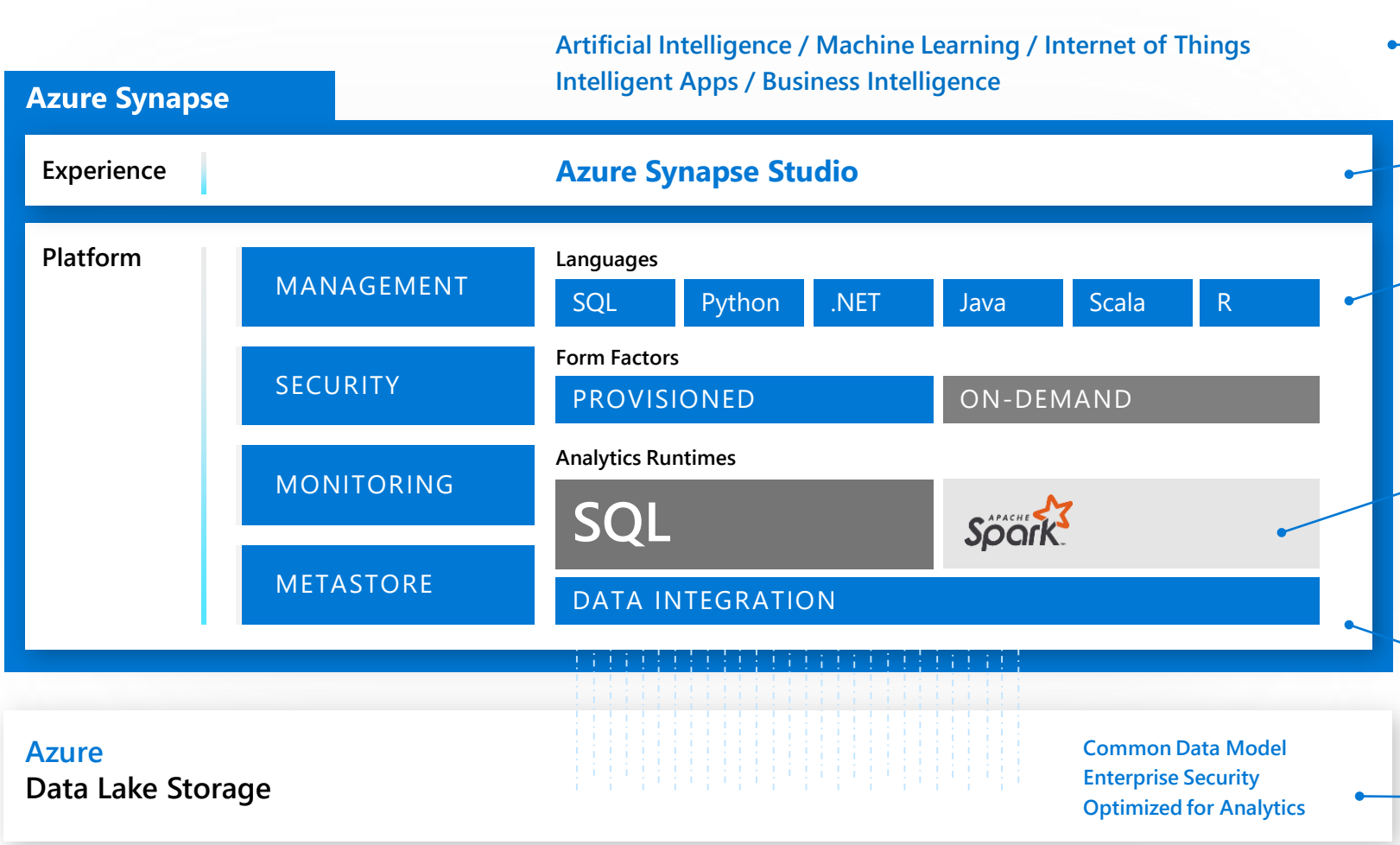
Microsoft

Azure Synapse Analytics

Azure Synapse Analytics

Limitless analytics service with unmatched time to insight



Designed for analytics **workloads at any scale**

SaaS **developer experiences** for code free and code first

Multiple **languages** suited to different analytics workloads

Integrated analytics runtimes available provisioned and serverless on-demand

SQL Analytics offering T-SQL for batch, streaming and interactive processing

Spark for big data processing with Python, Scala, R and .NET

Integrated **platform services** for, management, security, monitoring, and metastore

Data **lake integrated** and Common Data Model aware

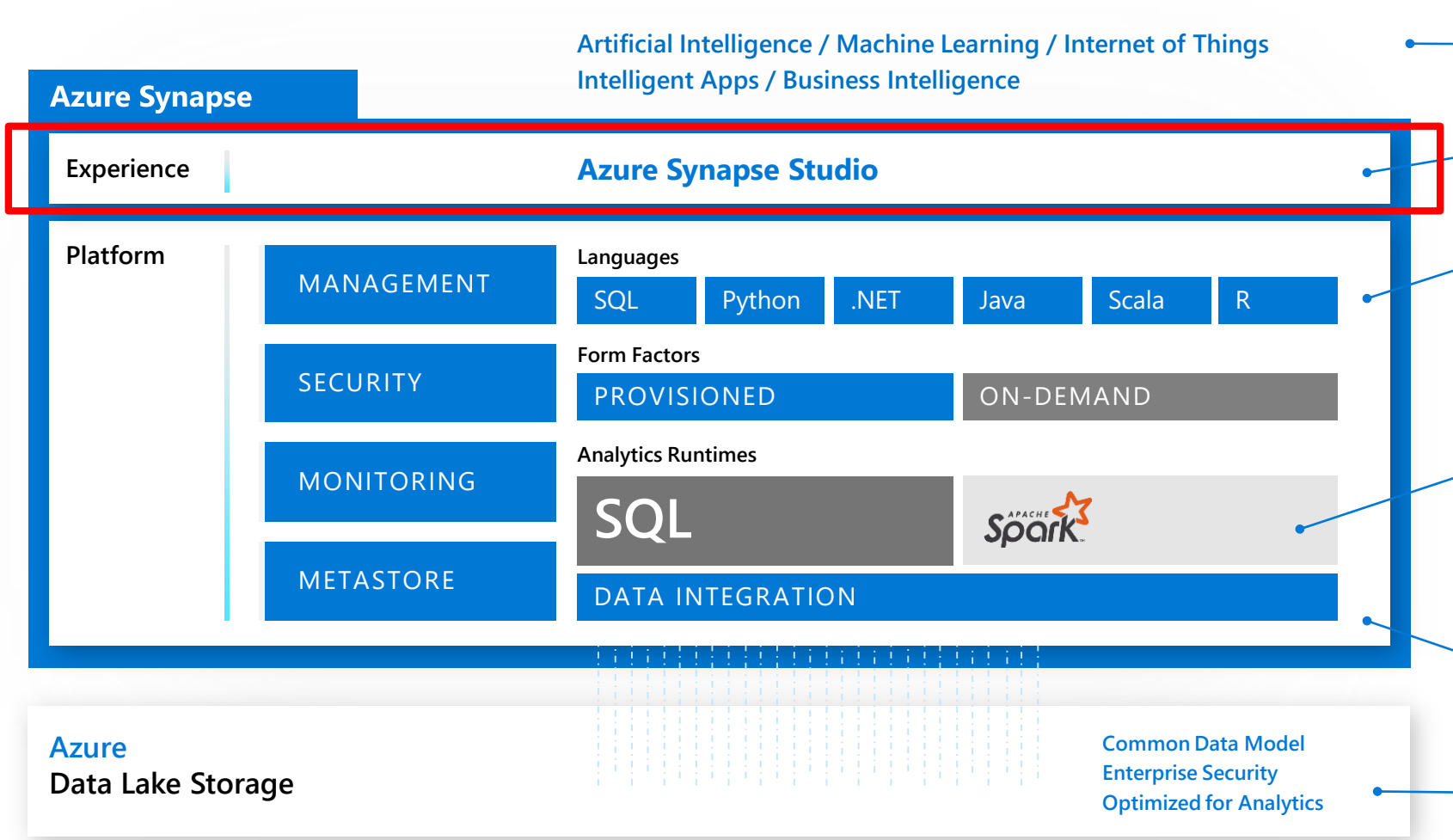
Microsoft

Azure Synapse Analytics

Studio

Azure Synapse Analytics

Limitless analytics service with unmatched time to insight



Designed for analytics **workloads at any scale**

SaaS **developer experiences** for code free and code first

Multiple **languages** suited to different analytics workloads

Integrated analytics runtimes available provisioned and serverless on-demand

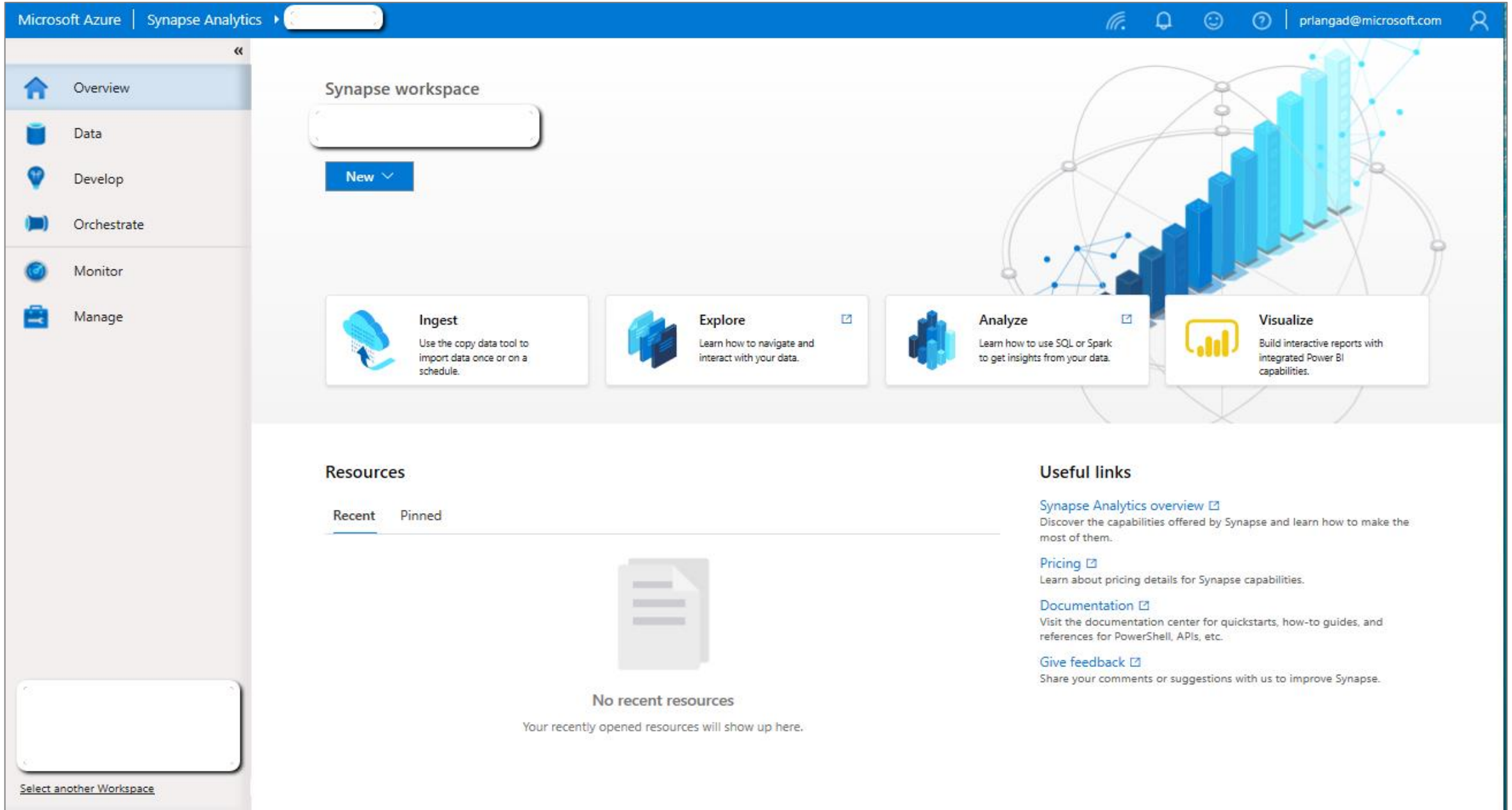
SQL Analytics offering T-SQL for batch, streaming and interactive processing

Spark for big data processing with Python, Scala, R and .NET

Integrated **platform services** for, management, security, monitoring, and metastore

Data **lake integrated** and Common Data Model aware

Studio


Microsoft Azure
Synapse Analytics
prlangad@microsoft.com
Overview
Data
Develop
Orchestrate
Monitor
Manage
Synapse workspace
New
Ingest
Use the copy data tool to import data once or on a schedule.
Explore
Learn how to navigate and interact with your data.
Analyze
Learn how to use SQL or Spark to get insights from your data.
Visualize
Build interactive reports with integrated Power BI capabilities.
Resources
Recent
Pinned
No recent resources
Your recently opened resources will show up here.
Useful links
Synapse Analytics overview
Discover the capabilities offered by Synapse and learn how to make the most of them.
Pricing
Learn about pricing details for Synapse capabilities.
Documentation
Visit the documentation center for quickstarts, how-to guides, and references for PowerShell, APIs, etc.
Give feedback
Share your comments or suggestions with us to improve Synapse.
Select another Workspace

Develop Hub

Overview

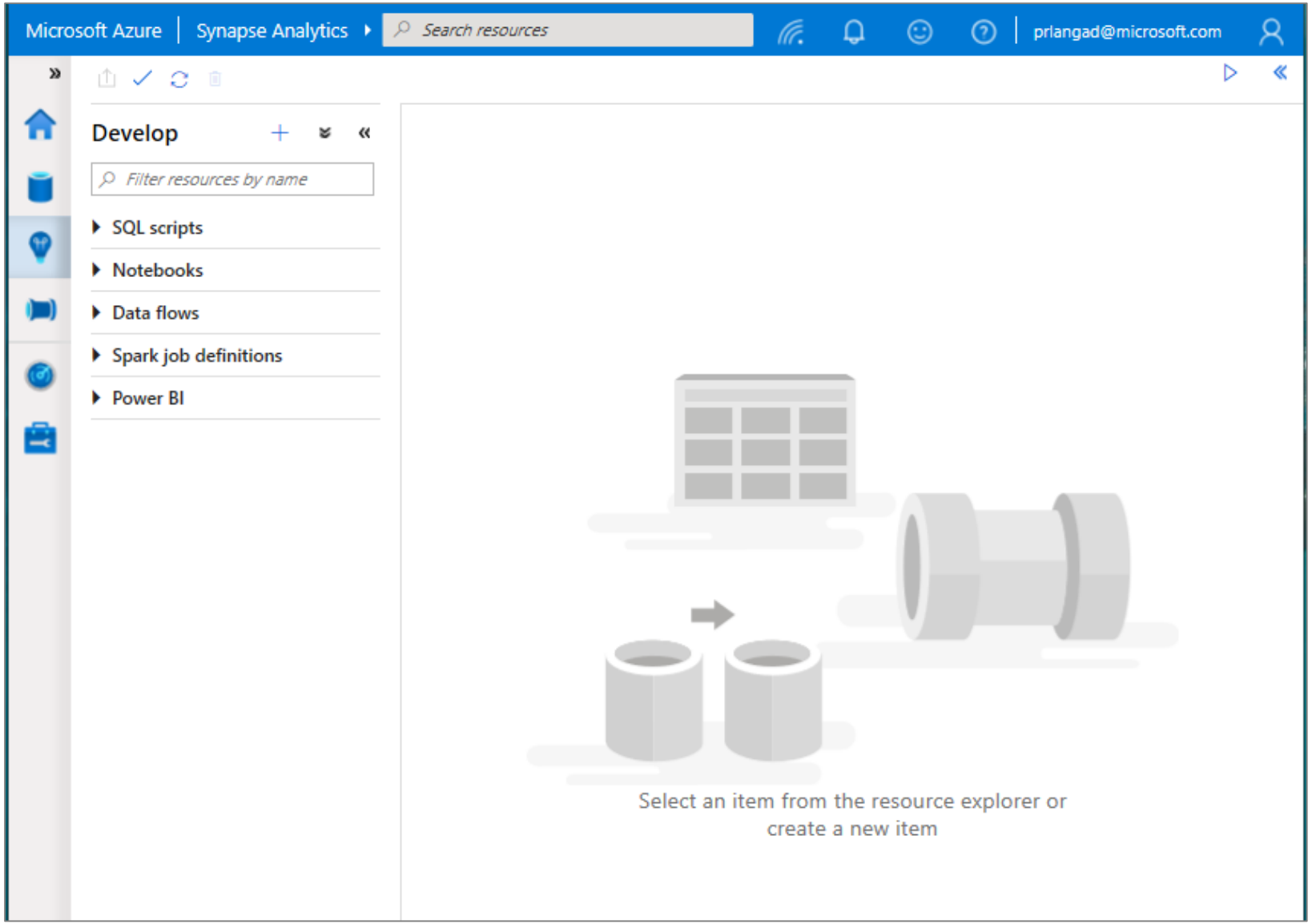
It provides development experience to query, analyze, model data

Benefits

Multiple languages to analyze data under one umbrella

Switch over notebooks and scripts without loosing content

Code intellisense offers reliable code development



Develop Hub/SQL scripts

SQL Script

Authoring SQL Scripts

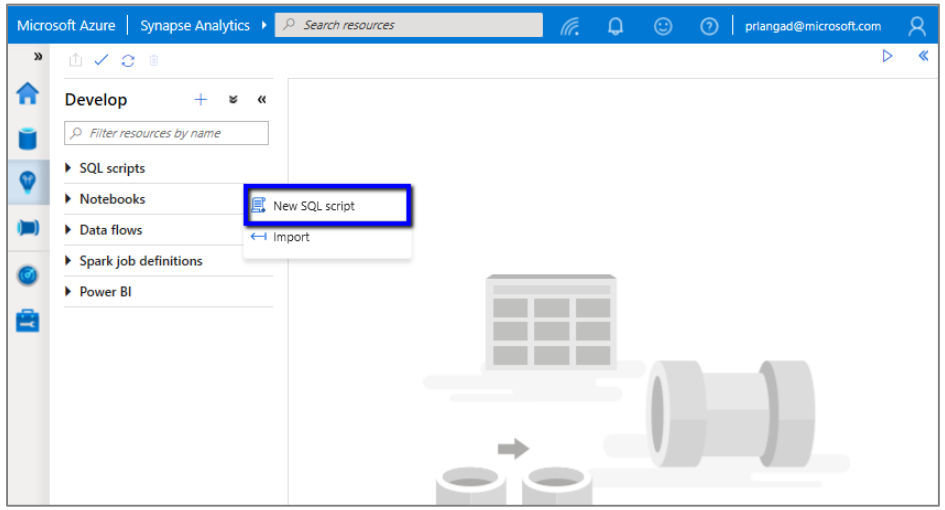
Execute SQL script on provisioned SQL Pool or SQL On-demand

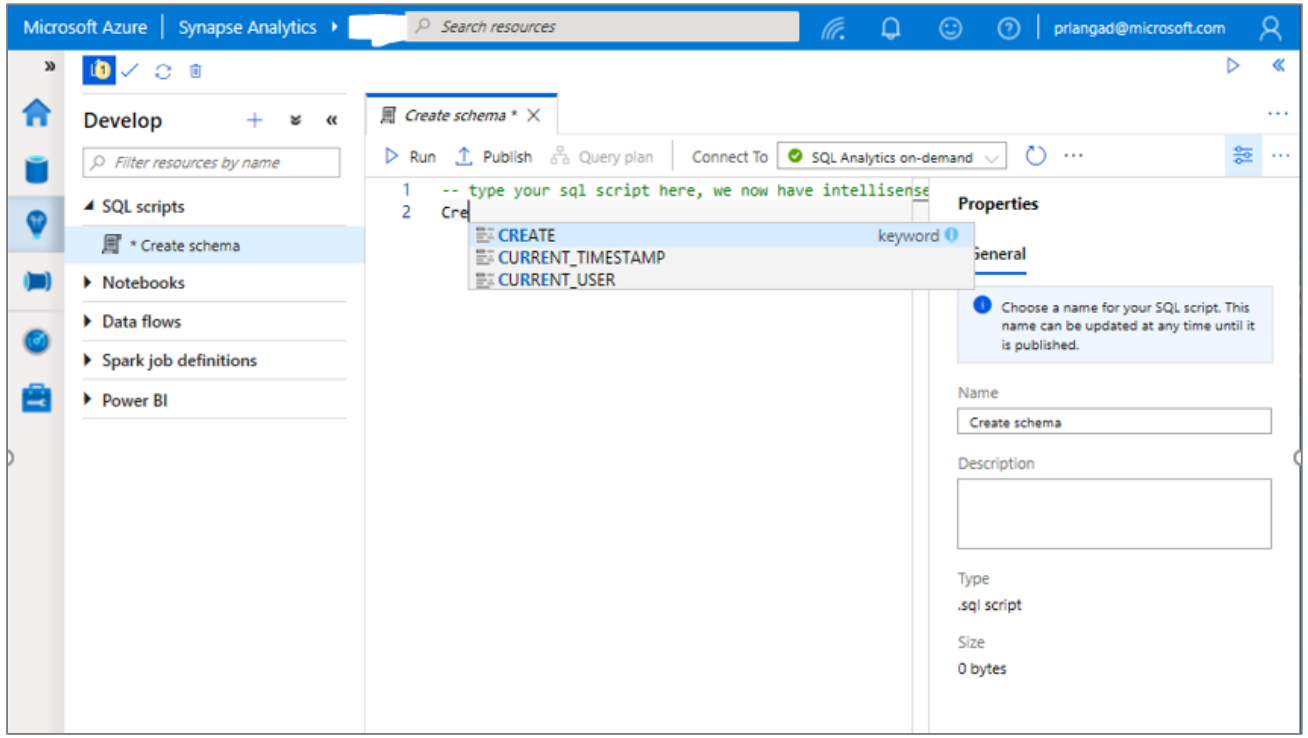
Publish individual SQL script or multiple SQL scripts through Publish all feature

