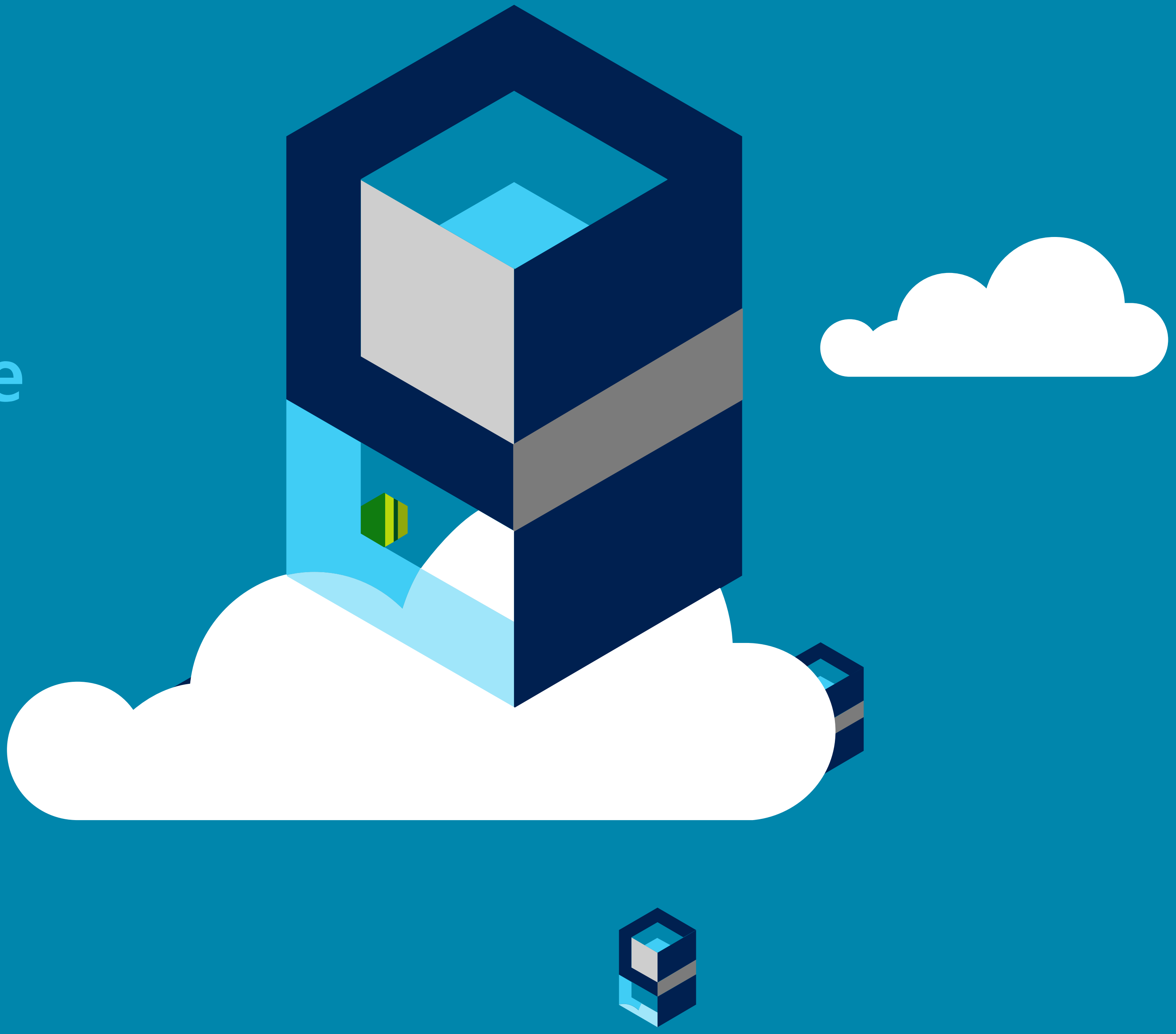
Microsoft

# Modernizing Your Data Warehouse

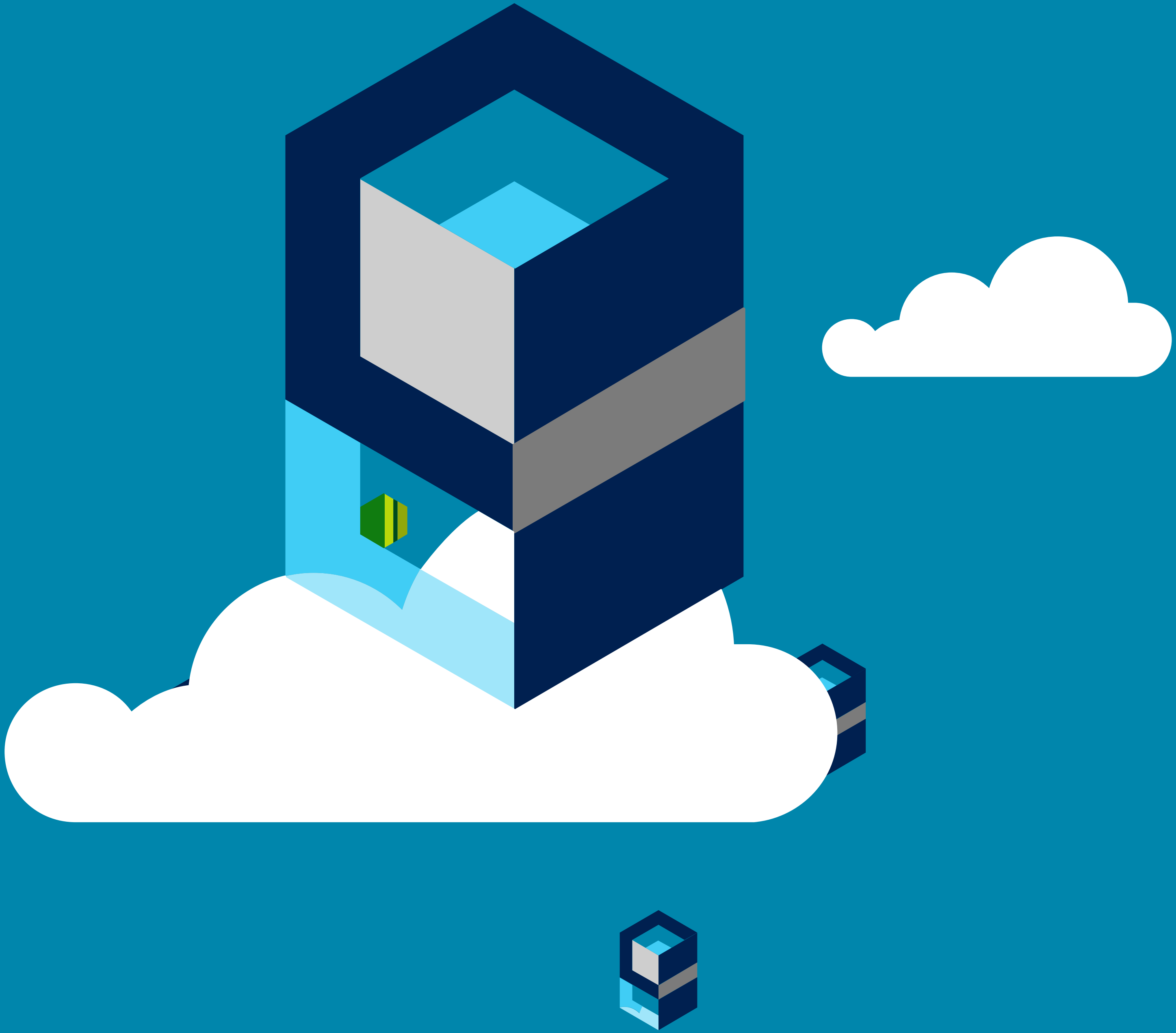


