https://github.com/joemk2/synapseanalytics
실습 파일을 다운 받아주세요.

Synapse 소개 자료는 Azure Synapse/Documents/Azure Synapse Analytics.pdf 입니다.

Azure Synapse Analytics

Contents

- Azure Synapse Analytics
- Synapse SQL
- Dedicated SQL Pool
- Serverless SQL Pool
- Aparch Spark

Azure Synapse Analytics

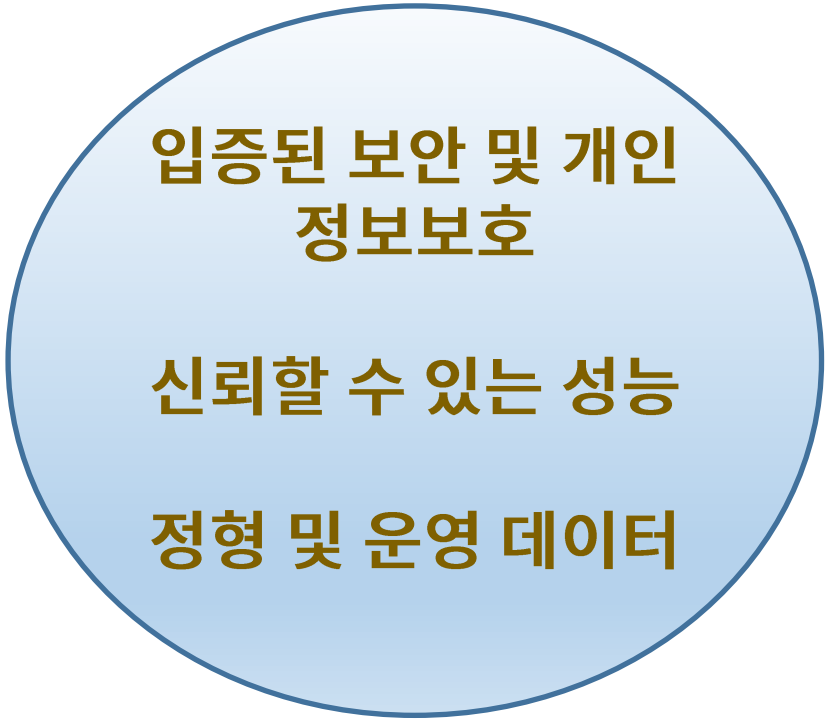
분석 플랫폼의 종류

Big Data

Data Lake

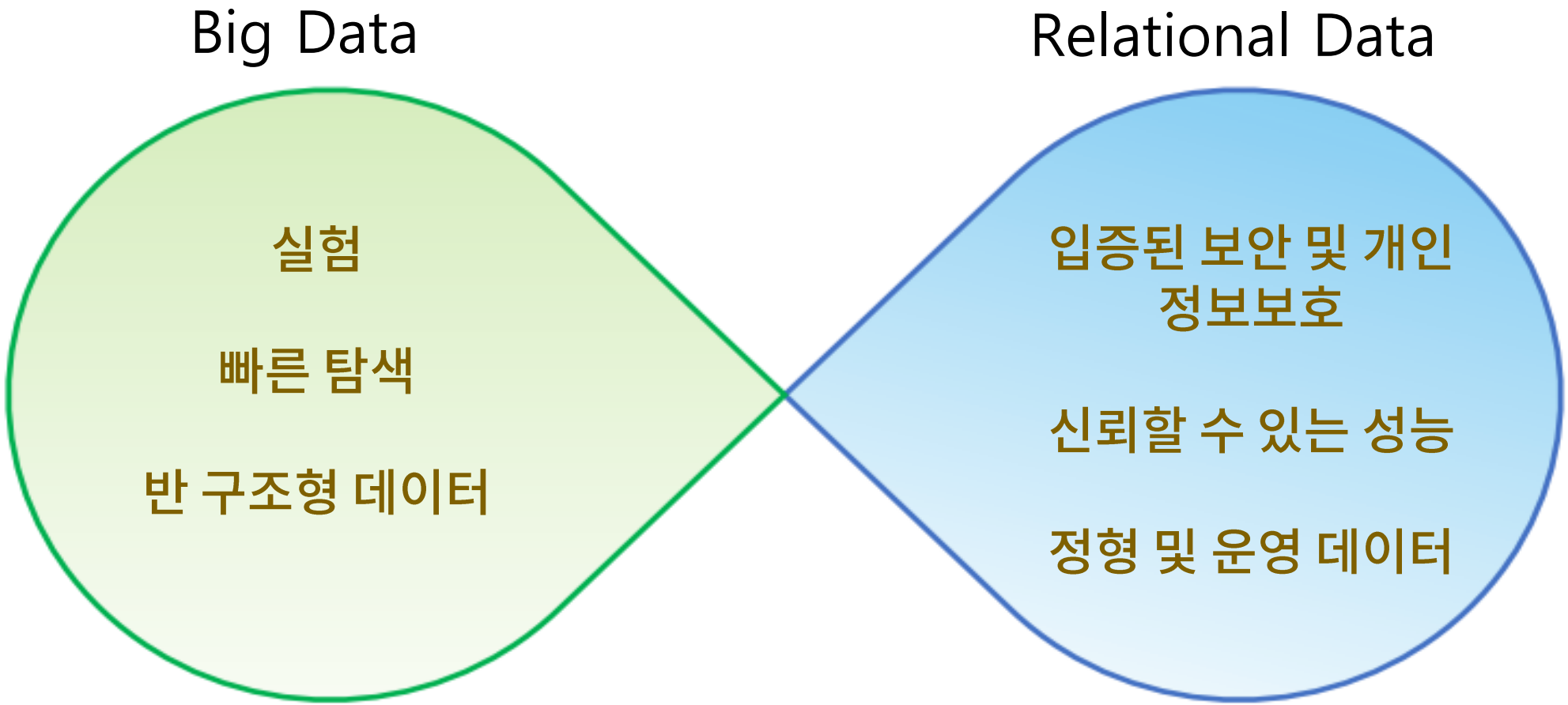
OR

Relational Data



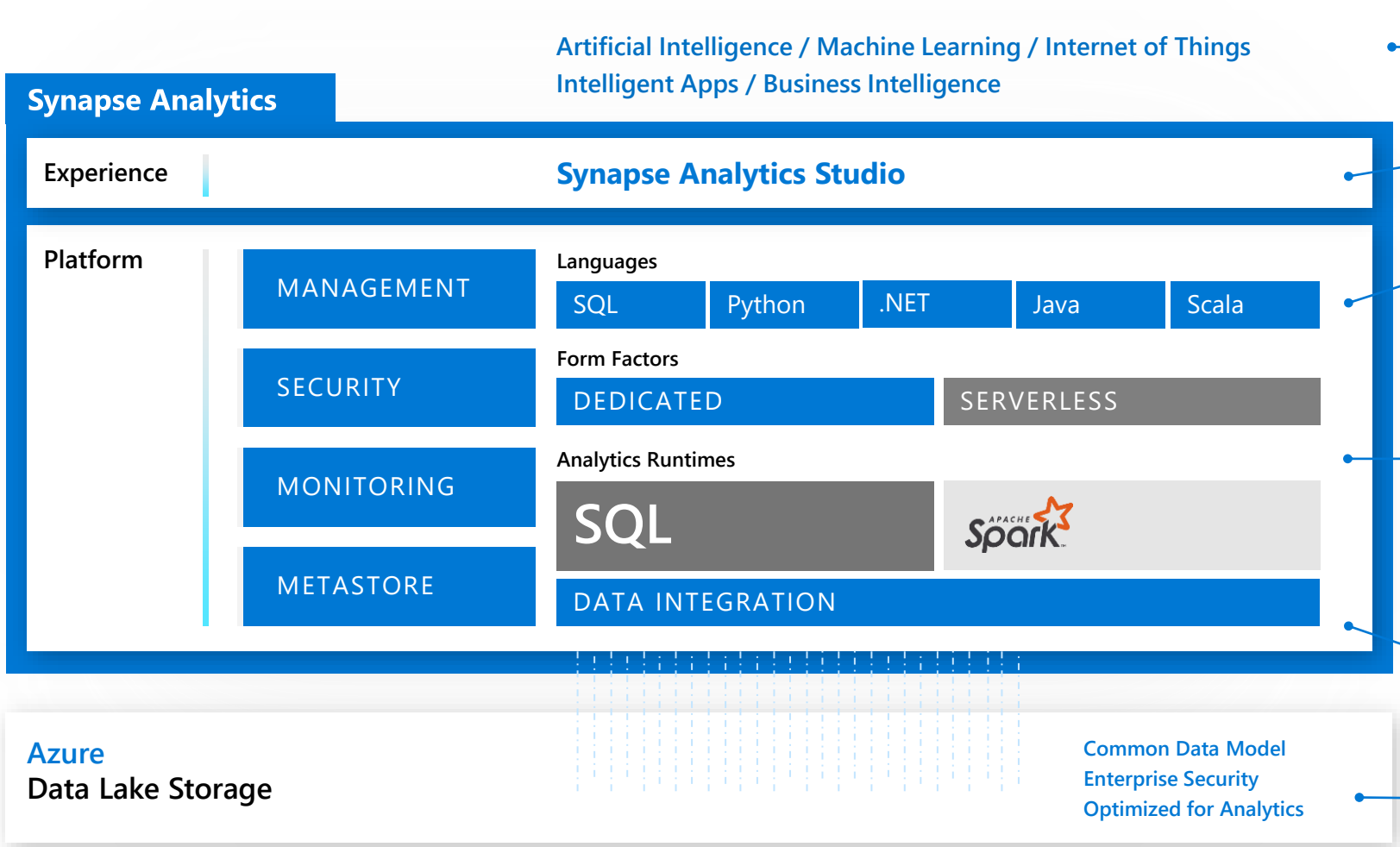
Data Warehouse

분석 플랫폼의 통합 – Azure Synapse Analytics



Big Data 분석 솔루션과 Data Warehouse 의 통합!

Synapse Analytics



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 dedicated and serverless

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

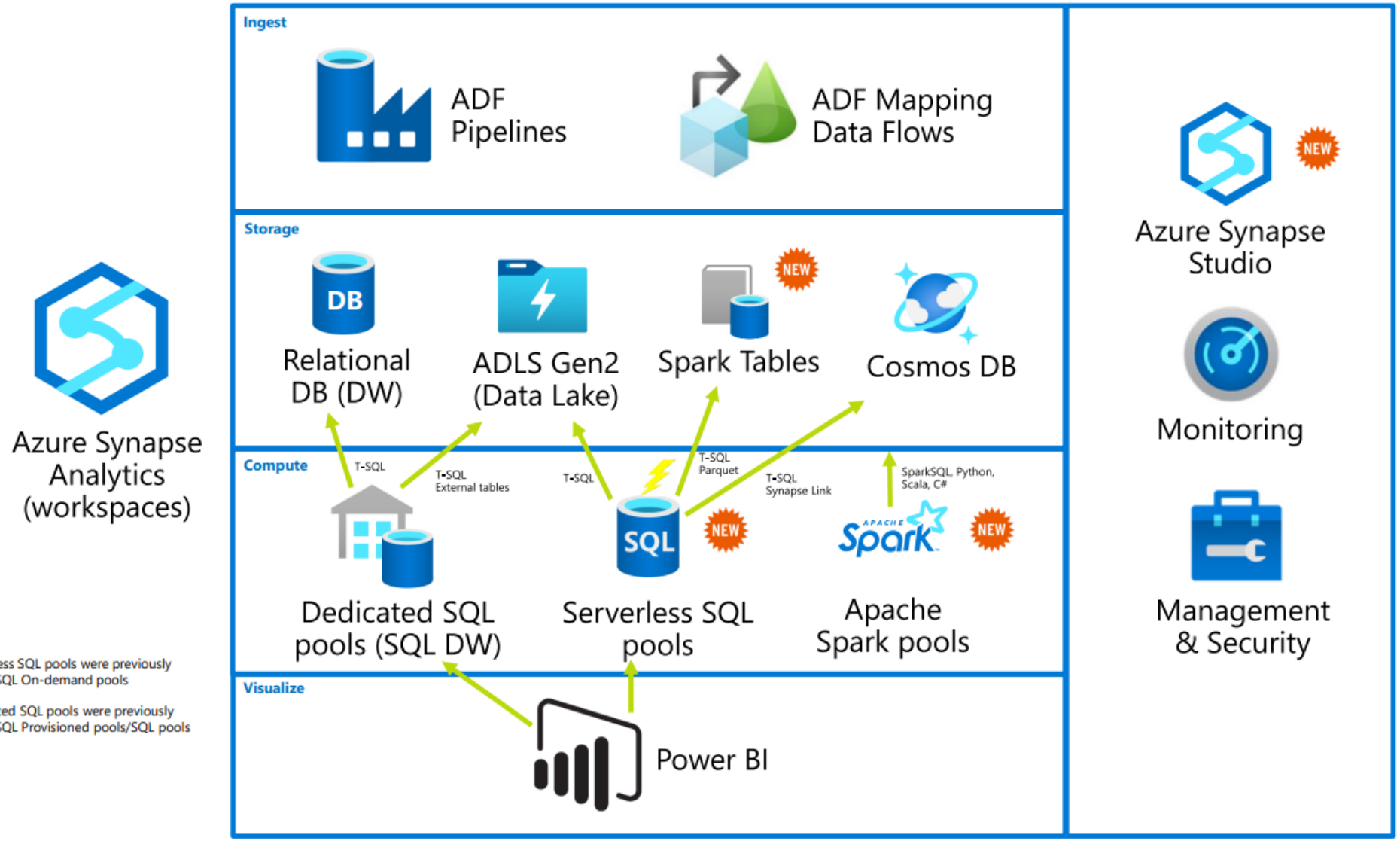
Apache Spark for big data processing with Python, Scala and .NET