Language support and intellisense

View result in tabular or chart format

Export result





Develop Hub/Notebooks

Notebooks

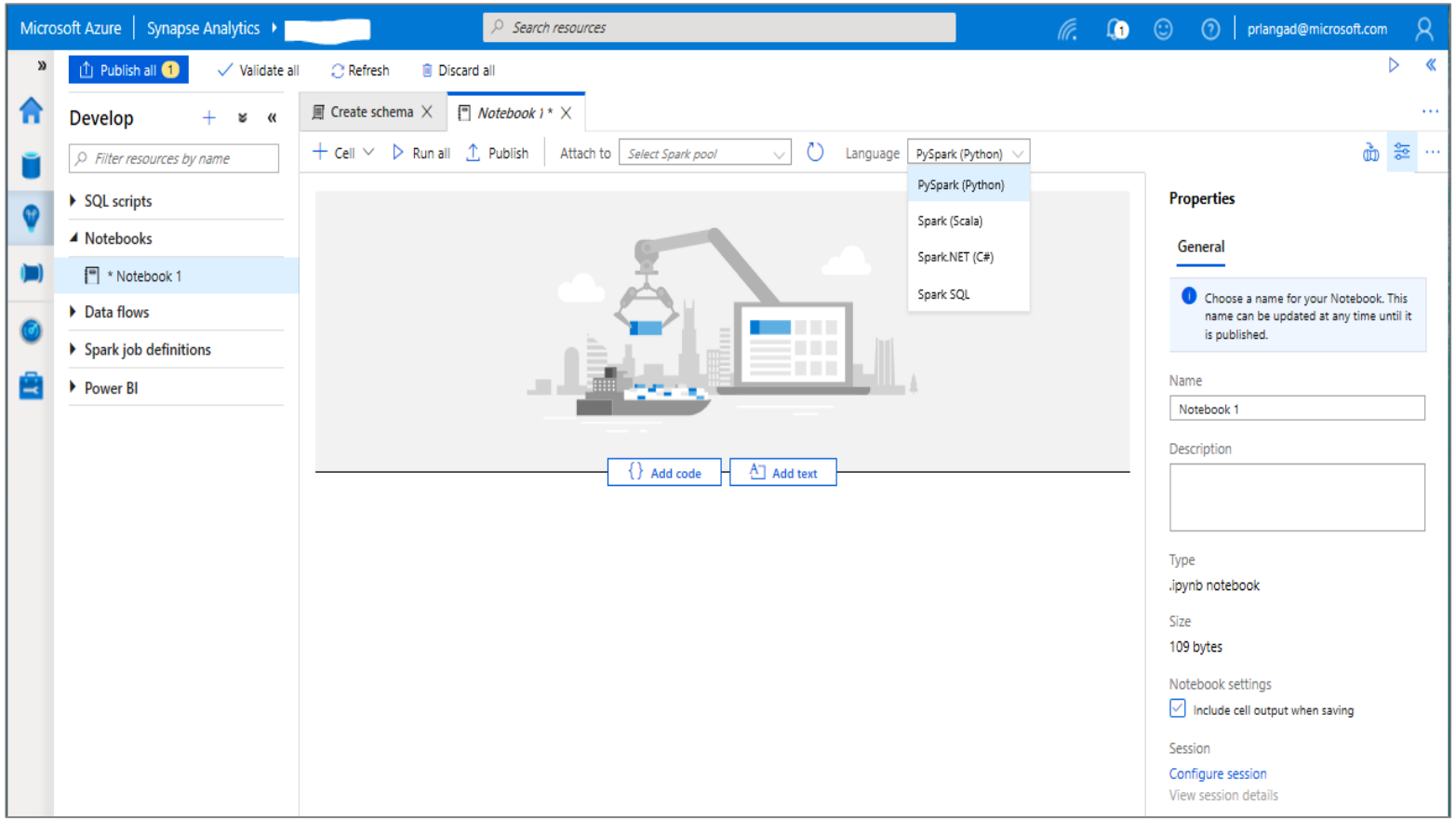
Allows to write multiple languages in one notebook

%%<Name of language>

Offers use of temporary tables across languages

Language support for Syntax highlight, syntax error, syntax code completion, smart indent, code folding

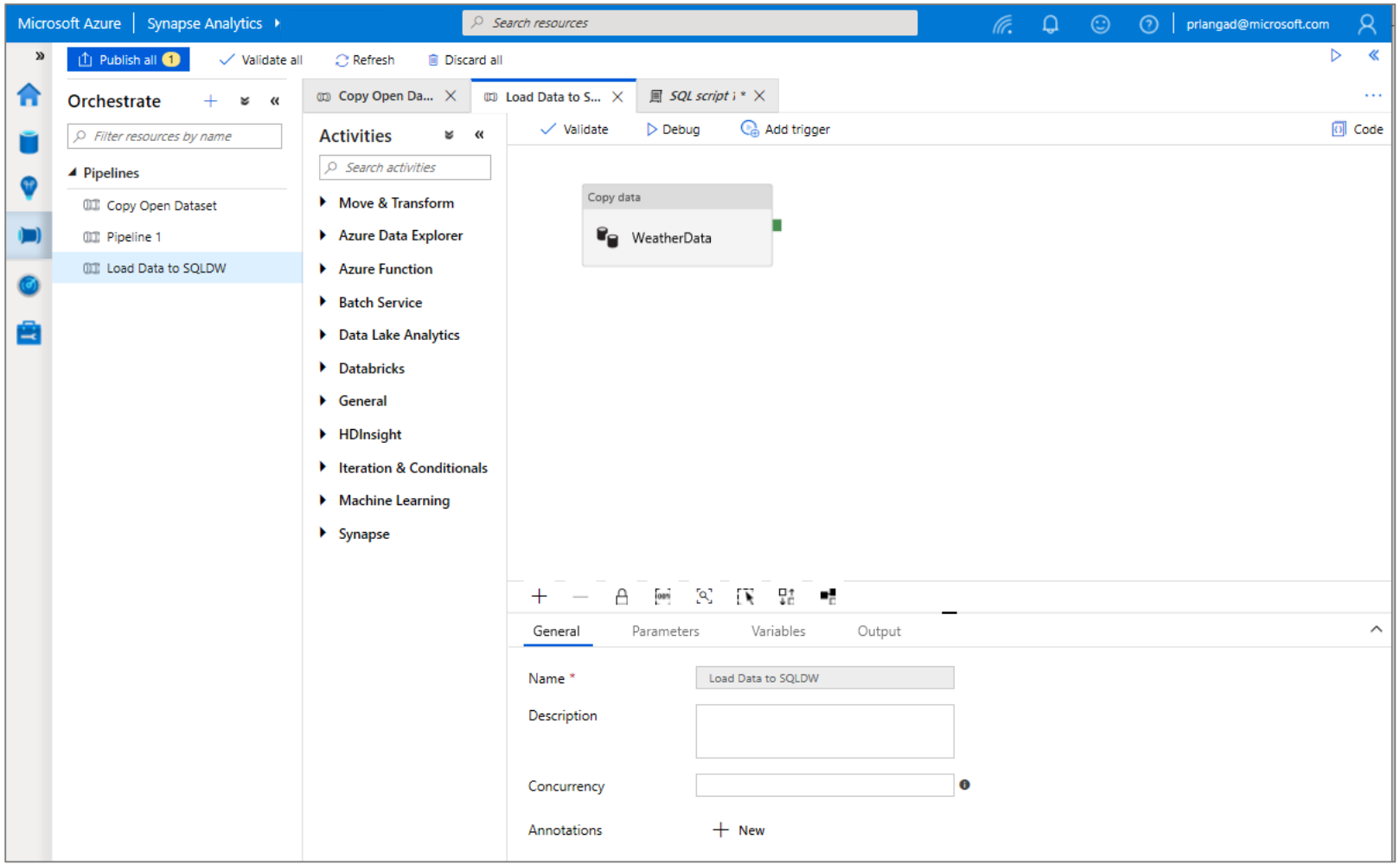
Export results



Orchestrate Hub

Overview

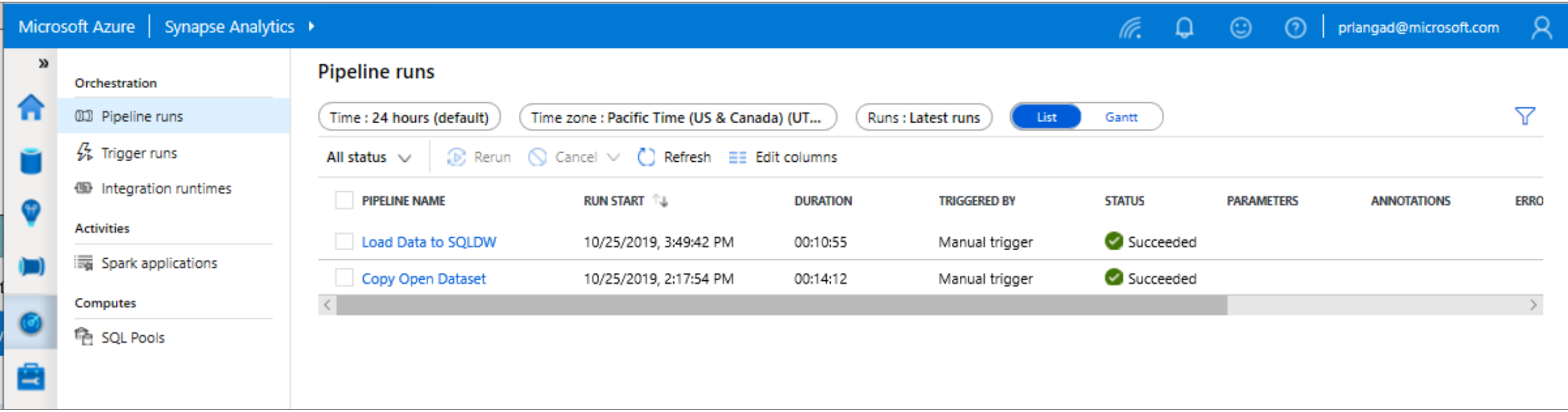
It provides ability to create pipelines to ingest, transform and load data with 80+ inbuilt connectors



Monitor Hub

Overview

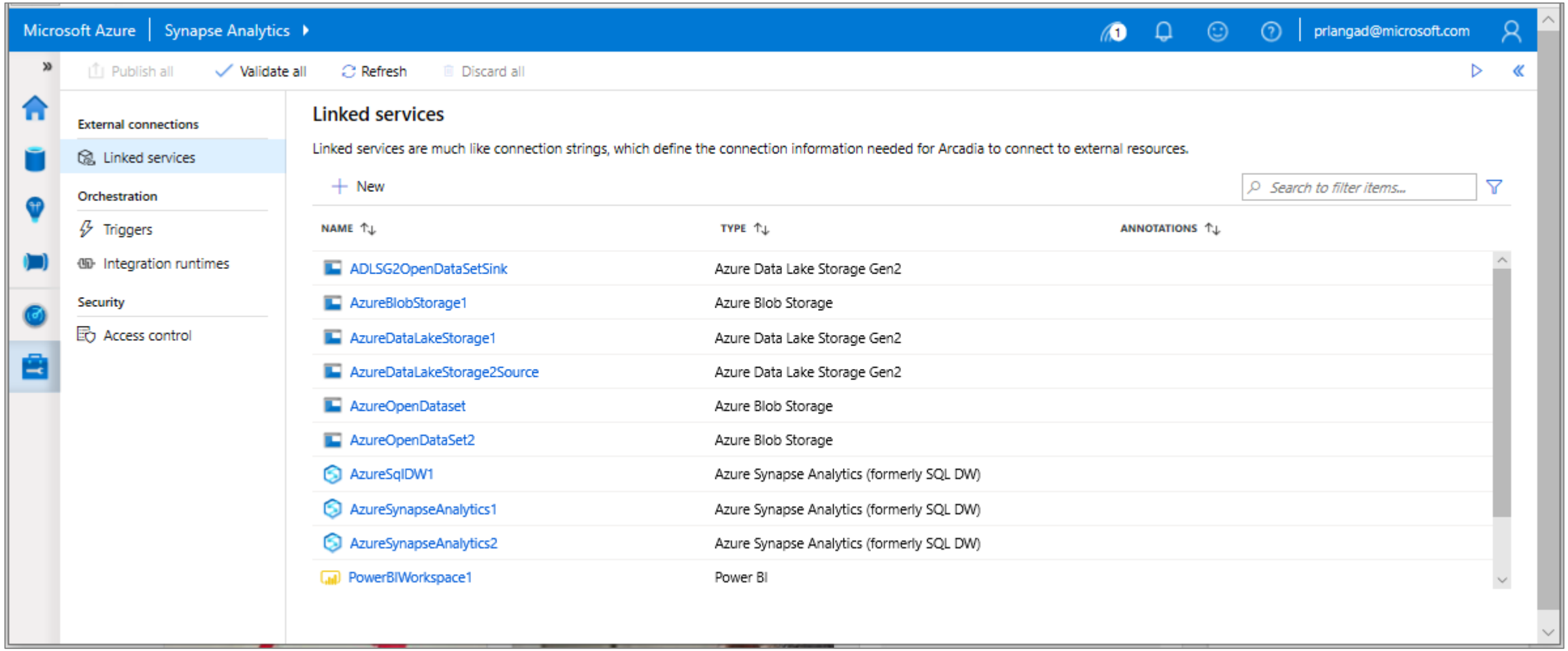
This feature provides ability to monitor orchestration, activities and compute resources.



Manage Hub

Overview

This feature provides ability to manage Linked Services, Orchestration and Security.



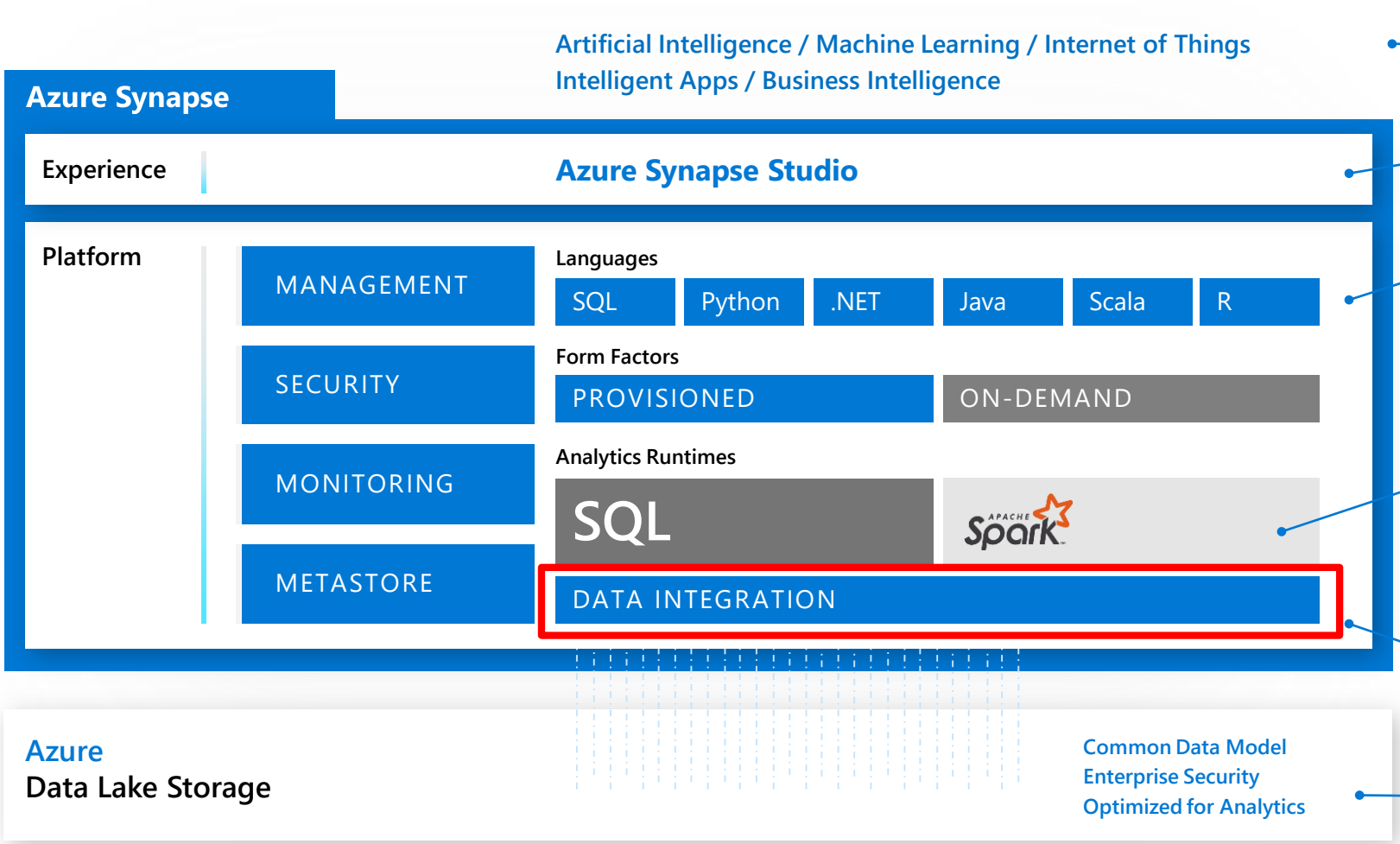
Microsoft

Azure Synapse Analytics

Data Integration

Azure Synapse Analytics

Limitless analytics service with unmatched time to insight



Designed for analytics **workloads at any scale**

SaaS **developer experiences** for code free and code first

Multiple **languages** suited to different analytics workloads

Integrated analytics runtimes available provisioned and serverless on-demand

SQL Analytics offering T-SQL for batch, streaming and interactive processing

Spark for big data processing with Python, Scala, R and .NET

Integrated **platform services** for, management, security, monitoring, and metastore

Data **lake integrated** and Common Data Model aware

Manage – Linked Services

Overview

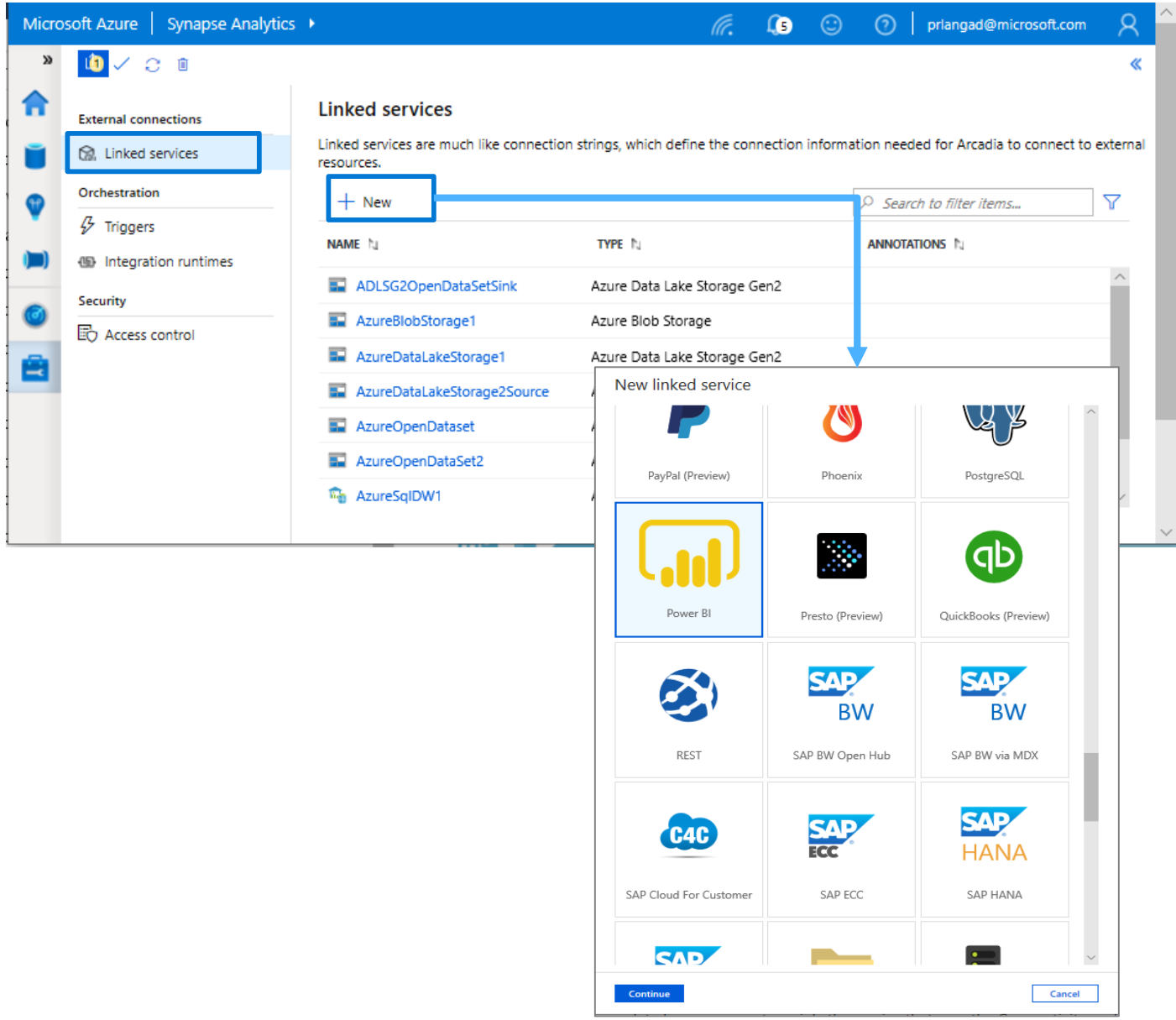
It defines the connection information needed for Pipeline to connect to external resources.

Benefits

Offers pre-build 85+ connectors

Easy cross platform data migration

Represents data store or compute resources



Manage – Integration runtimes

Overview

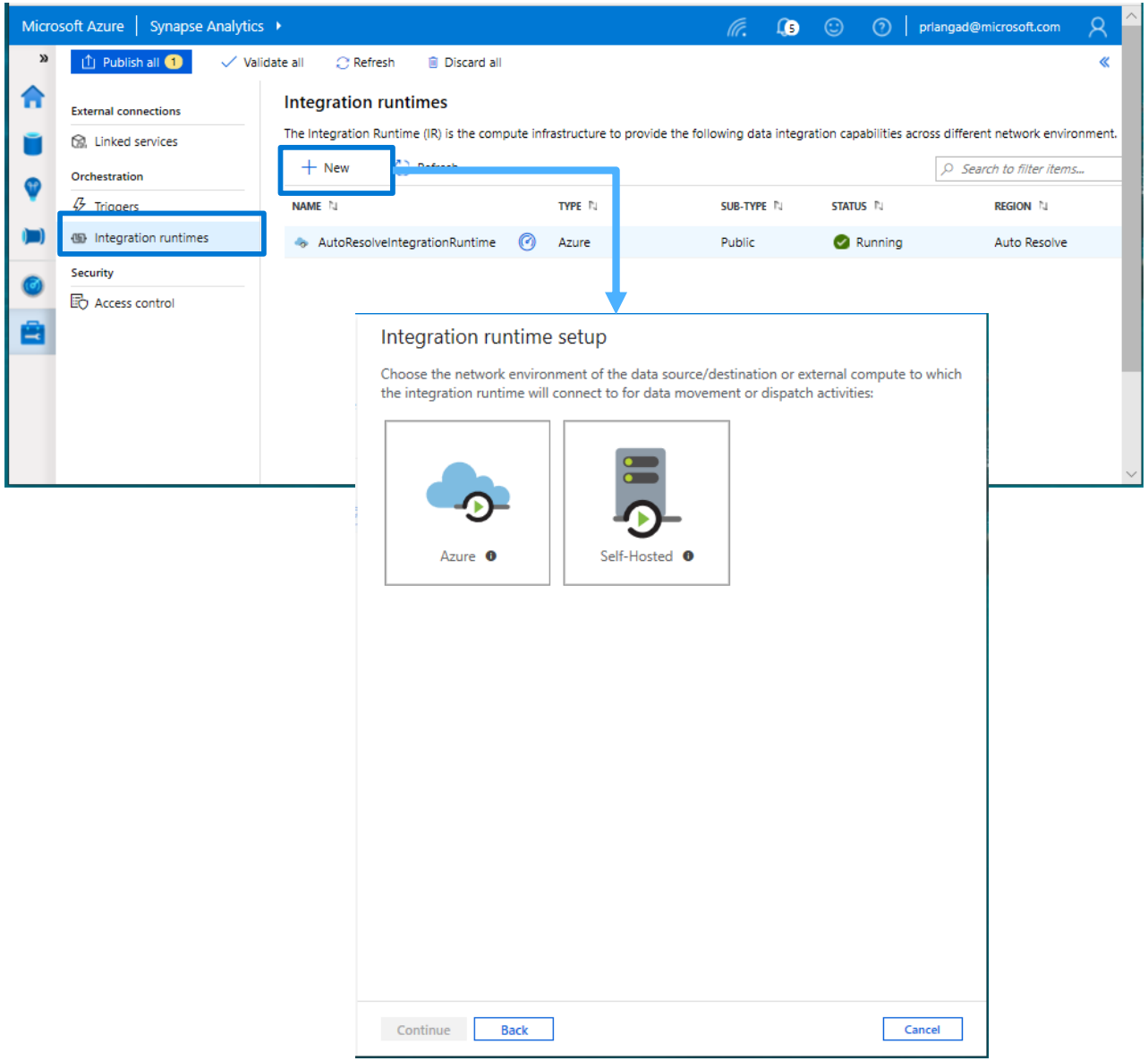
It is the compute infrastructure used by Pipelines to provide the data integration capabilities across different network environments. An integration runtime provides the bridge between the activity and linked Services.