Microsoft

# SQL DW Data Loading Best Practices

Casey Karst

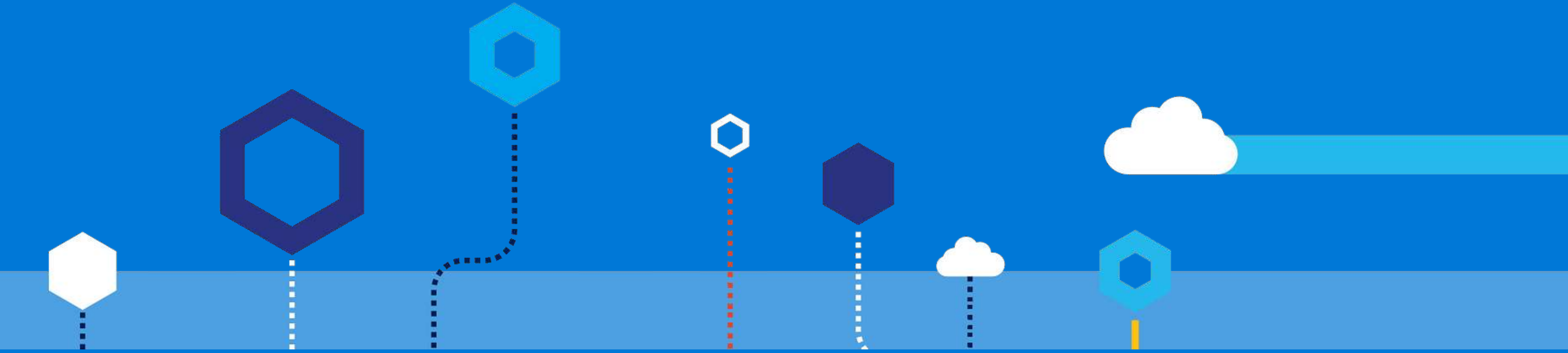