Integrated **platform services** for, management, security, monitoring, and meta-store

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

Synapse 를 이용한 데이터의 수집과 분석

Synapse 를 이용한 데이터의 수집과 분석



Serverless SQL pools were previously called SQL On-demand pools

Dedicated SQL pools were previously called SQL Provisioned pools/SQL pools

Azure Synapse Analytics

Synapse SQL

Synapse SQL 특징

Rich surface area

- T-SQL을 사용하여 데이터 분석
- 엔터프라이즈급 보안

Dedicated SQL Pool

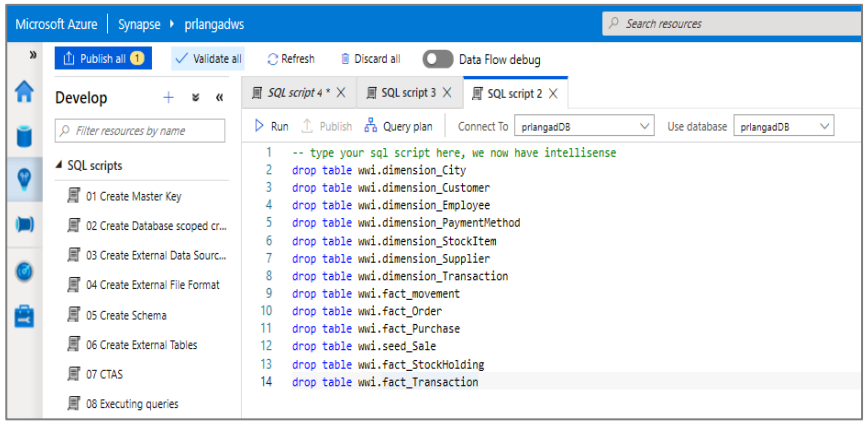
- 엔터프라이즈급 DW
- 인덱싱 & 캐싱 지원
- 외부 데이터 조회 및 적재
- 워크로드 관리

Serverless SQL Pool

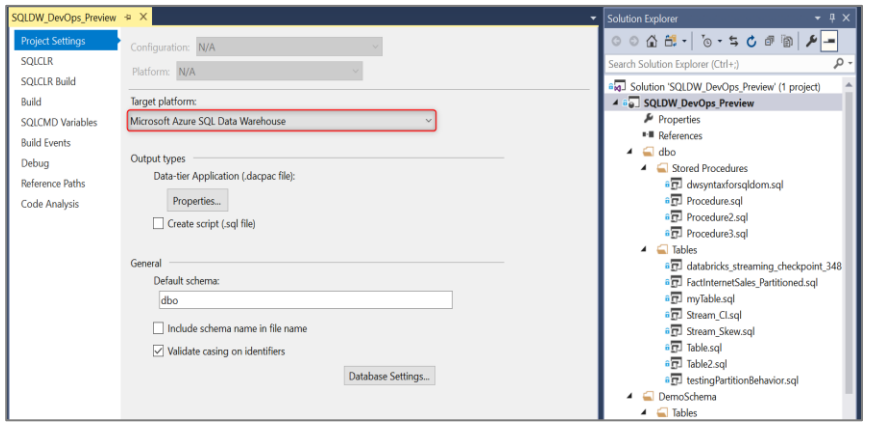
- 외부 데이터 조회
- 원시 데이터를 테이블 및 뷰로 모델화
- 쉬운 데이터 포맷 변환

Synapse SQL - Client and Tools

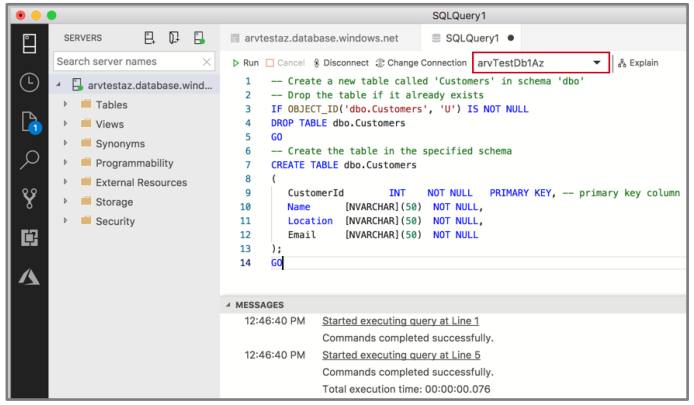
Azure Synapse Analytics



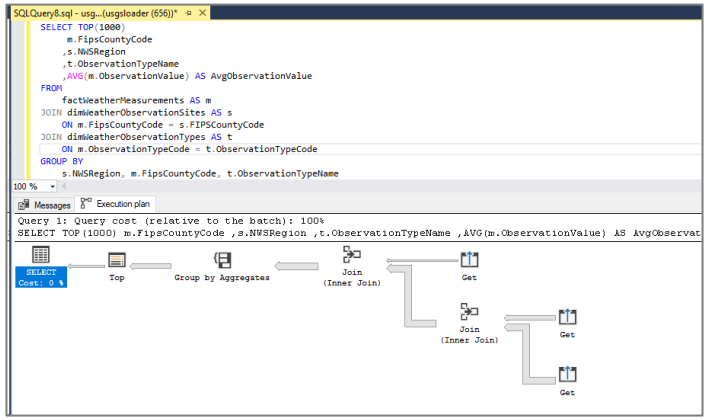
Visual Studio - SSDT database projects



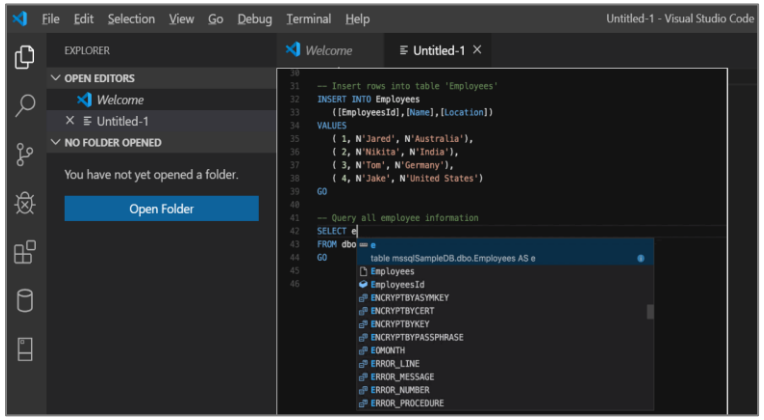
Azure Data Studio (queries, extensions etc.)