Benefits

Offers Azure Integration Runtime or Self-Hosted Integration Runtime

Azure Integration Runtime – provides fully managed, serverless compute in Azure

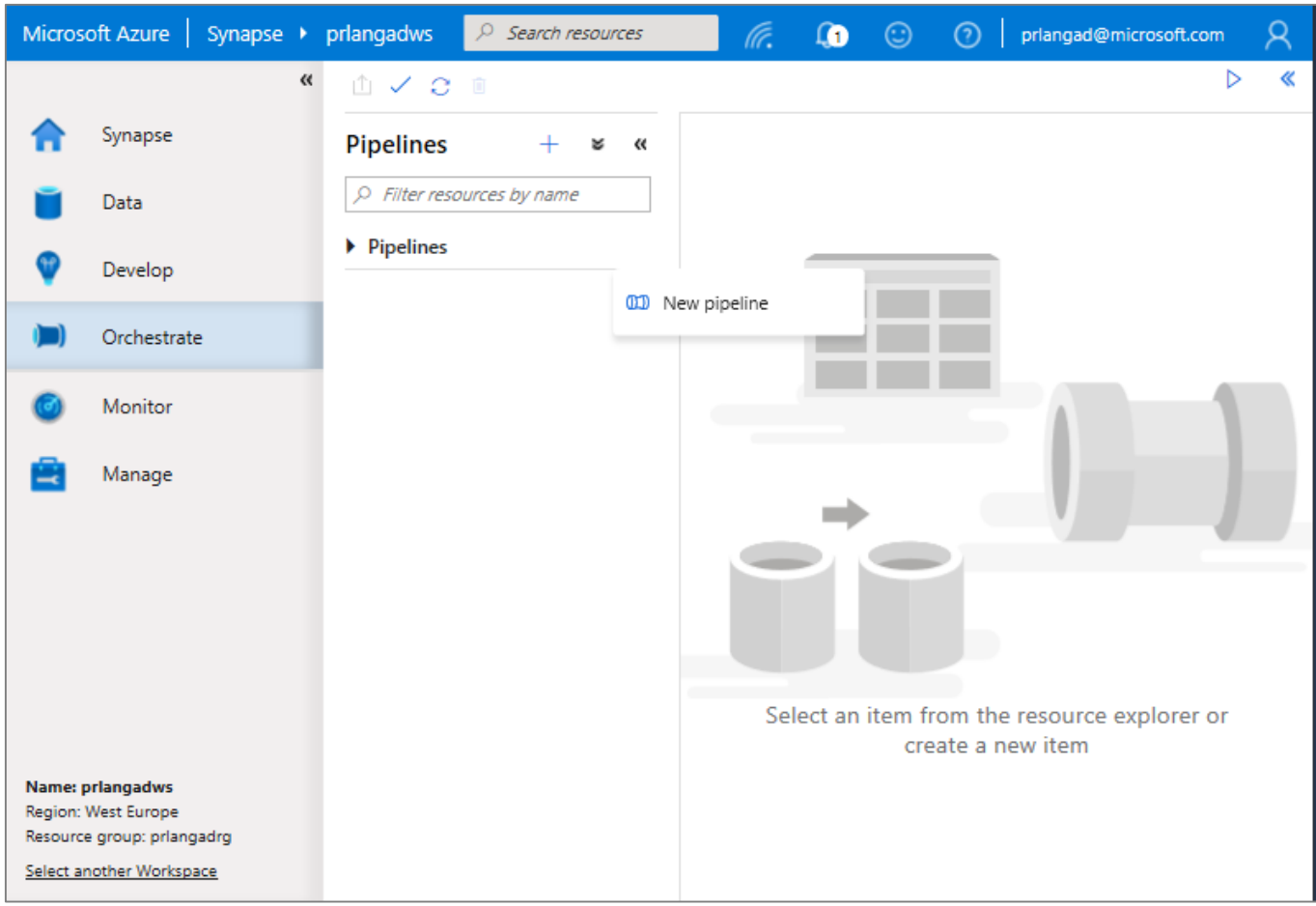
Self-Hosted Integration Runtime – use compute resources in on-premises machine or a VM inside private network



Pipelines

Overview

- Scalable – per job elasticity up to 4GB/s
- Visually author or via code (Python, Spark(Scala), Spark.NET,(C#) etc.)
- Serverless, no infrastructure to manage
- 24 points of presence worldwide
- Self-hostable Integration Runtime for hybrid movement
- Supported file formats: CSV, AVRO, ORC, Parquet, JSON



90+ Connectors out of the box

Azure (15)	Database & DW (26)		File Storage (6)	NoSQL (3)	Services and App (28)		Generic (4)
Blob storage	Amazon Redshift	Oracle	Amazon S3	Cassandra	Amazon MWS	Oracle Service Cloud	Generic HTTP
Cosmos DB - SQL API	DB2	Phoenix	File system	Couchbase	Common Data Service	PayPal	Generic OData
Cosmos DB - MongoDB API	Drill	PostgreSQL	FTP	MongoDB	Concur	QuickBooks	Generic ODBC
Data Explorer	Google BigQuery	Presto	Google Cloud Storage		Dynamics 365	Salesforce	Generic REST
Data Lake Storage Gen1	Greenplum	SAP BW Open Hub	HDFS		Dynamics AX	Salesforce Service Cloud	
Data Lake Storage Gen2	HBase	SAP BW via MDX	SFTP		Dynamics CRM	Salesforce Marketing Cloud	
Database for MariaDB	Hive	SAP HANA			Google AdWords	SAP Cloud for Customer (C4C)	
Database for MySQL	Apache Impala	SAP table			HubSpot	SAP ECC	
Database for PostgreSQL	Informix	Spark			Jira	ServiceNow	
File Storage	MariaDB	SQL Server			Magento	Shopify	
SQL Database	Microsoft Access	Sybase			Marketo	Square	
SQL Database MI	MySQL	Teradata			Office 365	Web table	
SQL Data Warehouse	Netezza	Vertica			Oracle Eloqua	Xero	
Search index					Oracle Responsys	Zoho	
Table storage							

Prep & Transform Data

Overview

It offers data cleansing, transformation, aggregation, conversion, etc

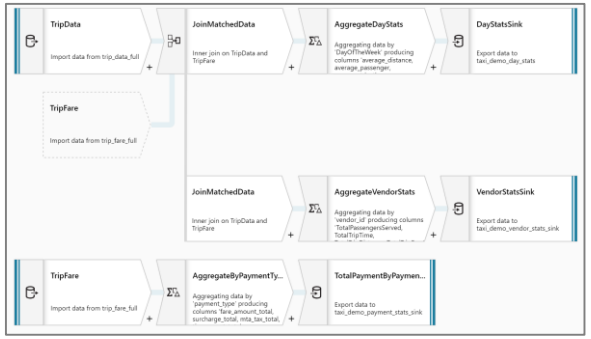
Benefits

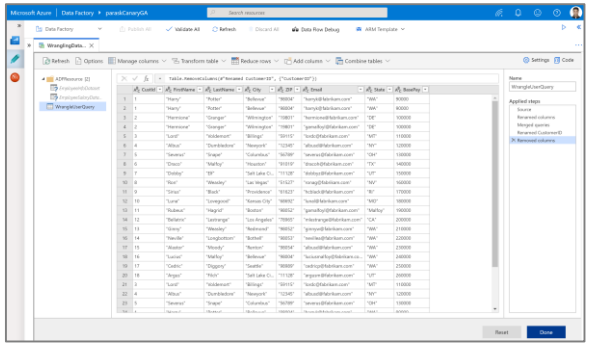
Cloud scale via Spark execution

Guided experience to easily build resilient data flows

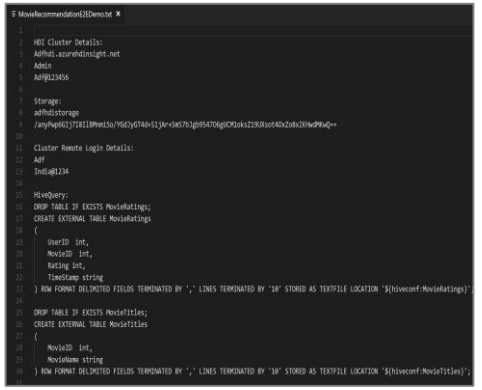
Flexibility to transform data per user's comfort

Monitor and manage dataflows from a single pane of glass





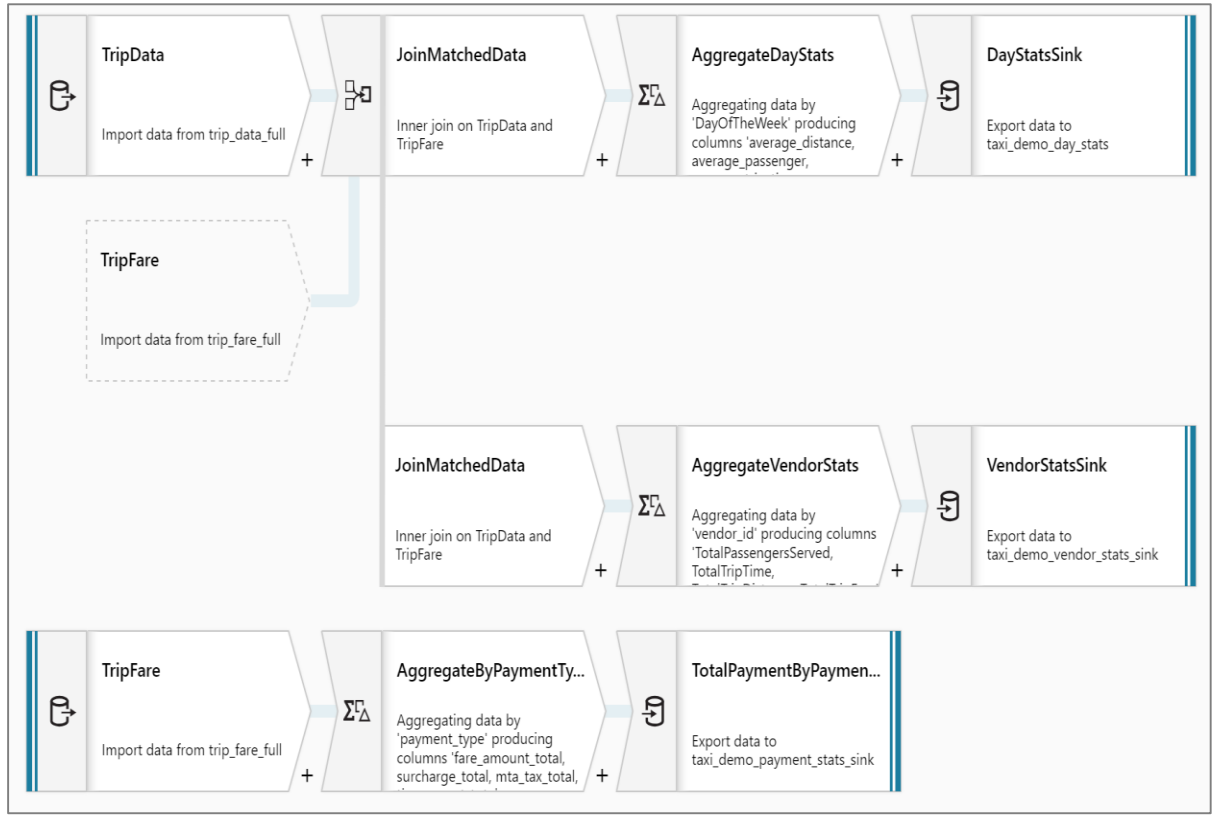
...
not



Prep & Transform Data

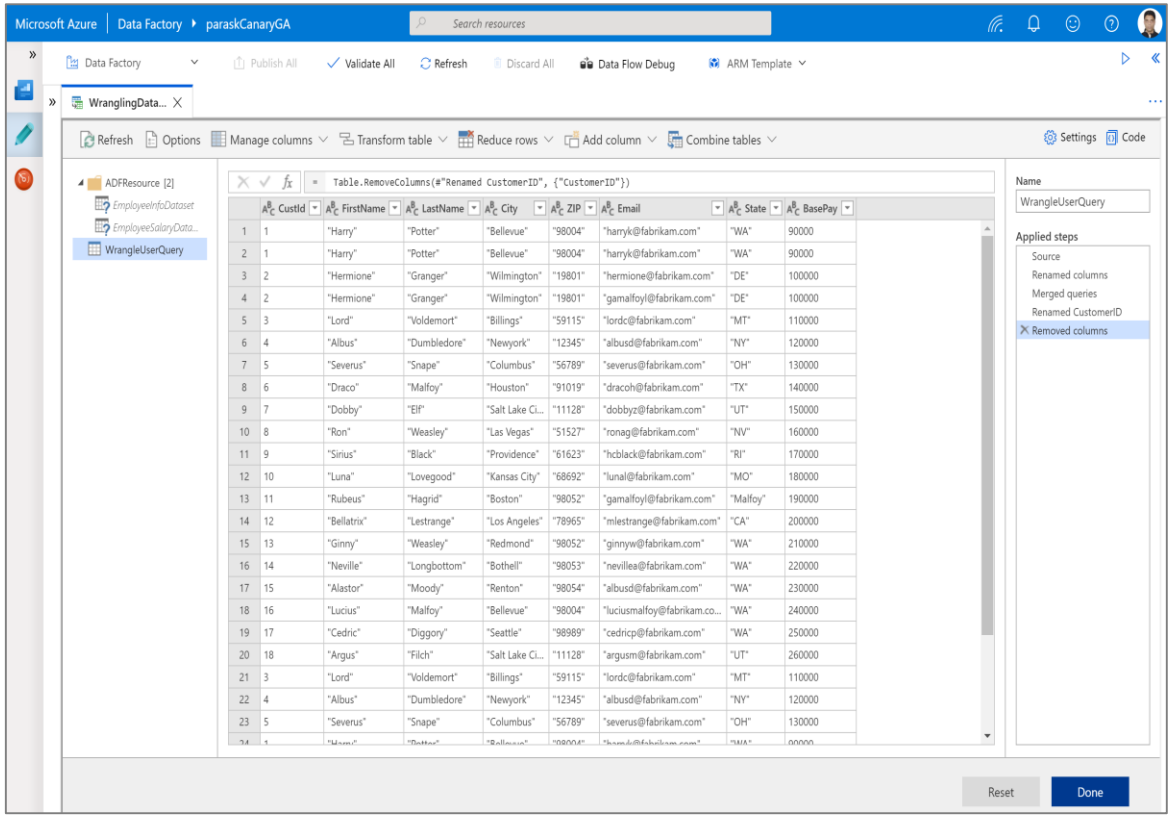
Mapping Dataflow

Code free data transformation @scale



Wrangling Dataflow

Code free data preparation @scale



Load Data

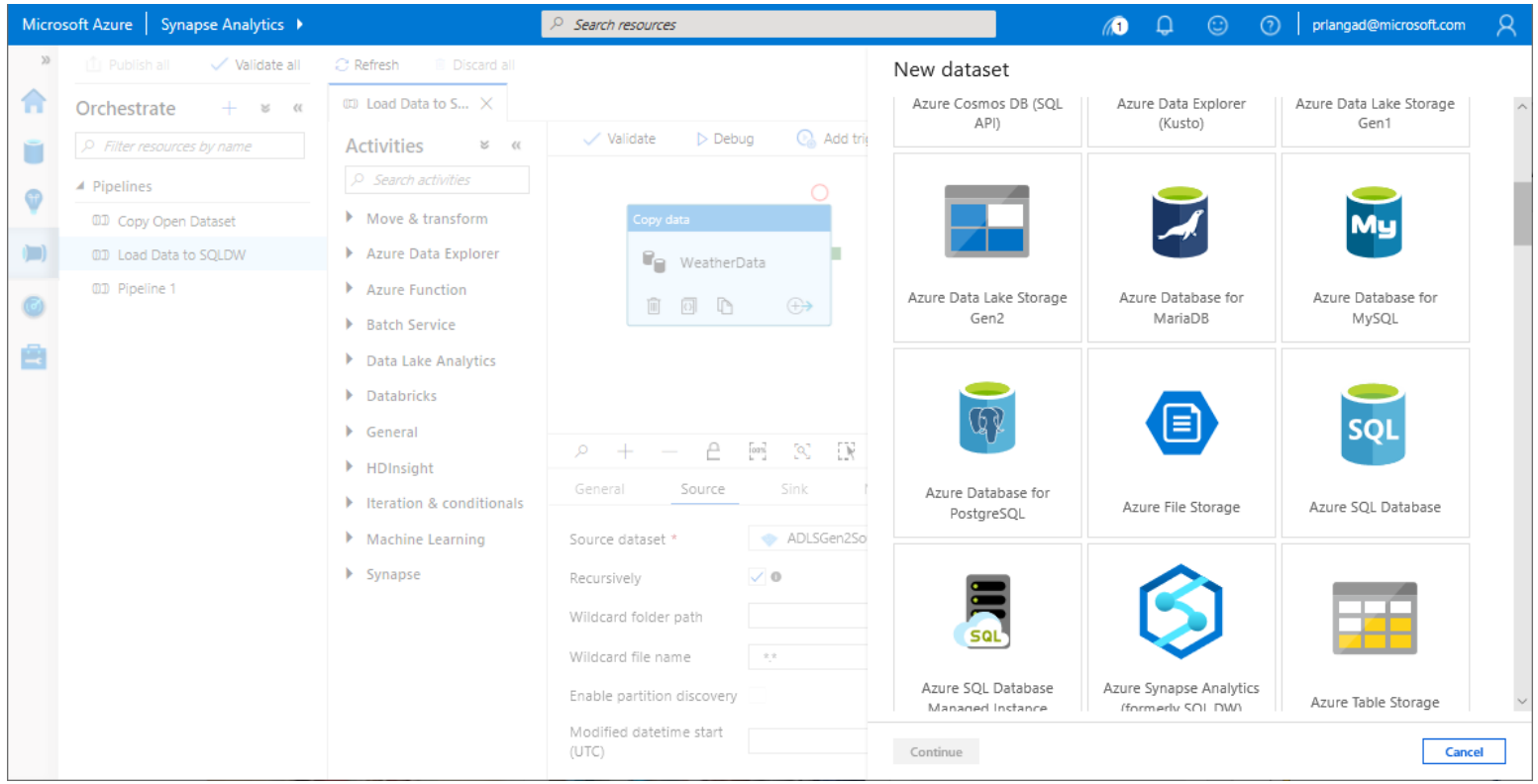
Overview

It provides ability to load data from storage account to desired linked service. Load data by manual execution of pipeline or by orchestration

Benefits

Supports common loading patterns

Polybase for data loading @ scale



Orchestrate @ Scale

Overview

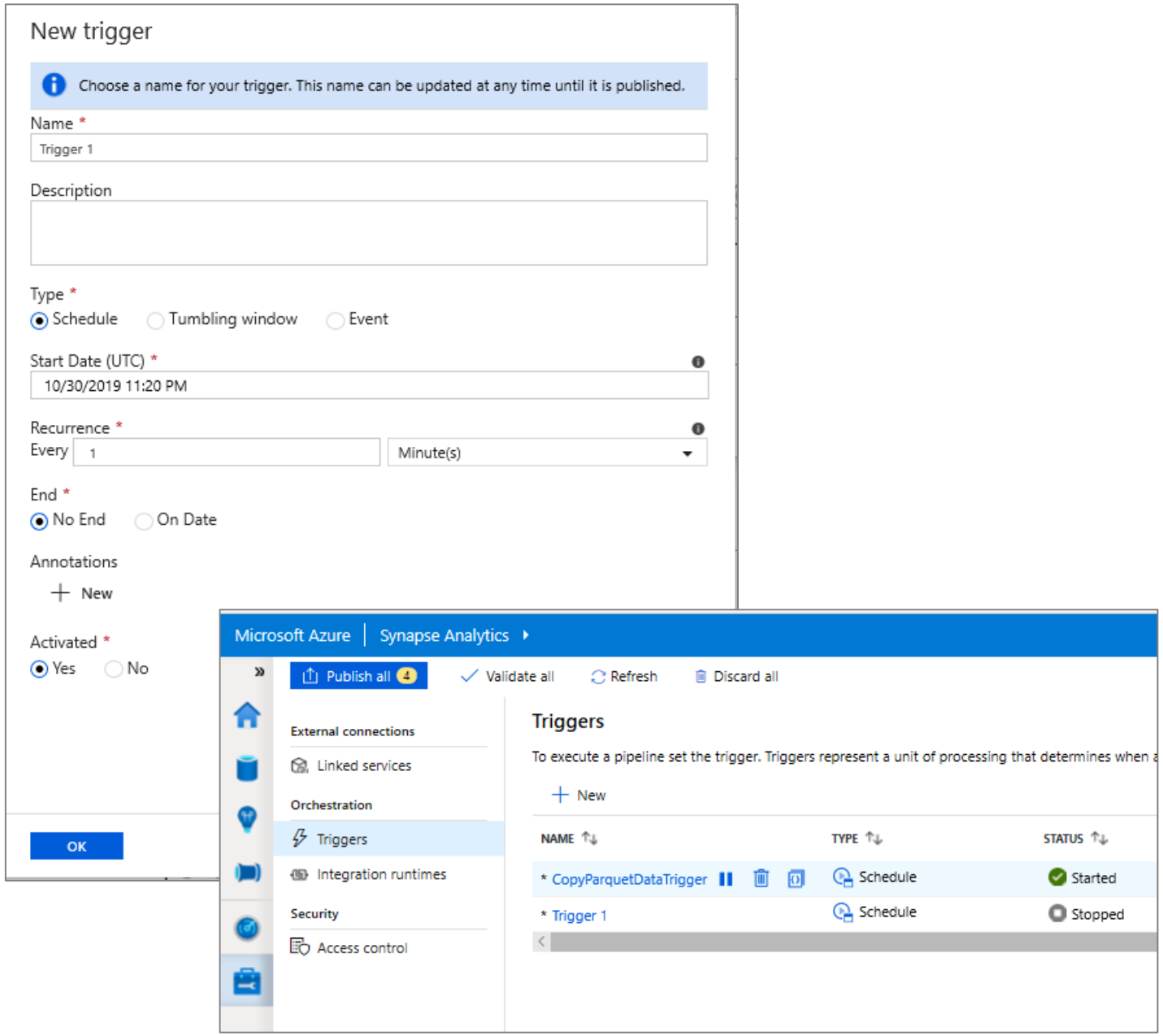
It offers trigger types as - schedule, event, tumbling window. Monitor pipeline runs, control trigger execution.