cakarst@Microsoft.com



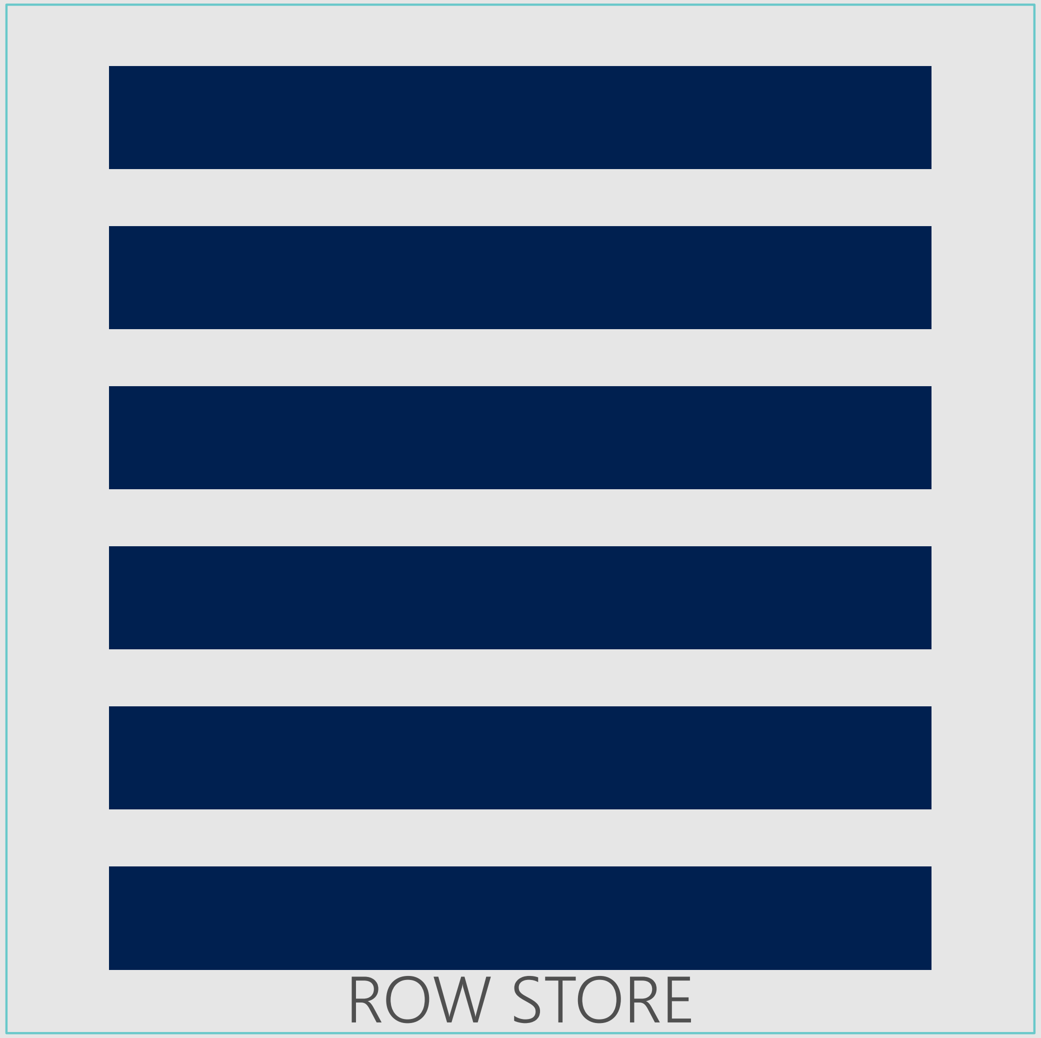
## Agenda

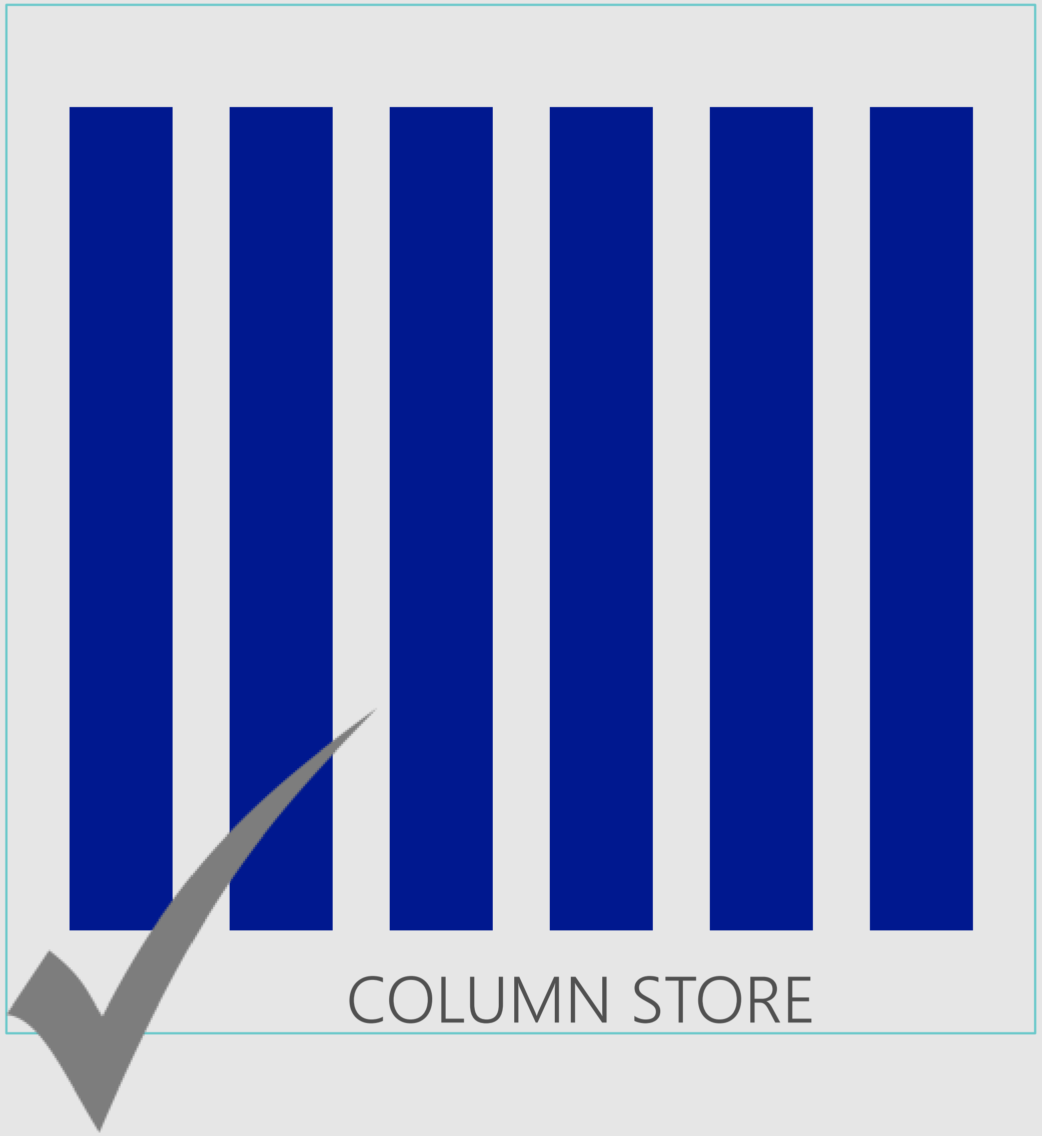
- Table Structure and impacts to loading
- Loading patterns
- Loading tools
- Loading best practices