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



Visual Studio Code

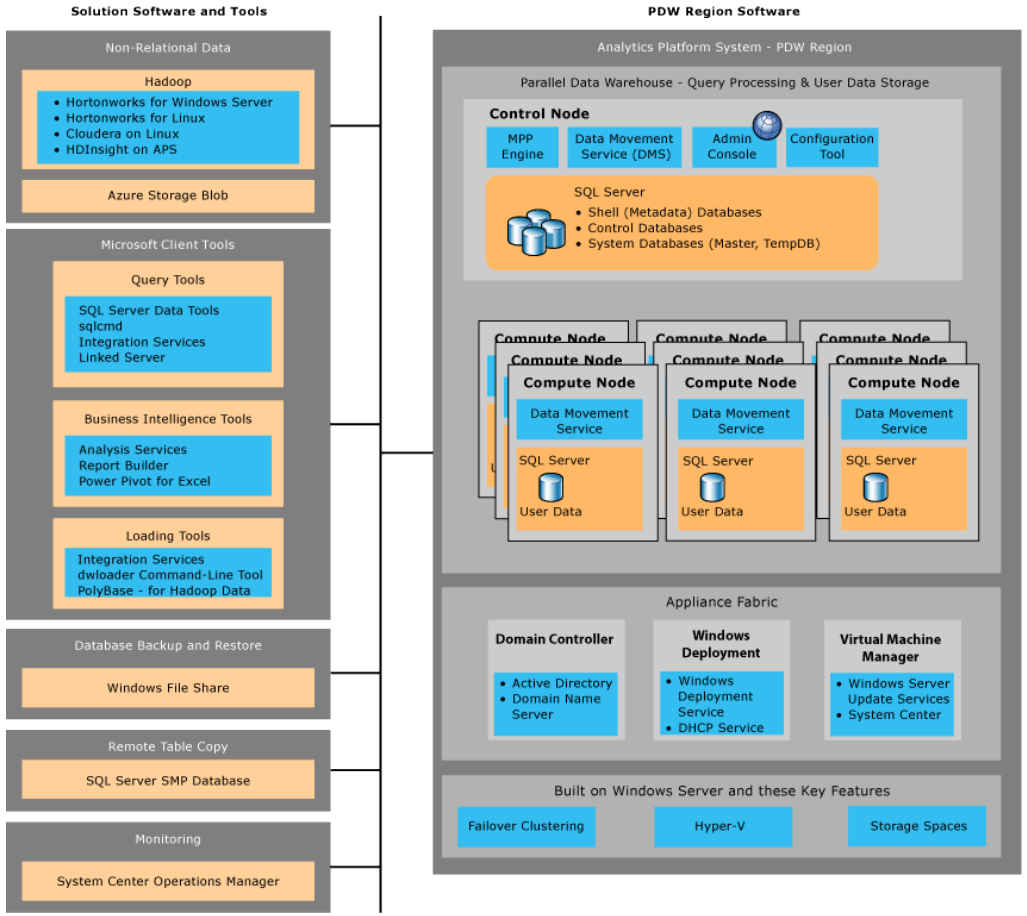


Azure Synapse Analytics

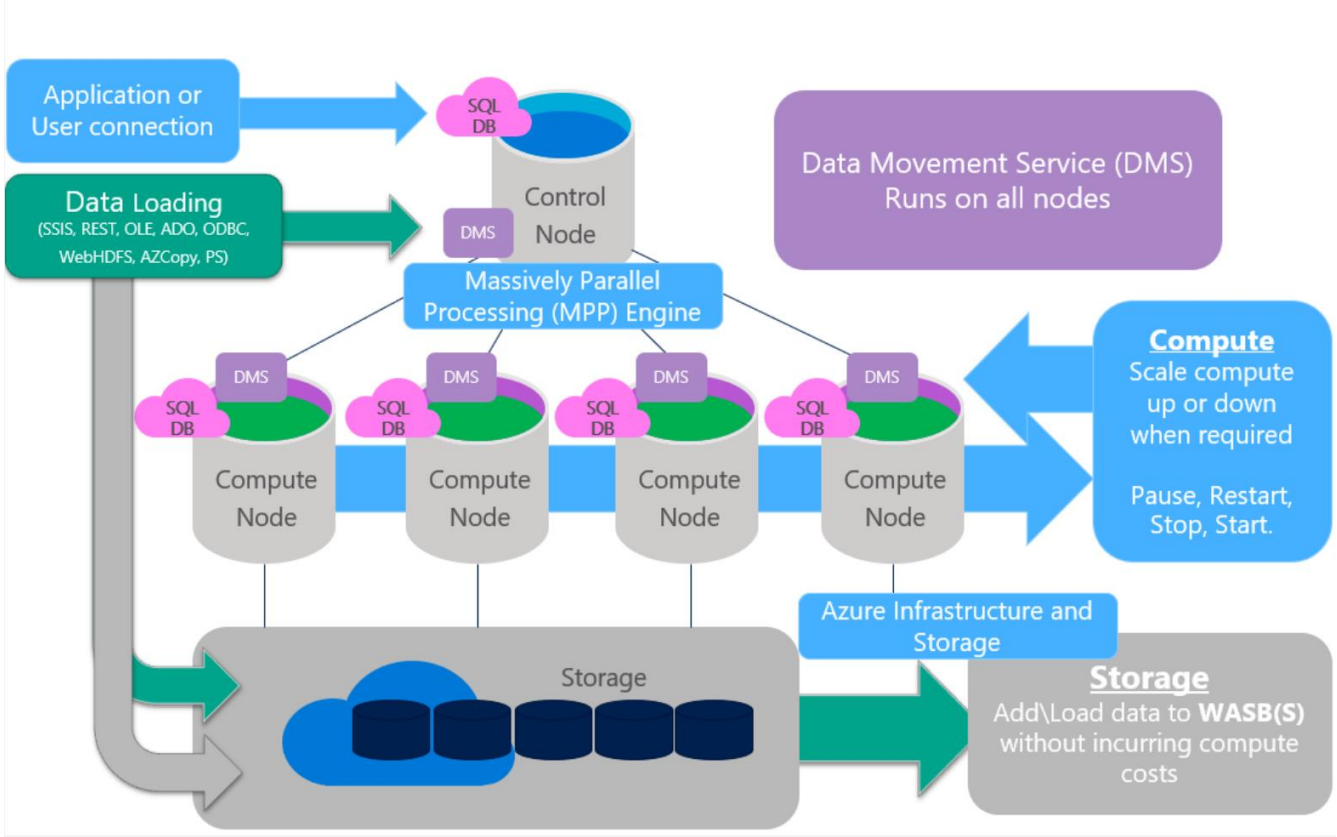
Dedicated SQL Pool

On-Premise PDW vs Synapse SQL Pool

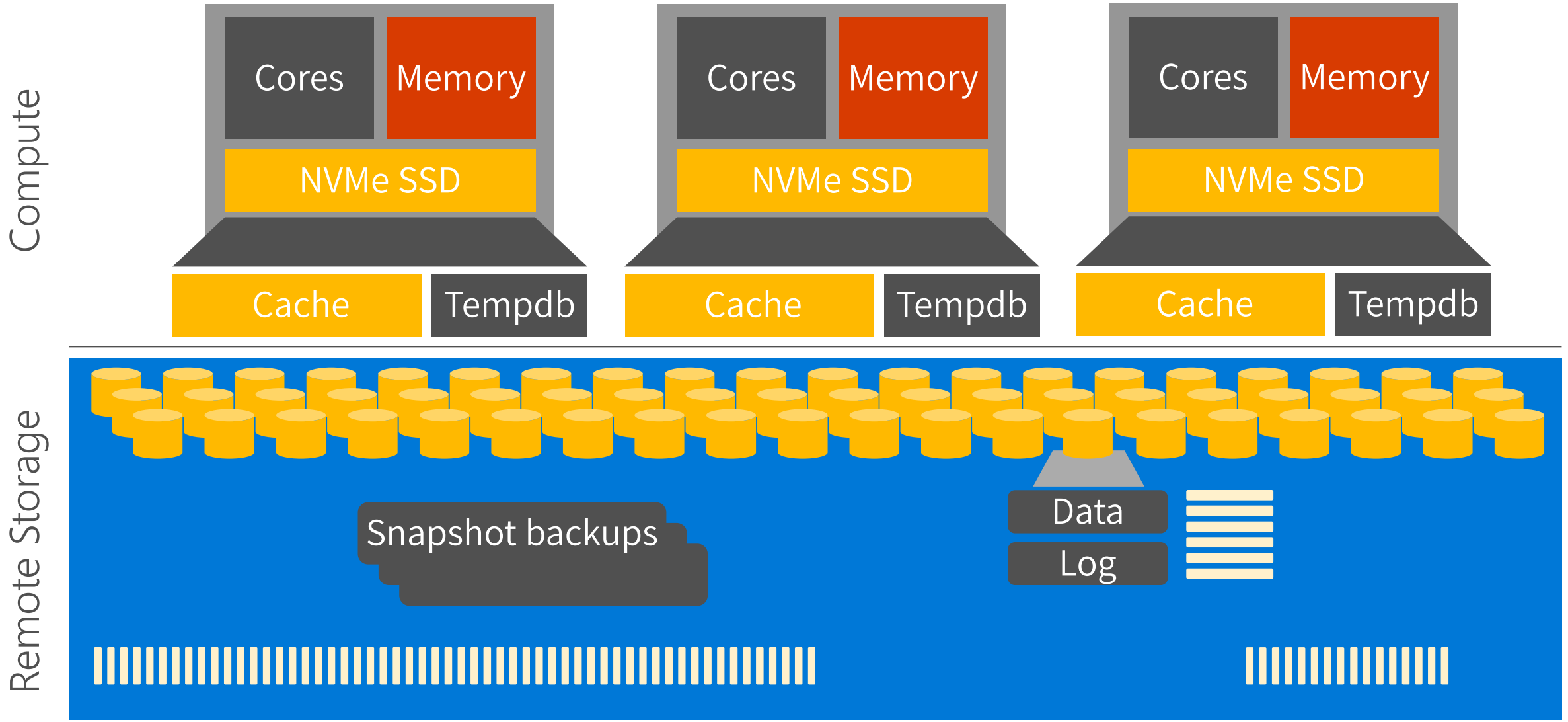
PDW Architecture (Parallel Data Warehouse)



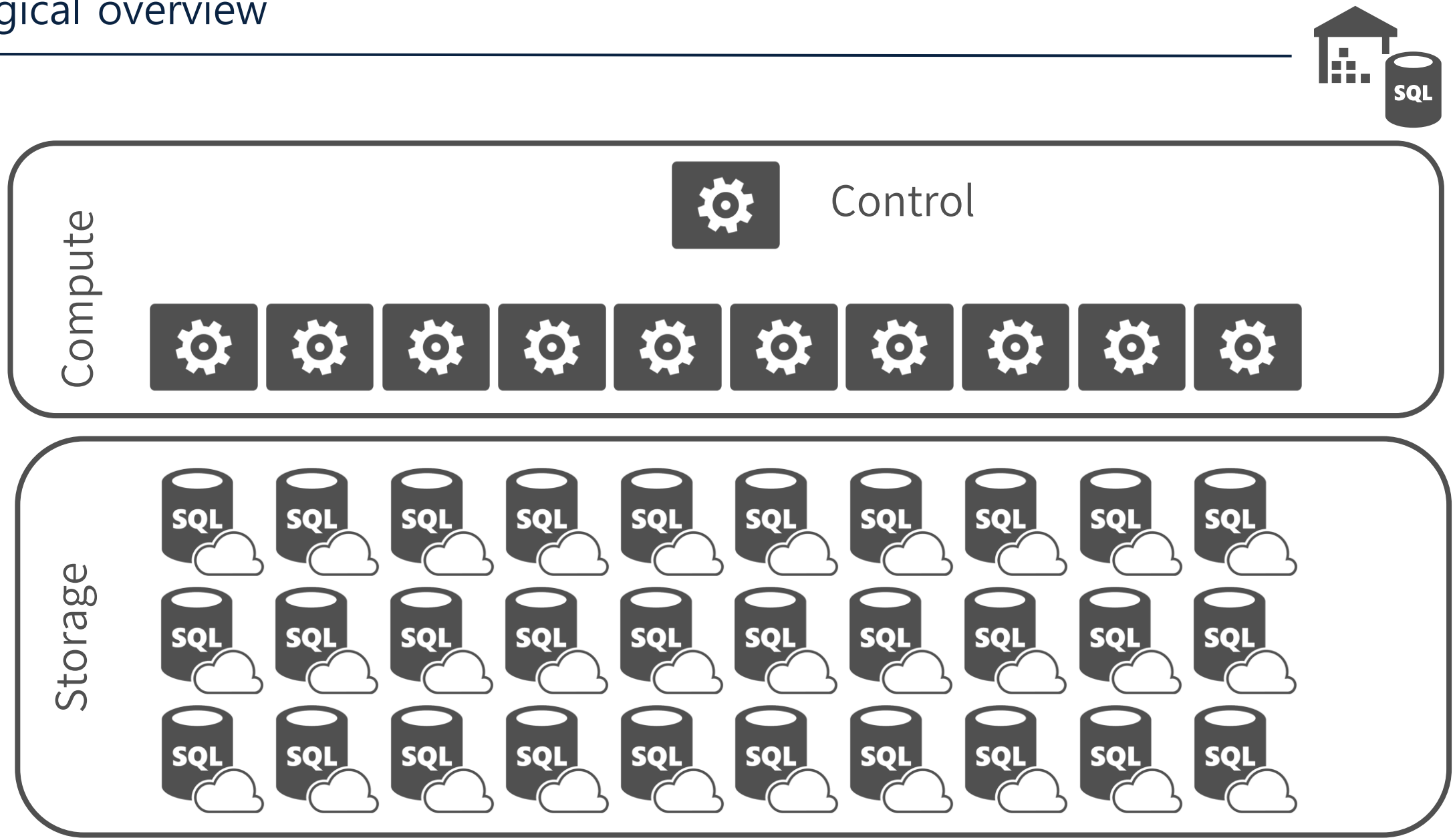
Synapse SQL Pool Architecture



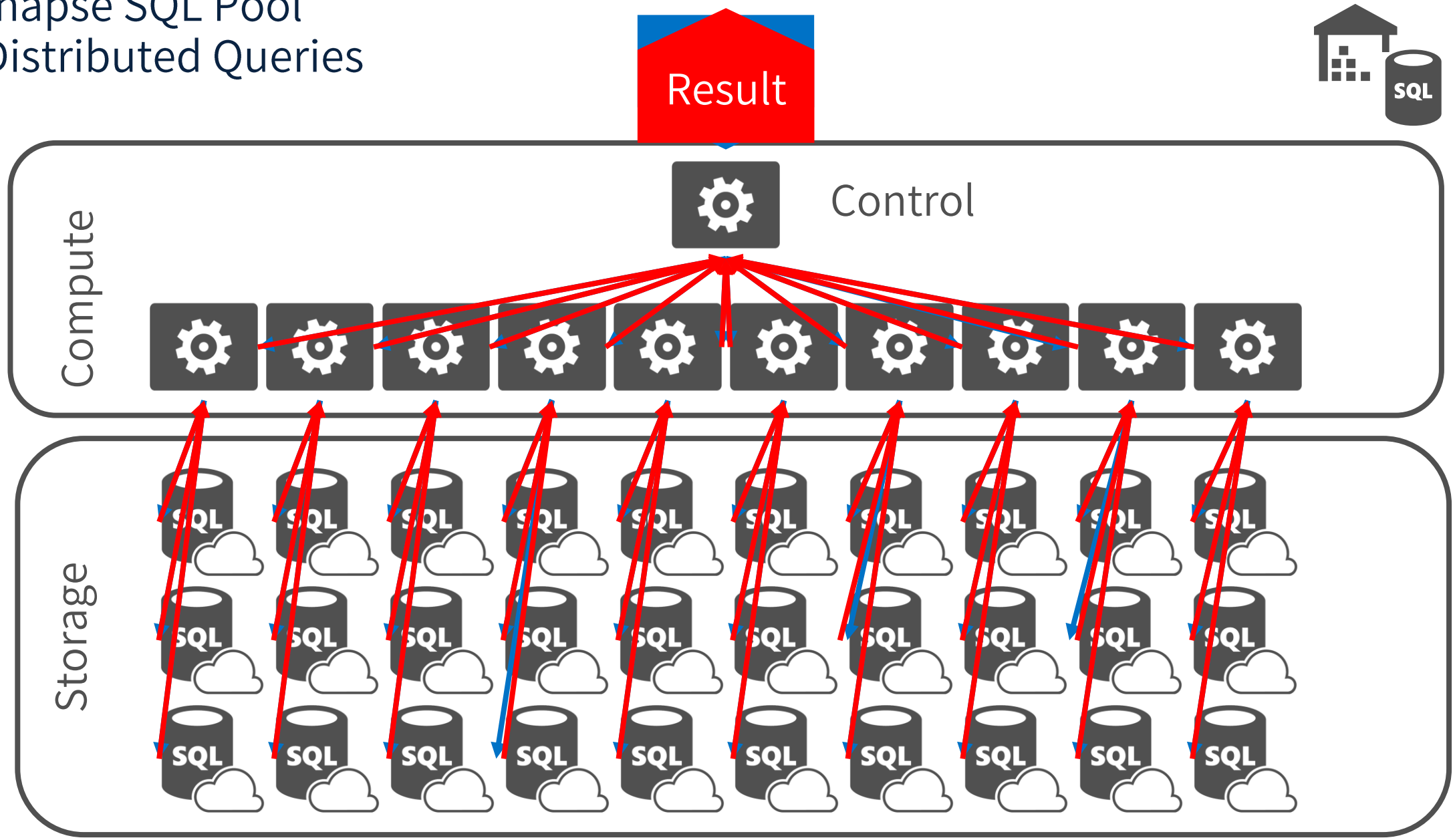
Compute, Storage 내부 구조



Logical overview



Synapse SQL Pool
- Distributed Queries



Data Warehouse Units

Normalized amount of compute

Converts to billing units i.e. what you pay

DWUc
100
200
300
400
500
600
1000
1200
1500
2000
3000
6000
...
30000

Sizing Node Count / Cpu & Memory

Node Count

Performance level	Compute nodes
DW100c	1
DW200c	1
DW300c	1
DW400c	1
DW500c	1
DW1000c	2
DW1500c	3
DW2000c	4
DW2500c	5
DW3000c	6
DW5000c	10
DW6000c	12
DW7500c	15
DW10000c	20
DW15000c	30
DW30000c	60

CPU (Hyperthreaded vCore)

Convert to DTU = DWU * 9

vCore = DTU / 100

= DWU * 9 / 100

= (6000 * 9) / 100

= 54000 / 100

= 540 vCore

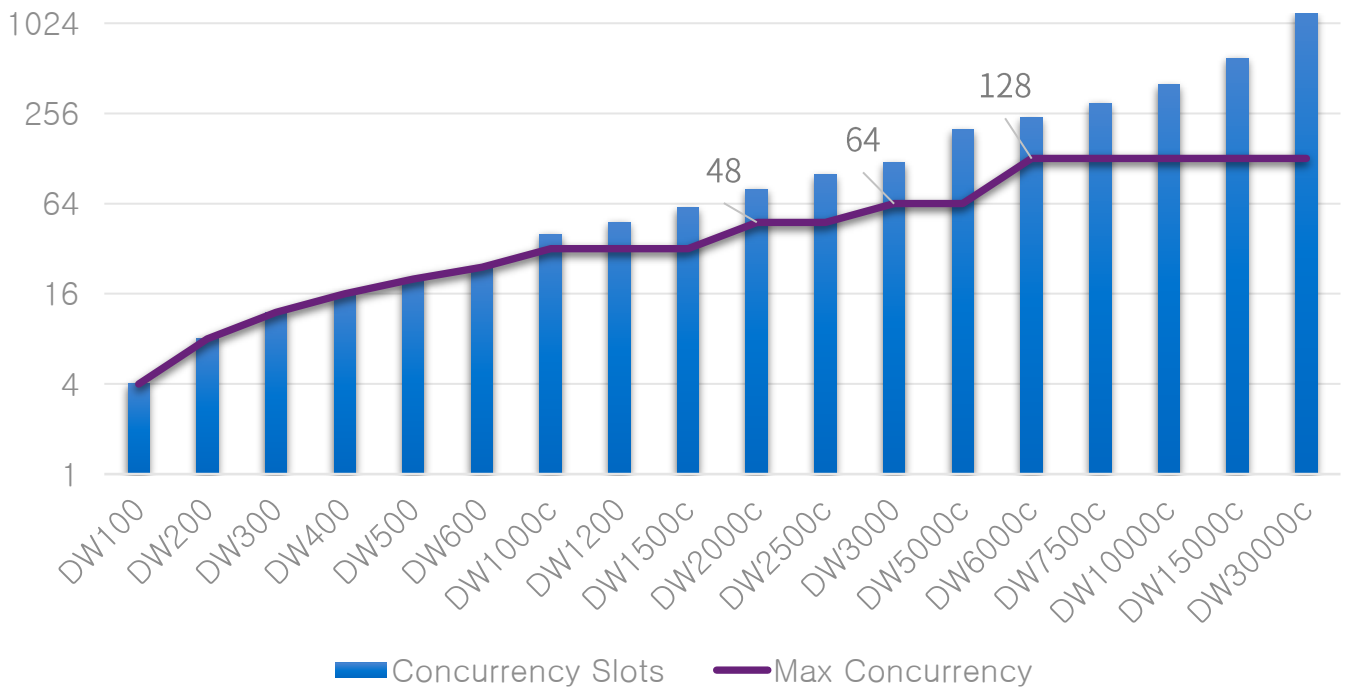
Memory

DWU * 60

(DW6000c * 60 = 3.6 Tb)