Benefits

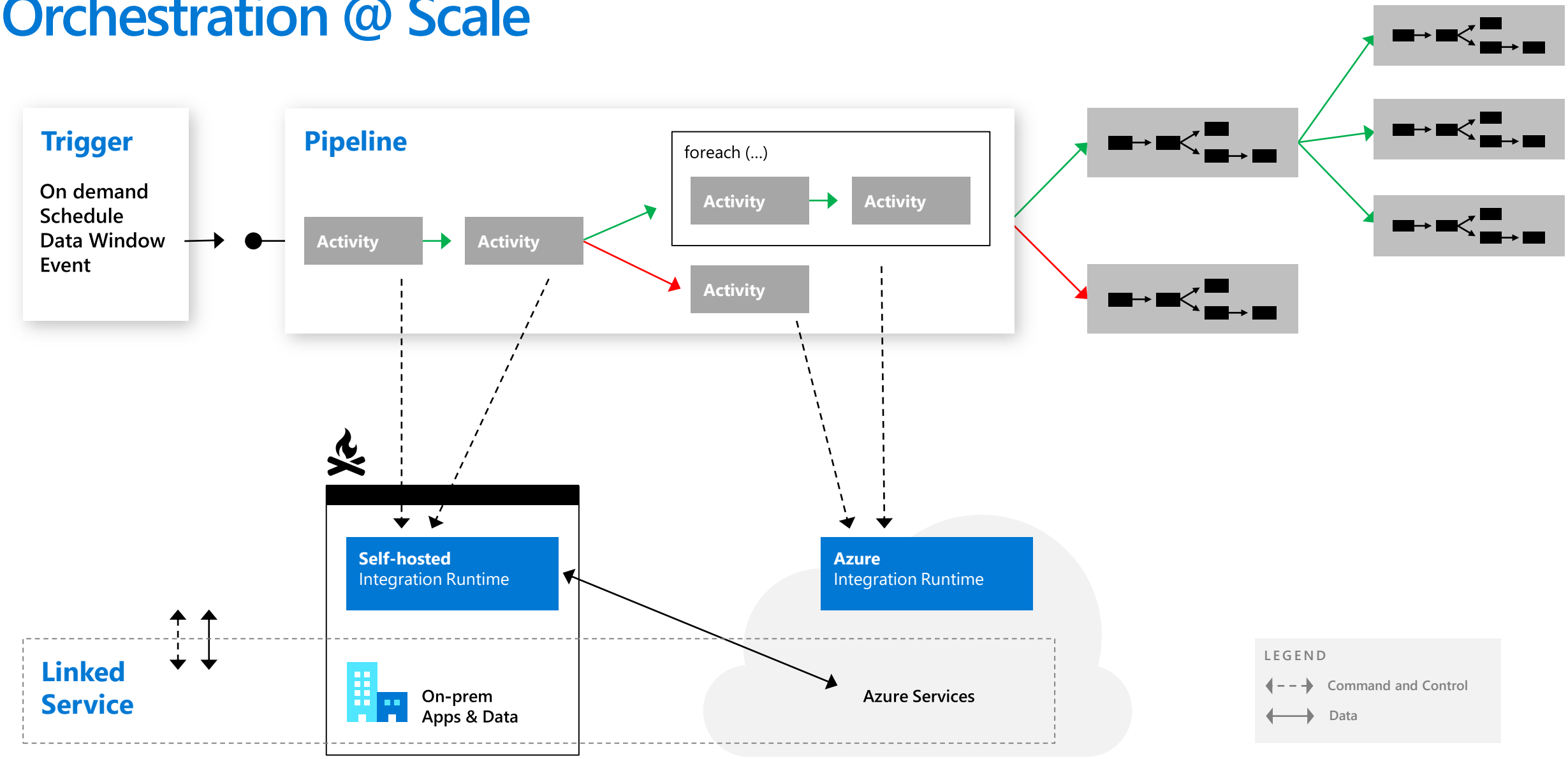
Flexible invocation

Control execution flows, conditional logic

Ability to monitor execution



Orchestration @ Scale



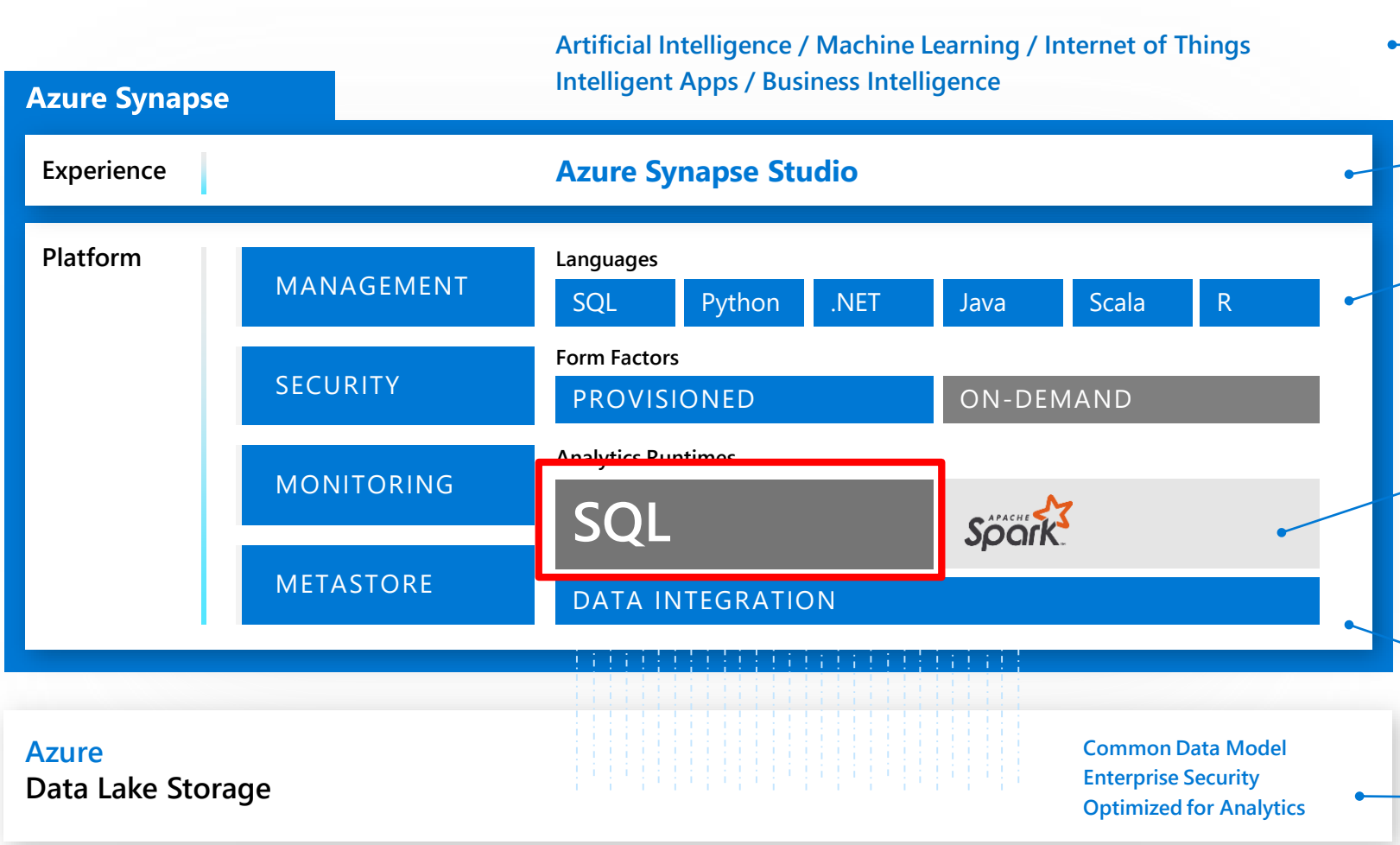
Microsoft

Azure Synapse Analytics

SQL Analytics

Azure Synapse Analytics

Limitless analytics service with unmatched time to insight



Designed for analytics **workloads at any scale**

SaaS **developer experiences** for code free and code first

Multiple **languages** suited to different analytics workloads

Integrated analytics runtimes available provisioned and serverless on-demand

SQL Analytics offering T-SQL for batch, streaming and interactive processing

Spark for big data processing with Python, Scala, R and .NET

Integrated **platform services** for, management, security, monitoring, and metastore

Data **lake integrated** and Common Data Model aware

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

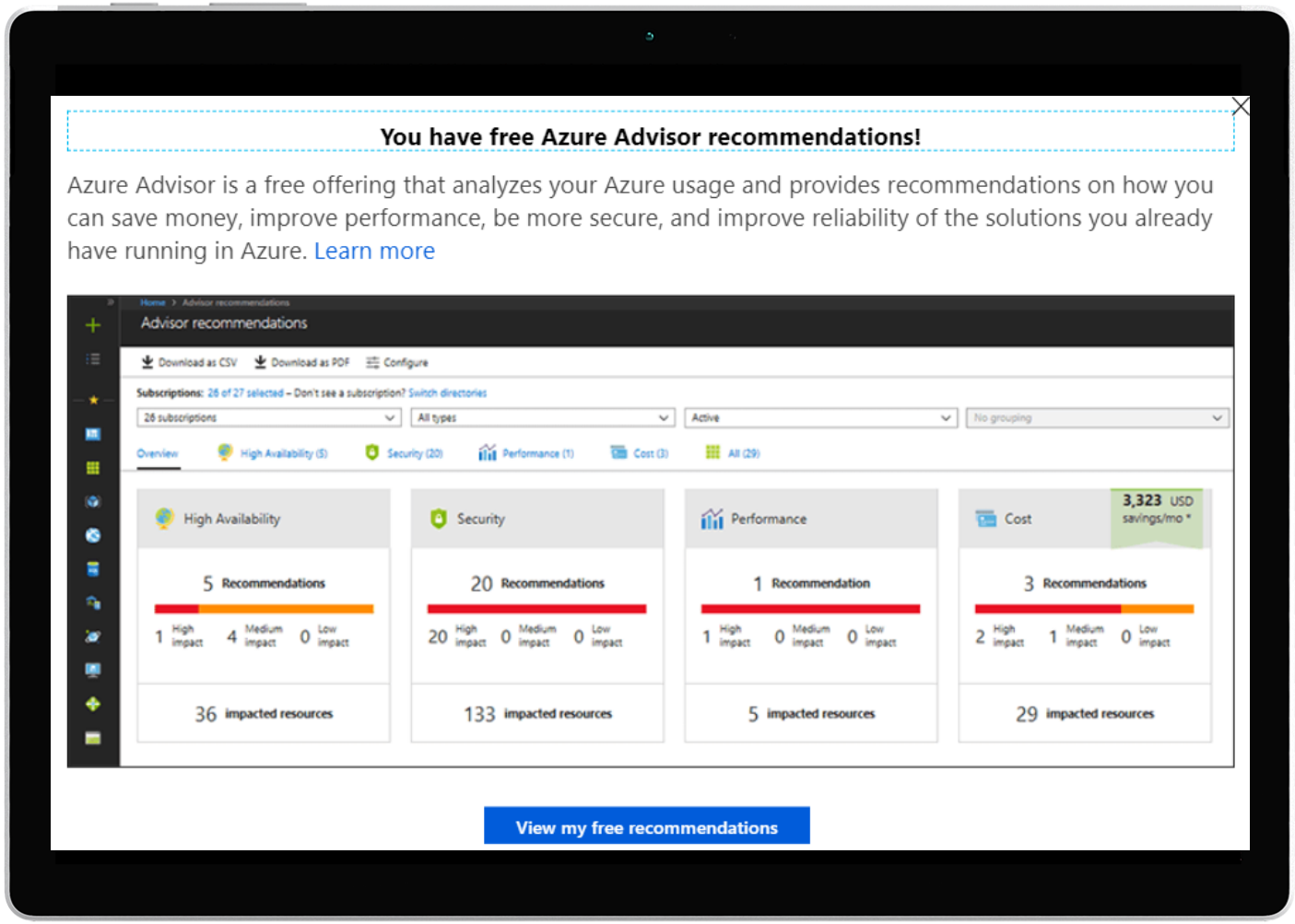
Provision additional capacity

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.

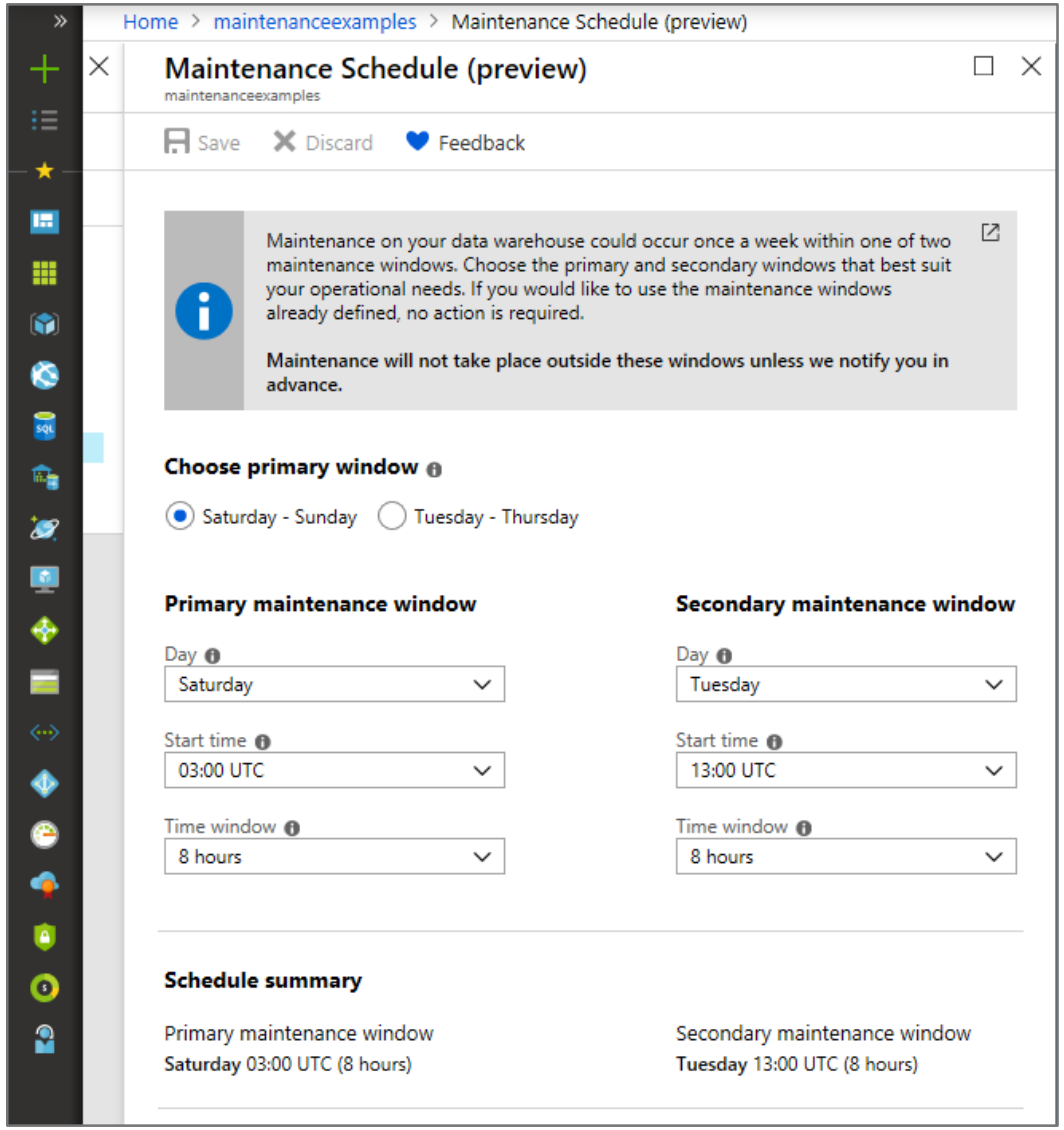
24-hour advance notification for maintenance events.

Benefits

Ensure upgrades happen on your schedule.

Predictable planning for long-running jobs.

Stay informed of start and end of maintenance.



Automatic statistics management

Overview

Statistics are automatically created and maintained in Synapse Analytics. 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
```

Comprehensive SQL language

Performance optimizations

- Table partitions
- Distributed tables
- Isolation modes
- Materialized Views
- Indexing
- Columnstore index
- Result-set caching

T-SQL Querying

- Windowing aggregates
- Approximate execution (Hyperloglog)
- GROUP BY ROLLUP

Complete SQL object model

- Tables
- Views
- Stored procedures
- Functions
- JSON data support
- and more

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

Analytical functions

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

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

Will return 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

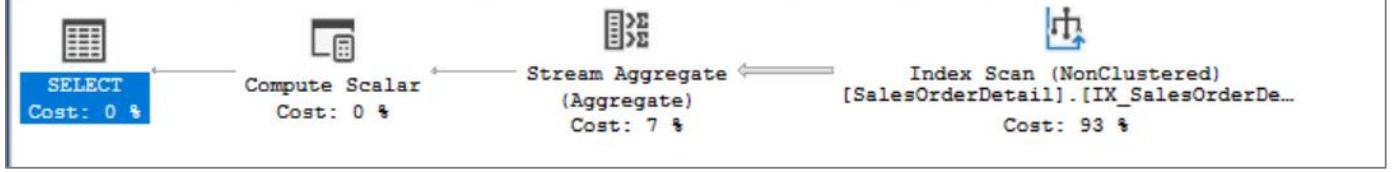
	Approx_Distinct_OrderKey
1	540

```
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 [Sa
```



COUNT DISTINCT

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

100 %

Results | Messages

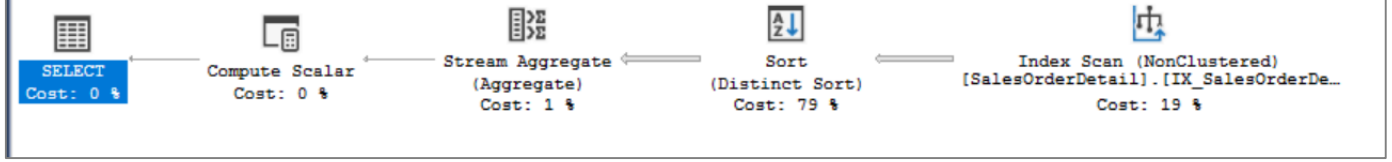
	Distinct_OrderKey
1	542

```
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.

Isolation level

- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE
- READ UNCOMMITTED

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.

```
ALTER DATABASE MyDatabase
SET ALLOW_SNAPSHOT_ISOLATION ON

ALTER DATABASE MyDatabase SET
READ_COMMITTED_SNAPSHOT ON
```

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 runtime language (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;
```

Data Storage and performance optimizations

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);
```

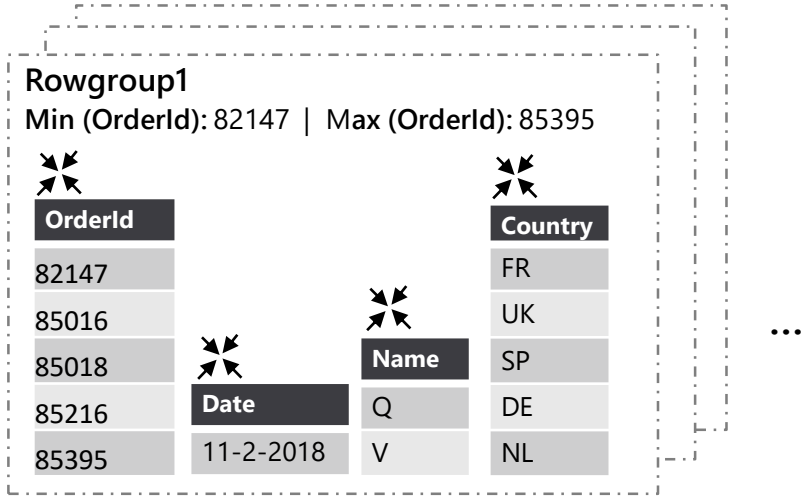
Tables – Azure Synapse Analytics Indexes

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)



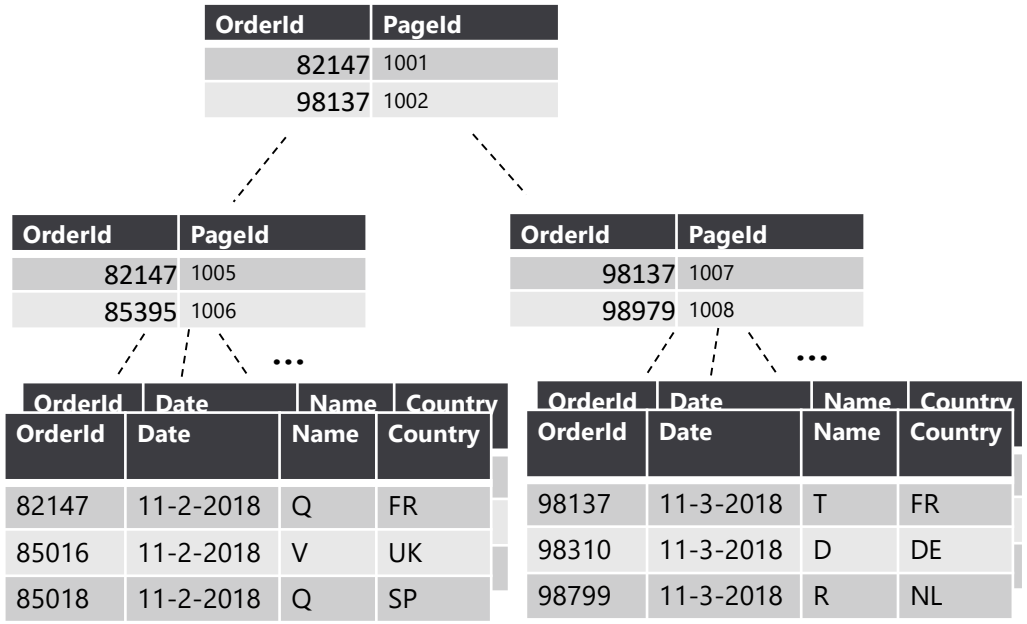
Delta Rowstore

OrderId	Date	Name	Country
98137	11-3-2018	T	FR
98310	11-3-2018	D	DE
98799	11-3-2018	R	NL
98979	11-3-2018	Z	DE

- 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)



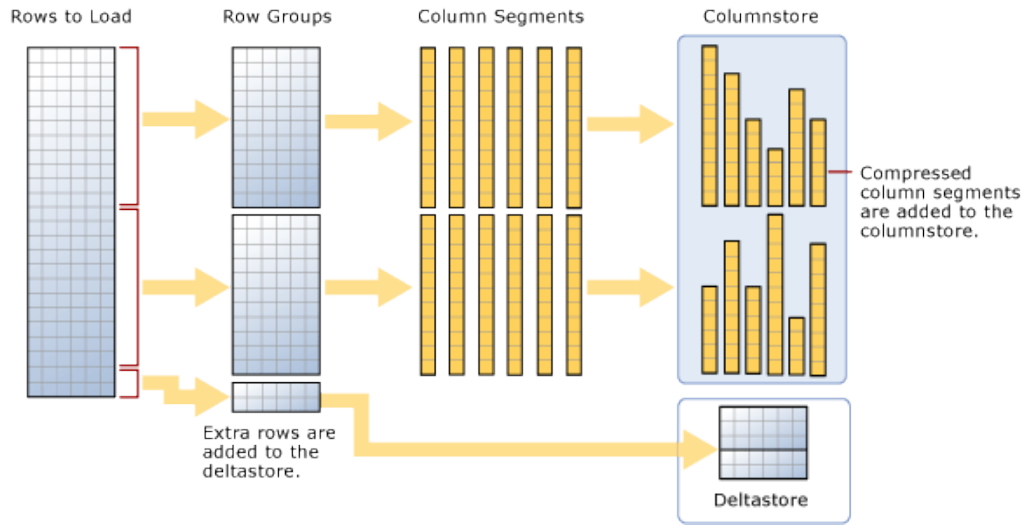
- Data is stored in a B-tree index structure for performant lookup queries for particular rows.
- Clustered rowstore index: The leaf nodes in the structure store the data values in a row (as pictured above)
- Non-clustered (secondary) rowstore index: The leaf nodes store pointers to the data values, 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.

