBy('hash', 1)) ~> OutputDistinctData
```

Source projection (1)

Source properties

Distinct Aggregate (3)

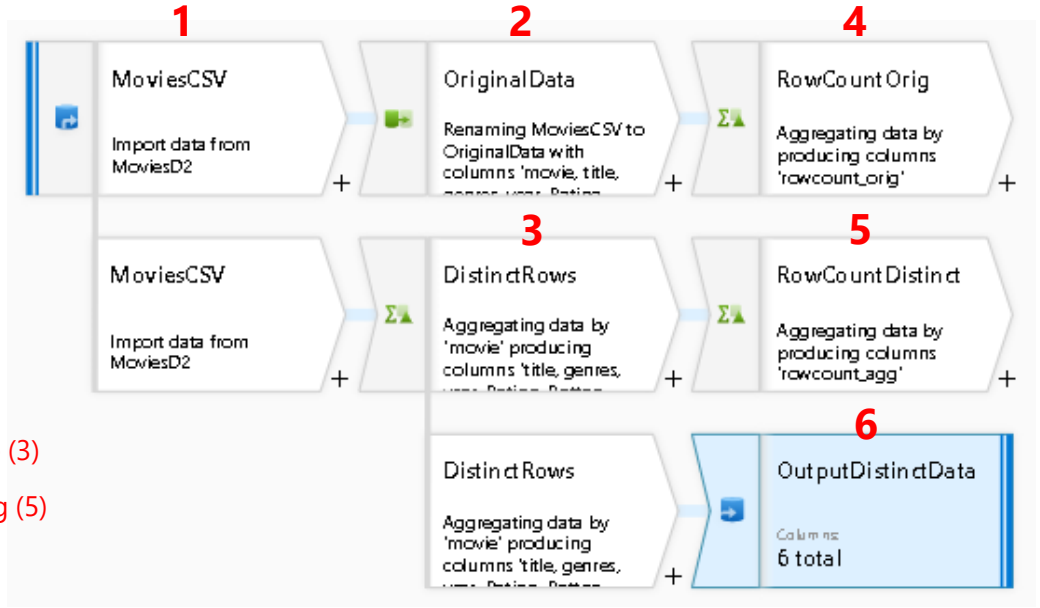
Row Count Agg (5)

Select transformation mappings (2)

Select properties (2)

Row Count Agg (4)

Sink transformation (6)



- ~> name_of_transform
- New branch does not require any script element

Azure Synapse SQLPool security and compliance

Enterprise-grade security

Industry-leading compliance

ISO 27001

SOC 1 Type 2

SOC 2 Type 2

PCI DSS Level 1

Cloud Controls Matrix

ISO 27018

Content Delivery and Security Association

Shared Assessments

FedRAMP JAB P-ATO

HIPAA / HITECH

FIPS 140-2

21 CFR Part 11

FERPA

DISA Level 2

CJIS

IRS 1075

ITAR-ready

Section 508 VPAT

European Union Model Clauses

EU Safe Harbor

United Kingdom G-Cloud

China Multi Layer Protection Scheme

China GB 18030

China CCCPPF

Singapore MTCS Level 3

Australian Signals Directorate

New Zealand GCIO

Japan Financial Services

ENISA IAF

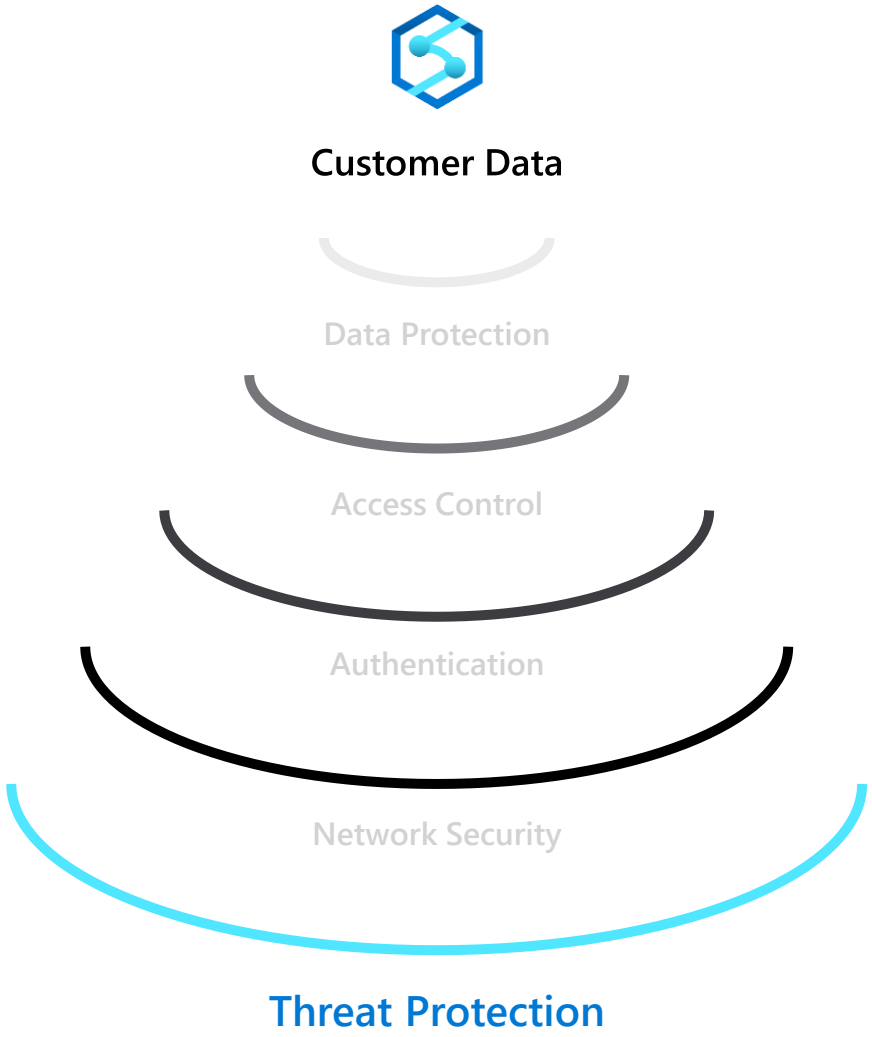
Threat Protection - Business requirements

How do we enumerate and track potential SQL vulnerabilities?

To mitigate any security misconfigurations before they become a serious issue.

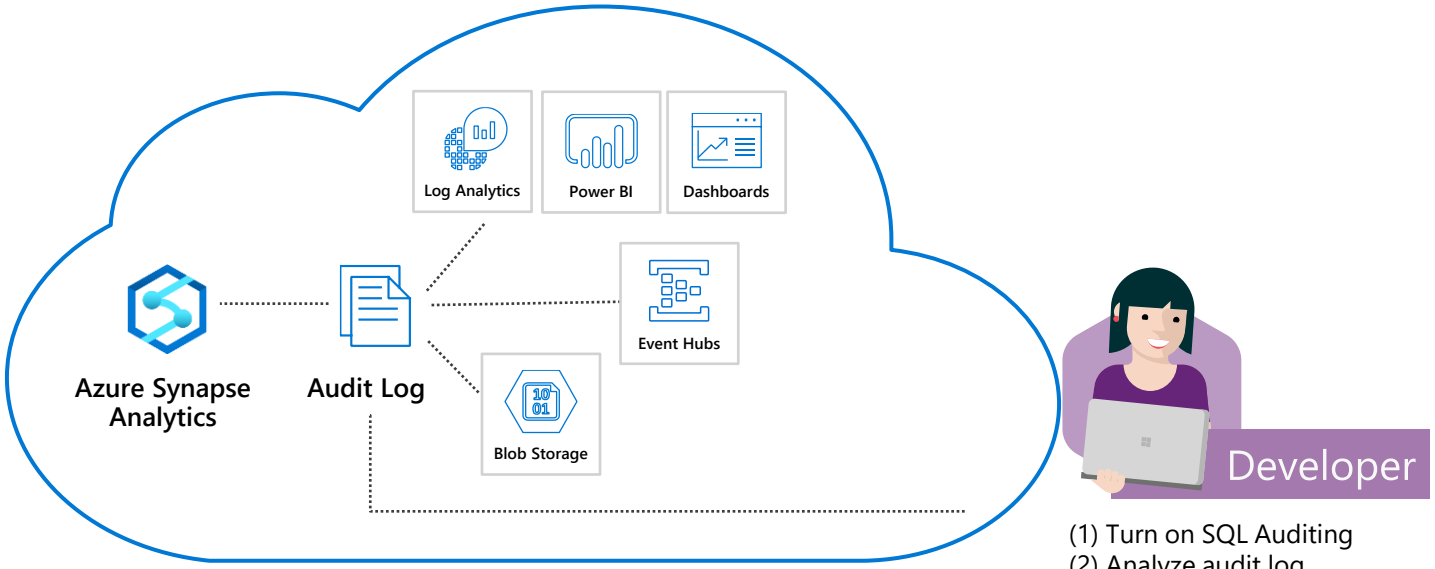
How do we discover and alert on suspicious database activity?

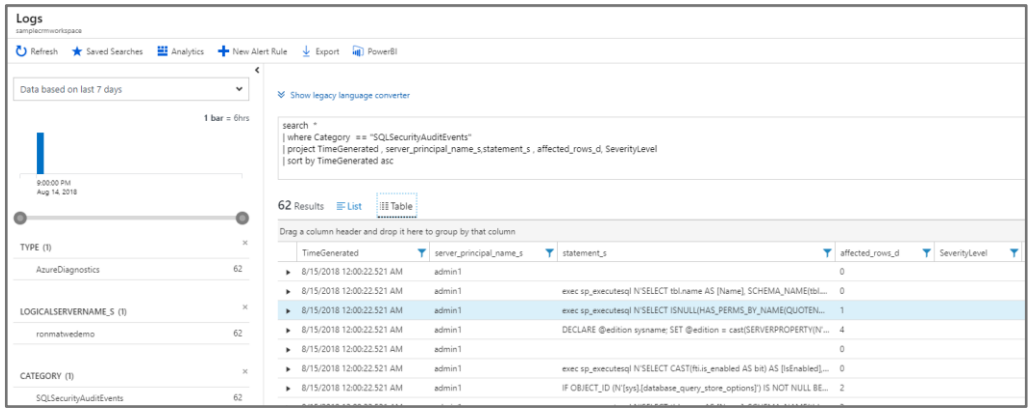
To detect and resolve any data exfiltration or SQL injection attacks.



SQL auditing in Azure Log Analytics and Event Hubs

Gain insight into database audit log

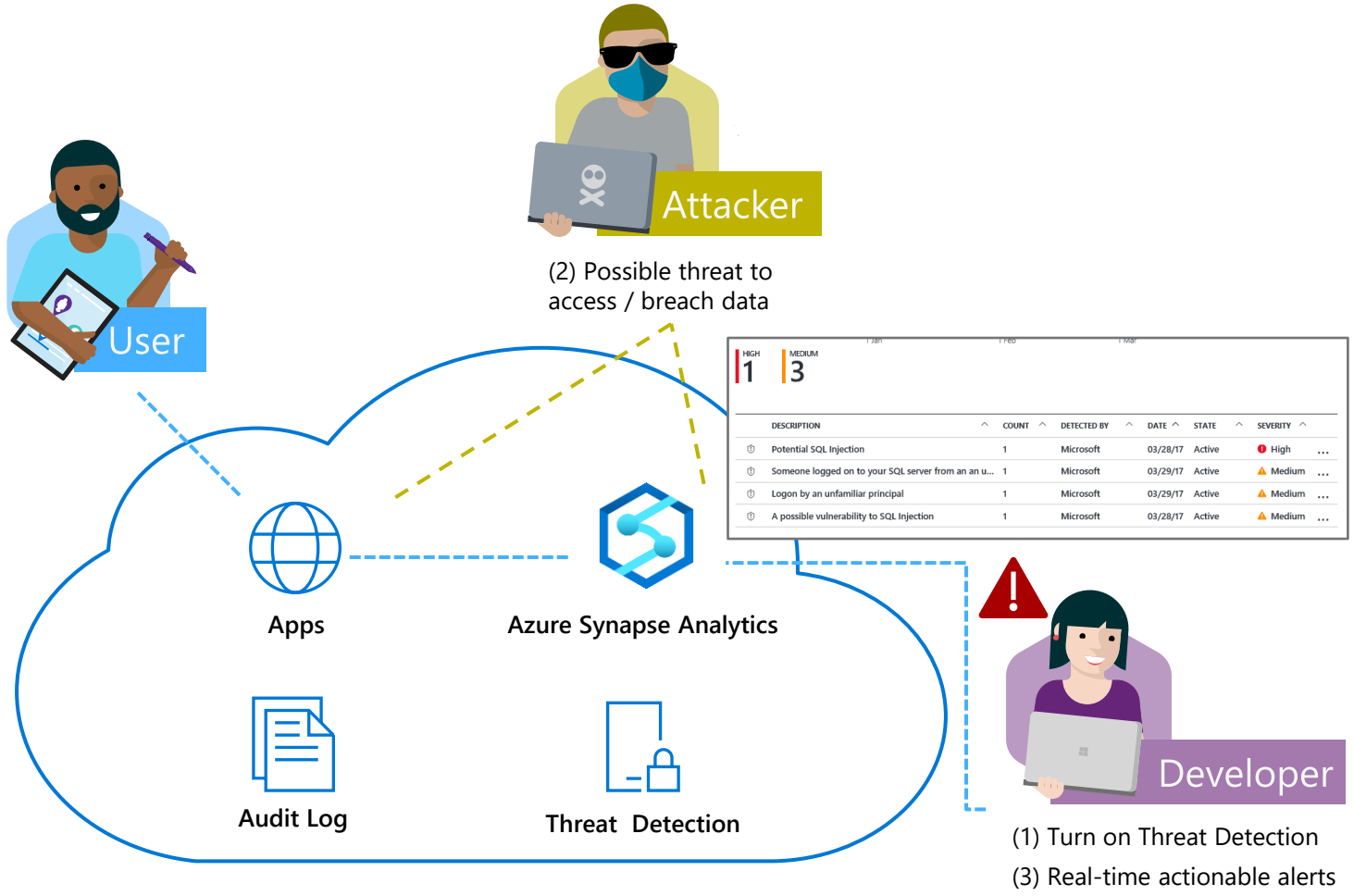




- ✓ **Configurable via audit policy**
- ✓ **SQL audit logs can reside in**
  - Azure Storage account
  - Azure Log Analytics
  - Azure Event Hubs
- ✓ **Rich set of tools for**
  - Investigating security alerts
  - Tracking access to sensitive data

SQL threat detection

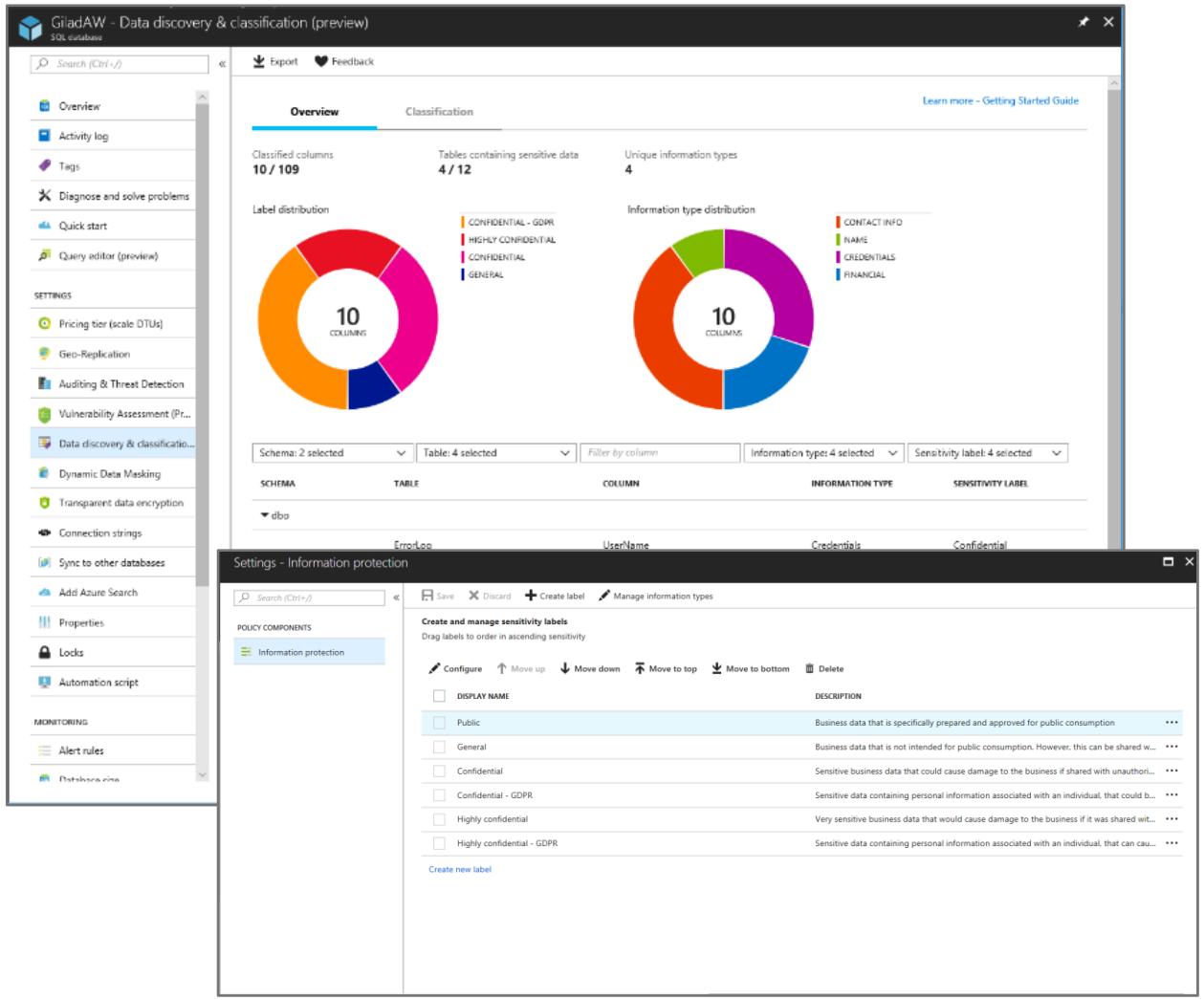
Detect and investigate anomalous database activity



- ✓ Detects potential SQL injection attacks
- ✓ Detects unusual access & data exfiltration activities
- ✓ Actionable alerts to investigate & remediate
- ✓ View alerts for your entire Azure tenant using Azure Security Center

SQL Data Discovery & Classification