Performance level	Memory per data warehouse (GB)
DW100c	60
DW200c	120
DW300c	180
DW400c	240
DW500c	300
DW1000c	600
DW1500c	900
DW2000c	1,200
DW2500c	1,500
DW3000c	1,800
DW5000c	3,000
DW6000c	3,600
DW7500c	4,500
DW10000c	6,000
DW15000c	9,000
DW30000c	18,000

수행 가능한 동시 쿼리와 슬롯



서비스 수준	최대 동시 쿼리 수	사용 가능한 동시성 슬롯 수	쿼리당 필요한 최소의 리소스% (작업 그룹 사용시)
DW100c	4	4	25%
DW200c	8	8	12.5%
DW300c	12	12	8%
DW400c	16	16	6.25%
DW500c	20	20	5%
DW1000c	32	40	3%
DW1500c	32	60	3%
DW2000c	48	80	2%
DW2500c	48	100	2%
DW3000c	64	120	1.5%
DW5000c	64	200	1.5%
DW6000c	128	240	0.75%
DW7500c	128	300	0.75%
DW10000c	128	400	0.75%
DW15000c	128	600	0.75%
DW30000c	128	1200	0.75%

리소스 클래스

정적 리소스 클래스

- 데이터의 집합 또는 크기가 고정적일 때 사용
- 현재 성능 수준에 관계없이 동일한 양의 메모리를 할당
- 서비스 성능 수준을 높이면 더 많은 쿼리를 수행 가능

- 정적 리소스 클래스 종류
  -. Staticrc10, Staticrc20, Staticrc30, Staticrc40, Staticrc50, Staticrc60, Staticrc70, Staticrc80

동적 리소스 클래스

- 데이터 양이 증가하거나 변화하는 경우에 적합
- 서비스 수준을 높이면 쿼리에 더 많은 메모리 할당

- 동적 리소스 클래스 종류
  -. smallrc, mediumrc, largerc, xlargerc

- 리소스 클래스 메모리 할당

서비스 수준	smallrc	mediumrc	Largerc	xlargerc
DW100c	25%	25%	25%	70%
DW200c	12.5%	12.5%	22%	70%
DW300c	8%	10%	22%	70%
DW400c	6.25%	10%	22%	70%
DW500c	5%	10%	22%	70%
DW1000c – DW30000c	3%	10%	22%	70%

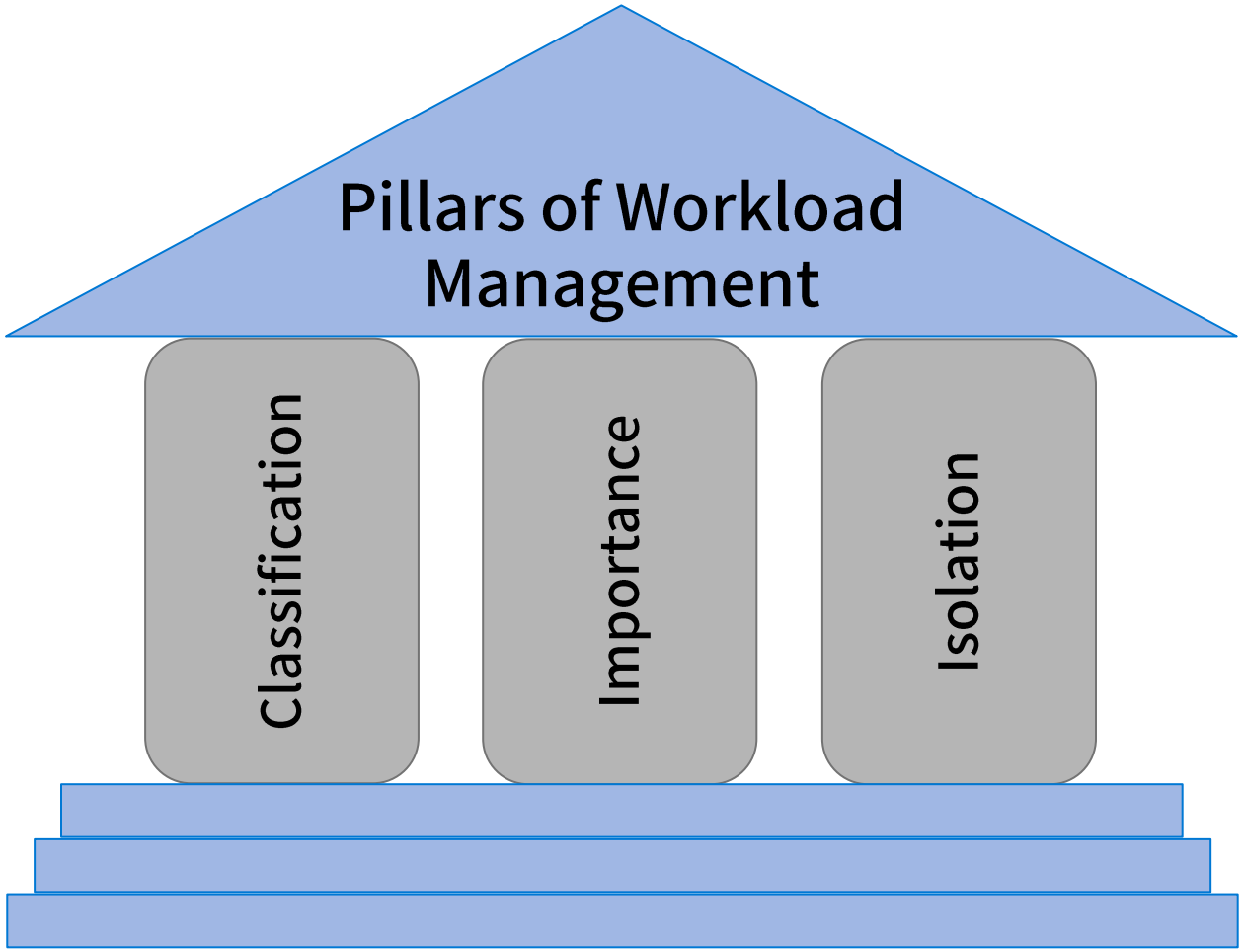
워크로드 관리

Overview

리소스의 효율적인 관리를 위해 사용

워크로드 관리 3대 요소

- 워크로드 분류 – 작업 요청을 그룹에 할당하고 중요도 설정
- 워크로드 중요도 – 그룹에 설정된 중요도에 따라 리소스의 액세스에 영향 받음
- 워크로드 독립성 – 워크로드 그룹에 리소스 예약



워크로드 관리

Overview

리소스의 효율적인 관리를 위해 사용

워크로드 관리 3대 요소

- 워크로드 분류 – 작업 요청을 그룹에 할당하고 중요도 설정
- 워크로드 중요도 – 그룹에 설정된 중요도에 따라 리소스의 액세스에 영향 받음
- 워크로드 독립성 – 워크로드 그룹에 리소스 예약

```
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*

```
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 ]
)[ ; ]
```

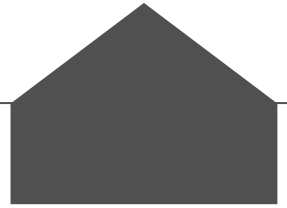
테이블 분산 옵션

	Hash Distributed	Round Robin (Default)	Replicated
분산 방법	• 해쉬 알고리즘 리턴 값에 의해 노드로 데이터 분산 • 해쉬 알고리즘에 의한 결과값이 같으면 같은 노드에 데이터 분산 • 해쉬 키 값은 Single Column Only	• 모든 노드에 데이터가 분산 • 비용 대비 단순한 저장 방식	• 모든 노드에 데이터 복제
장점	• 아래 작업 수행 시 속도가 빠름 COUNT (DISTINCT <hashed_key>) OVER PARTITION BY <hashed_key> JOIN <table_name> ON <hashed_key> GROUP BY <hashed_key>	• 초기 적재 속도가 빠름	• 쿼리 계획을 단순화하고 데이터 이동 감소 • Hash 테이블과 조인시 최적
사용 예	• Fact Table	• 초기 데이터 이관시 Data Load • 스테이징 테이블에 사용	• Dimension Table (Fact Table 을 설명해주는 Table) • Small Table
유의 사항	• 특정 노드로 데이터 쏠림 발생 • NULLS, Default 값 사용시 부적합 • Hash key 값은 update 되면 안됨	• 쿼리 수행 시 노드간 데이터 이동 발생	• 한 노드에서 두개의 복제 테이블이 조인 시 많은 공간이 소비됨

테이블 생성 옵션 : Round Robin, Hash, Replicated

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

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



```
CREATE TABLE dbo.DimCustomer
(   CustomerKey          int           NOT NULL
,   GeographyKey         int           NULL
,   CustomerAlternateKey nvarchar(15)  NOT NULL
,   Title                nvarchar(8)   NULL
,   FirstName            nvarchar(50)  NULL
,   LastName             nvarchar(50)  NULL
,   BirthDate            date          NULL
,   Gender               nvarchar(1)   NULL
,   EmailAddress         nvarchar(50)  NULL
,   YearlyIncome         money         NULL
,   DateFirstPurchase    date          NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = REPLICATED
)
;
```

테이블 Partitions

Overview

- 테이블 데이터를 작은 그룹으로 분리하는 작업
- 대부분 날짜 컬럼을 파티션 키를 사용
- 모든 테이블에 파티션 기능 지원

RANGE RIGHT – Used for time partitions

RANGE LEFT – Used for number partitions

Benefits

- 데이터 로딩, 조회에 효율성 및 속도 증가
- 데이터 조회 시, 불필요한 스캔의 감소와 이로 인한 I/O 의 감소로 상당한 쿼리 속도 증가

```
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'
    )
  )
);
```

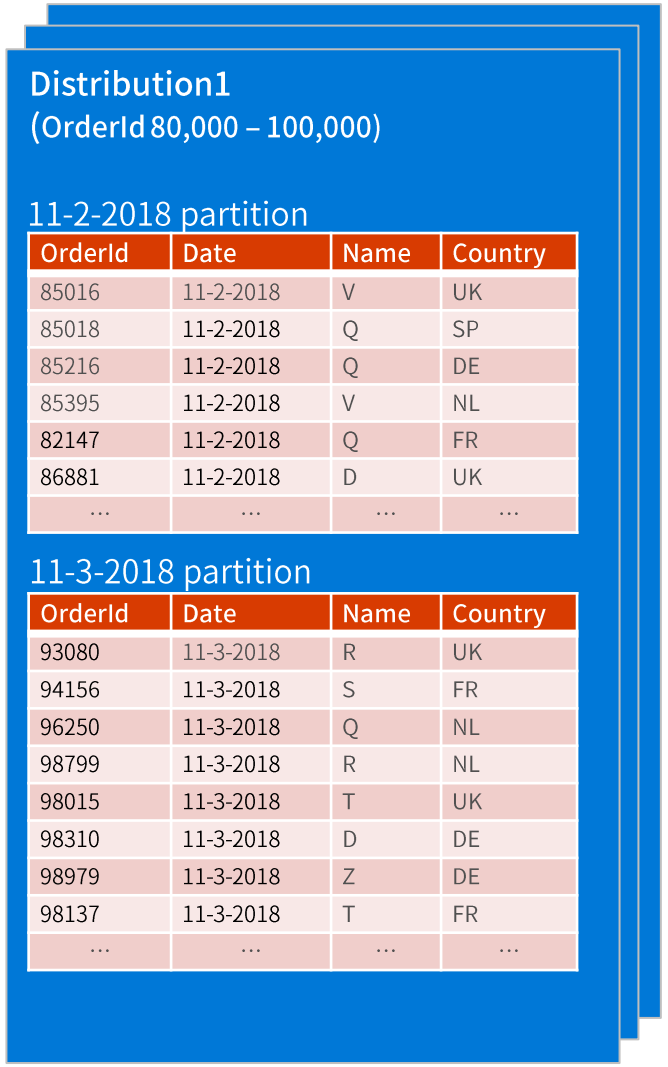
테이블 Partitions

논리적 Table 구조

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
…	…	…	…

물리적 Data distribution

(Hash distribution (OrderId), Date partitions)



…

x 60 distributions (shards)

- 각 Shard 는 같은 Date로 파티션으로 되어있음
- 최적의 압축과 Clustered Columnstore table의 퍼포먼스 향상을 위해서는 각 파티션에 최소 100만건의 row 가 필요

Index

Clustered Columnstore index (Default Primary)

- 최적의 데이터 압축 지원
- 최고의 쿼리 퍼포먼스 지원

Clustered index (Primary)

- 단일 행, 극소수 행을 조회할 때 최적

(Multi-Column Index 가능, But 1개 이상의 Index 생성 불가)

Heap (Primary)

- 임시 데이터를 Loading 시 최적
- 사이즈가 작은 Look up Table 조회 시 사용

Nonclustered indexes (Secondary)

- Table에 Multiple Columns Index 를 허용
- Single Table에 여러개의 Multiple Non-Clustered 허용
- Primary Indexes 대안으로 Non-Clustered Indexes 생성 가능
- Lookup 테이블 조회에 성능 기대

Clustered Columnstore index order

- 정렬된 Clustered Columnstore Index를 사용하면 테이블 조회 시
- 필요한 세그먼트만 읽음으로써 쿼리 시간을 크게 단축 가능

```
-- 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);