Columnstore indexes requires manual rebuild if data is inserted, updated, or deleted in tables




```
-- 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')
```

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 – Azure Synapse Analytics 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.

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 INDEXED 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

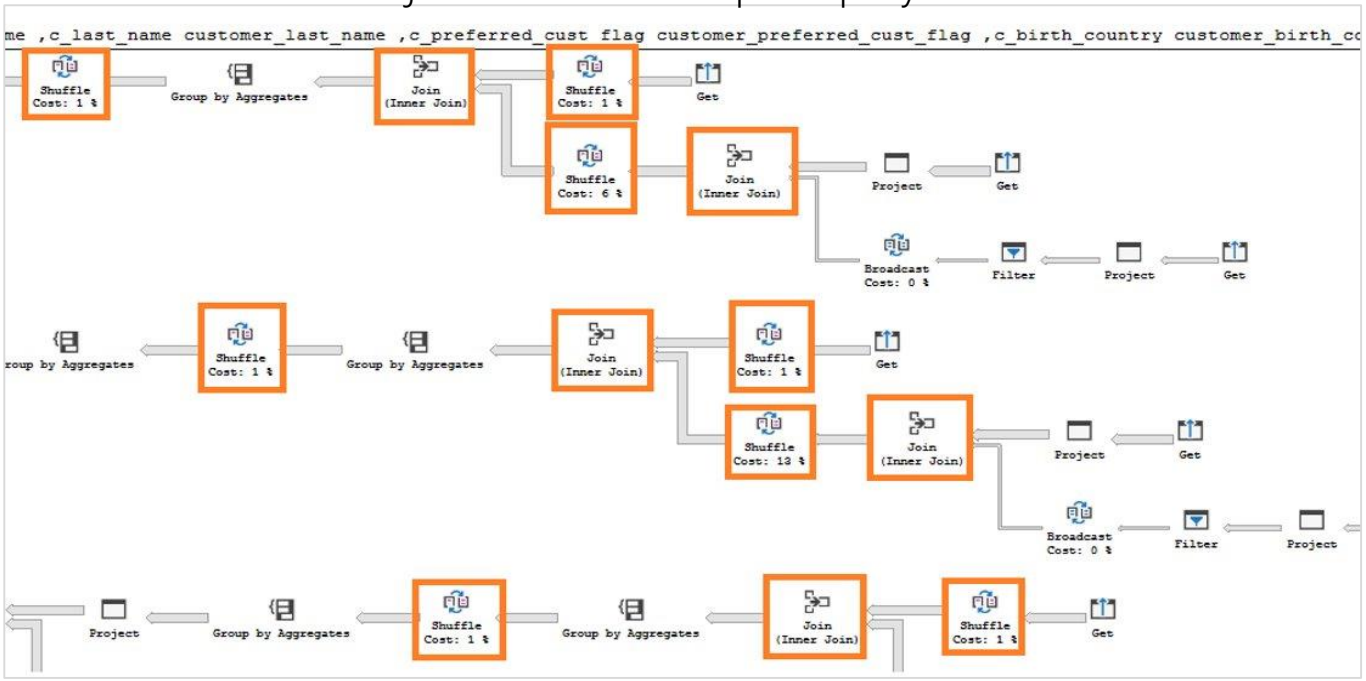
In this example, a query to get the year total sales per customer is shown to have a lot of data shuffles and joins that contribute to slow performance:

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

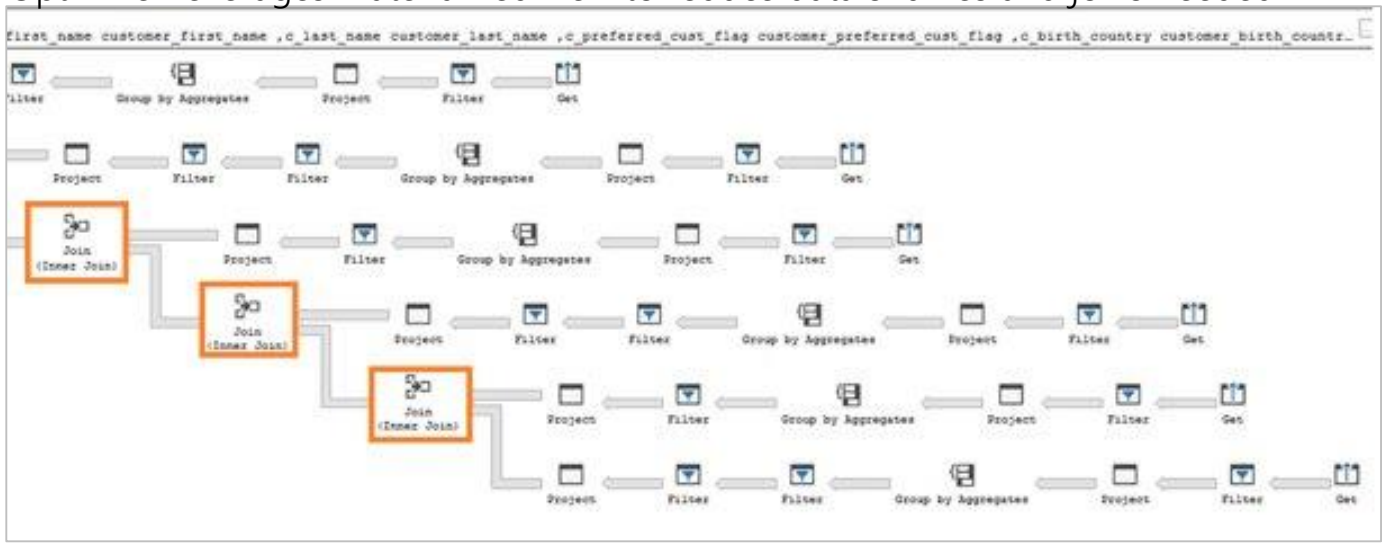
Azure Synapse Analytics query optimizer automatically leverages the indexed view to speed up the same query.
Notice that the query does not need to reference the view directly

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 Azure Synapse Analytics 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
```

COPY command

Overview

Copies data from source to destination

Benefits

Retrieves data from all files from the folder and all its subfolders.

Supports multiple locations from the same storage account, separated by comma

Supports Azure Data Lake Storage (ADLS) Gen 2 and Azure Blob Storage.

Supports CSV, PARQUET, ORC file formats

```
COPY INTO test_1
FROM
'https://XXX.blob.core.windows.net/customerdatasets/test_1.txt'
WITH (
    FILE_TYPE = 'CSV',
    CREDENTIAL=(IDENTITY= 'Shared Access Signature',
SECRET='<Your_SAS_Token>'),
    FIELDQUOTE = '"',
    FIELDTERMINATOR=';',
    ROWTERMINATOR='0X0A',
    ENCODING = 'UTF8',
    DATEFORMAT = 'ymd',
    MAXERRORS = 10,
    ERRORFILE = '/errorsfolder/'--path starting from
the storage container,
    IDENTITY_INSERT
    )
```

```
COPY INTO test_parquet
FROM
'https://XXX.blob.core.windows.net/customerdatasets/test
.parquet'
WITH (
    FILE_FORMAT = myFileFormat
    CREDENTIAL=(IDENTITY= 'Shared Access Signature',
SECRET='<Your_SAS_Token>')
)
```

Result-set caching

Overview

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

The result-set cache persists even if Azure Synapse Analytics 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

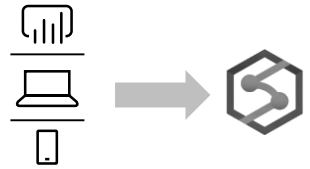
Enhances performance when same result is requested repetitively

Reduced load on server for repeated queries

Offers monitoring of query execution with a result cache hit or miss

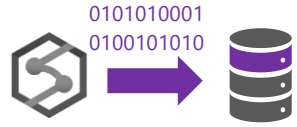
```
-- Turn on/off result-set caching for a database
-- Must be run on the MASTER database
ALTER DATABASE {database_name}
SET RESULT_SET_CACHING { ON | OFF }

-- 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

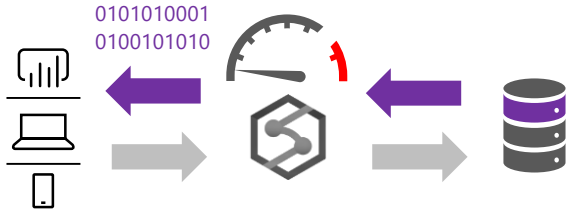


1. Client sends query to Azure Synapse Analytics

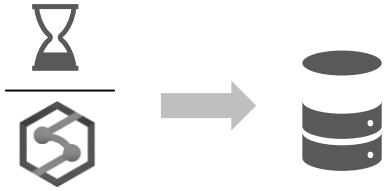
2. Query is processed using compute nodes which pull data from remote storage, process query and output back to client app



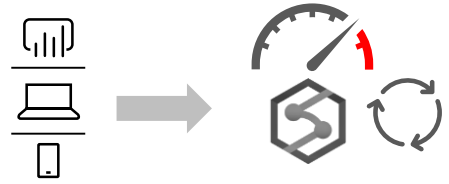
Query results are cached in remote storage so subsequent requests can be served immediately



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

Resource classes

Overview

Pre-determined resource limits defined for a user or role.

Benefits

Govern the system memory assigned to each query.

Effectively used to control the number of concurrent queries that can run on Azure Synapse Analytics.

Exemptions to concurrency limit:

CREATE|ALTER|DROP (TABLE|USER|PROCEDURE|VIEW|LOGIN)

CREATE|UPDATE|DROP (STATISTICS|INDEX)

SELECT from system views and DMVs

EXPLAIN

Result-Set Cache

TRUNCATE TABLE

ALTER AUTHORIZATION

CREATE|UPDATE|DROP STATISTICS

```sql
/*View resource classes in the Azure Synapse Analytics*/
SELECT name
FROM   sys.database_principals
WHERE  name LIKE '%rc%' AND type_desc = 'DATABASE_ROLE';

/* Change user's resource class to 'largerc' */
EXEC sp_addrolemember 'largerc', 'loaduser';

/* Decrease the loading user's resource class */
EXEC sp_droprolemember 'largerc', 'loaduser';
```

Resource class types

Static Resource Classes

Allocate the same amount of memory independent of the current service-level objective (SLO).

Well-suited for fixed data sizes and loading jobs.

Dynamic Resource Classes

Allocate a variable amount of memory depending on the current SLO.

Well-suited for growing or variable datasets.

All users default to the *smallrc* dynamic resource class.

Static resource classes:

`staticrc10 | staticrc20 | staticrc30 | staticrc40 | staticrc50 | staticrc60 | staticrc70 | staticrc80`

Dynamic resource classes:

`smallrc | mediumrc | largerc | xlargerc`

Resource Class	Percentage Memory	Max. Concurrent Queries
smallrc	3%	32
mediumrc	10%	10
largerc	22%	4
xlargerc	70%	1

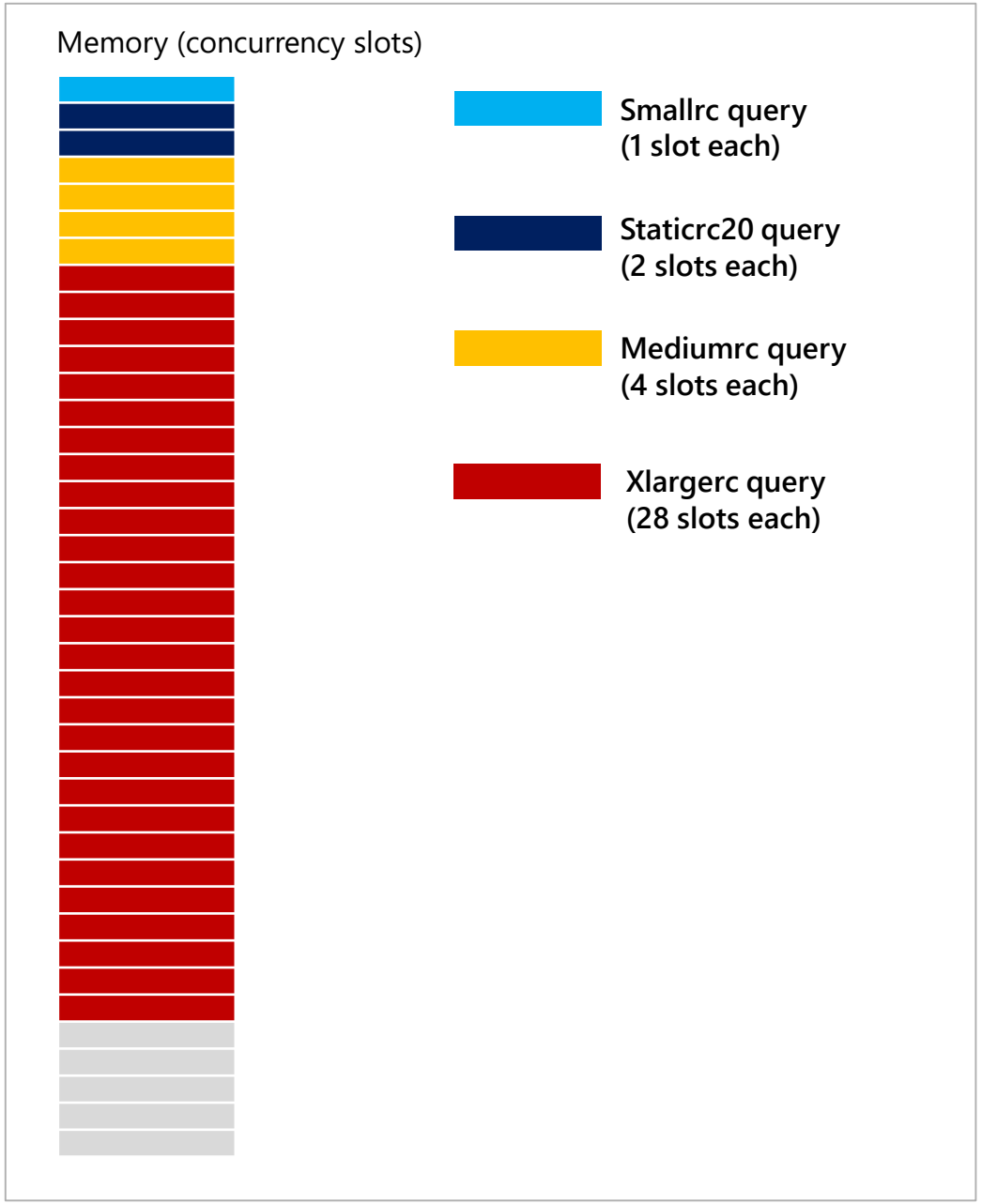
Concurrency slots

Overview

Queries running on Azure Synapse Analytics compete for access to system resources
(CPU, IO, and memory).

To guarantee access to resources, running queries are assigned a chunk of system memory (**a concurrency slot**) for processing the query. The amount given is determined by the resource class of the user executing the query. Higher DW SLOs provide more memory and concurrency slots

@DW1000c: **40 concurrency slots**



Concurrent query limits

Overview

The limit on how many queries can run at the same time is governed by two properties:

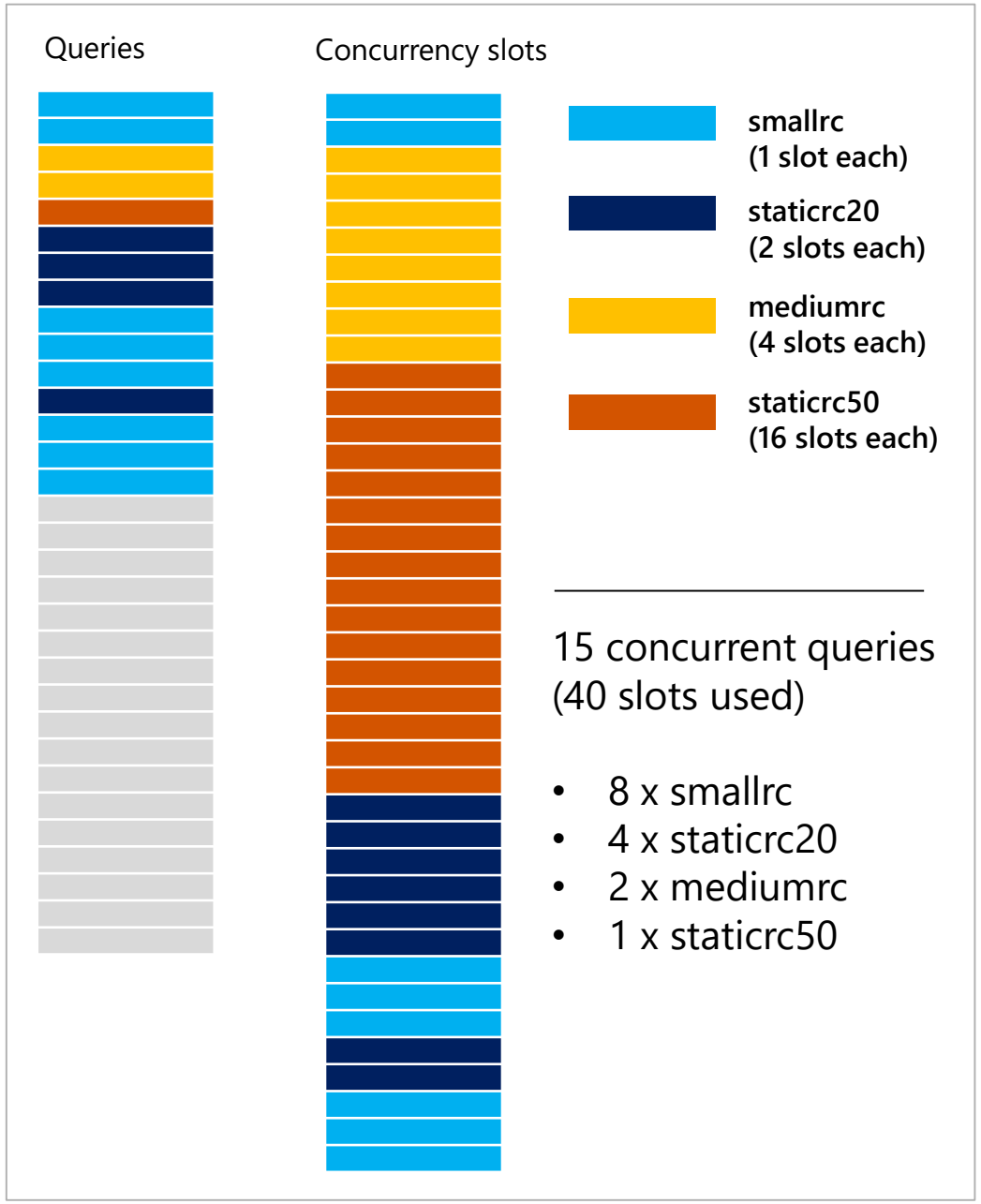
- The max. concurrent query count for the DW SLO
- The total available memory (concurrency slots) for the DW SLO

Increase the concurrent query limit by:

- Scaling up to a higher DW SLO (up to 128 concurrent queries)
- Using lower resource classes that use less memory per query

[Concurrency limits based on resource classes]

@DW1000c: **32 max concurrent queries, 40 slots**



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 (smallrc resource class).

Benefits

Map queries to both Resource Management and Workload Isolation concepts.

Manage groups of users with only a few classifiers.

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 = '<Resource Class>' ]
   [IMPORTANCE = { LOW                  |
                                BELOW_NORMAL      |
                                NORMAL            |
                                ABOVE_NORMAL      |
                                HIGH
                         }
   ]
   [MEMBERNAME = 'security_account']
)
```

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.

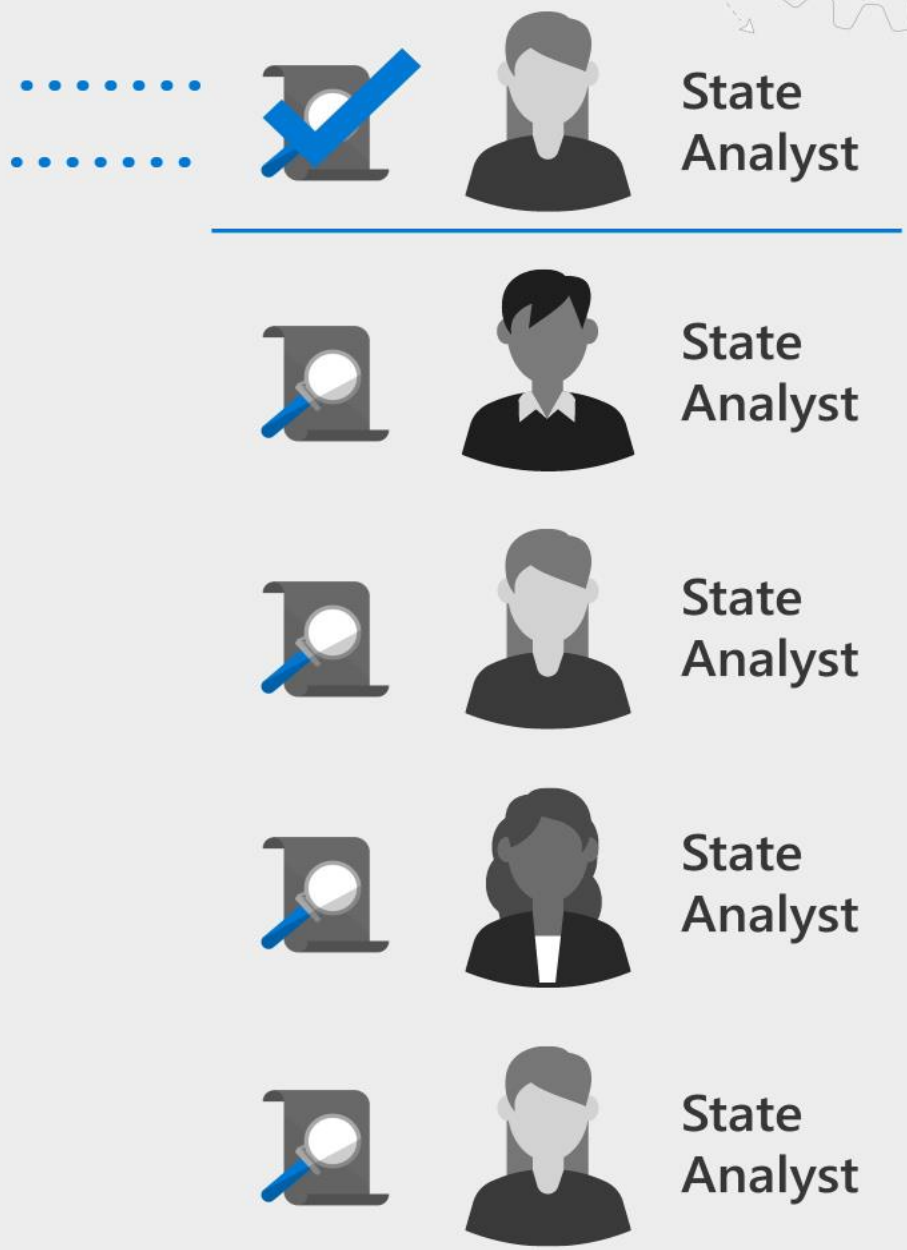
National analyst is assigned high importance.

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 = 'smallrc']
    [IMPORTANCE = HIGH]
    [MEMBERNAME = 'National_Analyst_Login']
```

Azure Synapse Analytics



Workload Isolation

Overview

Allocate fixed resources to workloads.