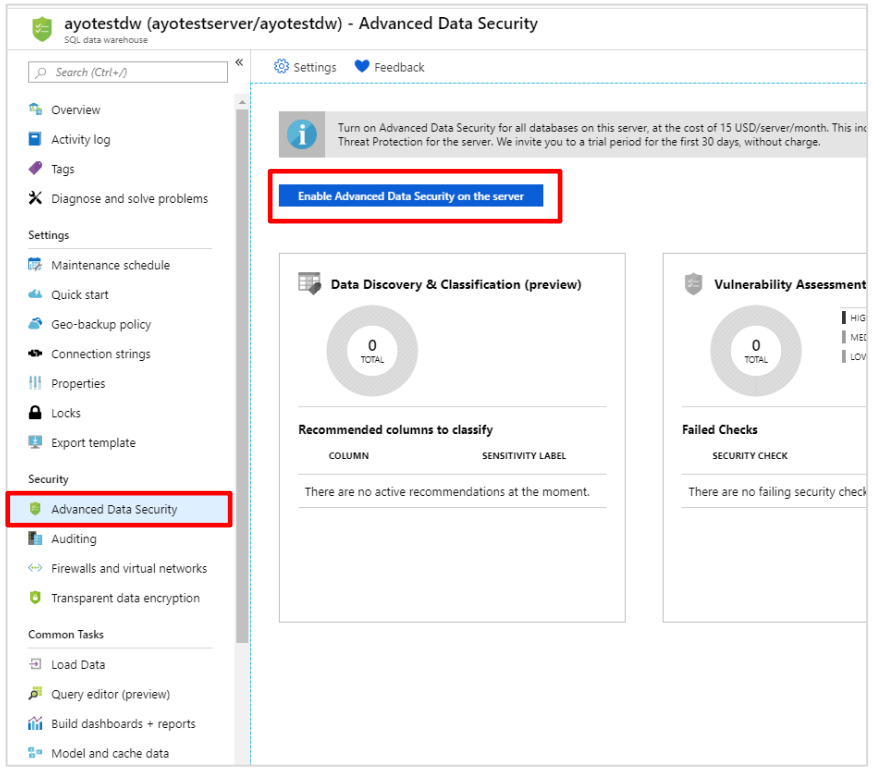
Discover, classify, protect and track access to sensitive data



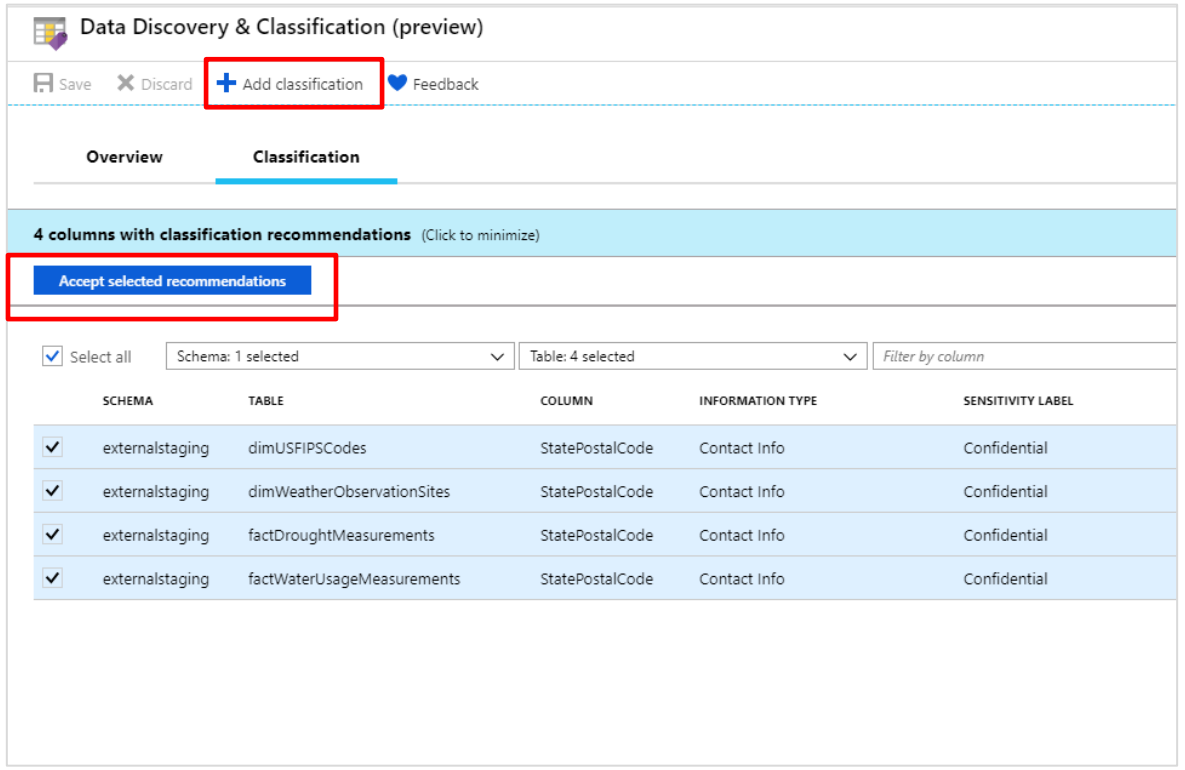
- ✓ Automatic discovery of columns with sensitive data
- ✓ Add persistent sensitive data labels
- ✓ Audit and detect access to the sensitive data
- ✓ Manage labels for your entire Azure tenant using Azure Security Center

SQL Data Discovery & Classification - setup

Step 1: Enable Advanced Data Security on the logical SQL Server

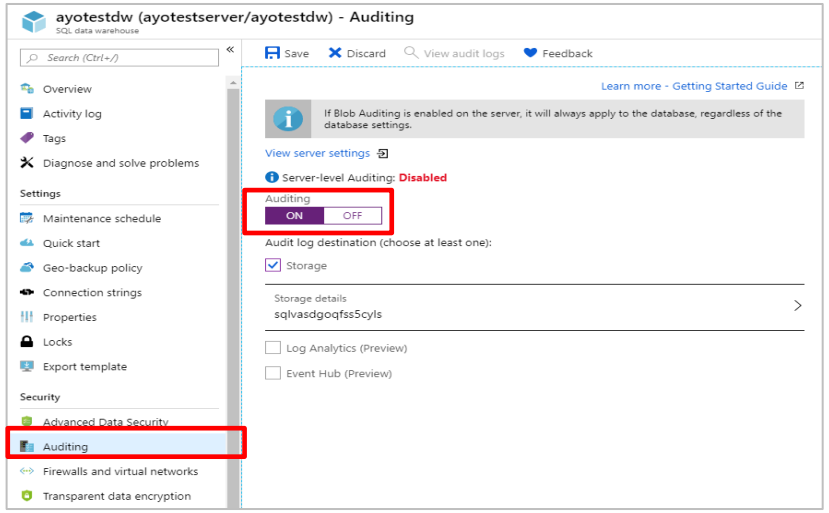


Step 2: Use recommendations and/or manual classification to classify all the sensitive columns in your tables

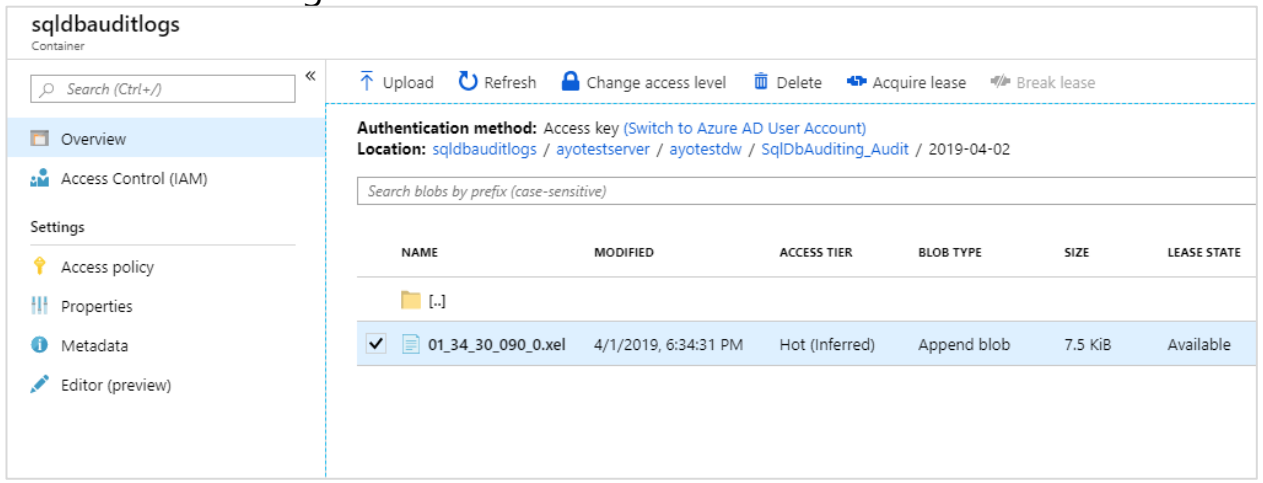


SQL Data Discovery & Classification – audit sensitive data access

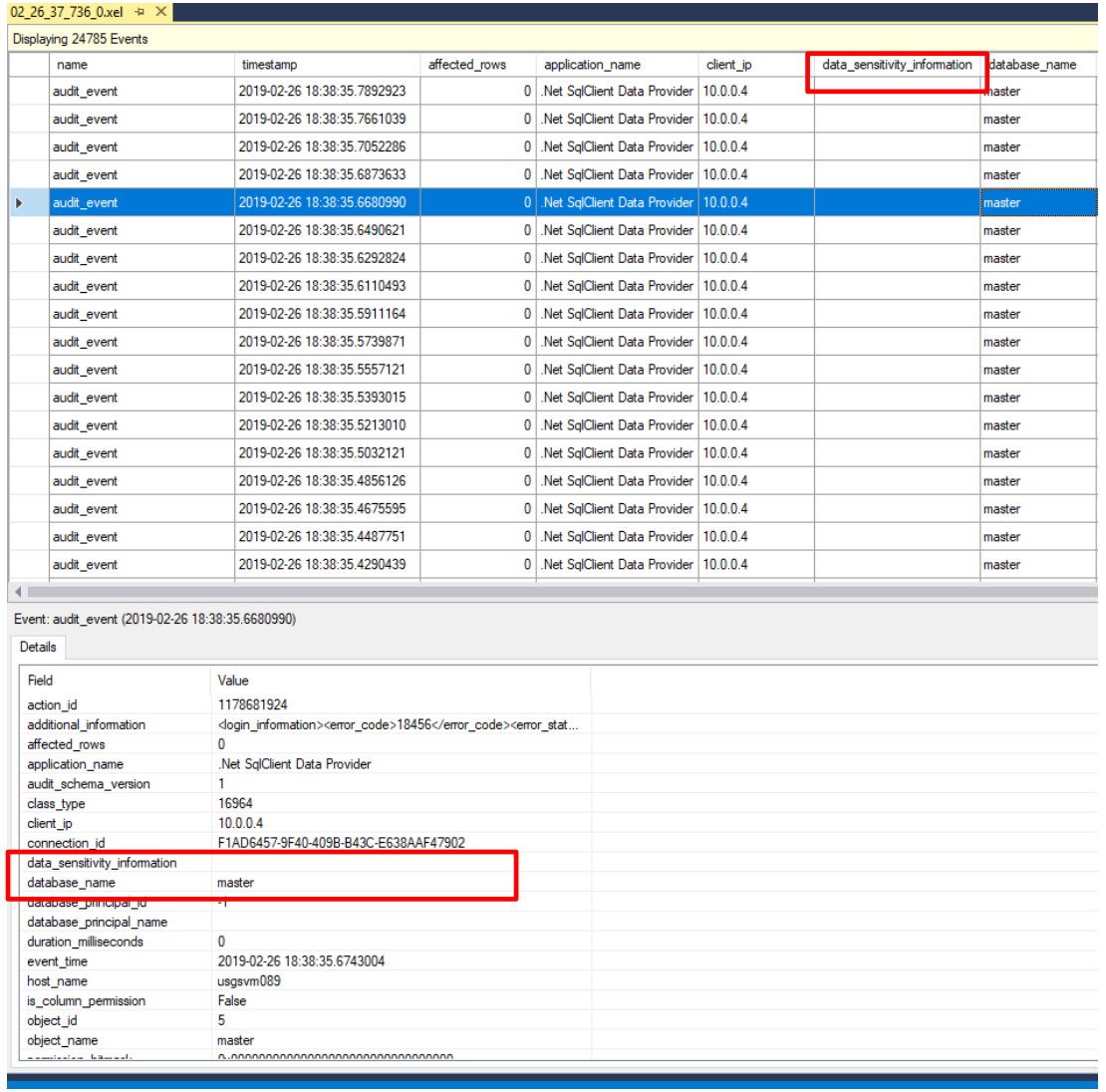
Step 1: Configure auditing for your target Data warehouse. This can be configured for just a single data warehouse or all databases on a server.



Step 2: Navigate to audit logs in storage account and download 'xel' log files to local machine.



Step 3: Open logs using extended events viewer in SSMS. Configure viewer to include 'data_sensitivity_information' column



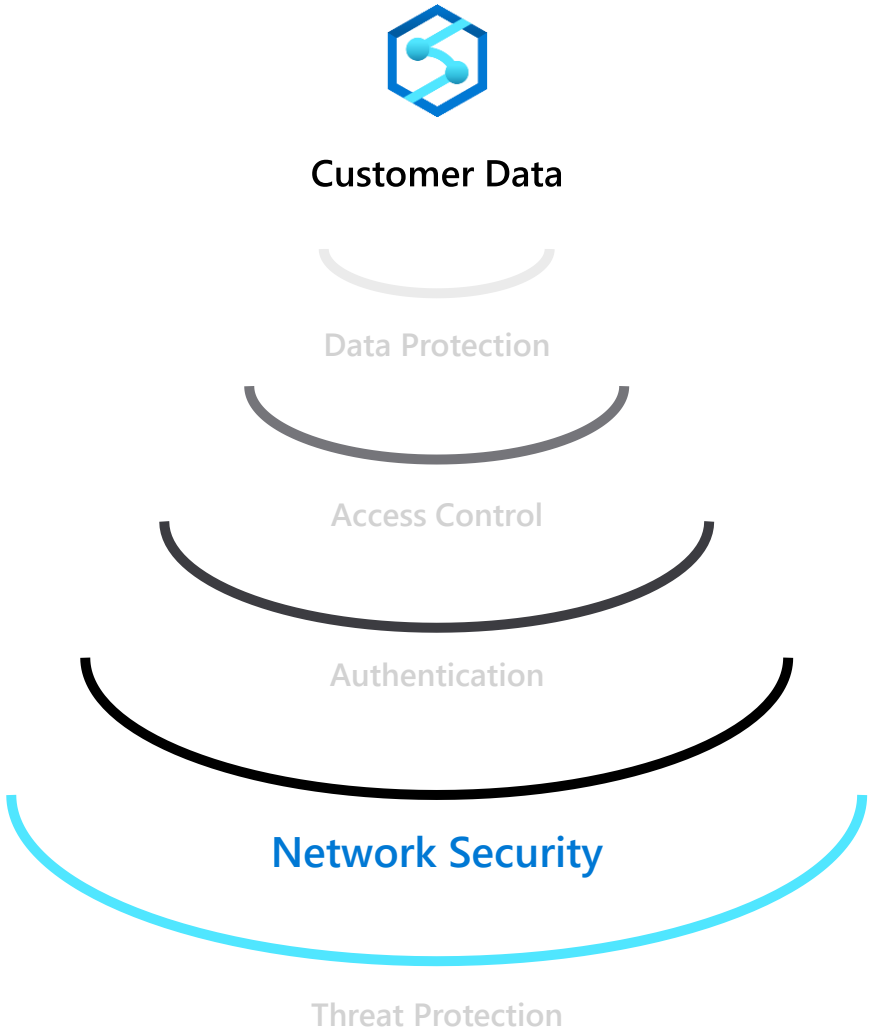
Network Security - Business requirements

How do we implement network isolation?

Data at different levels of security needs to be accessed from different locations.

How do we achieve separation?

Disallowing access to entities outside the company's network security boundary.

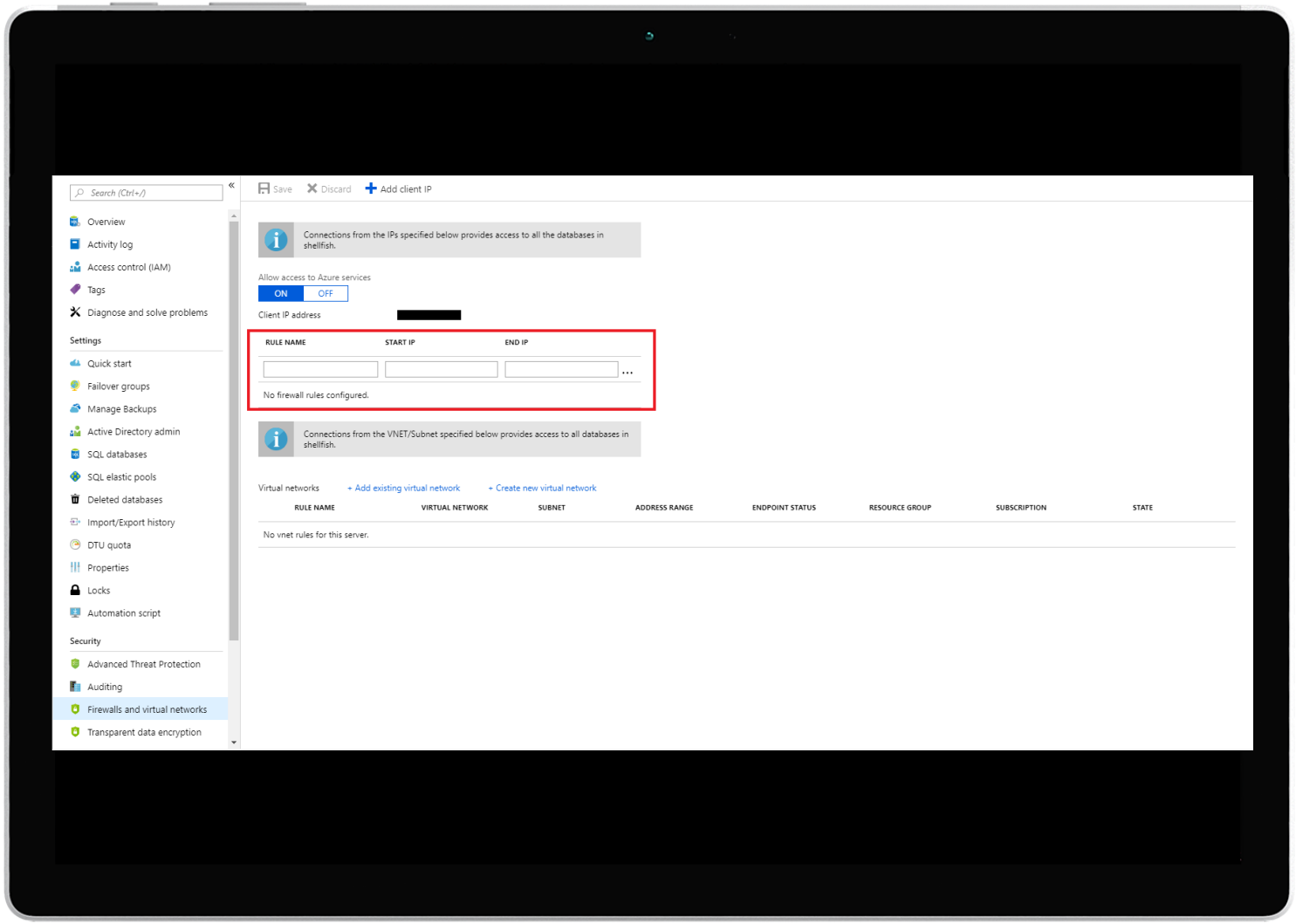


Firewall configuration on the portal

By default, Azure blocks all external connections to port 1433

Configure with the following steps:

Azure Synapse Analytics Resource:
Server name > Firewalls and virtual networks



Firewall configuration using REST API

Managing firewall rules through REST API must be authenticated.

For information, see Authenticating Service Management Requests.

Server-level rules can be created, updated, or deleted using REST API.

To create or update a server-level firewall rule, execute the PUT method.

To remove an existing server-level firewall rule, execute the DELETE method.

To list firewall rules, execute the GET.