-- When Create Clustered Columnstore index order
CREATE CLUSTERED COLUMNSTORE INDEX ORDER (OrderId);
```

Azure Synapse Analytics

Serverless SQL Pool

Serverless SQL Pool 특징

Quick data exploration

- Azure storage 내부 파일의 스키마와 데이터 조회가 가능
- Parquet, CSV, JSON 파일 포맷 지원
- BI Tool 을 위해 Azure storage 에 액세스 할 수 있는 Direct connector 지원

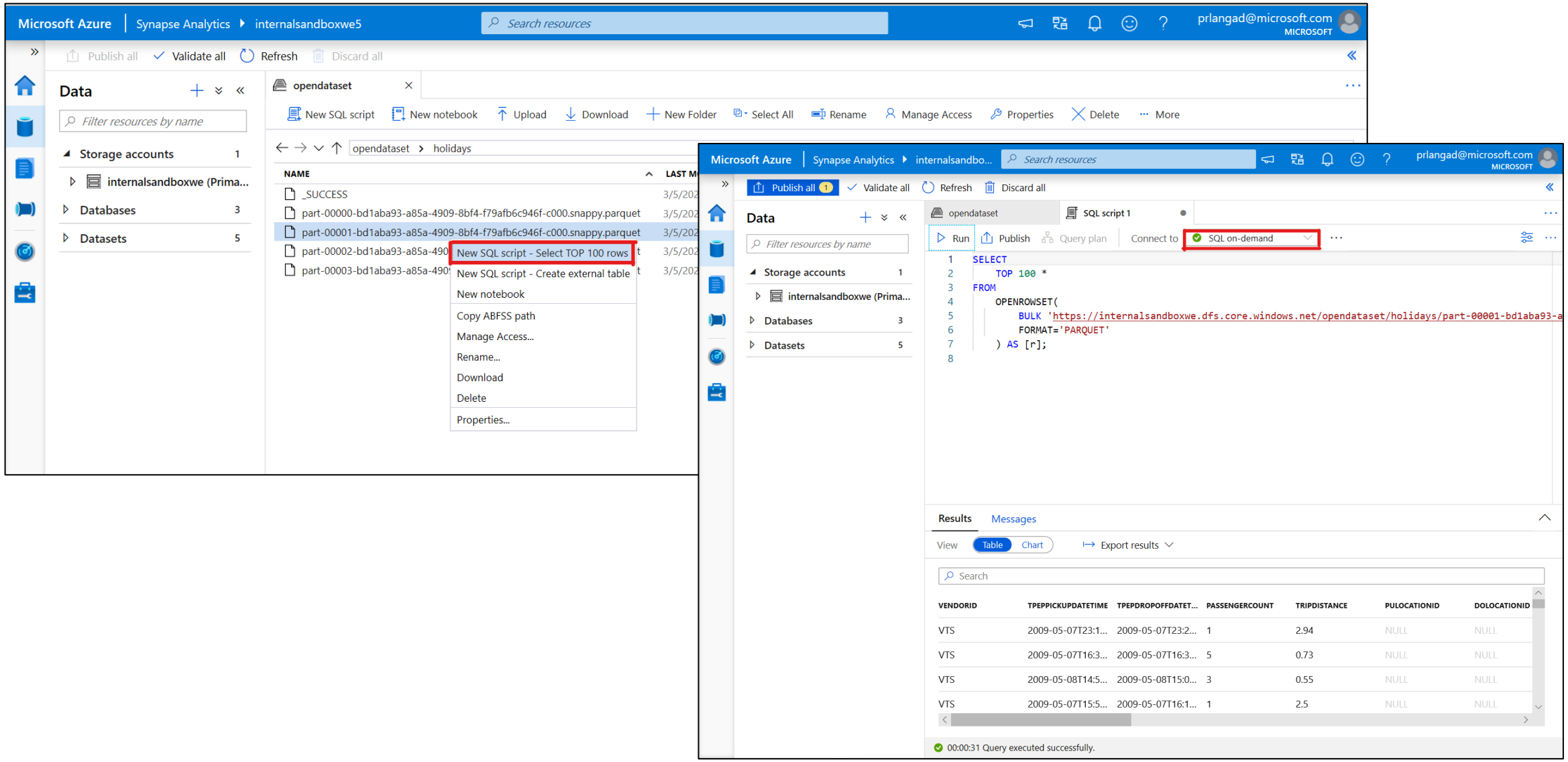
Logical Data Warehouse

- Azure storage 내부 raw files 을 virtual tables 와 views 로 생성 가능
- SQL 을 사용하는 Tool 을 이용하여 file 을 분석 가능
- enterprise-grade security model

Easy data transformation

- CSV 을 parquet 포맷으로 변경 지원
- 컨테이너와 Storage 계정들을 오가며 데이터 이동 가능
- 외부 스토리지에 대한 쿼리 결과를 저장 가능

Storage file의 쉬운 탐색 기능



다양한 파일 포맷의 쿼리 가능

Overview

다양한 파일 포맷으로 저장된 데이터를 조회하기 위해 OPENROWSET 함수를 사용한다

Benefits

- CSV, parquet, and JSON 파일 조회 가능
- 모든 파일에 대해 통합된 T-SQL 사용 가능
- 데이터의 변환과 조회 결과 데이터 분석을 위해 standard SQL 사용

- Use JSON functions to get the data from underlying files.
- Use JSON functions to get data from PARQUET nested types

```
SELECT TOP 10 *
FROM OPENROWSET(
        BULK 'https://XYZ.blob.core.windows.net/csv/taxi/*.csv',
        FORMAT = 'CSV')
WITH (
          country_code VARCHAR(4),
          country_name VARCHAR(50),
          year INT,
          population INT
          ) AS nyc
```

```
SELECT TOP 10 *
FROM OPENROWSET(
        BULK 'https://XYZ.blob.core.windows.net/csv/taxi/*.parquet',
        FORMAT = 'PARQUET') AS nyc
```

```
SELECT TOP 10 *
   JSON_VALUE(jsonContent, '$.countryCode') AS country_code,
   JSON_VALUE(jsonContent, '$.countryName') AS country_name
   JSON_VALUE(jsonContent, '$.year')  AS year
   JSON_VALUE(jsonContent, '$.population')  AS population
FROM  OPENROWSET(
      BULK 'https://XYZ.blob.core.windows.net/json/taxi/*.json',
      FORMAT='CSV',
      FIELDTERMINATOR ='0x0b',
      FIELDQUOTE = '0x0b',
      ROWTERMINATOR = '0x0b'
   )
   WITH ( jsonContent varchar(MAX)   ) AS json_line
```

	country_code	country_name	year	population
1	LU	Luxembourg	2017	594130

스키마 타입 자동 정의

Overview

OPENROWSET 은 외부 파일의 컬럼과 컬럼의 데이터 타입을 자동으로 정의한다.

```
SELECT TOP 10 *
FROM OPENROWSET(
        BULK 'https://XYZ.blob.core.windows.net/csv/taxi/*.parquet',
        FORMAT = 'PARQUET') AS nyc
```

Benefits

No need to up-front analyze file structure to query the file

OPENROWSET identifies columns and their types based on underlying file metadata.

Perfect solution for data exploration where schema is unknown.

The functionality is available for both parquet & CSV files.

	country_code	country_name	year	population
1	LU	Luxembourg	2017	594130

```
SELECT
  TOP 100 *
FROM
  OPENROWSET(
    BULK 'https://azuresynapsesa.dfs.core.windows.net/default/RetailData/StoreDemoGraphics.csv',
    FORMAT = 'CSV',
    PARSER_VERSION='2.0',
HEADER_ROW = TRUE) AS [result]
```

StoreId	RatioAge60	CollegeRatio	Income	HighIncome15...	LargeHH	MinoritiesRatio	More1FullTime...	DistanceNeare...	SalesN
2	0.232864734	0.248934934	10.55320518	0.463887065	0.103953406	0.114279949	0.303585347	2.110122129	1.1428
5	0.117368032	0.32122573	10.92237097	0.535883355	0.103091585	0.053875277	0.410568032	3.801997814	0.6818

스키마 타입 수동 정의

Overview

사용자가 쿼리의 WITH 절에 컬럼명과 컬럼의 데이터 타입을 명시할 수 있다.

Benefits

Define result schema at query time in WITH clause.

No need for external format files.

Explicitly define exact return types, their sizes, and collations.

Improve performance by column elimination in parquet files.

```
SELECT TOP 10 *
FROM OPENROWSET(
        BULK 'https://XYZ.blob.core.windows.net/csv/taxi/*.csv',
        FORMAT = 'CSV')
WITH (
        country_code VARCHAR(4),
        country_name VARCHAR(50),
        year INT,
        population INT
        ) AS nyc
```

	country_code	country_name	year	population
1	LU	Luxembourg	2017	594130

사용자 의한 구문 분석 정의

Overview

OPENROWSET 함수를 이용하여 사용자가 파일의 구분자와
원하는 컬럼만 선택적으로 명시하여 결과 값을 불러올 수 있습니다.

Benefits

Ability to read CSV files with custom format

- With or without header row
- Handle any new-line terminator (Windows or Unix style)
- Use custom field terminator and quote character
- Read UTF-8 and UTF-18 encoded files
- Use only a subset of columns by specifying column position after column types

```
SELECT *
FROM OPENROWSET(
      BULK 'https://XYZ.blob.core.windows.net/csv/population/population.csv',
      FORMAT = 'CSV',
      FIELDTERMINATOR =',',
      ROWTERMINATOR = '\n'
    )
WITH (
    [country_code] VARCHAR (5) 2,
    [country_name] VARCHAR (100) 4,
    [year] smallint 7,
    [population] bigint 9
) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017
```

Second, fourth, seventh and ninth columns are returned

	country_code	country_name	year	population
1	LU	Luxembourg	2017	594130

와일드 카드(*) 를 사용한 다수의 파일 조회

Overview

OPENROWSET 함수에서 WildCards 를 사용하여
여러 폴더안에 있는 다수의 파일을 동시에 질의하여
결과를 얻을 수 있습니다.

```
SELECT YEAR(pickup_datetime) as [year],
        SUM(passenger_count) AS passengers_total,
        COUNT(*) AS [rides_total]
FROM OPENROWSET(
       BULK 'https://XYZ.blob.core.windows.net/csv/taxi/year=*/month=1/*.parquet',
       FORMAT = 'PARQUET')  AS nyc
GROUP BY YEAR(pickup_datetime)
ORDER BY YEAR(pickup_datetime)
```

Benefits

Offers reading multiple files/folders through usage of wildcards

Offers reading specific file/folder

Supports use of multiple wildcards

	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

폴더 구조를 이용한 일부의 데이터 조회

Overview

OPENROWSET 함수를 사용하여 여러 개의 파일로부터 원하는 데이터만을 조회할 수 있습니다.

Benefits

Use filepath() function to access actual values from file paths.

Eliminate sub-folders/partitions before the query starts execution

Query Spark/Hive partitioned data sets

```
SELECT
    r.filepath(1) AS [year]
   ,r.filepath(2) AS [month]
   ,COUNT_BIG(*) AS [rows]
FROM OPENROWSET(
       BULK 'https://XYZ.blob.core.windows.net/year=*/month=*/*.parquet',
       FORMAT = 'PARQUET') 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
```

year	month	rows
2017	10	9768815
2017	11	9284803
2017	12	9508276

Logical Data Warehouse - Views

Overview

Serverless SQL pool 은 논리적 Data Warehouse 로서 Azure Storage 에 있는 파일들을 사용하여 View 를 생성할 수 있습니다.

Benefits

Create SQL views on externally stored data

Access files using the view from various tools and language

Leverage rich T-SQL language to process and analyze data in external files exposed via views

Create PowerBI reports on the views created on external data

```
USE [mydbname]
GO

DROP VIEW IF EXISTS  populationView
GO

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

Logical Data Warehouse - Tables

Overview

Serverless SQL Pool 은 논리적 Data Warehouse 로서 Azure Storage 에 있는 파일들을 사용하여 External Table 을 생성할 수 있습니다.

Benefits

Create external tables that reference set of files on Azure storage.

Join and transform multiple tables in the same query.

Enables you to analyze external files with the same experience that you have in classic databases.

Manage column statistics in external tables.

Manage access rights per table.