These limits are strictly enforced for memory, and only enforced under load for CPU and IO.

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

Benefits

Flexibly allocate resources within a single cluster

Workload isolation that responds immediately to cluster usage

Guarantee queries minimum resources regardless of Azure Synapse Analytics load

Monitoring DMVs

`sys.workload_management_workload_groups`

Query to view configured workload group.

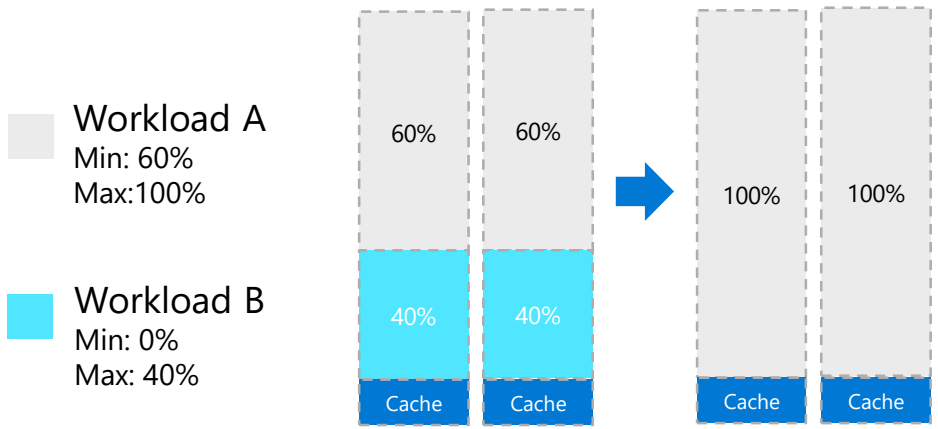
```
CREATE WORKLOAD GROUP group_name
WITH
(
    [ MIN_PERCENTAGE_RESOURCE = value ]
    [ CAP_PERCENTAGE_RESOURCE = value ]
    [ MAX_CONCURRENCY = value ]
)
```

MIN_PERCENTAGE_RESOURCE: *percentage of resources guaranteed for all requests in this group.*

CAP_PERCENTAGE_RESOURCE: *maximum percentage of resources that can be used for all requests in this group.*

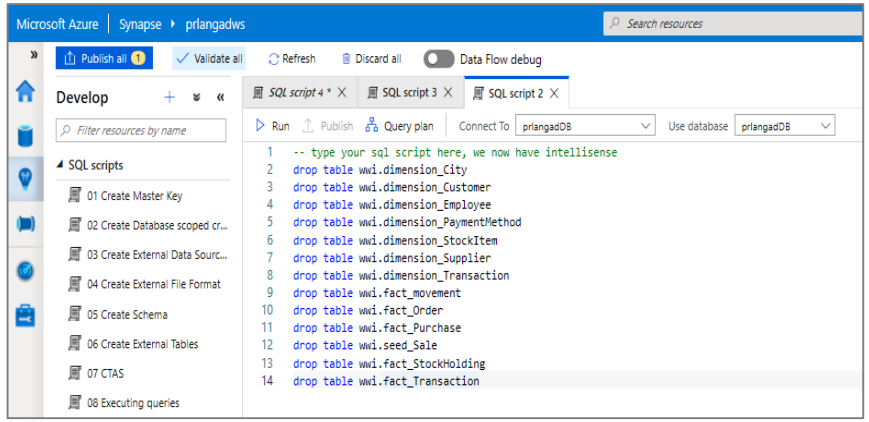
MAX_CONCURRENCY: *maximum number of queries that can be run concurrently in this group.*

Workload Isolation automatically reassigns resources as Workload B ends

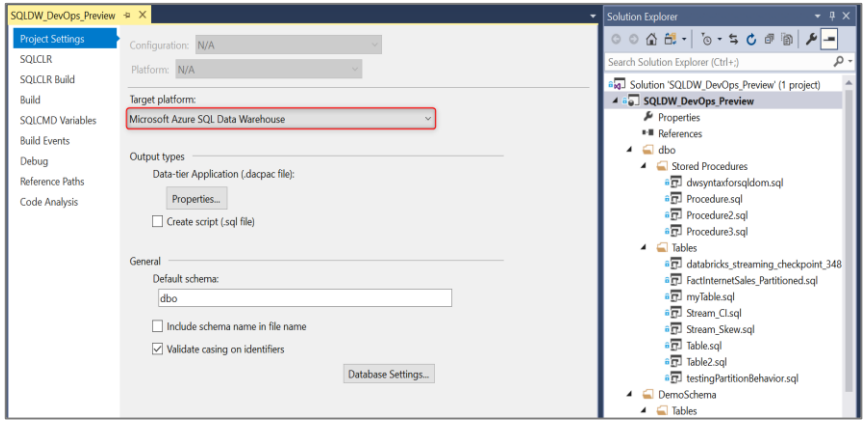


Developer Tools

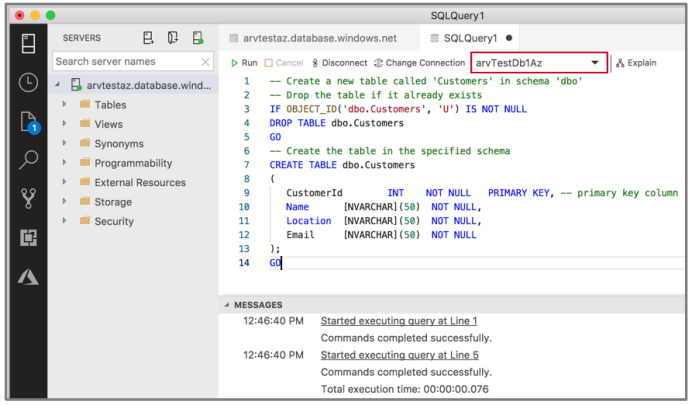
Azure Synapse Analytics



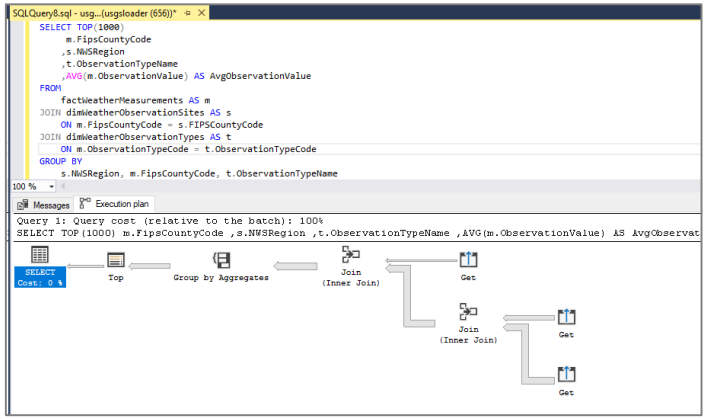
Visual Studio - SSDT database projects



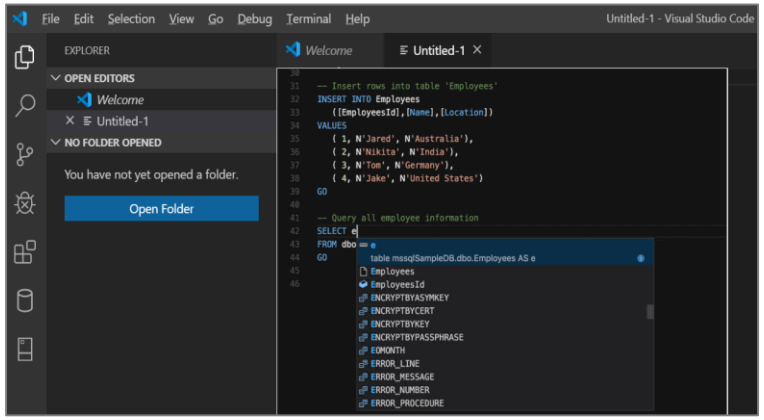
Azure Data Studio (queries, extensions etc.)



SQL Server Management Studio (queries, execution plans etc.)

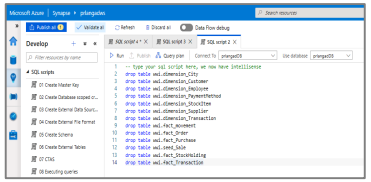


Visual Studio Code



Developer Tools

Azure Synapse Analytics

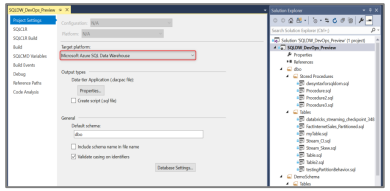


Azure Cloud Service

Offers end-to-end lifecycle for analytics

Connects to multiple services

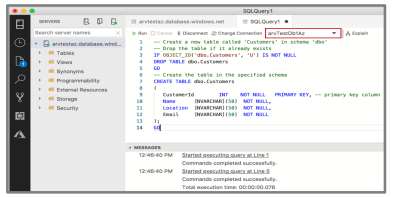
Visual Studio - SSDT database projects



Runs on Windows

Create, maintain database code, compile, code refactoring

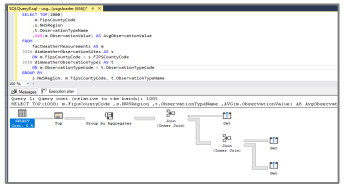
Azure Data Studio



Runs on Windows, Linux, macOS

Light weight editor, (queries and extensions)

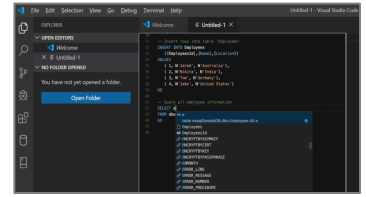
SQL Server Management Studio



Runs on Windows

Offers GUI support to query, design and manage

Visual Studio Code



Runs on Windows, Linux, macOS

Offers development experience with light-weight code editor

Continuous integration and delivery (CI/CD)

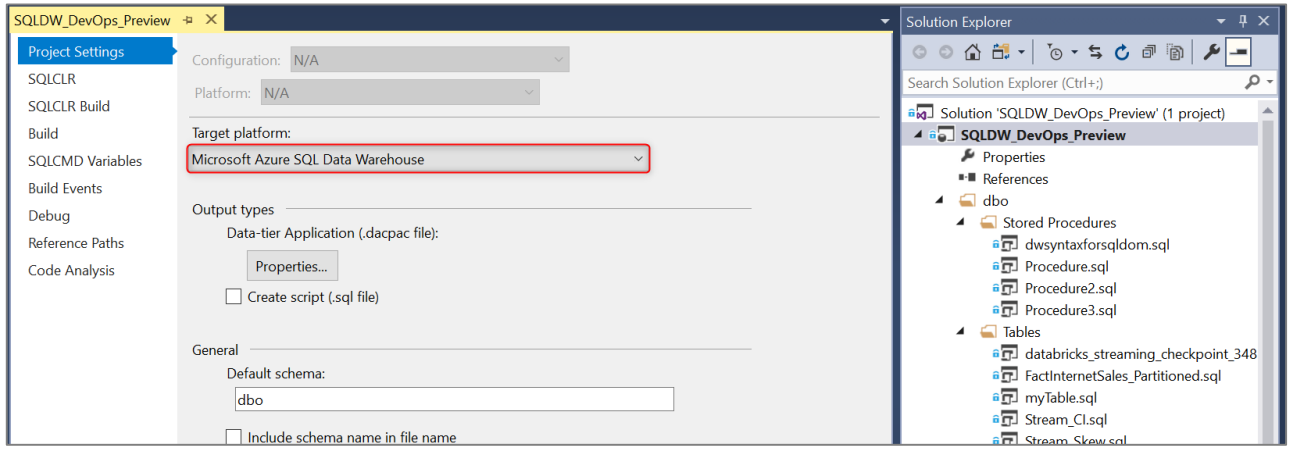
Overview

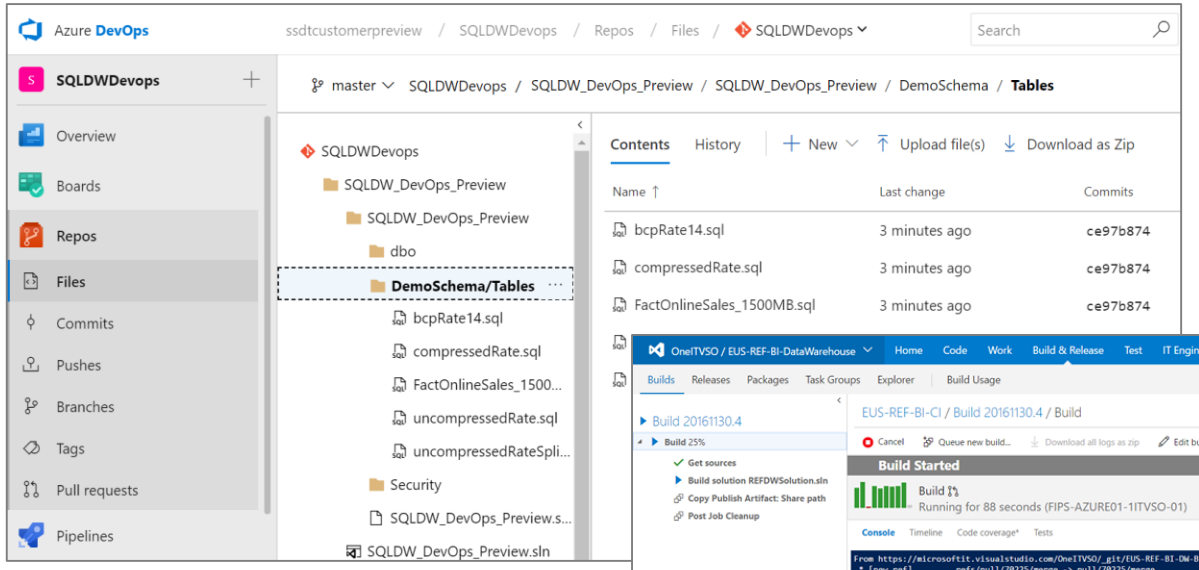
Database project support in SQL Server Data Tools (SSDT) allows teams of developers to collaborate over a version-controlled Azure Synapse Analytics, and track, deploy and test schema changes.

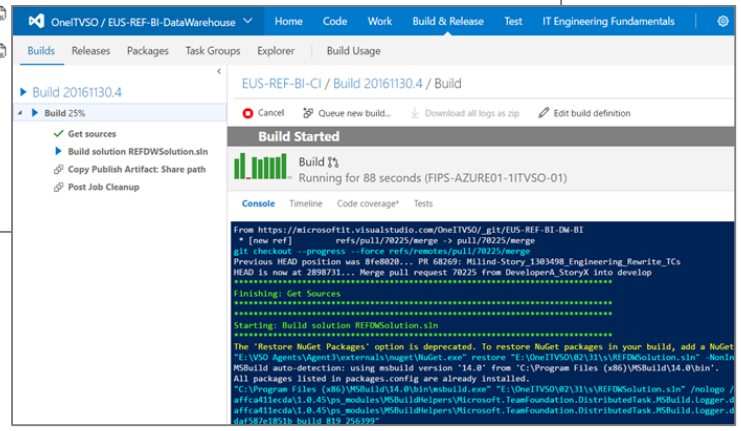
Benefits

Database project support includes first-class integration with Azure DevOps. This adds support for:

- **Azure Pipelines** to run CI/CD workflows for any platform (Linux, macOS, and Windows)
- **Azure Repos** to store project files in source control
- **Azure Test Plans** to run automated check-in tests to verify schema updates and modifications
- Growing ecosystem of third-party integrations that can be used to complement existing workflows (Timetracker, Microsoft Teams, Slack, Jenkins, etc.)







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

SQL On-Demand

SQL On Demand

SQL On-Demand

Overview

An interactive query service that provides T-SQL queries over high scale data in Azure Storage.

Benefits

Serverless

No infrastructure

Pay only for query execution

No ETL

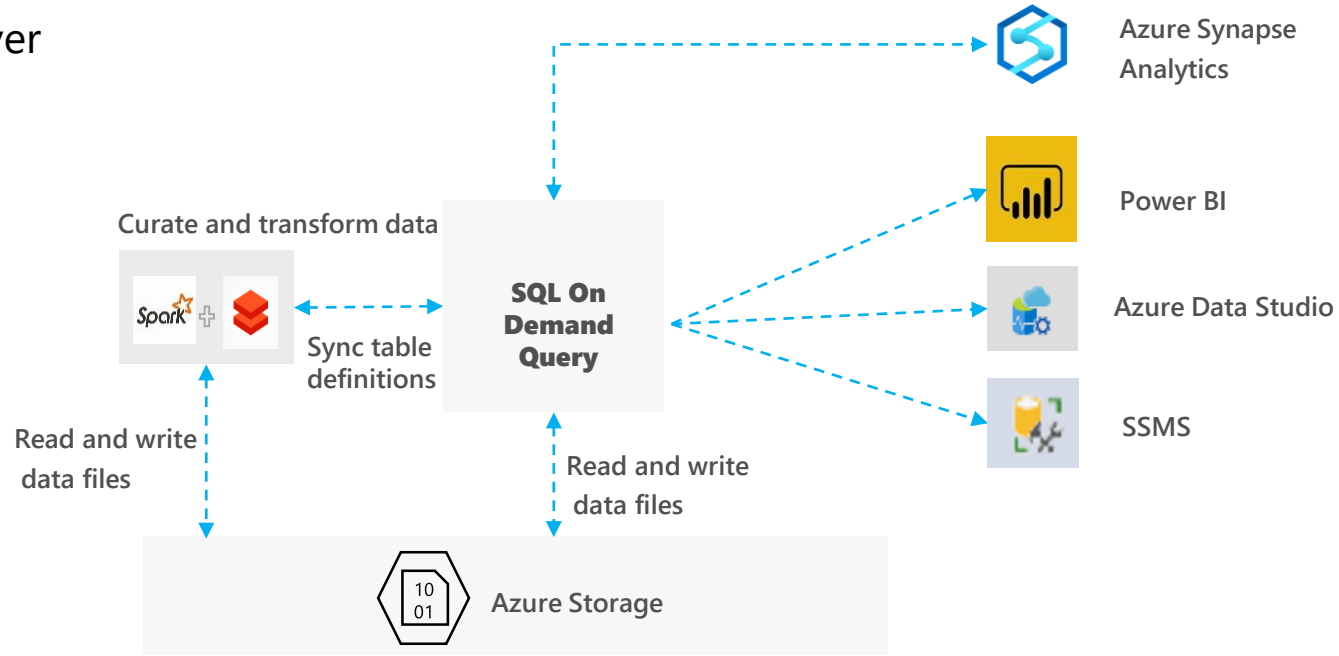
Offers security

Data integration with Databricks, HDInsight

T-SQL syntax to query data

Supports data in various formats (Parquet, CSV, JSON)

Support for BI ecosystem



SQL On Demand – Querying CSV File

Overview

Uses OPENROWSET function to access data

Benefits

Ability to read CSV File with

- no header row, Windows style new line
- no header row, Unix-style new line
- header row, Unix-style new line
- header row, Unix-style new line, quoted
- header row, Unix-style new line, escape
- header row, Unix-style new line, tab-delimited
- without specifying all columns

```
SELECT *
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/population/population.csv',
        FORMAT = 'CSV',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '\n'
    )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017
```

	country_code	country_name	year	population
1	LU	Luxembourg	2017	594130

SQL On Demand – Querying CSV File

Read CSV file - header row, Unix-style new line

```
SELECT *
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/population-
unix-hdr/population.csv',
        FORMAT = 'CSV',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '0x0a',
        FIRSTROW = 2
    )
    WITH (
        [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
        [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
        [year] smallint,
        [population] bigint
    ) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017
```

	country_code	country_name	year	population
1	LU	Luxembourg	2017	594130

Read CSV file - without specifying all columns

```
SELECT
    COUNT(DISTINCT country_name) AS countries
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/popul
ation/population.csv',
        FORMAT = 'CSV',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '\n'
    )
WITH (
        [country_name] VARCHAR (100) COLLATE Latin1_Gener
al_BIN2 2
) AS [r]
```

	countries
1	228

SQL On Demand – Querying folders

Overview

Uses OPENROWSET function to access data from multiple files or folders

Benefits

Offers reading multiple files/folders through usage of wildcards

Offers reading specific file/folder

Supports use of multiple wildcards

```
SELECT YEAR(pickup_datetime) as [year], SUM(passenger_count) AS passengers_total,
COUNT(*) AS [rides_total]
FROM OPENROWSET(
BULK 'https://XXX.blob.core.windows.net/csv/taxi/*.*',
FORMAT = 'CSV'
, FIRSTROW = 2 )
WITH (
        vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count INT,
        trip_distance FLOAT,
        rate_code INT,
        store_and_fwd_flag VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_location_id INT,
        dropoff_location_id INT,
        payment_type INT,
        fare_amount FLOAT,
        extra FLOAT, mta_tax FLOAT,
        tip_amount FLOAT,
        tolls_amount FLOAT,
        improvement_surcharge FLOAT,
        total_amount FLOAT
        ) AS nyc
GROUP BY YEAR(pickup_datetime)
ORDER BY YEAR(pickup_datetime)
```

	year	passengers_total	rides_total
1	2001	14	10
2	2002	29	16
3	2003	22	16
4	2008	378	188
5	2009	594	353
6	2016	102093687	61758523
7	2017	184464988	113496932
8	2018	86272771	53925040
9	2019	37	29
…	2020	6	6

SQL On Demand – Querying folders

Read all files from multiple folders

```
SELECT YEAR(pickup_datetime) as [year],
    SUM(passenger_count) AS passengers_total,
    COUNT(*) AS [rides_total]
FROM OPENROWSET(
    BULK 'https://XXX.blob.core.windows.net/csv/t*i/',
        FORMAT = 'CSV',
        FIRSTROW = 2    )
    WITH (
        vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count INT,
        trip_distance FLOAT,
        <… columns>
    ) AS nyc
GROUP BY YEAR(pickup_datetime)
ORDER BY YEAR(pickup_datetime)
```

	year	passengers_total	rides_total
1	2001	14	10
2	2002	29	16
3	2003	22	16
4	2008	378	188
5	2009	594	353
6	2016	102093687	61758523
7	2017	184464988	113496932
8	2018	86272771	53925040
9	2019	37	29
…	2020	6	6