```
PUT
https://management.azure.com/subscriptions/{subscriptionI
d}/resourceGroups/{resourceGroupName}/providers/Microsoft
.Sql/servers/{serverName}/firewallRules/{firewallRuleName
}?api-version=2014-04-01REQUEST BODY
{
  "properties": {
    "startIpAddress": "0.0.0.3",
    "endIpAddress": "0.0.0.3"
  }
}

DELETE
https://management.azure.com/subscriptions/{subscriptionI
d}/resourceGroups/{resourceGroupName}/providers/Microsoft
.Sql/servers/{serverName}/firewallRules/{firewallRuleName
}?api-version=2014-04-01

GET
https://management.azure.com/subscriptions/{subscriptionI
d}/resourceGroups/{resourceGroupName}/providers/Microsoft
.Sql/servers/{serverName}/firewallRules/{firewallRuleName
}?api-version=2014-04-01
```

Firewall configuration using PowerShell/T-SQL

Windows PowerShell Azure cmdlets

New-AzureRmSqlServerFirewallRule

Get-AzureRmSqlServerFirewallRule

Set-AzureRmSqlServerFirewallRule

Transact SQL

sp_set_firewall_rule

sp_delete_firewall_rule

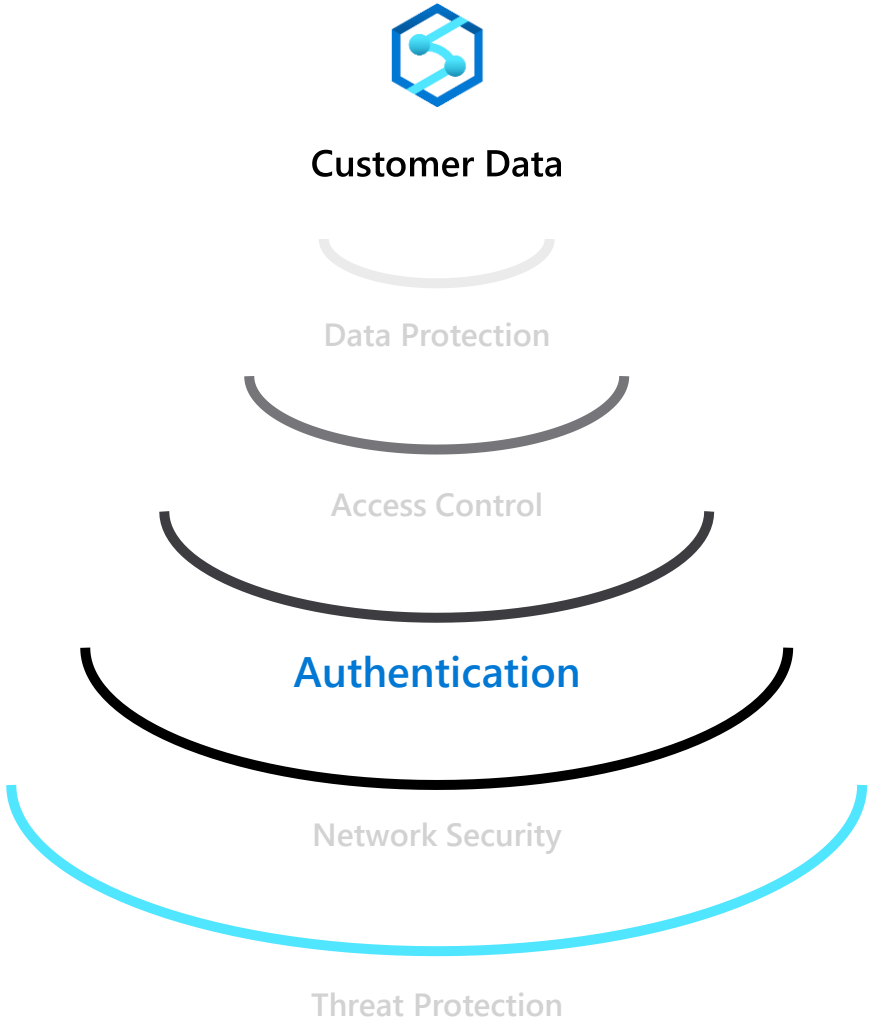
```
# PS Allow external IP access to SQL DW
PS C:\> New-AzureRmSqlServerFirewallRule
          -ResourceGroupName "myResourceGroup" `
          -ServerName $servername `
          -FirewallRuleName "AllowSome"
          -StartIpAddress "0.0.0.0"
          -EndIpAddress "0.0.0.0“