## Physical Structures



# Row store & Column store





# Table and Index Terminology

## Primary Data Stores

Heap = Base Row Store

Clustered Index (CI) = Base Row Store maintained as a B-Tree

Clustered Columnstore Index (CCI) = Base Column Store

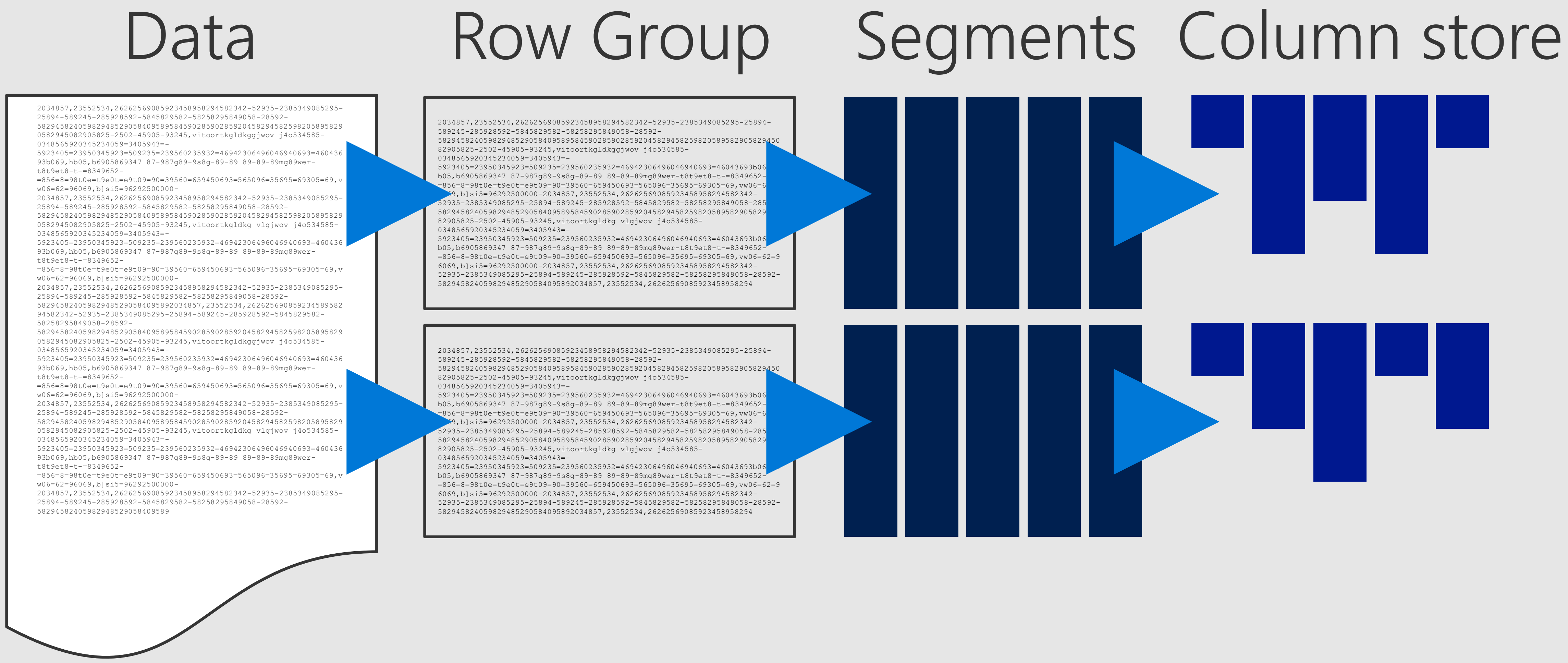
## Secondary Data Stores

Non Clustered Index (NCI) = Secondary B-Tree Index

- NCI can be on Heap
- NCI can be on Clustered Columnstore Index (NCI on CCI)

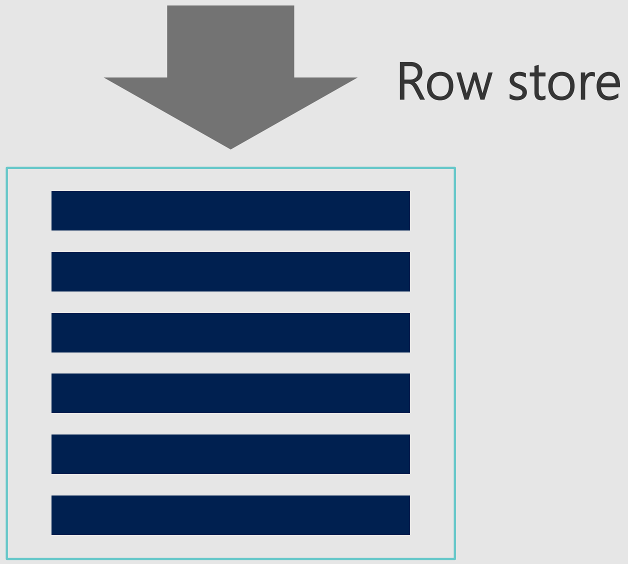