Read subset of files in folder

```
SELECT
    payment_type,
    SUM(fare_amount) AS fare_total
FROM OPENROWSET(
    BULK 'https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_2017-*.csv',
        FORMAT = 'CSV',
        FIRSTROW = 2    )
    WITH (
        vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count INT,
        trip_distance FLOAT,
        <…columns>
    ) AS nyc
GROUP BY payment_type
ORDER BY payment_type
```

	payment_type	fare_total
1	1	1026072325.579…
2	2	441093322.8000…
3	3	10435183.04
4	4	3304550.99
5	5	14

SQL On Demand – Querying specific files

Overview

filename – Provides file name that originates row result

filepath – Provides full path when no parameter is passed or part of path when parameter is passed that originates result

Benefits

Provides source name/path of file/folder for row result set

Example of filename function

```
SELECT
    r.filename() AS [filename]
    ,COUNT_BIG(*) AS [rows]
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_201
7-1*.csv',
        FORMAT = 'CSV',
        FIRSTROW = 2
    )
    WITH (
        vendor_id INT,
        pickup_datetime DATETIME2,
        dropoff_datetime DATETIME2,
        passenger_count SMALLINT,
        trip_distance FLOAT,
        <…columns>
    ) AS [r]

GROUP BY r.filename()

ORDER BY [filename]
```

	filename	rows
1	yellow_tripdata_2017-10.csv	9768815
2	yellow_tripdata_2017-11.csv	9284803
3	yellow_tripdata_2017-12.csv	9508276

SQL On Demand – Querying specific files

Example of filepath function

```
SELECT
    r.filepath() AS filepath
    ,r.filepath(1) AS [year]
    ,r.filepath(2) AS [month]
    ,COUNT_BIG(*) AS [rows]
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_*-*.csv',
        FORMAT = 'CSV',
        FIRSTROW = 2    )
WITH (
    vendor_id INT,
    pickup_datetime DATETIME2,
    dropoff_datetime DATETIME2,
    passenger_count SMALLINT,
    trip_distance FLOAT,
   <… columns>
) AS [r]

WHERE r.filepath(1) IN ('2017')
  AND r.filepath(2) IN ('10', '11', '12')

GROUP BY    r.filepath() ,r.filepath(1) ,r.filepath(2)
ORDER BY    filepath
```

filepath	year	month	rows
https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_2017-10.csv	2017	10	9768815
https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_2017-11.csv	2017	11	9284803
https://XXX.blob.core.windows.net/csv/taxi/yellow_tripdata_2017-12.csv	2017	12	9508276

SQL On Demand – Querying Parquet files

Overview

Uses OPENROWSET function to access data

Benefits

Ability to specify column names of interest

Offers auto reading of column names and data types

Provides target specific partitions using filepath function

```
SELECT
        YEAR(pickup_datetime),
        passenger_count,
        COUNT(*) AS cnt
FROM
    OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/parquet/taxi/*/*/*',
        FORMAT='PARQUET'
    ) WITH (
        pickup_datetime DATETIME2,
        passenger_count INT
    ) AS nyc
GROUP BY
    passenger_count,
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime),
    passenger_count
```

	(No column name)	passenger_count	cnt
1	2016	0	2557
2	2016	1	43735845
3	2016	2	9056714
4	2016	3	2610541
5	2016	4	1309639
6	2016	5	3086097
7	2016	6	1956607

SQL On Demand – Creating views

Overview

Create views using SQL On Demand queries

Benefits

Works same as standard views

```
USE [mydbname]
GO

IF EXISTS(select * FROM sys.views where name = 'populationView')
DROP VIEW populationView
GO

CREATE VIEW populationView AS
SELECT *
FROM OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/csv/population/population.csv',
        FORMAT = 'CSV',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '\n'
    )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r]
```

```
SELECT
    country_name, population
FROM populationView
WHERE
    [year] = 2019
ORDER BY
    [population] DESC
```

	country_name	population
1	China	1389618778
2	India	1311559204
3	United States	331883986
4	Indonesia	264935824
5	Pakistan	210797836
6	Brazil	210301591
7	Nigeria	208679114
8	Bangladesh	161062905
9	Russia	141944641
10	Mexico	127318112

SQL On Demand – Querying JSON files

Overview

Read JSON files and provides data in tabular format

Benefits

Supports OPENJSON, JSON_VALUE and JSON_QUERY functions

```
SELECT *
FROM
    OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/json/books/book
1.json’,
        FORMAT='CSV',
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x0b',
        ROWTERMINATOR = '0x0b'
    )
    WITH (
        jsonContent varchar(8000)
    ) AS [r]
```

	jsonContent
1	{"_id": "kim95", "type": "Book", "title": "Modern Databas…

SQL On Demand – Querying JSON files

Example of JSON_VALUE function

```
SELECT
    JSON_VALUE(jsonContent, '$.title') AS title,
    JSON_VALUE(jsonContent, '$.publisher') as publisher,
    jsonContent
FROM
    OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/json/books/*.json',
        FORMAT='CSV',
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x0b',
        ROWTERMINATOR = '0x0b'
    )
    WITH (
        jsonContent varchar(8000)
    ) AS [r]
WHERE
    JSON_VALUE(jsonContent, '$.title') = 'Probabilistic and Statist
ical Methods in Cryptology, An Introduction by Selected Topics'
```

	title	publisher	jsonContent
1	Probabilistic and Statistical Methods in Cryptology, An Int…	Springer	{"_id": "neuen…

Example of JSON_QUERY function

```
SELECT
    JSON_QUERY(jsonContent, '$.authors') AS authors,
    jsonContent
FROM
    OPENROWSET(
        BULK 'https://XXX.blob.core.windows.net/json/books/*.json',
        FORMAT='CSV',
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x0b',
        ROWTERMINATOR = '0x0b'
    )
    WITH (
        jsonContent varchar(8000)
    ) AS [r]
WHERE
    JSON_VALUE(jsonContent, '$.title') = 'Probabilistic and Statist
ical Methods in Cryptology, An Introduction by Selected Topics'
```

	authors	jsonContent
1	["Daniel Neuenschwander"]	{"_id": "neuenschwander04", "type": "Book", "title": "Probabi…

Create External Table As Select

Overview

Creates an external table and then exports results of the Select statement. These operations will import data into the database for the duration of the query

Steps:

1. Create Master Key
2. Create Credentials
3. Create External Data Source
4. Create External Data Format
5. Create External Table

```
-- Create a database master key if one does not already exist
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'S0me!nfo'
;
-- Create a database scoped credential with Azure storage account key
as the secret.
CREATE DATABASE SCOPED CREDENTIAL AzureStorageCredential
WITH
     IDENTITY   = '<my_account>'
,    SECRET     = '<azure_storage_account_key>'
;
-- Create an external data source with CREDENTIAL option.
CREATE EXTERNAL DATA SOURCE MyAzureStorage
WITH
(    LOCATION   = 'wasbs://daily@logs.blob.core.windows.net/'
,    CREDENTIAL = AzureStorageCredential
,    TYPE       = HADOOP
)
-- Create an external file format
CREATE EXTERNAL FILE FORMAT MyAzureCSVFormat
WITH (FORMAT_TYPE = DELIMITEDTEXT,
      FORMAT_OPTIONS(
          FIELD_TERMINATOR = ',',
          FIRST_ROW = 2)
--Create an external table
CREATE EXTERNAL TABLE dbo.FactInternetSalesNew
WITH(
        LOCATION = '/files/Customer',
        DATA_SOURCE = MyAzureStorage,
        FILE_FORMAT = MyAzureCSVFormat
    )
AS SELECT T1.* FROM dbo.FactInternetSales T1 JOIN dbo.DimCustomer T2
ON ( T1.CustomerKey = T2.CustomerKey )
OPTION ( HASH JOIN );
```

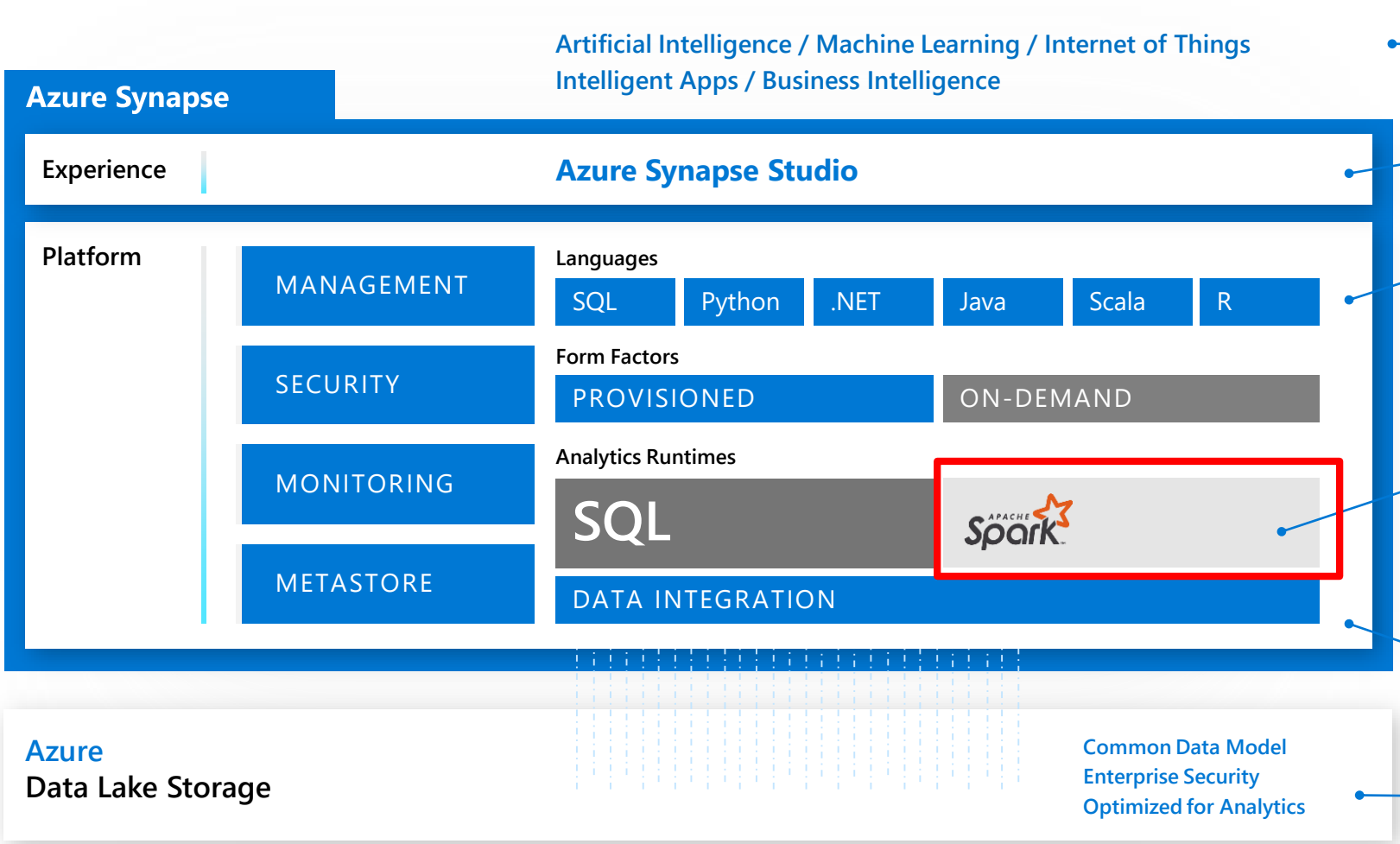
Microsoft

Azure Synapse Analytics

Spark

Azure Synapse Analytics

Limitless analytics service with unmatched time to insight



Designed for analytics **workloads at any scale**

SaaS **developer experiences** for code free and code first

Multiple **languages** suited to different analytics workloads

Integrated analytics runtimes available provisioned and serverless on-demand

SQL Analytics offering T-SQL for batch, streaming and interactive processing

Spark for big data processing with Python, Scala, R and .NET

Integrated **platform services** for, management, security, monitoring, and metastore

Data **lake integrated** and Common Data Model aware

Library Management - Python

Overview

Customers can add new python libraries at Spark pool level

Benefits

Input requirements.txt in simple pip freeze format

Add new libraries to your cluster

Update versions of existing libraries on your cluster

Libraries will get installed for your Spark pool during cluster creation

Ability to specify different requirements file for different pools within the same workspace

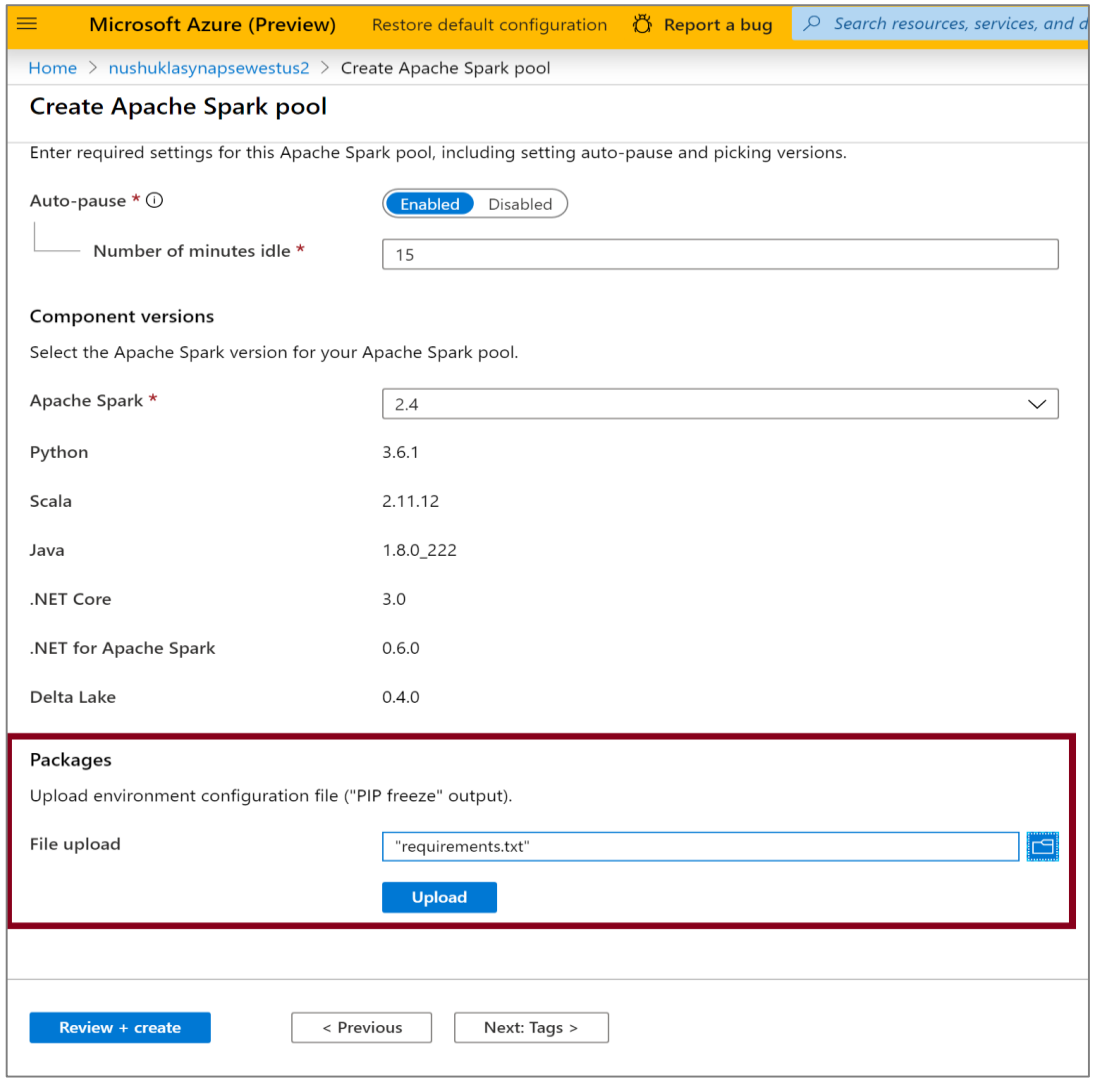
Constraints

The library version must exist on PyPI repository

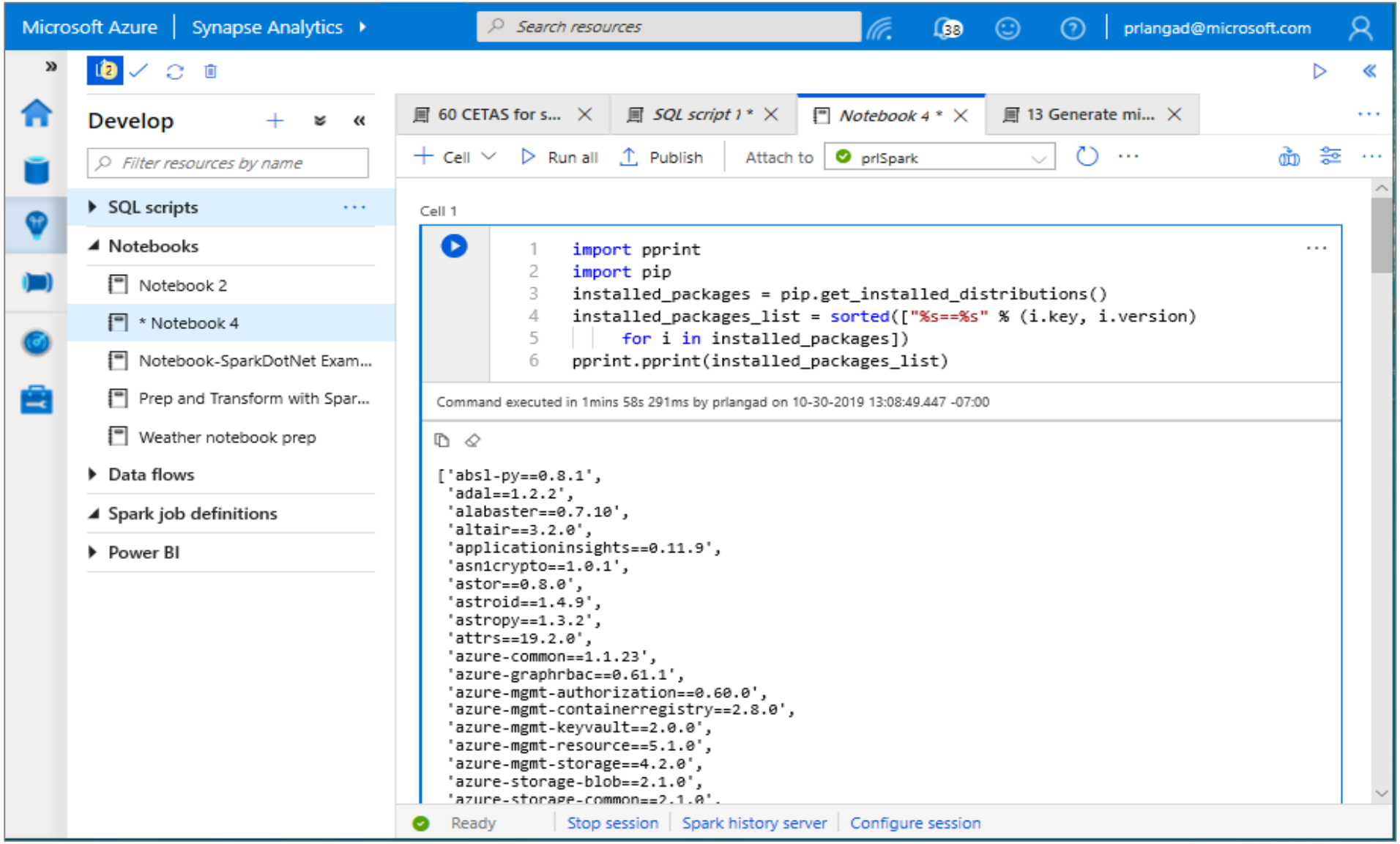
Version downgrade of an existing library not allowed

In the Portal

Specify the new requirements while creating Spark Pool in Additional Settings blade



Library Management - Python



Languages

Overview

Supports multiple languages to develop notebook

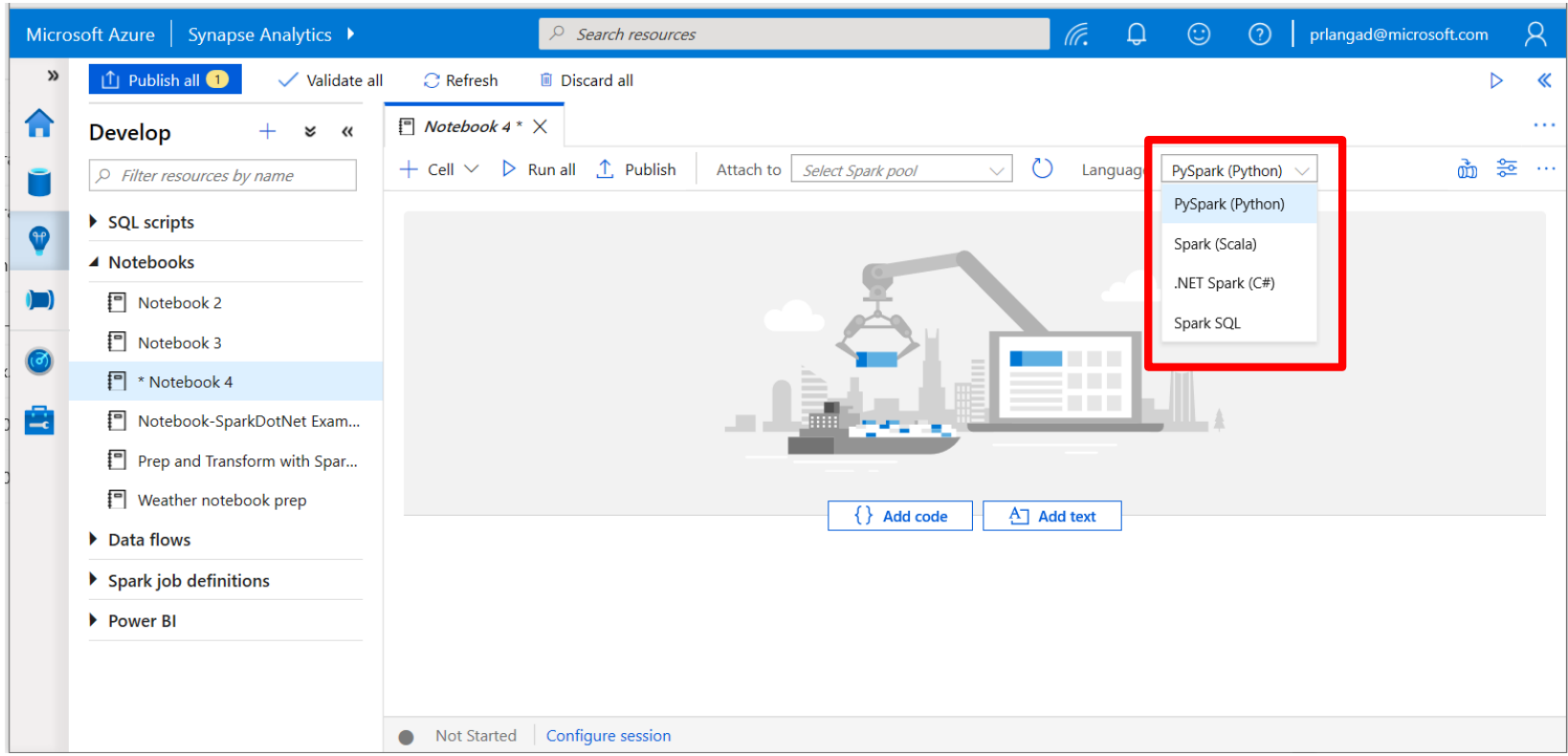
- PySpark (Python)
- Spark (Scala)
- .NET Spark (C#)
- Spark SQL