-- T-SQL Allow external IP access to SQL DW
EXECUTE sp_set_firewall_rule
          @name = N'ContosoFirewallRule’,
          @start_ip_address = '192.168.1.1’,
          @end_ip_address = '192.168.1.10'
```

Authentication - Business requirements

How do I configure Azure Active Directory with Azure Synapse Analytics?

I want additional control in the form of multi-factor authentication

How do I allow non-Microsoft accounts to be able to authenticate?

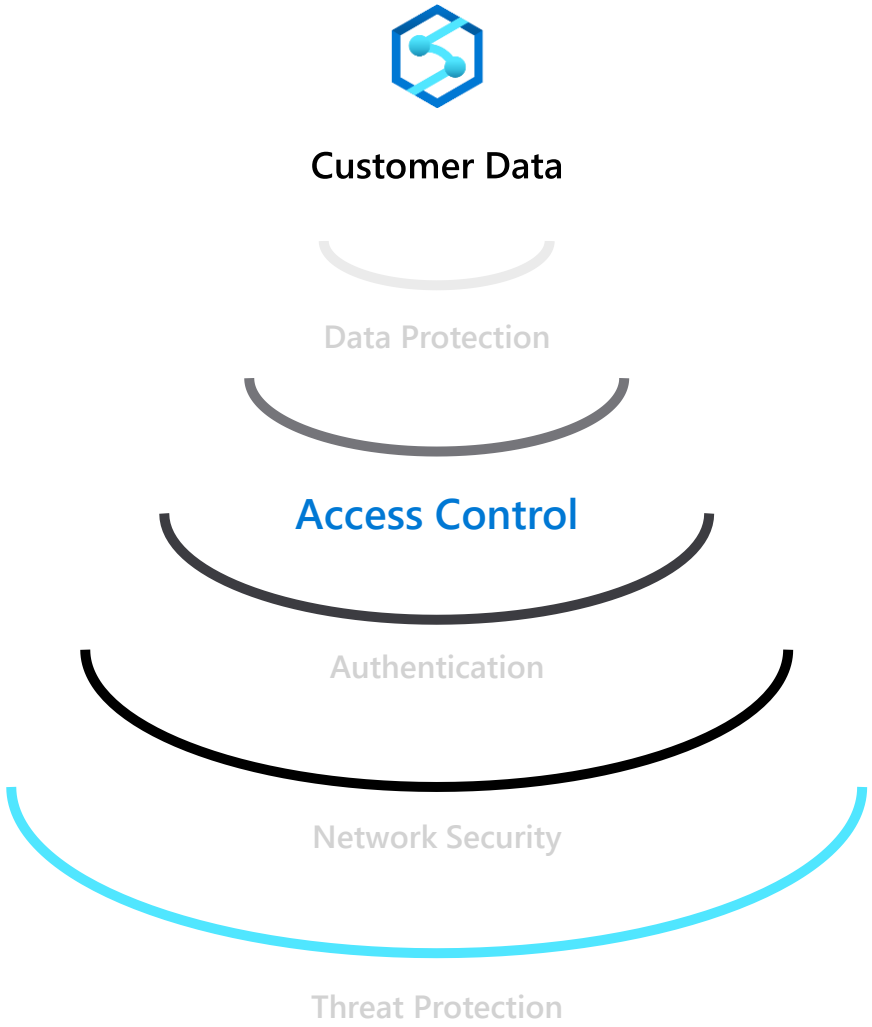


Access Control - Business requirements

How do I restrict access to sensitive data to specific database users?

How do I ensure users only have access to relevant data?

For example, in a hospital only medical staff should be allowed to see patient data that is relevant to them—and not every patient's data.



Object-level security (tables, views, and more)

Overview

GRANT controls permissions on designated tables, views, stored procedures, and functions.

Prevent unauthorized queries against certain tables.

Simplifies design and implementation of security at the database level as opposed to application level.

```
-- Grant SELECT permission to user RosaQdM on table Person.Address in the AdventureWorks2012 database
GRANT SELECT ON OBJECT::Person.Address TO RosaQdM;
GO
-- Grant REFERENCES permission on column BusinessEntityID in view HumanResources.vEmployee to user Wanida
GRANT REFERENCES(BusinessEntityID) ON OBJECT::HumanResources.vEmployee to Wanida with GRANT OPTION;
GO
-- Grant EXECUTE permission on stored procedure HumanResources.uspUpdateEmployeeHireInfo to an application role called Recruiting11
USE AdventureWorks2012;
GRANT EXECUTE ON OBJECT::HumanResources.uspUpdateEmployeeHireInfo TO RECRUITING 11;
GO
```

Row-level security (RLS)

Overview

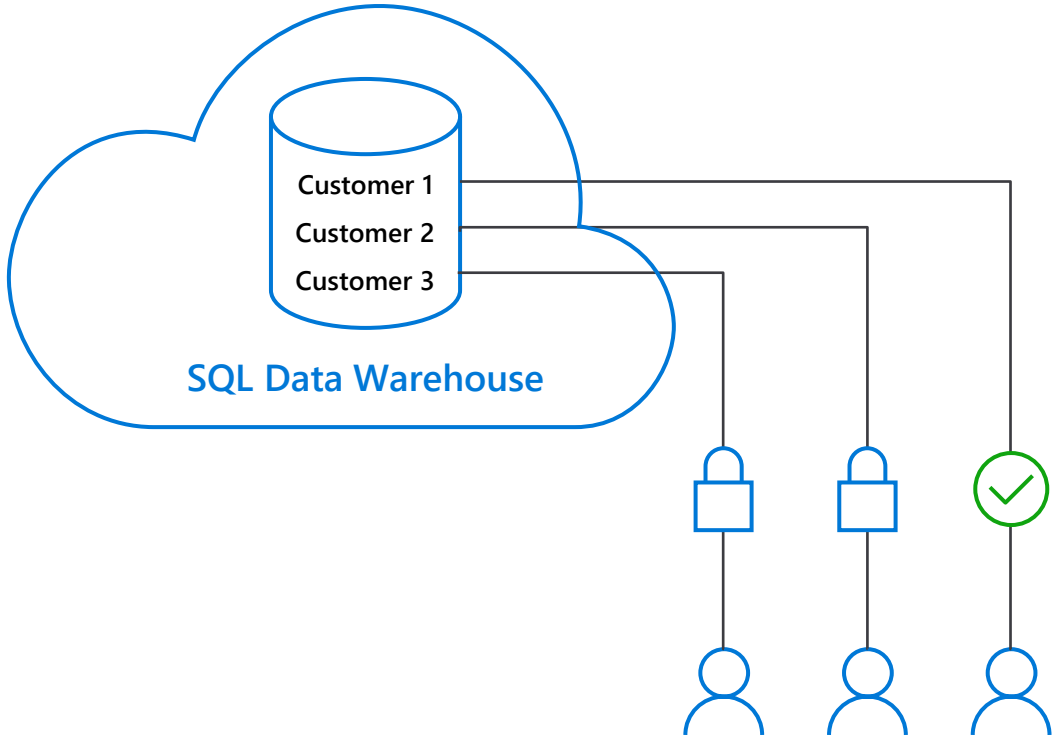
Fine grained access control of specific rows in a database table.

Help prevent unauthorized access when multiple users share the same tables.

Eliminates need to implement connection filtering in multi-tenant applications.

Administer via SQL Server Management Studio or SQL Server Data Tools.

Easily locate enforcement logic inside the database and schema bound to the table.



Row-level security

Creating policies

Filter predicates silently filter the rows available to read operations (SELECT, UPDATE, and DELETE).

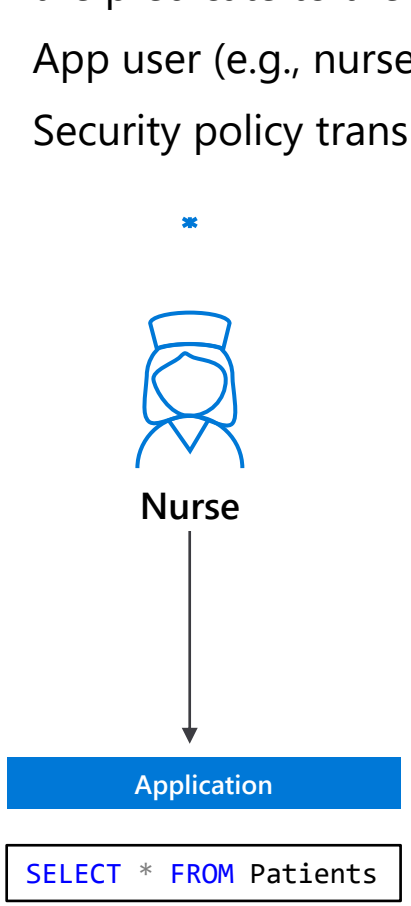
The following examples demonstrate the use of the CREATE SECURITY POLICY syntax

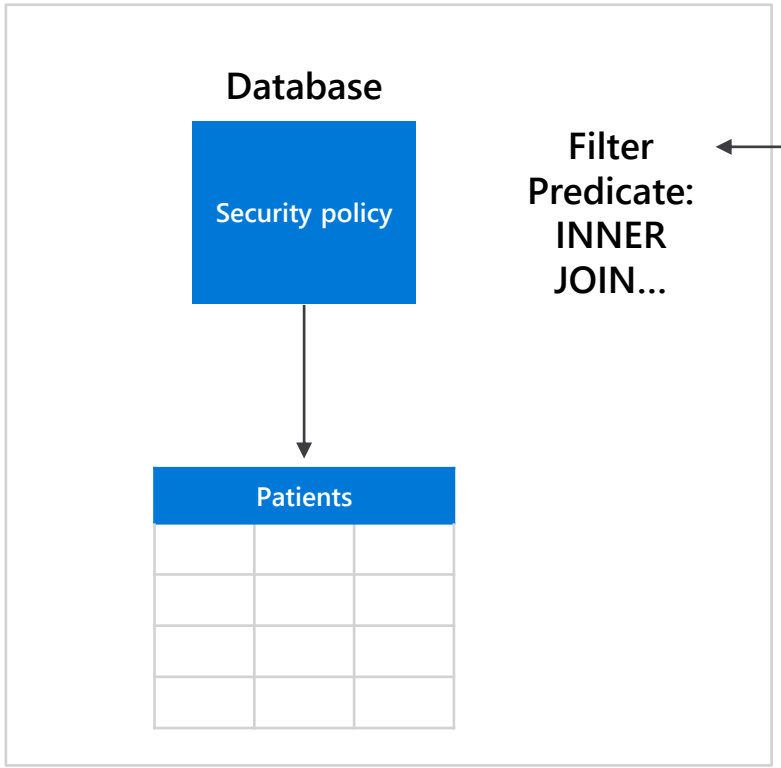
```
-- The following syntax creates a security policy with a filter predicate for the Customer table
CREATE SECURITY POLICY [FederatedSecurityPolicy]
ADD FILTER PREDICATE [rls].[fn_securitypredicate]([CustomerId])
ON [dbo].[Customer];


-- Create a new schema and predicate function, which will use the application user ID stored in CONTEXT_INFO to filter rows.
CREATE FUNCTION rls.fn_securitypredicate (@AppUserId int)
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN (
SELECT 1 AS fn_securitypredicate_result
WHERE
DATABASE_PRINCIPAL_ID() = DATABASE_PRINCIPAL_ID('dbo') -- application context
AND CONTEXT_INFO() = CONVERT(VARBINARY(128), @AppUserId));
GO
```

Row-level security

Three steps:

1. Policy manager creates filter predicate and security policy in T-SQL, binding the predicate to the patients table.
2. App user (e.g., nurse) selects from Patients table.
3. Security policy transparently rewrites query to apply filter predicate.





Filter Predicate: INNER JOIN...

Policy manager

```
CREATE FUNCTION dbo.fn_securitypredicate(@wing int)
    RETURNS TABLE WITH SCHEMABINDING AS
    return SELECT 1 as [fn_securitypredicate_result] FROM
        StaffDuties d INNER JOIN Employees e
        ON (d.EmpId = e.EmpId)
        WHERE e.UserSID = SUSER_SID() AND @wing = d.Wing;

CREATE SECURITY POLICY dbo.SecPol
    ADD FILTER PREDICATE dbo.fn_securitypredicate(Wing) ON Patients
    WITH (STATE = ON)
```

```
SELECT * FROM Patients
   SEMIJOIN APPLY dbo.fn_securitypredicate(patients.Wing);
```

```
SELECT Patients.* FROM Patients,
   StaffDuties d INNER JOIN Employees e ON (d.EmpId = e.EmpId)
   WHERE e.UserSID = SUSER_SID() AND Patients.wing = d.Wing;
```

Column-level security

Overview

Control access of specific columns in a database table based on customer's group membership or execution context.

Simplifies the design and implementation of security by putting restriction logic in database tier as opposed to application tier.

Administer via GRANT T-SQL statement.

Both Azure Active Directory (AAD) and SQL authentication are supported.

Column-level security

Three steps:

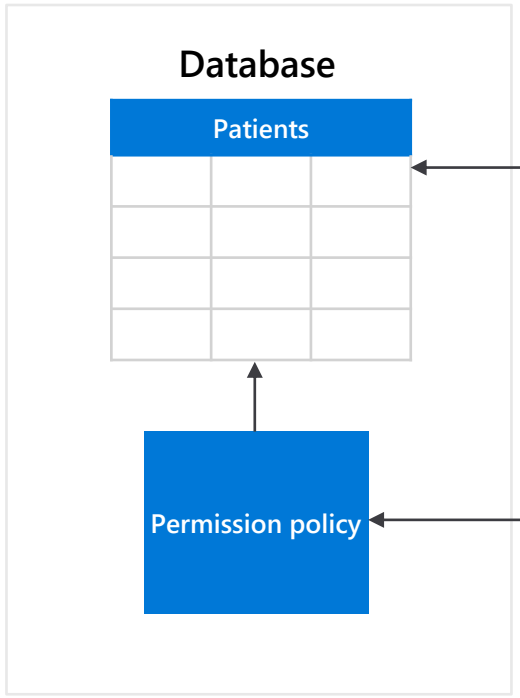
1. Policy manager creates permission policy in T-SQL, binding the policy to the Patients table on a specific group.
2. App user (for example, a nurse) selects from Patients table.
3. Permission policy prevents access on sensitive data.

Nurse

Application