Create PowerBI reports on the views created on external data

```
USE [mydbname]
GO

DROP TABLE IF EXISTS dbo.Population
GO
CREATE EXTERNAL TABLE dbo.Population (
   country_code VARCHAR (5) COLLATE Latin1_General_BIN2,
   country_name VARCHAR (100) COLLATE Latin1_General_BIN2,
   year smallint,
   population bigint
)
WITH(
      LOCATION = '/csv/population/population-*.csv',
      DATA_SOURCE = MyAzureStorage,
      FILE_FORMAT = MyAzureCSVFormat
)
```

```
CREATE STATISTICS stat_country_name
ON dbo.Population(country_name);
```

```
SELECT
   country_name, population
FROM population
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

데이터 포맷 변환

Overview

SQL queries 을 사용하여 Azure Storage Files 의 데이터 포맷을 쉽게 변환할 수 있습니다.

(CSV to Parquet, Json to Parquet)

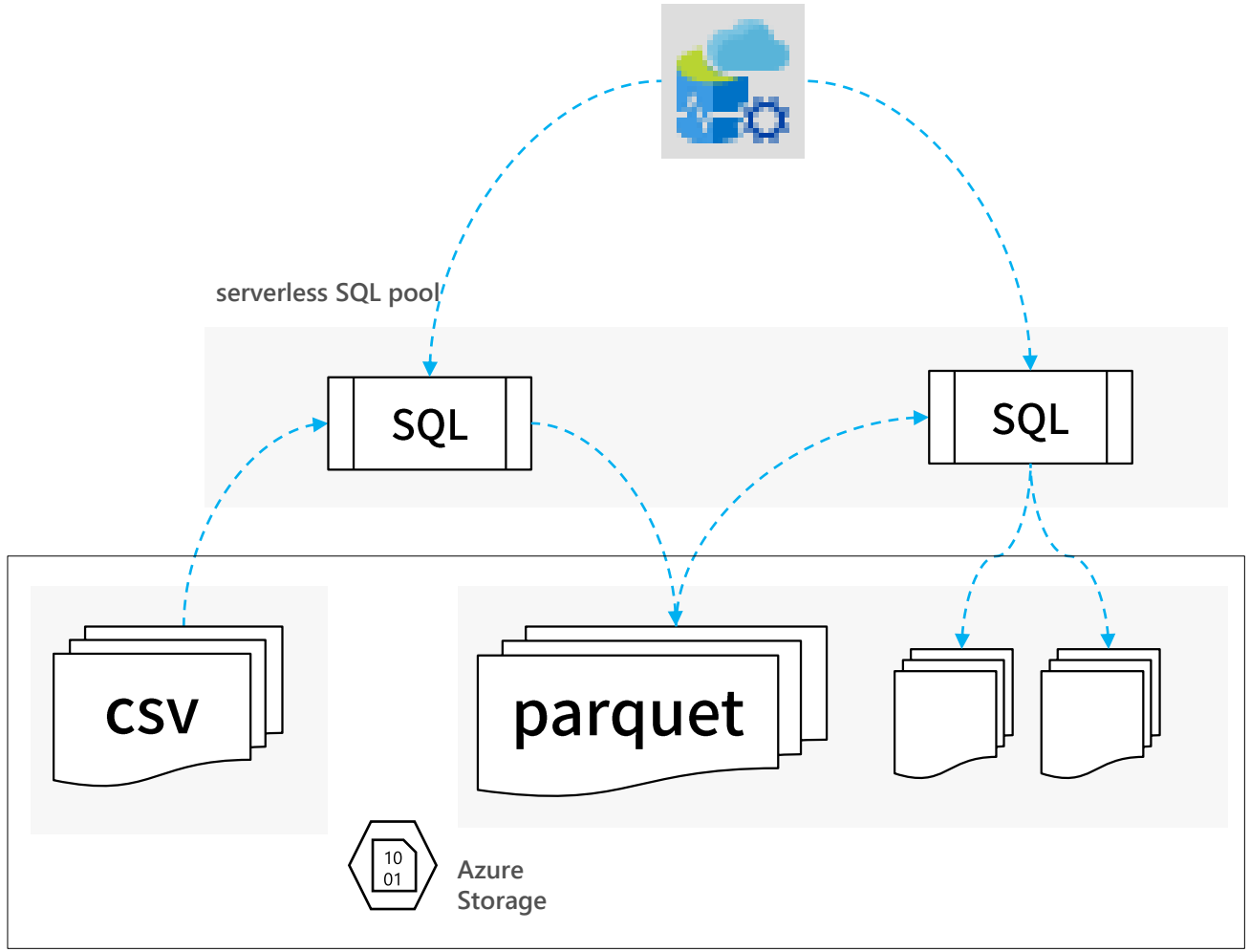
Benefits

Single statement transformations:

- convert CSV or JSON files to Parquet
- copy files from one storage account to another
- re-partition data to new location(s)
- store results of your query on Azure Storage

SQL ETL pipelines

- Use SQL commands to transform data
- Chain SQL statement for build ETL process
- Materialize reports created on the current snapshot of data



데이터 포맷 변환 with CETAS

Overview

Create External Tables As Select (CETAS) 를 이용하여 쉬운 데이터 포맷 변환이 가능하며 Azure storage 에 쿼리 결과를 저장할 수 있습니다.

Benefits

Select any data set and store it in parquet format.

Pre-calculate and store results of query and store them permanently on Azure storage.

Use saved data using external table.

Improve performance of your reports by permanently storing the result based on current snapshot of data as parquet files.

```
-- copy CSV dataset into parquet data set
CREATE EXTERNAL TABLE parquet.Population
WITH(
      LOCATION = '/parquet/population',
      DATA_SOURCE = MyAzureStorage,
      FILE_FORMAT = MyAzureParquetFormat  )
AS
SELECT *
FROM csv.Population

-- pre-create report using new parquet data-set
CREATE EXTERNAL TABLE parquet.PopulationByMonth2017
WITH(
      LOCATION = '/parquet/population/bymonth/2017',
      DATA_SOURCE = MyAzureStorage,
      FILE_FORMAT = MyAzureParquetFormat  )
AS
SELECT month = p.month, population = COUNT ( p.population )
FROM parquet.Population p
WHERE p.year = 2017
GROUP BY p.month