Benefits

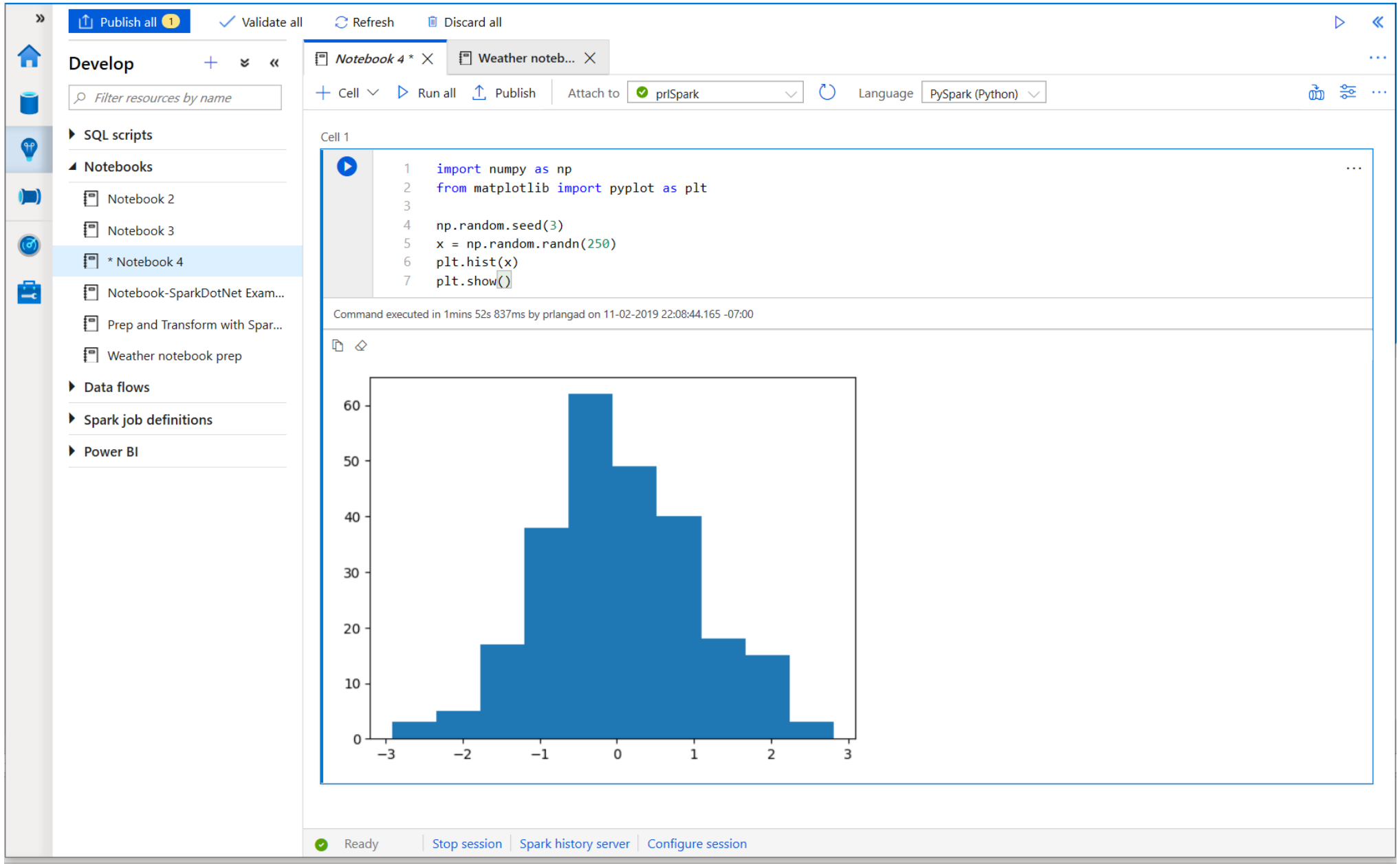
Allows to write multiple languages in one notebook

%%<Name of language>

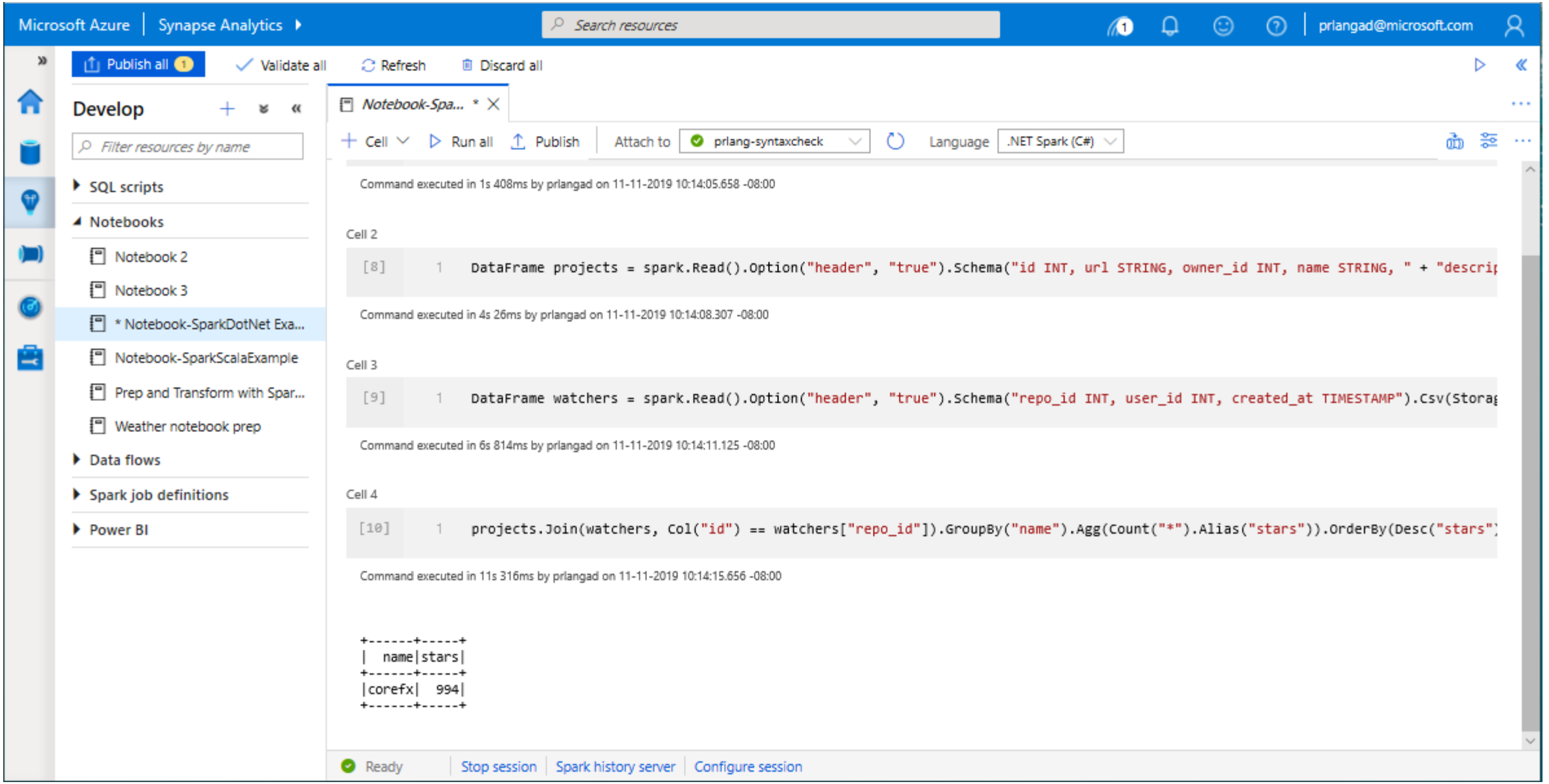
Offers use of temporary tables across languages



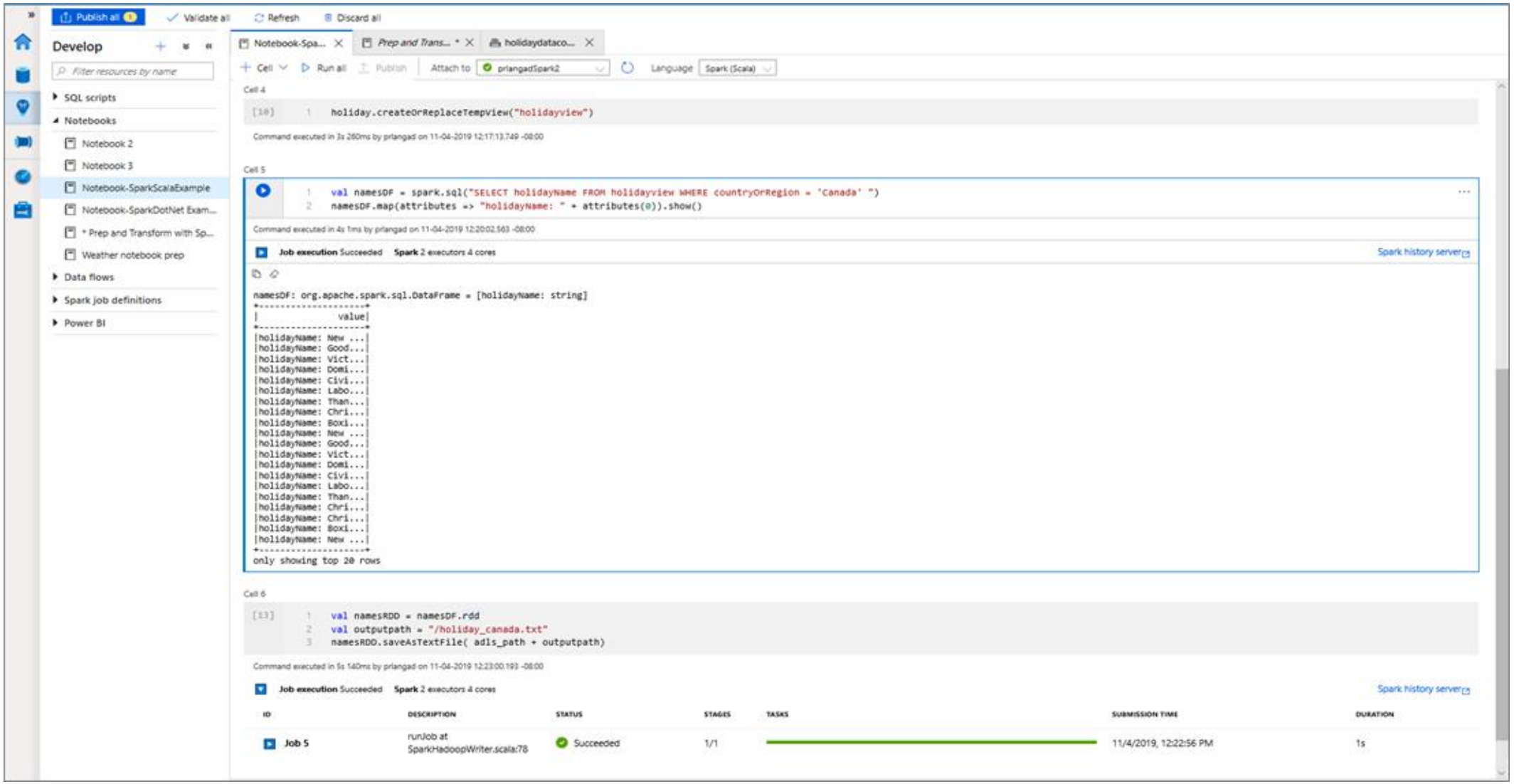
Languages – PySpark (Python)



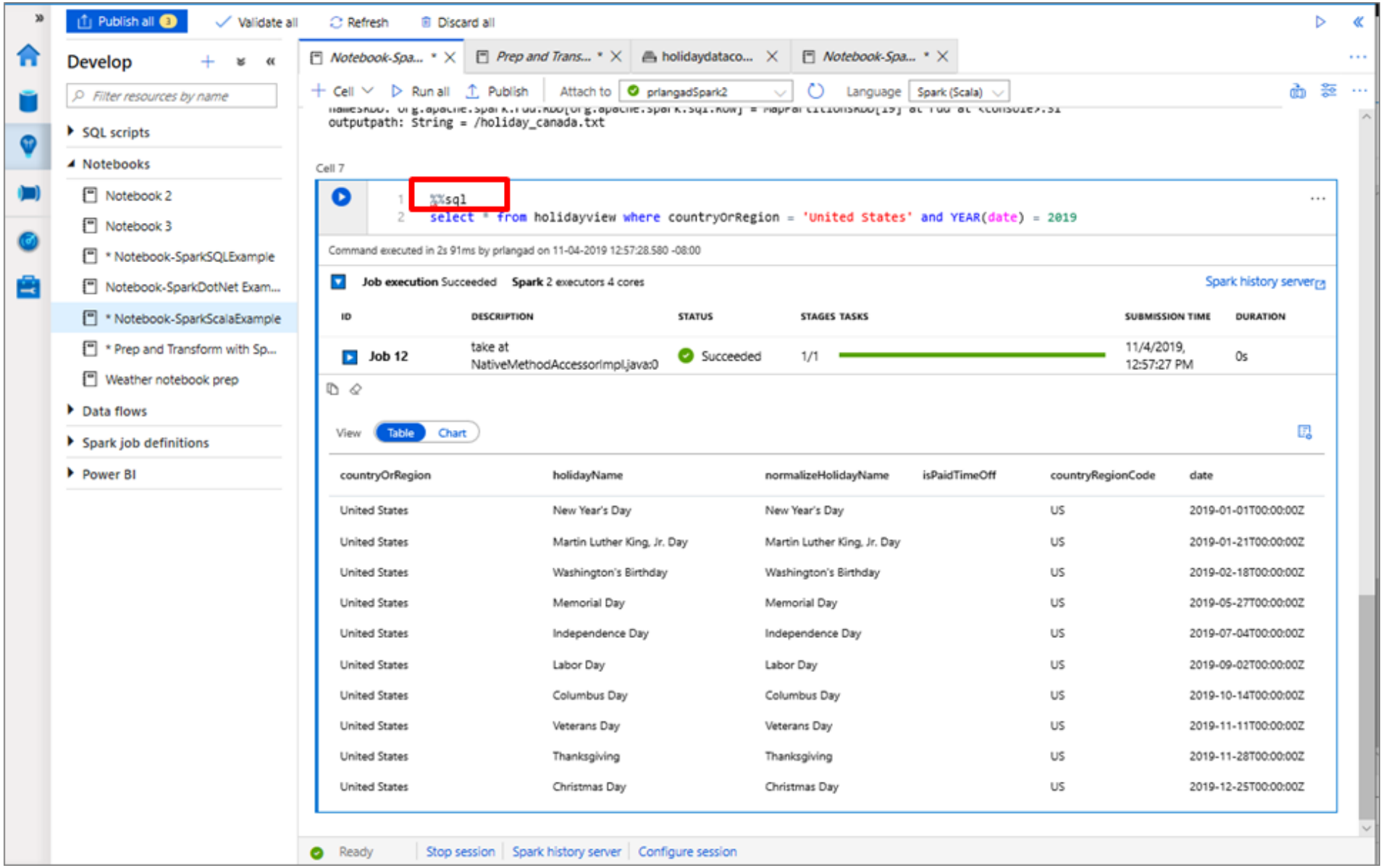
Languages – .NET Spark (C#)



Languages – Spark (Scala)



Languages – Spark SQL



Microsoft

Azure Synapse Analytics

Foundation

“Monitoring” scenarios go across a full range of experiences, personas, and activities. A unified **Monitoring Plane** supports all these scenarios.

PERSONAS	IT Pro	Data Engineer + Data Engineering Team	Advanced Data Engineer
EXPERIENCES	Azure Portal	Synapse Web UX	Ecosystem Tools
SCENARIOS	CLOUD INFRASTRUCTURE Resource Health Azure Recommendations Azure Alerts	USAGE AND ACTIVITY Running queries Historical usage	DEBUG & OPTIMIZE Job Graphs Skew Analysis

Manage – Access Control

Overview

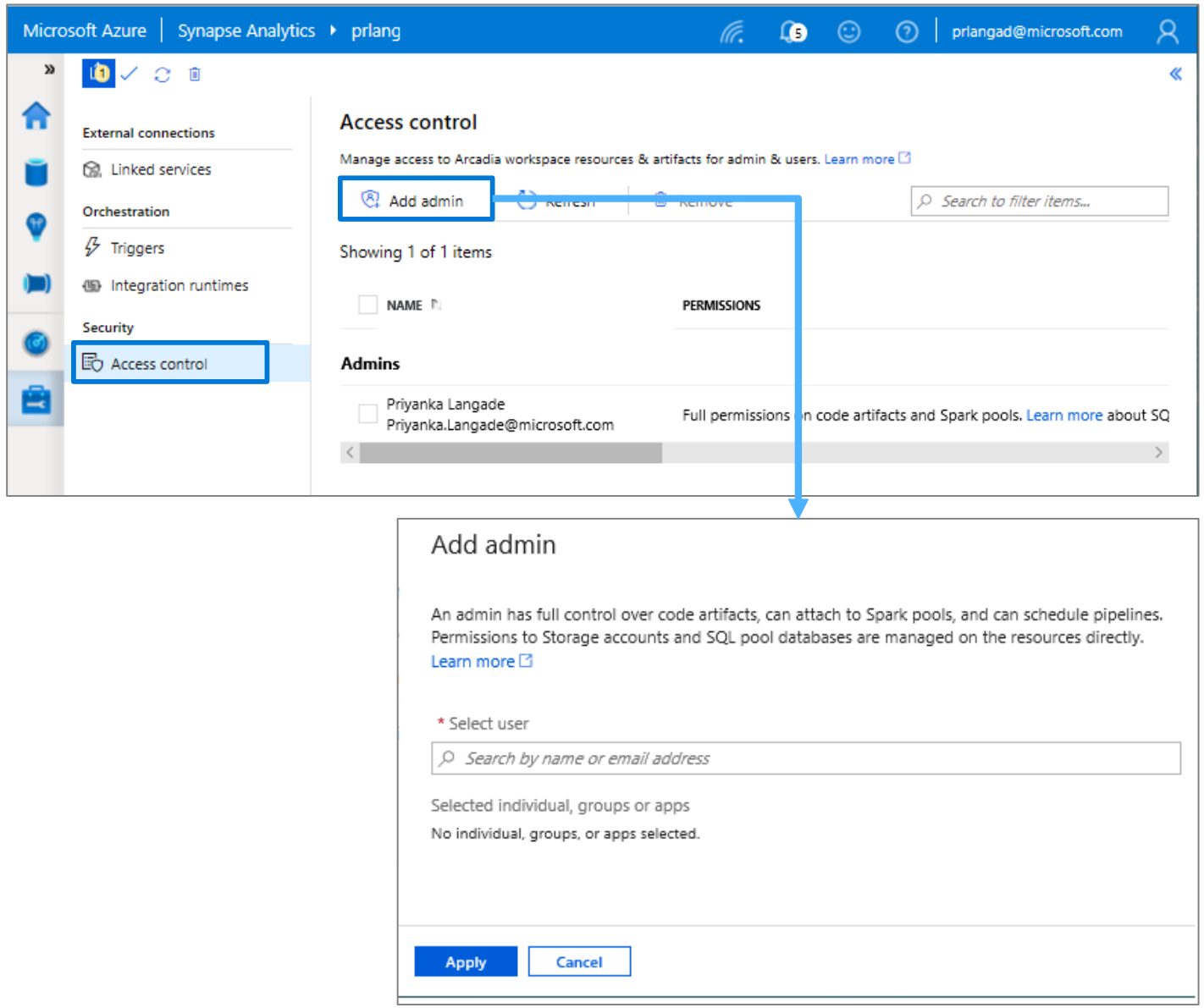
It provides access control management to workspace resources and artifacts for admin and users

Benefits

Share workspace with the team

Increases productivity

Manage permissions on code artifacts and Spark pools



Spark Monitoring

Overview

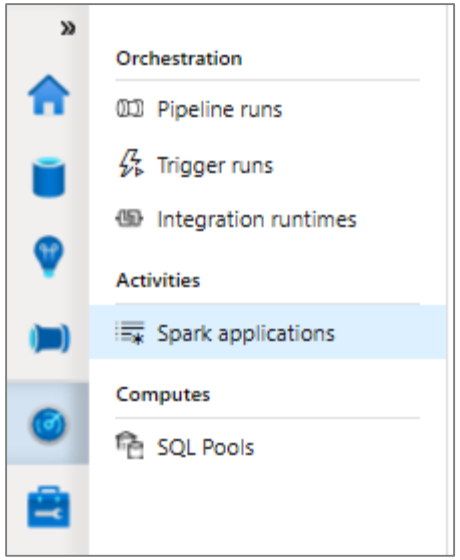
Monitor Spark pools, Spark applications for the progress and status of activities

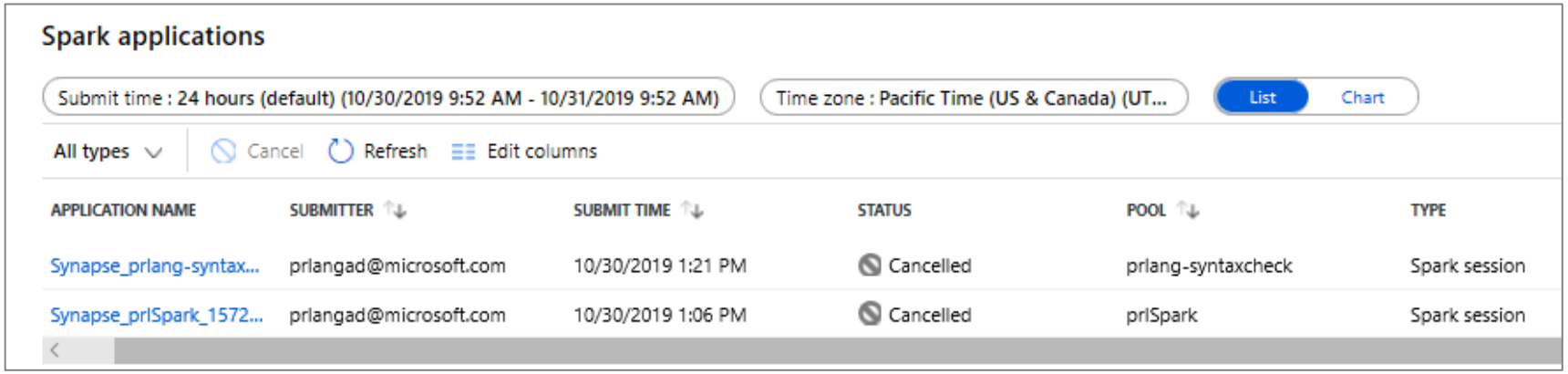
Benefits

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

Build a dashboard to monitor performance

Track the usage of resources





SQL Monitoring

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