```
SELECT * FROM Membership;

Msg 230, Level 14, State 1, Line 12
The SELECT permission was denied on the column
'SSN' of the object 'Membership', database
'CLS_TestDW', schema 'dbo'.
```

Queries executed as 'Nurse' will fail if they include the SSN column



Policy manager

```
CREATE TABLE Patients (
  PatientID int IDENTITY,
  FirstName varchar(100) NULL,
  SSN char(9) NOT NULL,
  LastName varchar(100) NOT NULL,
  Phone varchar(12) NULL,
  Email varchar(100) NULL
);
```

```
GRANT SELECT ON Patients (
  PatientID, FirstName, LastName, Phone, Email
) TO Nurse;
```

Allow 'Nurse' to access all columns except for sensitive SSN column

Data Protection - Business requirements

How do I protect sensitive data against unauthorized (high-privileged) users?

What key management options do I have?

Dynamic Data Masking

Overview

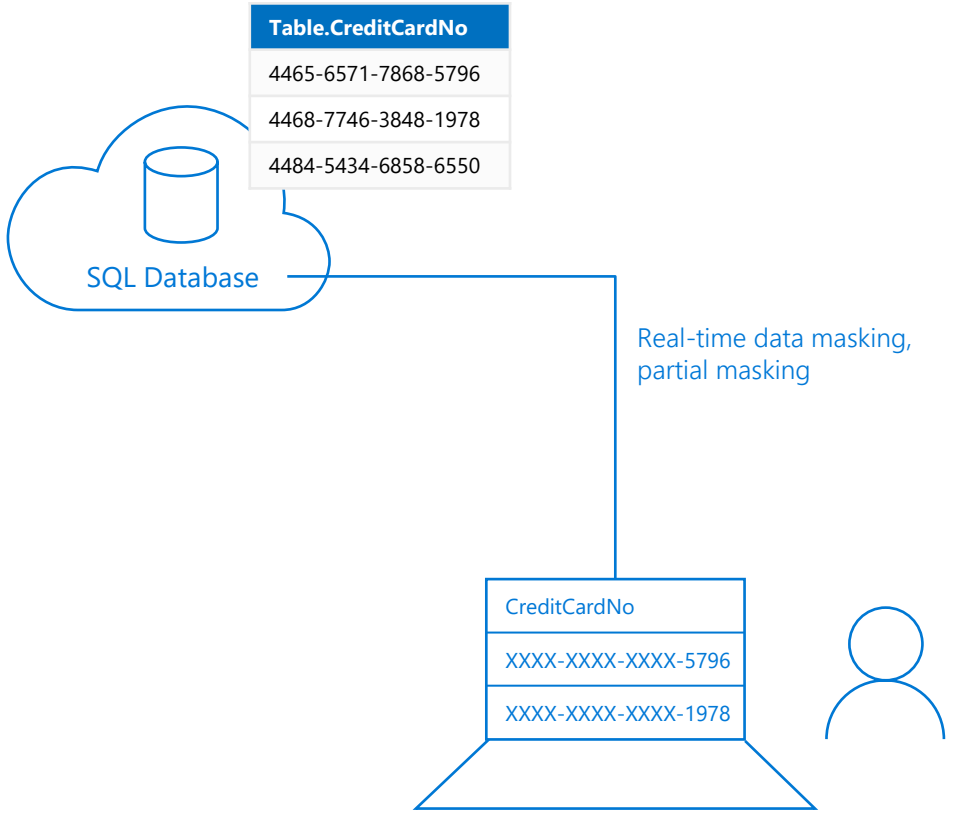
Prevent abuse of sensitive data by hiding it from users

Easy configuration in new Azure Portal

Policy-driven at table and column level, for a defined set of users

Data masking applied in real-time to query results based on policy

Multiple masking functions available, such as full or partial, for various sensitive data categories (credit card numbers, SSN, etc.)



Column Level Encryption

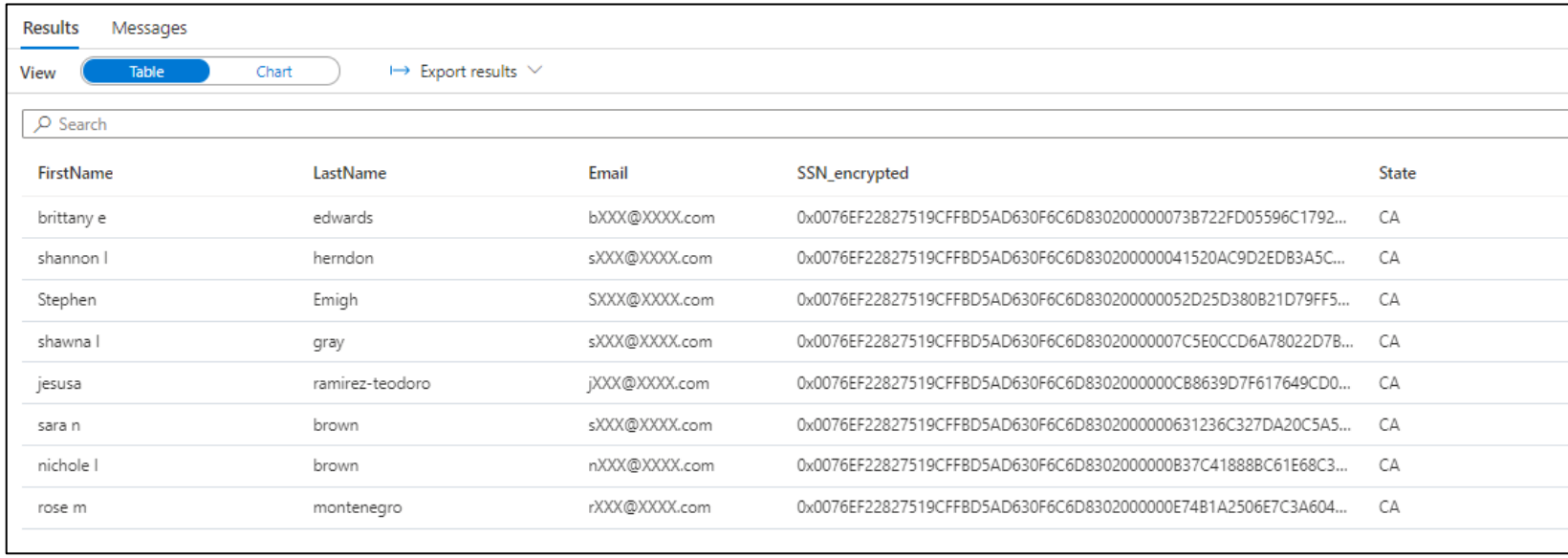
Overview

It helps to implement fine-grained protection of sensitive data within a table in dedicated SQL pool.

The data in CLE enforced columns is encrypted on disk. User need to use DECRYPTBYKEY function to decrypt it.

5 step process to set up CLE

1. Create master key
2. Create certificate
3. Configure symmetric key for encryption
4. Encrypt the column data
5. Close symmetric key



FirstName	LastName	Email	SSN_encrypted	State
brittany e	edwards	bXXX@XXXX.com	0x0076EF22827519CFFBD5AD630F6C6D830200000073B722FD05596C1792...	CA
shannon l	herndon	sXXX@XXXX.com	0x0076EF22827519CFFBD5AD630F6C6D830200000041520AC9D2EDB3A5C...	CA
Stephen	Emigh	SXXX@XXXX.com	0x0076EF22827519CFFBD5AD630F6C6D830200000052D25D380B21D79FF5...	CA
shawna l	gray	sXXX@XXXX.com	0x0076EF22827519CFFBD5AD630F6C6D83020000007C5E0CCD6A78022D7B...	CA
jesusa	ramirez-teodoro	jXXX@XXXX.com	0x0076EF22827519CFFBD5AD630F6C6D8302000000CB8639D7F617649CD0...	CA
sara n	brown	sXXX@XXXX.com	0x0076EF22827519CFFBD5AD630F6C6D8302000000631236C327DA20C5A5...	CA
nichole l	brown	nXXX@XXXX.com	0x0076EF22827519CFFBD5AD630F6C6D8302000000B37C41888BC61E68C3...	CA
rose m	montenegro	rXXX@XXXX.com	0x0076EF22827519CFFBD5AD630F6C6D8302000000E74B1A2506E7C3A604...	CA

Dynamic Data Masking

Three steps

1. Security officer defines dynamic data masking policy in T-SQL over sensitive data in the Employee table. The security officer uses the built-in masking functions (default, email, random)

2. The app-user selects from the Employee table

3. The dynamic data masking policy obfuscates the sensitive data in the query results for non-privileged users

Security officer

1

```
ALTER TABLE [Employee]
ALTER COLUMN [SocialSecurityNumber]
ADD MASKED WITH (FUNCTION = 'DEFAULT()')

ALTER TABLE [Employee]
ALTER COLUMN [Email]
ADD MASKED WITH (FUNCTION = 'EMAIL()')

ALTER TABLE [Employee]
ALTER COLUMN [Salary]
ADD MASKED WITH (FUNCTION = 'RANDOM(1,20000)')

GRANT UNMASK to admin1
```

2

Business app

```
SELECT [First Name],
       [Social Security Number],
       [Email],
       [Salary]
FROM   [Employee]
```

3

Non-masked data (admin login)

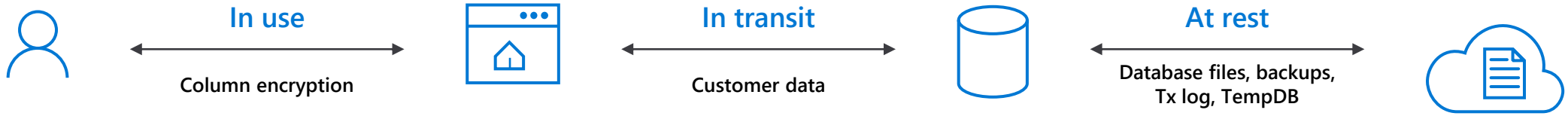
	First Name	Social Security Num...	Email	Salary
1	LILA	758-10-9637	lila.barnett@comcast.net	1012794
2	JAMIE	113-29-4314	jamie.brown@ntlworld.com	1025713
3	SHELLEY	550-72-2028	shelley.lynn@charter.net	1040131
4	MARCELLA	903-94-5665	marcella.estrada@comcast.net	1040753
5	GILBERT	376-79-4787	gilbert.juarez@verizon.net	1041308

Masked data (admin1 login)

	First Name	Social Security Number	Email	Salary
1	LILA	XXX-XX-XX37	lXX@XXXX.net	8940
2	JAMIE	XXX-XX-XX14	jXX@XXXX.com	19582
3	SHELLEY	XXX-XX-XX28	sXX@XXXX.net	3713
4	MARCELLA	XXX-XX-XX65	mXX@XXXX.net	11572
5	GILBERT	XXX-XX-XX87	gXX@XXXX.net	4487

Types of data encryption

Data Encryption	Encryption Technology	Customer Value
In transit	Transport Layer Security (TLS) from the client to the server TLS 1.2	Protects data between client and server against snooping and man-in-the-middle attacks
At rest	Transparent Data Encryption (TDE) for Azure Synapse Analytics	Protects data on the disk User or Service Managed key management is handled by Azure, which makes it easier to obtain compliance



Transparent data encryption (TDE)

Overview

All customer data encrypted at rest

TDE performs real-time I/O encryption and decryption of the data and log files.

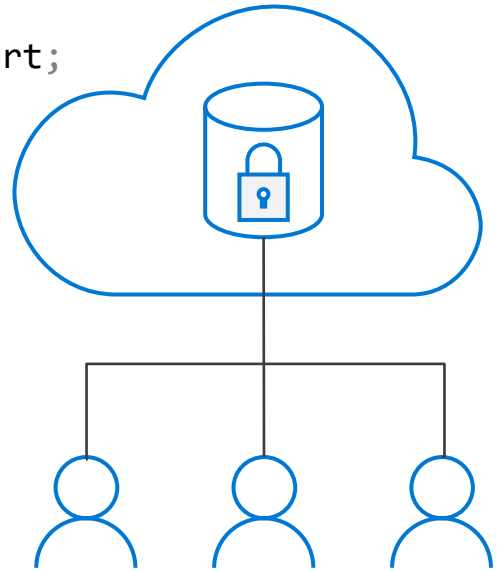
Service OR User managed keys.

Application changes kept to a minimum.

Transparent encryption/decryption of data in a TDE-enabled client driver.

Compliant with many laws, regulations, and guidelines established across various industries.