-- Reporting tools can now directly read data from pre-created report
SELECT *
FROM parquet.PopulationByMonth2017
```

Spark 테이블과의 자동 동기화

Overview

Spark Pool 에서 테이블을 생성하면 Serverless SQL Pool 에서 해당 테이블을 참조할 수 있는 Table 이 자동 생성됩니다.

Benefits

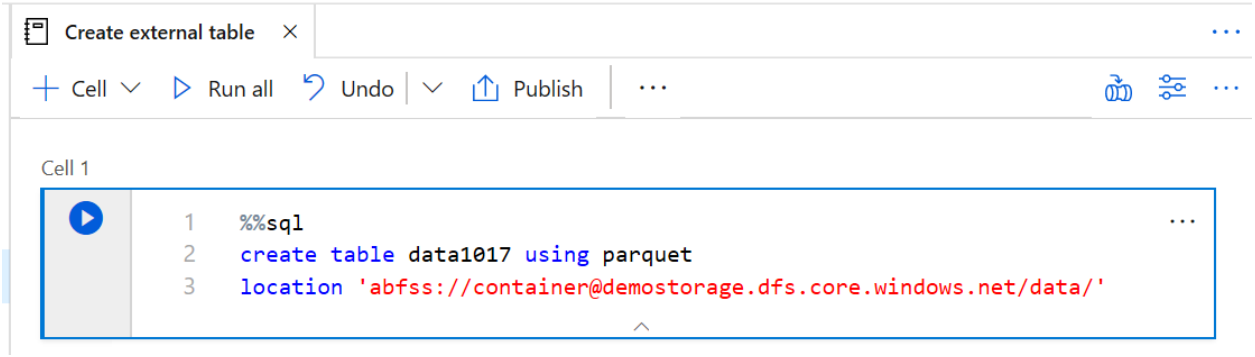
Tables designed using Spark languages are immediately available in serverless SQL pool.

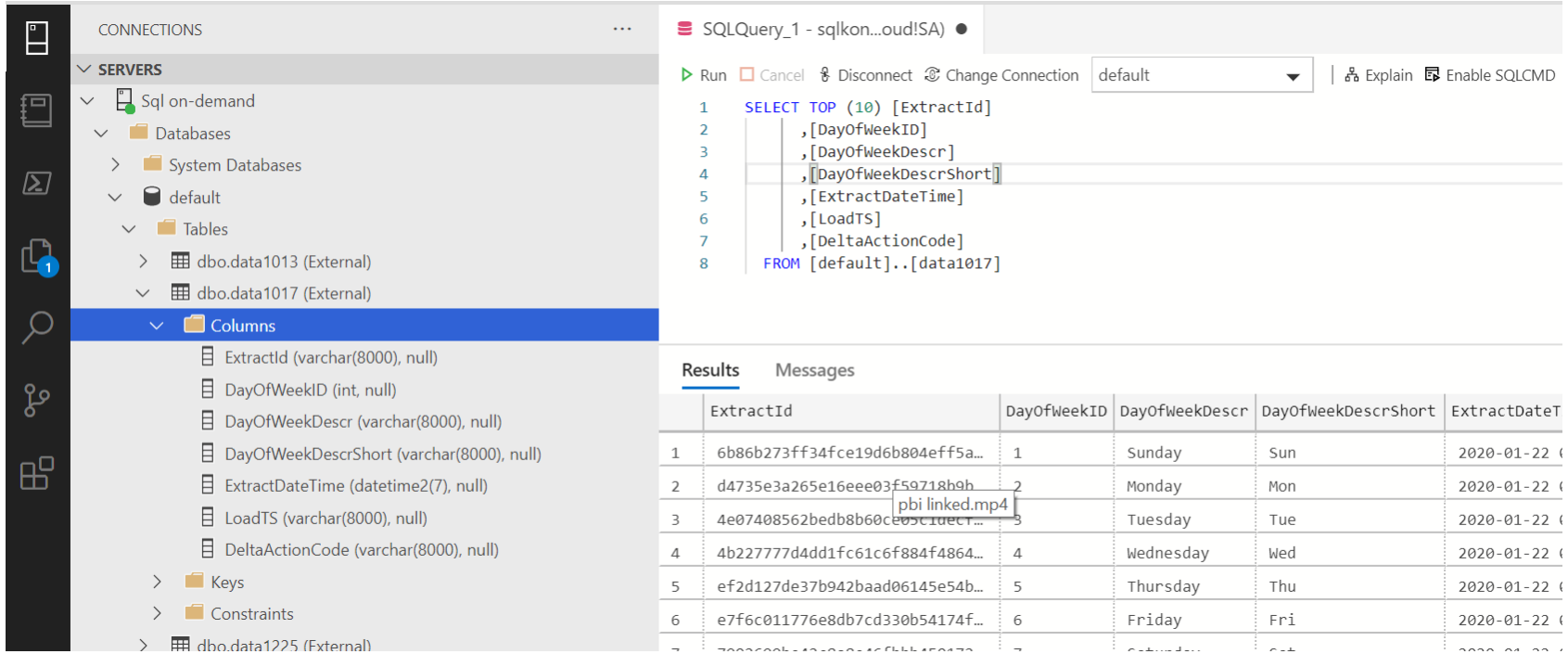
Schema definition matches original

Spark table updates are applied in serverless SQL pool

No need to manually create SQL tables that match Spark tables

Spark and serverless SQL pool tables references the same external files.





Azure Synapse Analytics

Apache Spark

Azure Synapse Apache Spark - 요약

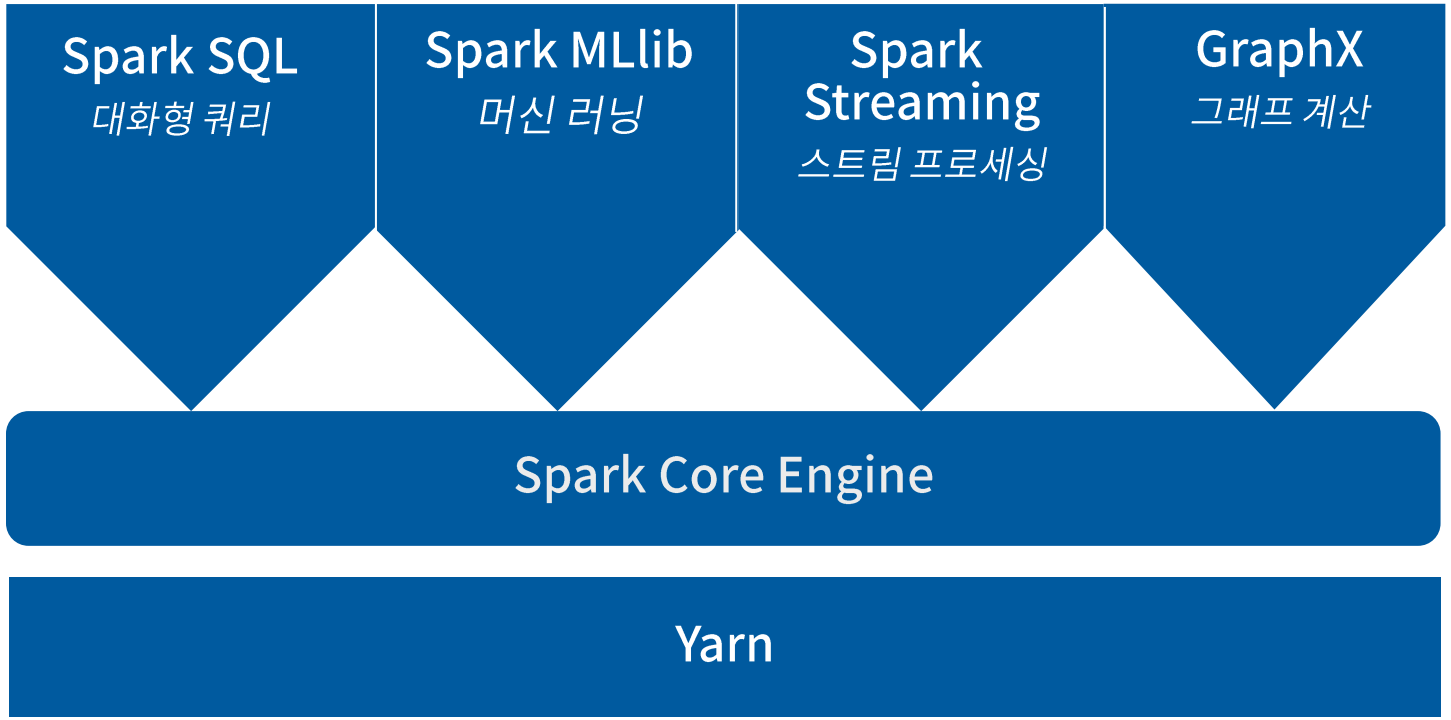
- Apache Spark 3.2
  - Delta Lake 1.2 지원
  - .Net Core 3.1 지원
  - Python 3.8 지원
- Tightly coupled to other Azure Synapse services
  - Integrated security and sign on
  - Integrated Metadata
  - Integrated and simplified provisioning
  - Integrated UX including nteract based notebooks
  - Fast load of Synapse SQL (provisioned) pools
- Core scenarios
  - Data Prep/Data Engineering/ETL
  - Machine Learning via Spark ML and Azure ML integration
  - Extensible through library management
- Efficient resource utilization
  - Fast Start
  - Auto scale (up and down)
  - Auto pause
  - Min cluster size of 3 nodes
- Multi Language Support
  - .Net (C#), PySpark, Scala, Spark SQL, Java

Apache Spark

빅데이터 분석을 위한 통합 오픈소스 병렬 데이터 처리 프레임 워크

Spark Unifies:

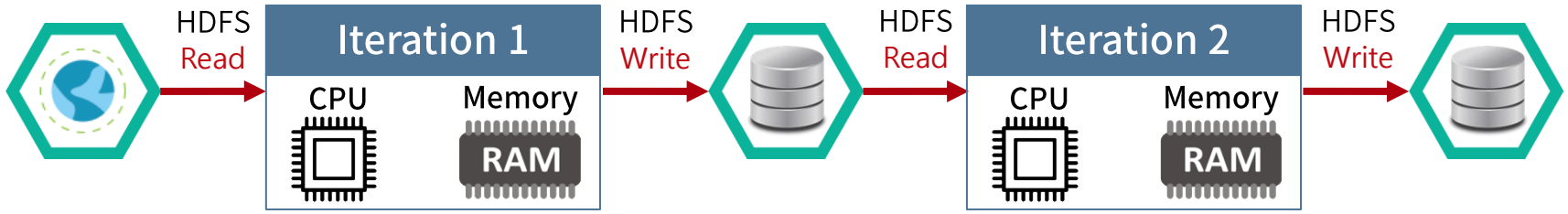
- 배치 처리
- Real-time 처리
- 대화형 쿼리
- 머신 러닝
- 딥 러닝
- 그래프 계산