```
USE master;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<UseStrongPasswordHere>';
go
CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'My DEK Certificate';
go
USE MyDatabase;
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE MyServerCert;
GO
ALTER DATABASE MyDatabase
SET ENCRYPTION ON;
GO
```



Transparent data encryption (TDE)

Key Vault

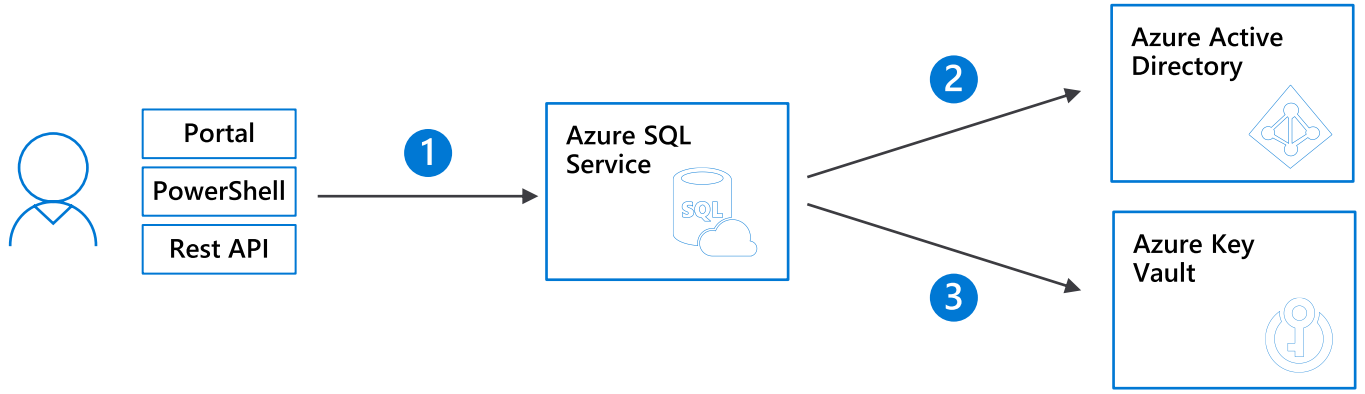
Benefits with User Managed Keys

Assume more control over who has access to your data and when.

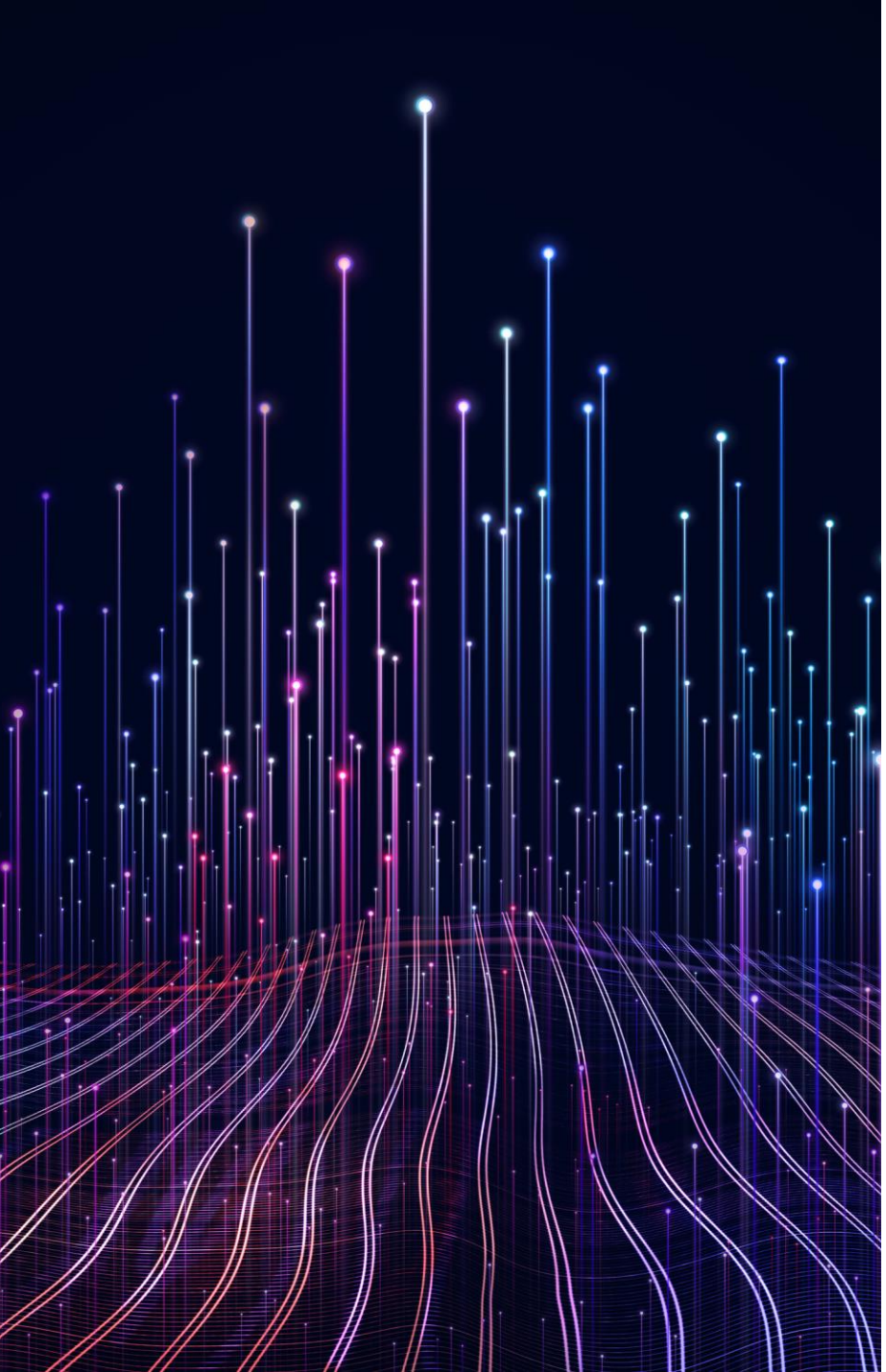
Highly available and scalable cloud-based key store.

Central key management that allows separation of key management and data.

Configurable via Azure Portal, PowerShell, and REST API.



1. The Key Vault admin grants vault access to the SQL Database server using its unique Azure Active Directory (AD) identity
2. The server uses its Azure AD identity to authenticate with Azure AD for access to your Key Vault
3. The server sends get, wrap key, and unwrap key request to the asymmetric key in key Vault for database encryption key protection.



Agenda

1 **Monitoring**

Monitor resources within Synapse Studio and the Azure Portal.

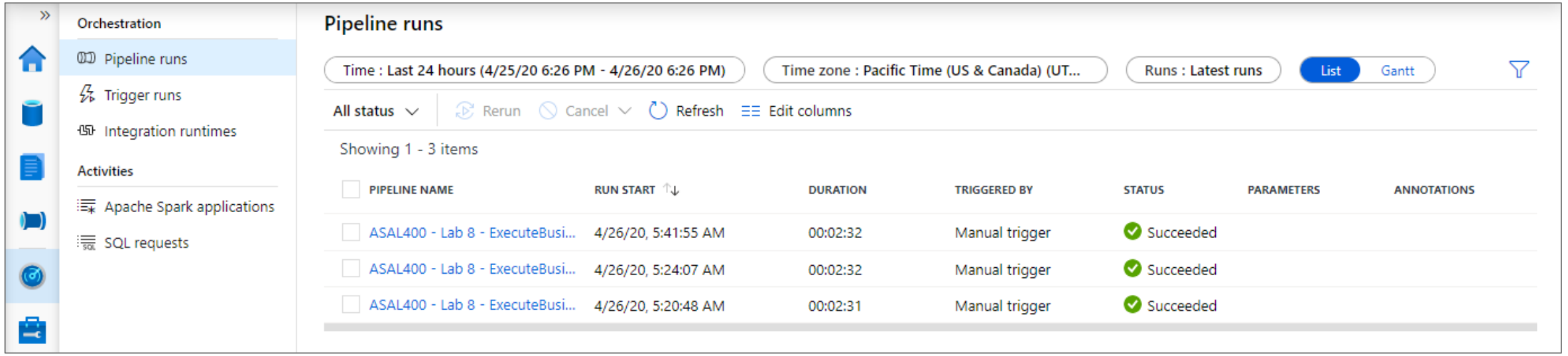
2 **Manage**

Manage resources for optimal utilization and performance.

Monitor within Monitor Hub

Overview

Within Synapse Studio you can monitor Pipeline Runs, Trigger Runs, Spark applications and SQL requests.



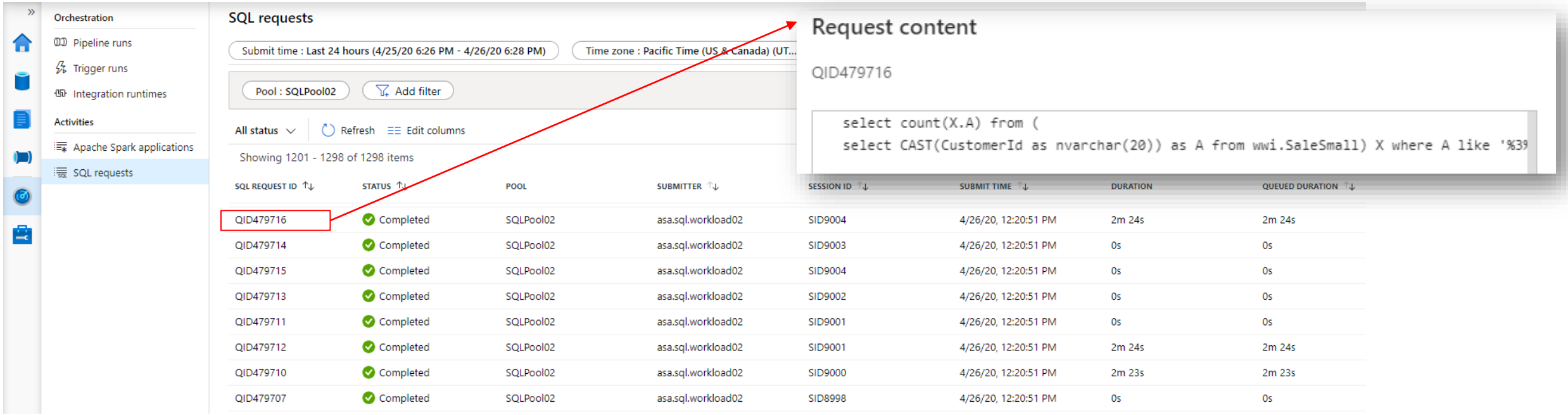
Monitor SQL Requests

Overview

Monitor SQL requests and view the content of the SQL queries.

Benefits

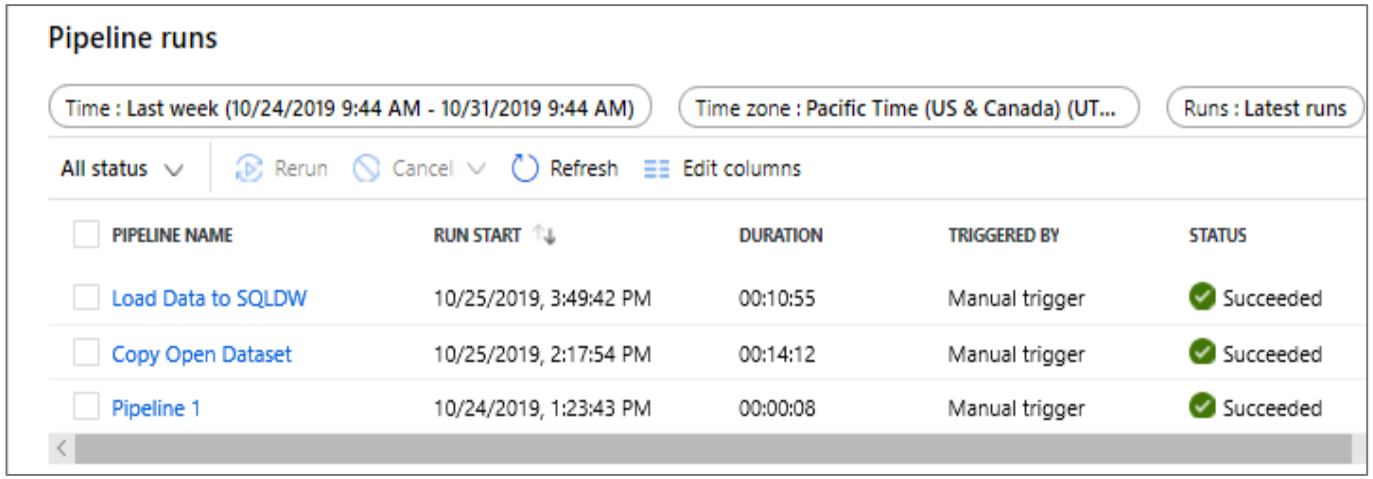
- View SQL requests, filter by time, user, workload group, and other properties.



Monitor Orchestration

Overview

Monitor orchestration in the Synapse workspace for the progress and status of pipeline



Benefits

Track all/specific pipelines

Monitor pipeline run and activity run details

Find the root cause of pipeline failure or activity failure

Monitor Spark applications

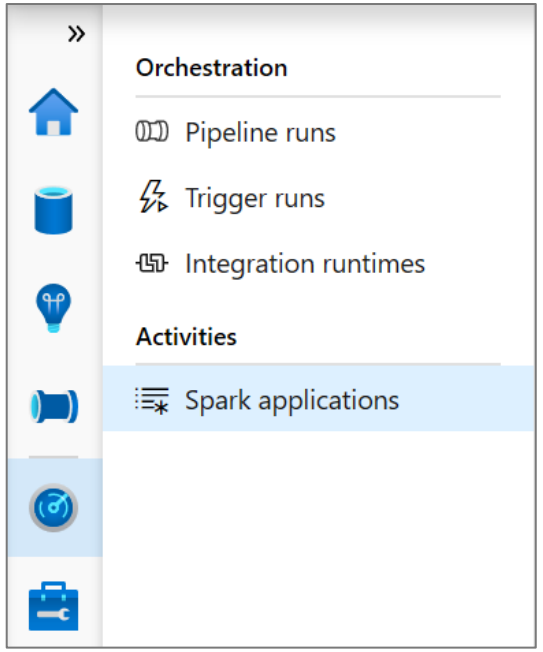
Overview

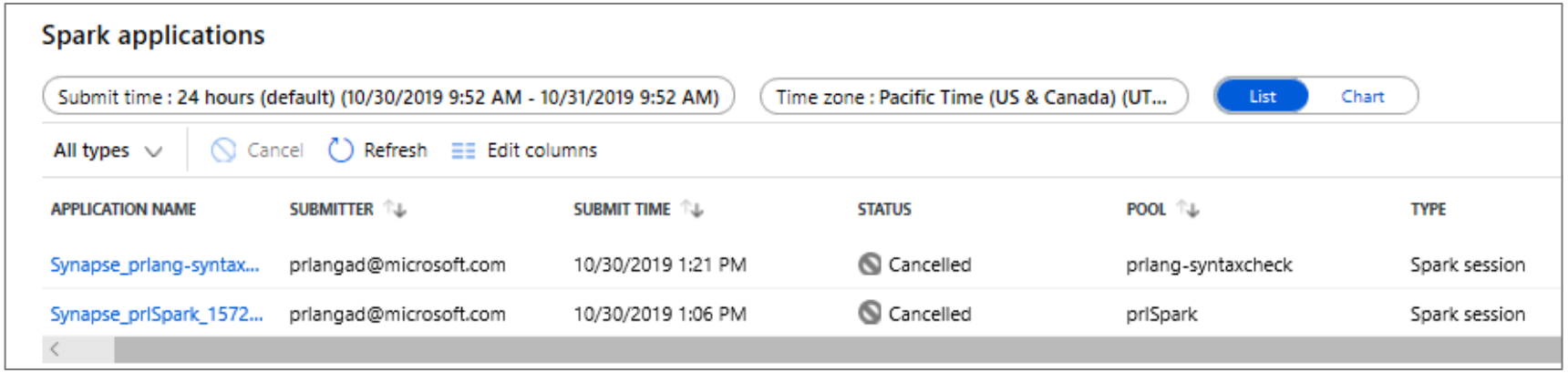
Monitor Spark pools, Spark applications for the progress and status of activities

Benefits

Monitor Spark pools for the status as paused, active, resume, scaling and upgrading

Track the usage of resources





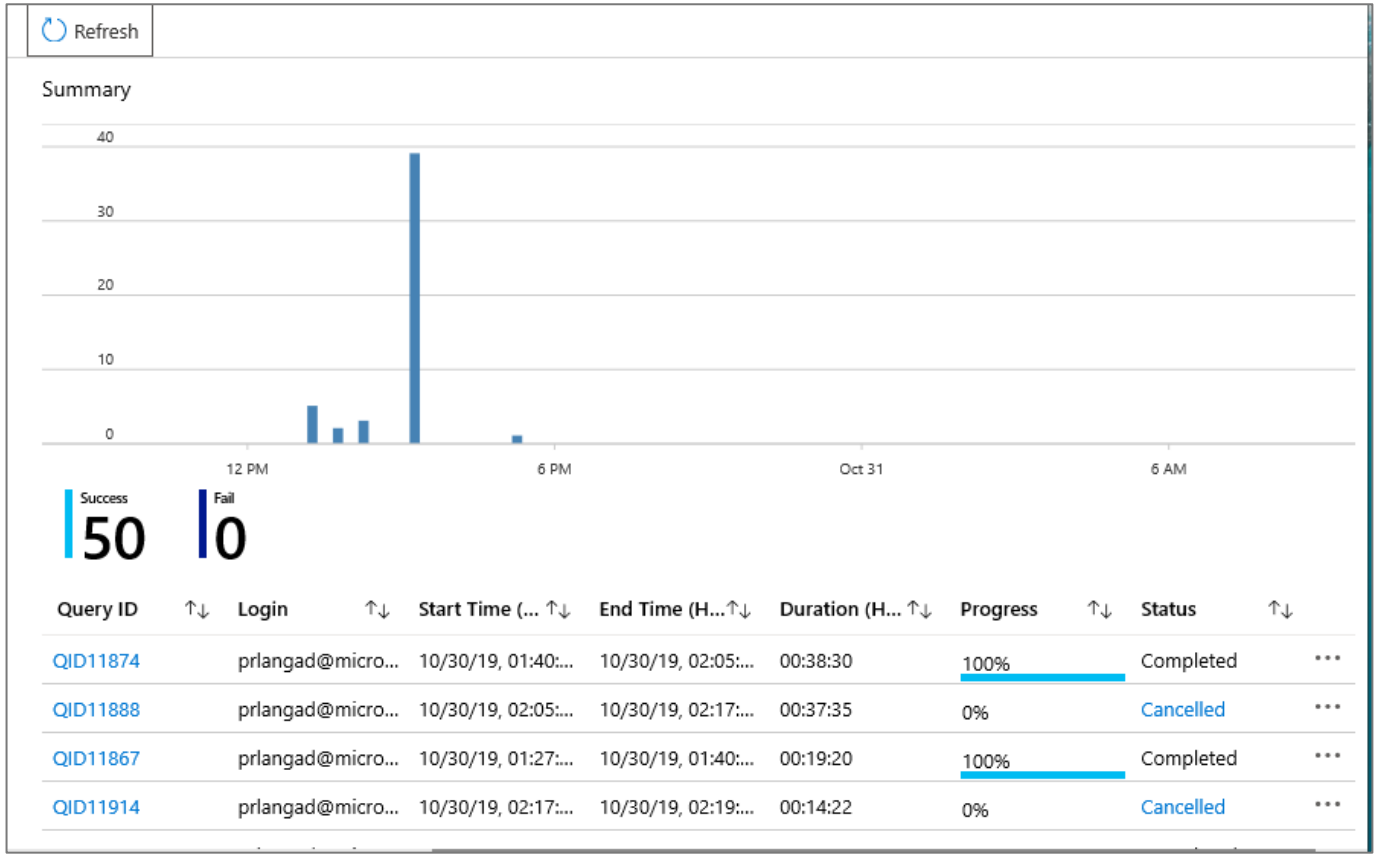
Monitor SQL Pools

Overview

Monitor SQL Pool in Azure Portal for overall usage and query activities.

Benefits

- Access SQL Audit Logs for my SQL computes
- Monitor status and progress of all/specific activities
- Dashboard view to monitor performance
- Get to know scale of SQL compute resource



Manage

Workload Management

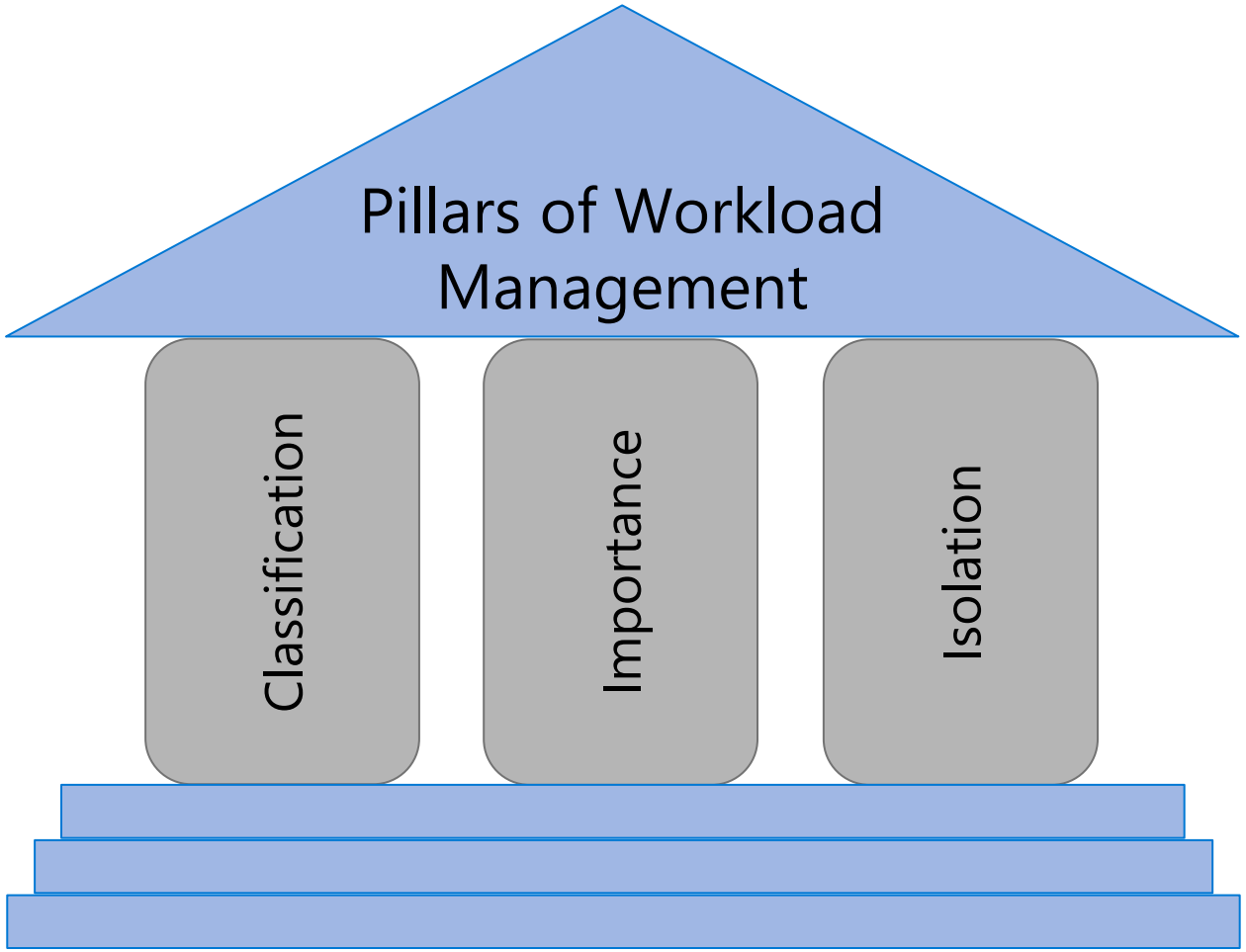
Overview

It manages resources, ensures highly efficient resource utilization, and maximizes return on investment (ROI).

Synapse is moving away from Resource Class and Concurrency Slots to Workload Management.

The three pillars of workload management are

- Workload Classification – To assign a request to a workload group and setting importance levels.
- Workload Importance – To influence the order in which a request gets access to resources.
- Workload Isolation – To reserve resources for a workload group.



Workload classification

Overview

Map queries to allocations of resources via pre-determined rules.

Use with workload importance to effectively share resources across different workload types.

If a query request is not matched to a classifier, it is assigned to the default workload group.

Benefits

Map queries to both Resource Management and Workload Isolation concepts.

Monitoring DMVs

sys.workload_management_workload_classifiers

sys.workload_management_workload_classifier_details

Query DMVs to view details about all active workload classifiers.

```
CREATE WORKLOAD CLASSIFIER classifier_name
WITH
(
      WORKLOAD_GROUP = 'name'
    , MEMBERNAME     = 'security_account'
[ [ , ] IMPORTANCE     = {LOW | BELOW_NORMAL | NORMAL | ABOVE_NORMAL | HIGH} ]
[ [ , ] WLM_LABEL      = 'label' ]
[ [ , ] WLM_CONTEXT    = 'name' ]
[ [ , ] START_TIME     = 'start_time' ]
[ [ , ] END_TIME       = 'end_time' ]
)[ ; ]
```

WORKLOAD_GROUP*: maps to an existing resource class*
IMPORTANCE*: specifies relative importance of request*
MEMBERNAME*: database user, role, AAD login or AAD group*

Workload importance

Overview

Queries past the concurrency limit enter a FiFo queue

By default, queries are released from the queue on a first-in, first-out basis as resources become available

Workload importance allows higher priority queries to receive resources immediately regardless of queue

Example Video

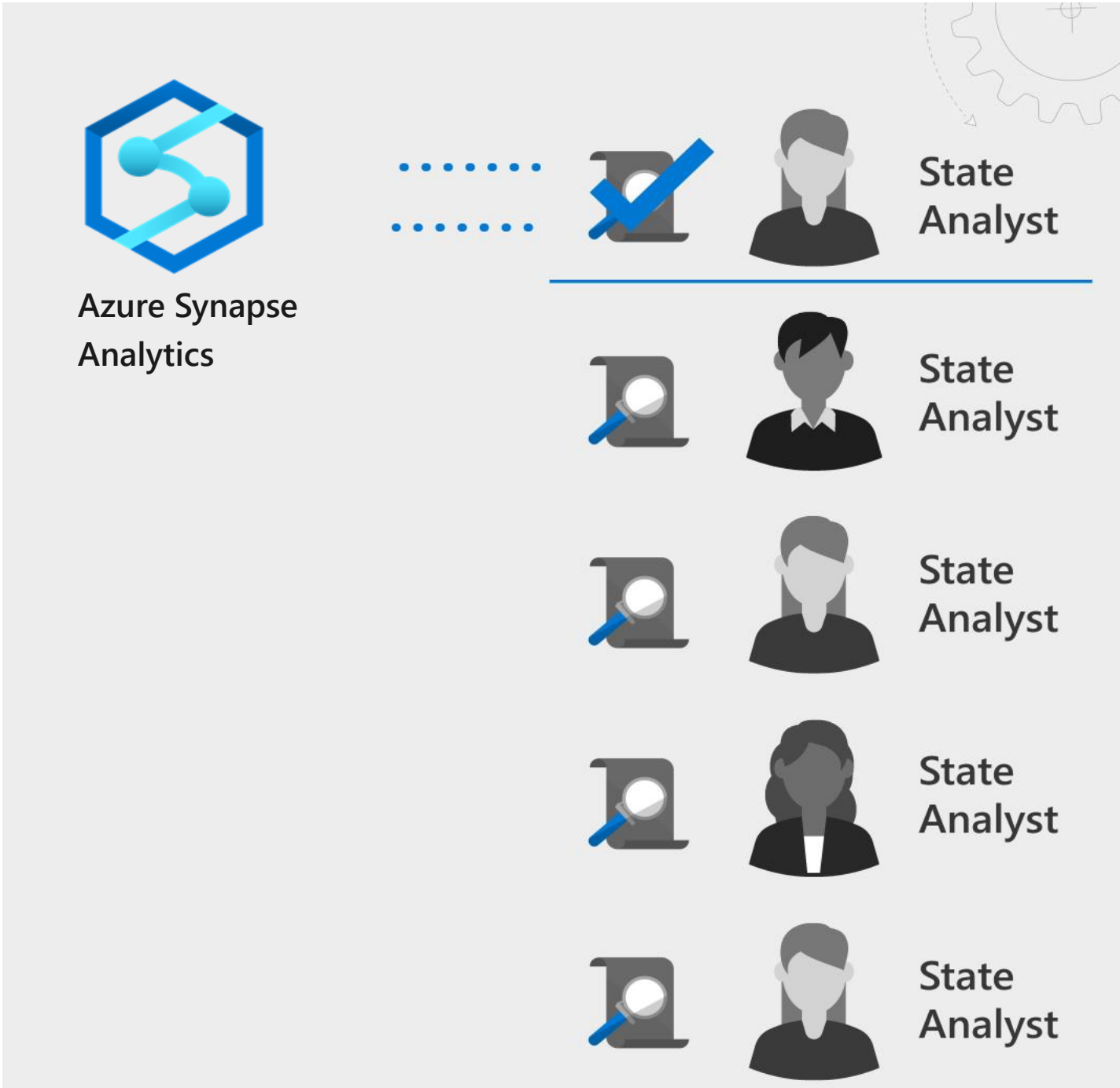
State analysts have normal importance.

National analyst is assigned high importance.

State analyst queries execute in order of arrival

When the national analyst's query arrives, it jumps to the top of the queue

```
CREATE WORKLOAD CLASSIFIER National_Analyst
WITH
(
  WORKLOAD_GROUP = 'analyst'
  ,IMPORTANCE    = HIGH
  ,MEMBERNAME    = 'National_Analyst_Login')
```



Workload Isolation

Overview

Allocate fixed resources to workload group.

Assign maximum and minimum usage for varying resources under load. These adjustments can be done live without having to take SQL Analytics offline.

Benefits

Reserve resources for a group of requests

Limit the amount of resources a group of requests can consume

Shared resources accessed based on importance level

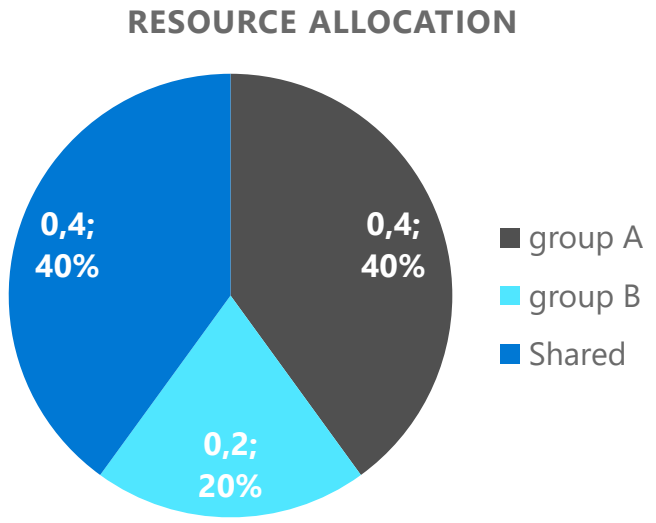
Set Query timeout value. Get DBAs out of the business of killing runaway queries

Monitoring DMVs

sys.workload_management_workload_groups

Query to view configured workload group.