http://spark.apache.org

Apache Spark

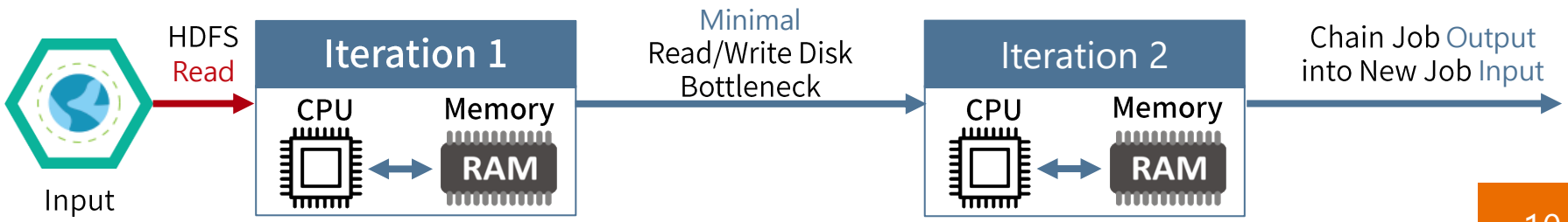
기존 접근 방식: 대화형 Query, Online event-hub 처리, 복잡한 작업을 위한 MapReduce jobs 에
많은 디스크 I/O 로 인한 속도가 저하됩니다.



Apache Spark

기존 접근 방식: 대화형 Query, Online event-hub 처리, 복잡한 작업을 위한 MapReduce jobs 에 많은 디스크 I/O 로 인한 속도가 저하됩니다.

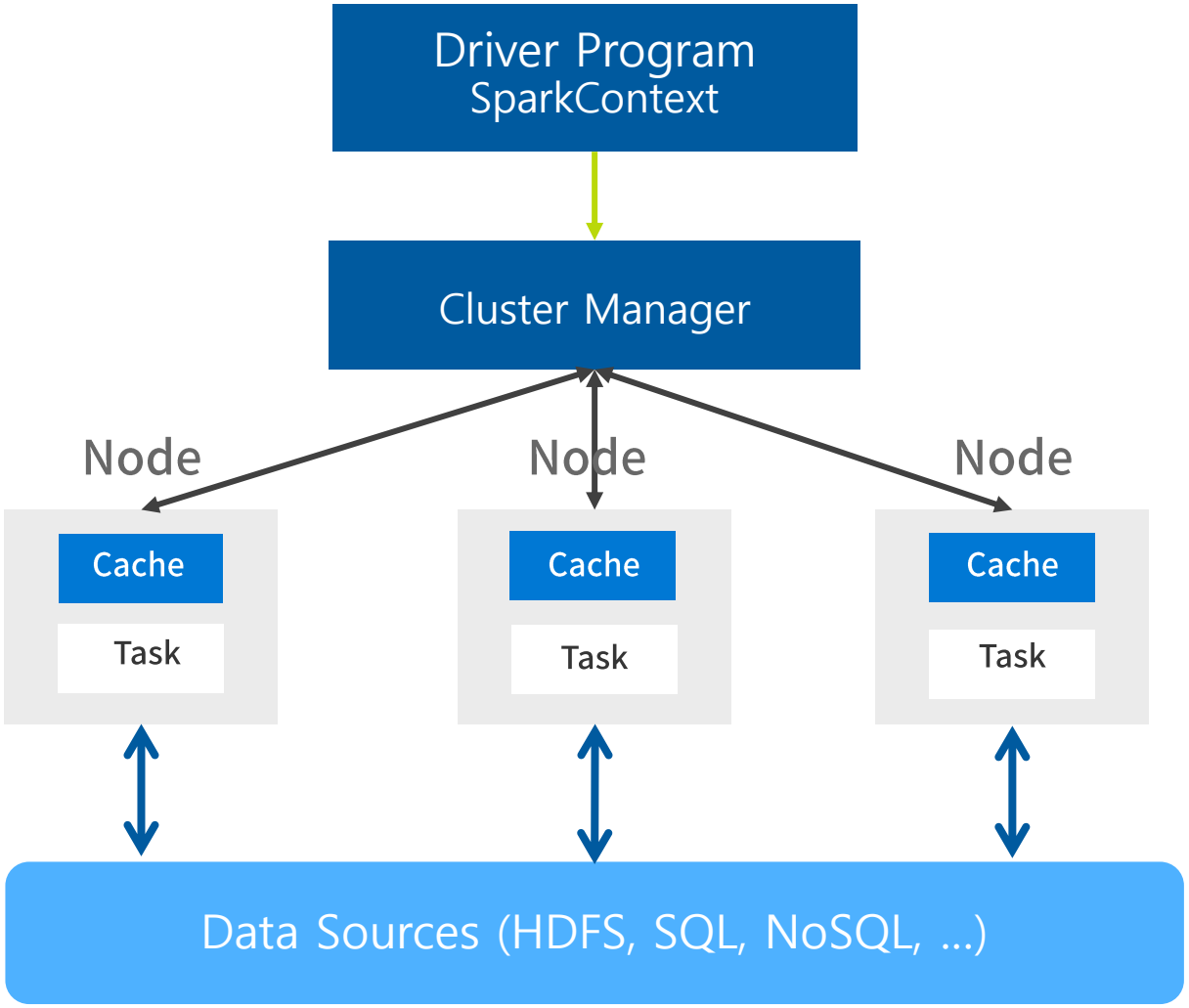
해결방안: 분산 처리 엔진에서 데이터를 **메모리**에 유지



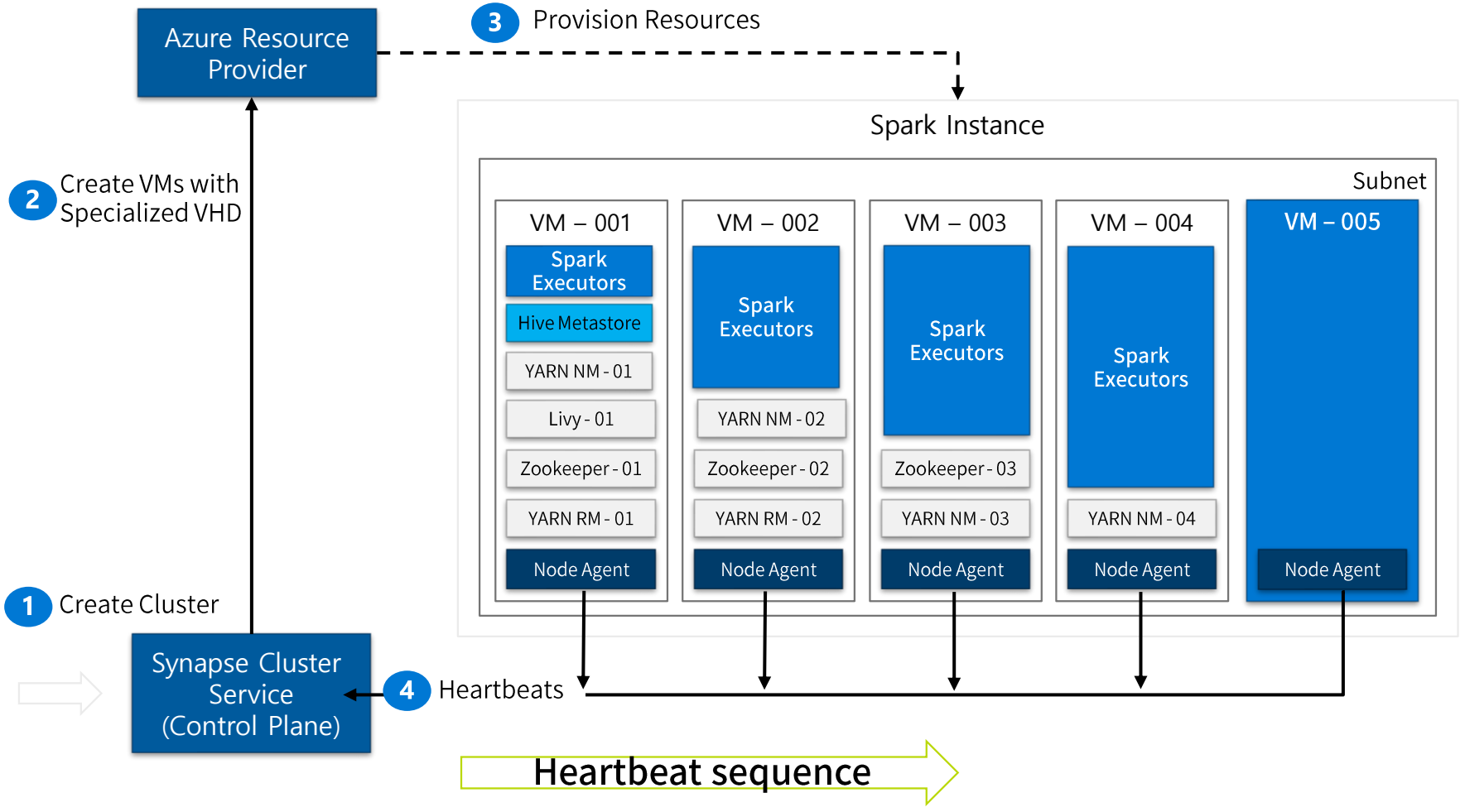
10–100x faster than network & disk

Apache Spark Architecture

- ‘Driver’는 사용자의 main() 함수를 구동하고, 다양한 병렬 작업들을 worker node에서 구동하게 합니다.
- 작업들의 결과는 ‘Driver’에 의해 취합되어 집니다.
- Worker node들은 HDFS, SQL과 같은 Data Source들로부터 데이터를 읽고 쓰게 됩니다.
- Worker node 변환된 데이터를 메모리에 RDD로 cache합니다.
- Worker node들과 Driver Node는 public cloud의 VM 들과 같이 실행됩니다.

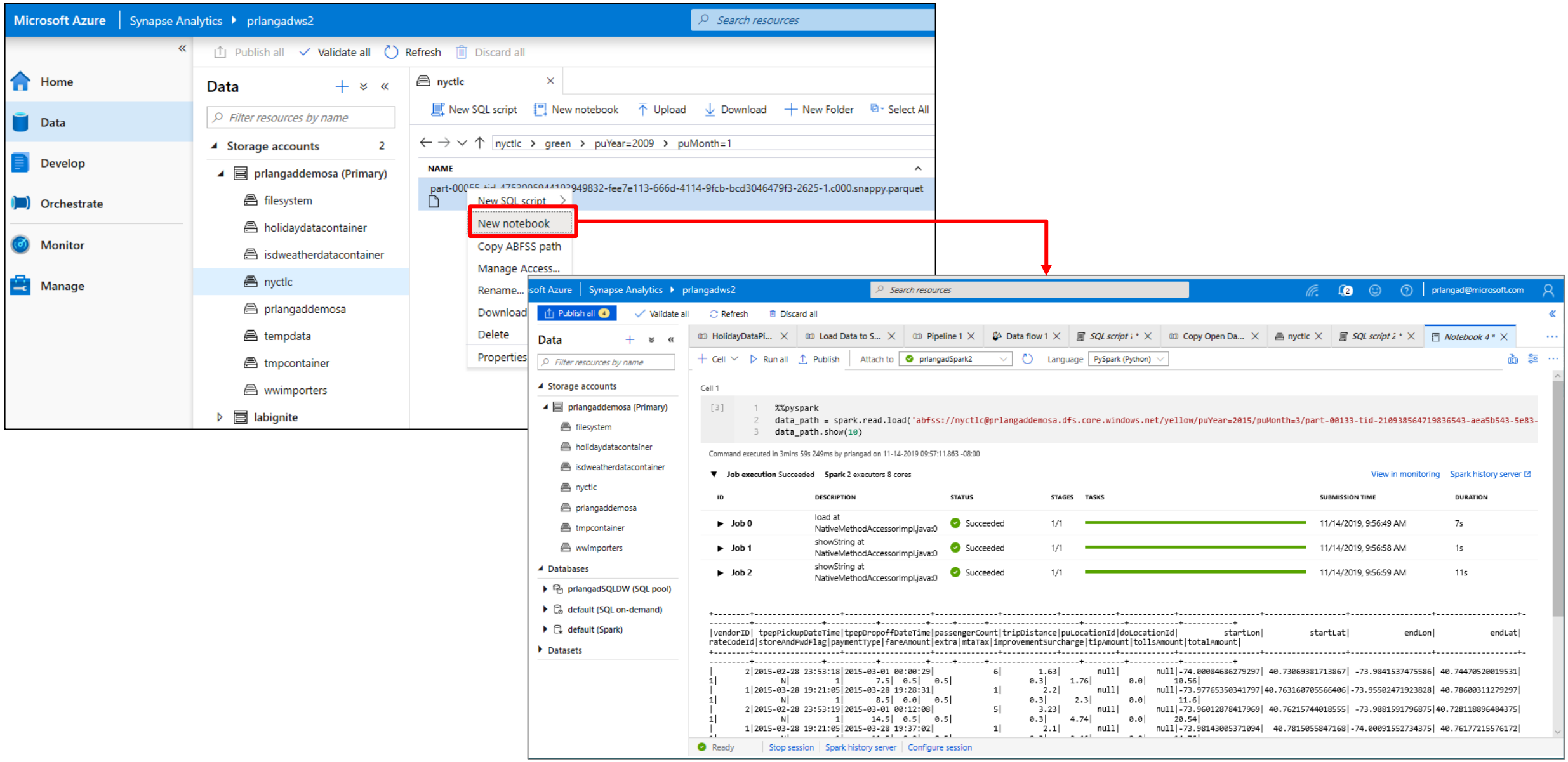


Synapse Spark Instance

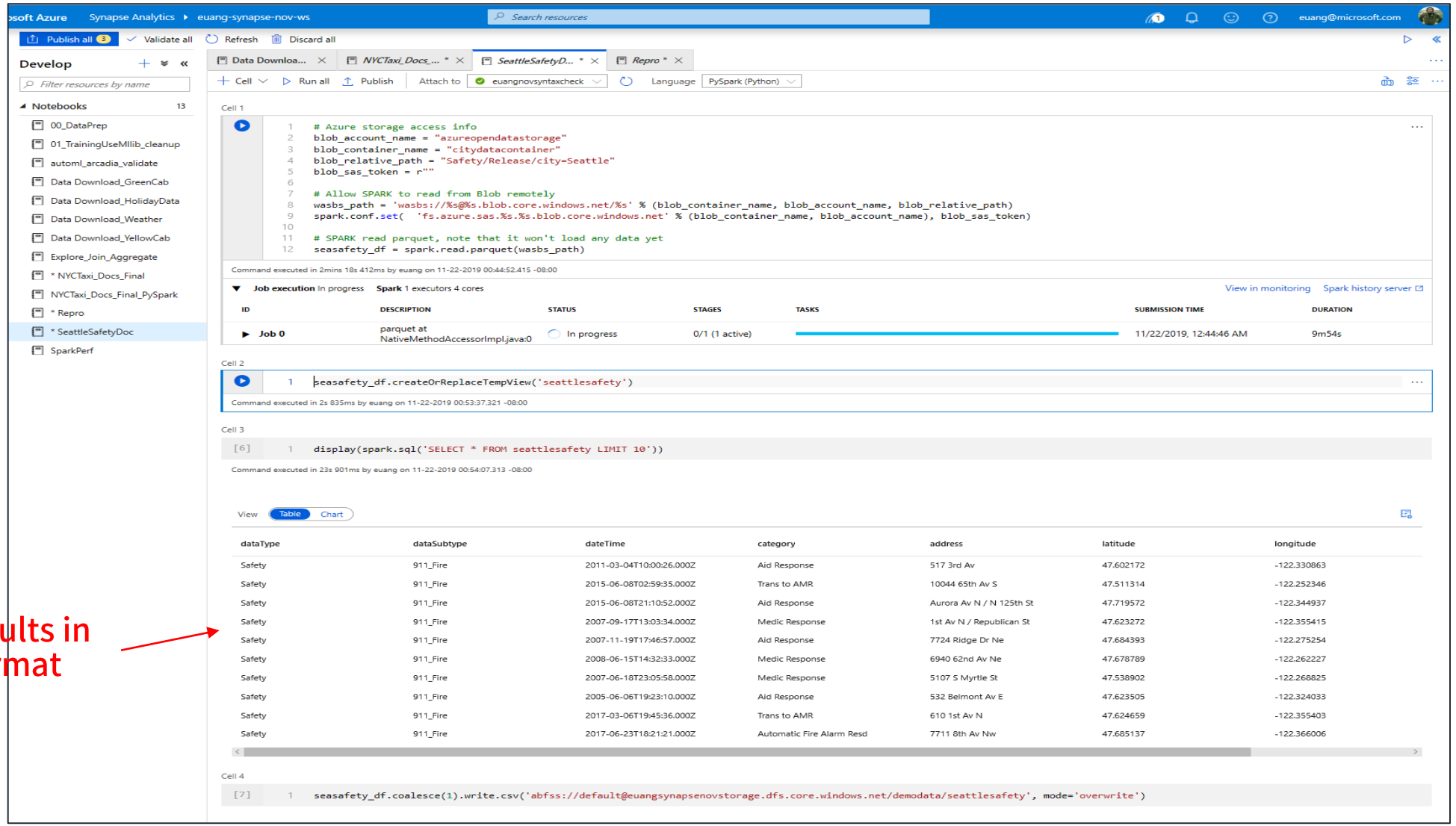


1. Synapse Job Service는 Spark pool에 설명에 따라, Cluster Service에게 cluster 생성 요청을 보냅니다.
2. Cluster Service는 Azure SDK를 사용하여 specialized VHD를 이용하여 VM을 만들도록 Azure에 요청합니다.
3. Specialized VHD는 cluster 타입(eg. Spark)에 의해 필요한 모든 서비스들이 포함되어져 있습니다.
4. VM이 부팅 되면, Node Agent는 heartbeat을 Cluster Service로 보내고 node configuration을 받습니다.
5. 첫번째 heartbeat을 기반으로 하여, node들은 초기화되고, role들이 assign 됩니다.
6. extra 노드들은 첫번째 heartbeat에서 삭제됩니다.
7. Cluster Service가 cluster가 준비되었다고 판단될 때, livy endpoint를 Job Service에게 반환합니다.

스토리지에 있는 파일을 이용한 Notebook 생성

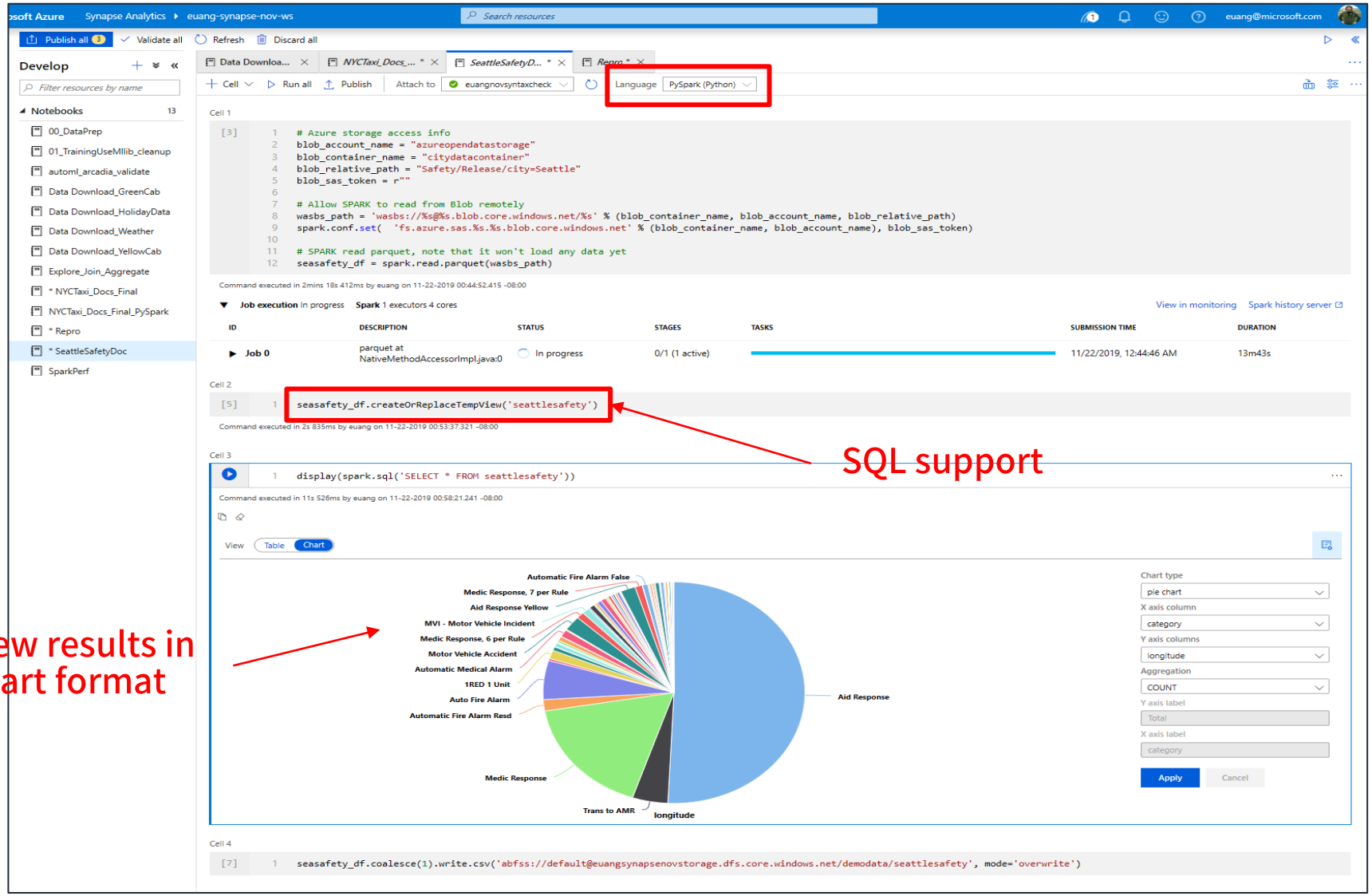


스토리지에 있는 파일을 이용한 Notebook 생성

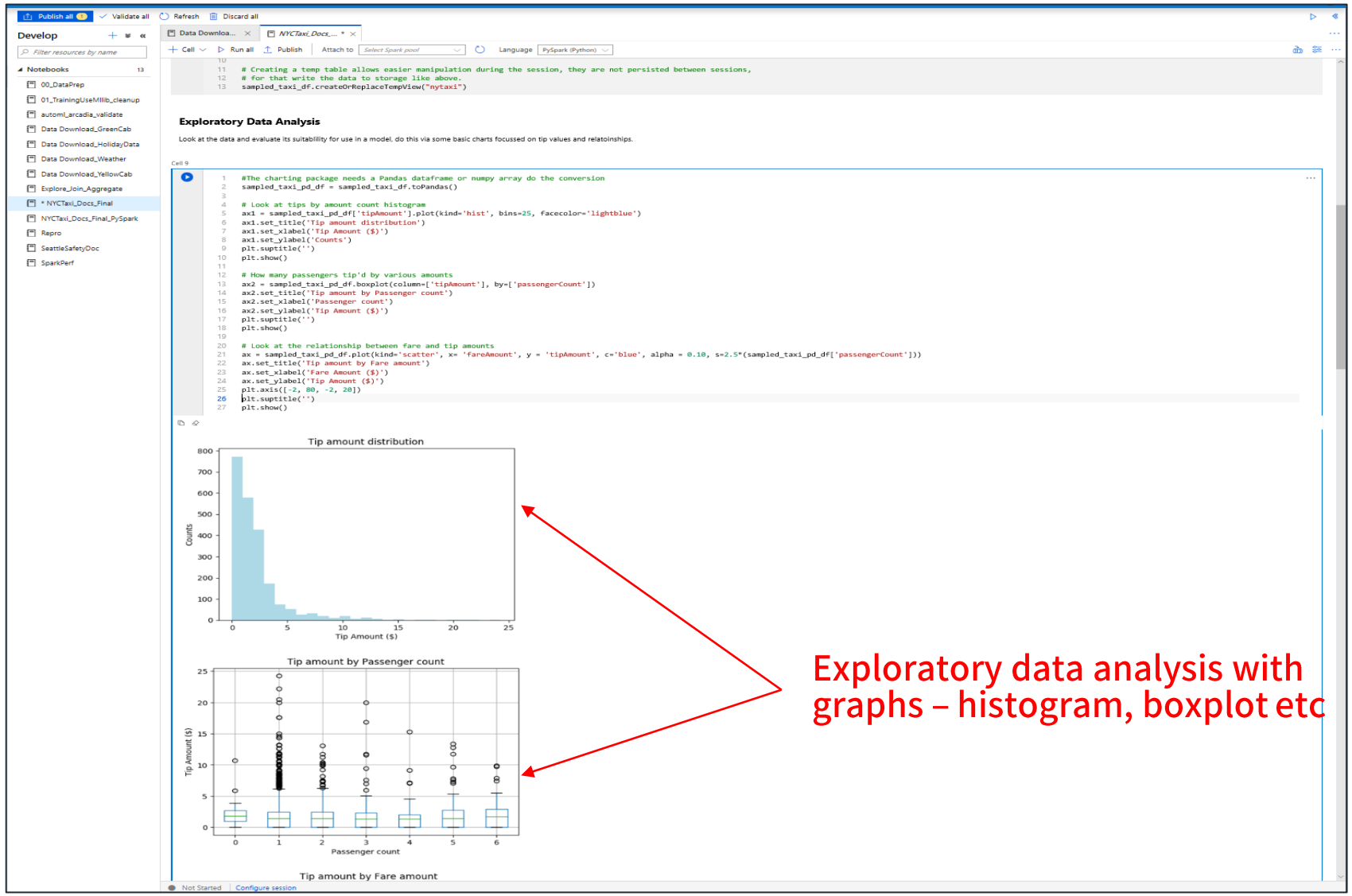


View results in table format

스토리지에 있는 파일을 이용한 Notebook 생성



스토리지에 있는 파일을 이용한 Notebook 생성



Q & A

Thank you