```
CREATE WORKLOAD GROUP group_name
WITH
(
   MIN_PERCENTAGE_RESOURCE = value
 , CAP_PERCENTAGE_RESOURCE = value
 , REQUEST_MIN_RESOURCE_GRANT_PERCENT = value
 [ [ , ] REQUEST_MAX_RESOURCE_GRANT_PERCENT = value ]
 [ [ , ] IMPORTANCE = {LOW | BELOW_NORMAL | NORMAL | ABOVE_NORMAL | HIGH} ]
 [ [ , ] QUERY_EXECUTION_TIMEOUT_SEC = value ]
)[ ; ]
```



Dynamic Management Views (DMVs)

Overview

Dynamic Management Views (DMV) are queries that return information about model objects, server operations, and server health.

Benefits:

Simple SQL syntax

Returns result in table format

Easier to read and copy result

Azure Advisor recommendations

Suboptimal Table Distribution

Reduce data movement by replicating tables

Data Skew

Choose new hash-distribution key

Slowest distribution limits performance

Cache Misses

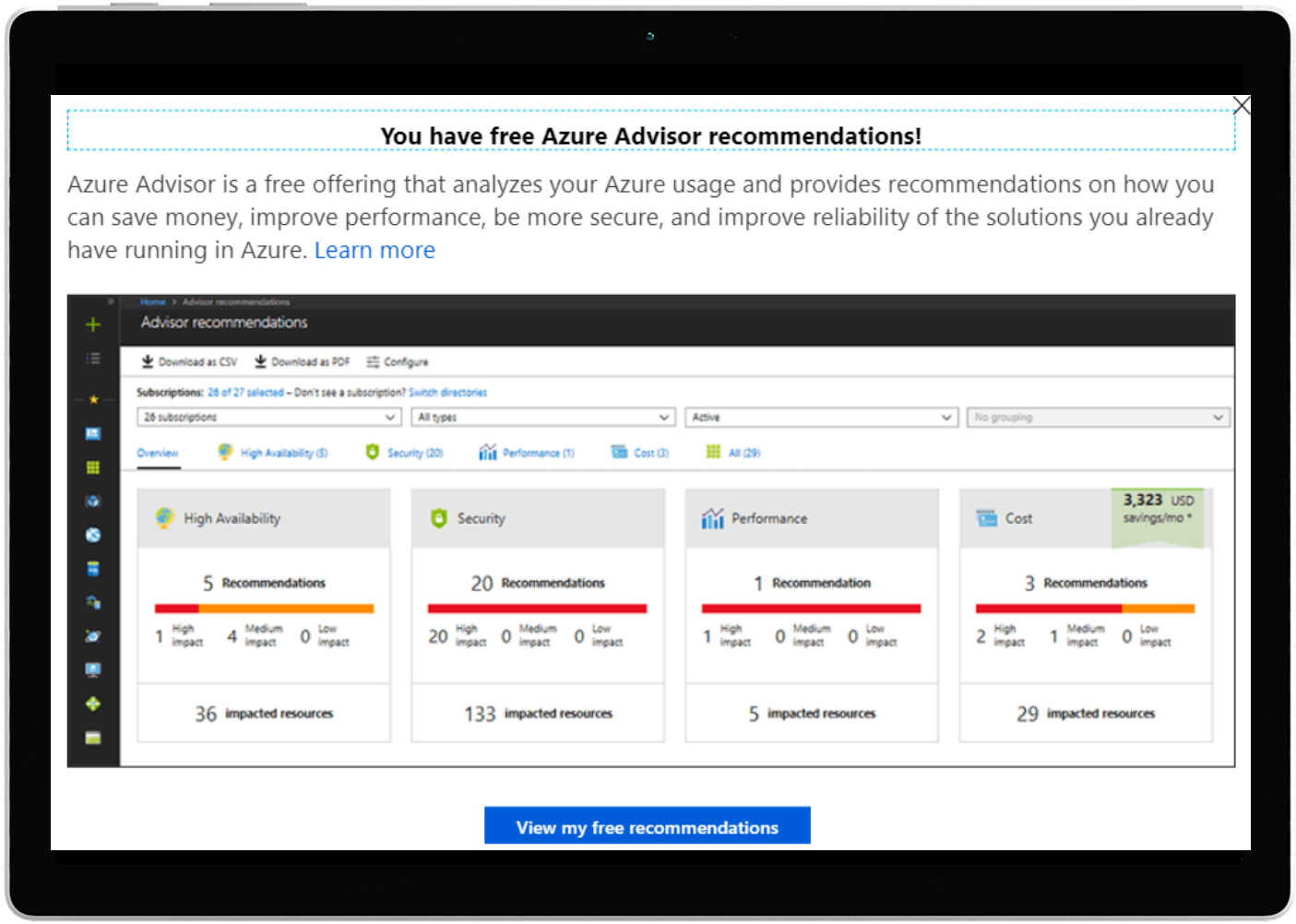
Provision additional capacity

Tempdb Contention

Scale or update user resource class

Suboptimal Plan Selection

Create or update table statistics



Maintenance windows

Overview

Choose a time window for your upgrades.

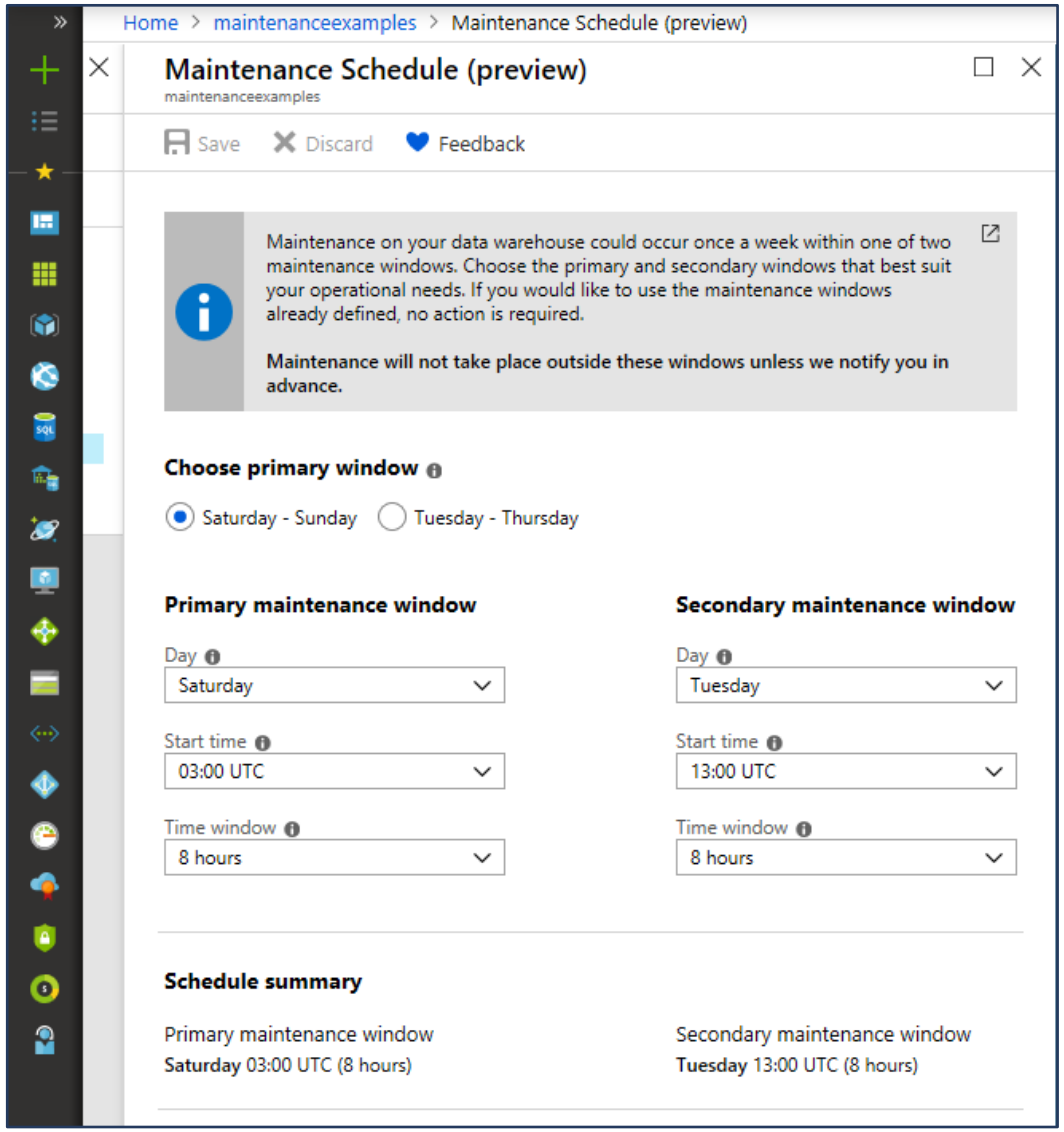
Select a primary and secondary window within a seven-day period.

Windows can be from 3 to 8 hours.

24-hour advance notification for maintenance events.

Benefits

- Ensure upgrades happen on your schedule.
- Predictable planning for long-running jobs.
- Stay informed of start and end of maintenance.



Manage Summary

- Resource contention can be mitigated by applying Workload Management in Azure Synapse Analytics.
  - **Workload Classification:** Assign a request to a workload group and set importance levels.
  - **Workload Importance:** Influence the order in which a request gets access to resources.
  - **Workload Isolation:** Reserve resources for a workload group.

- Use Azure Advisor recommendations to identify suboptimal table distribution, data skew, cache misses, tempdb contention, and suboptimal plan selection.

- Avoid disruptive system downtime due to system upgrades by configuring maintenance windows and notifications in Azure Synapse Analytics.

SQL Monitor with DMVs

Overview

Offers monitoring of

-all open, closed sessions

-count sessions by user

-count completed queries by user

-all active, complete queries

-longest running queries

-memory consumption

Count sessions by user

```
--count sessions by user
SELECT login_name, COUNT(*) as session_count FROM sys.dm_pdw_exec_sessions
where status = 'Closed' and session_id <> session_id() GROUP BY login_name;
```

List all open sessions

```
-- List all open sessions
SELECT * FROM sys.dm_pdw_exec_sessions where status <> 'Closed' and session_id <>
session_id();
```

List all active queries

```
-- List all active queries
SELECT * FROM sys.dm_pdw_exec_requests WHERE status not in
('Completed','Failed','Cancelled') AND session_id <> session_id() ORDER BY submit_time
DESC;
```

Azure Monitor

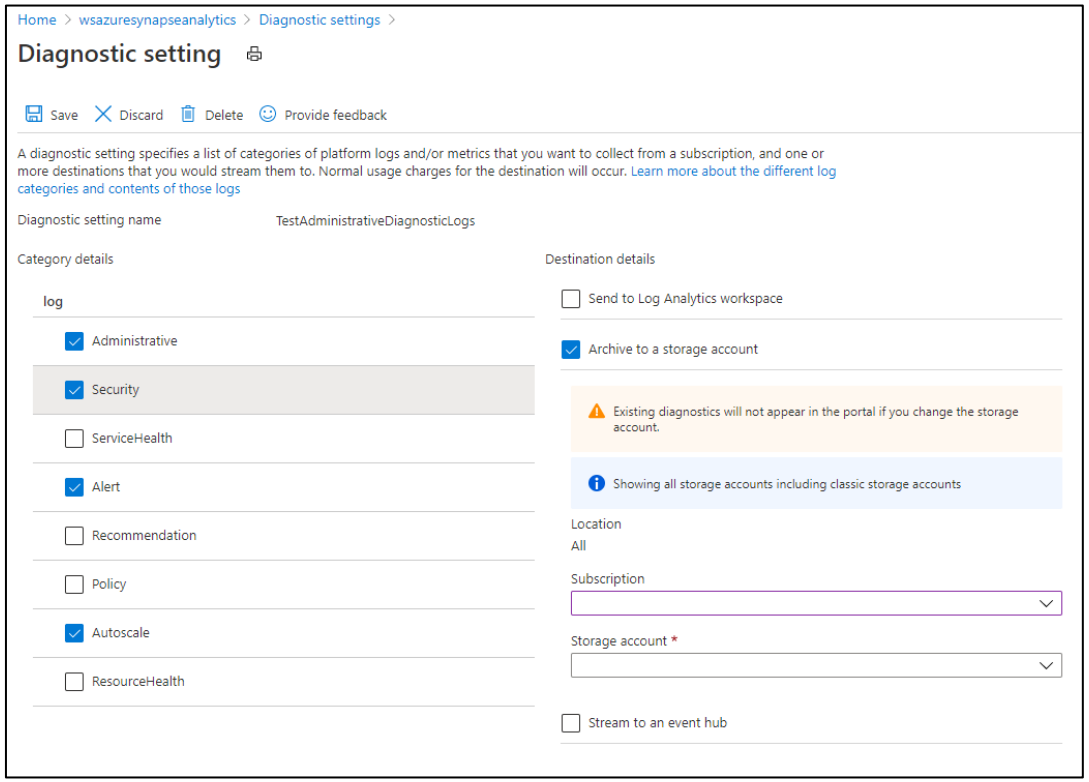
Overview

Leverage Azure Monitor to obtain metrics, alerts, and logs.

Azure Synapse Analytics can write diagnostic logs in Azure Monitor.

Workspace level metrics, dedicated SQL pool level metrics, Apache Spark pool metrics are supported.

With Azure Monitor diagnostic settings, you can route diagnostic logs for analysis to multiple different targets.



With Azure Monitor diagnostic settings, you can route diagnostic logs for analysis to multiple different targets

- Storage Account
- Log Analytics workspace
- Event Hub

End