Non Clustered Columnstore Index (NCCI) = not supported in SQLDW

# Column store taxonomy

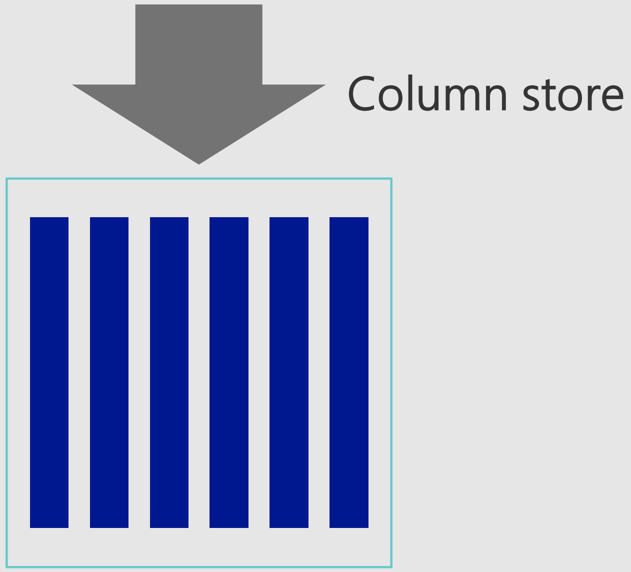


# Row store or Column store?

- Small Data Set (< 60 million rows)
- Frequent updates
- Small Dimension tables



- Large Data Set (> 60 million rows)
- Mostly append only data
- Fact tables or large dimension tables



# Why Partition?

## Benefit to Loads

Data Lifecycle Management

- Drop partition avoids transaction logging
- Insert to empty table/partition avoids transaction logging
- ⇒ Partition Switching pattern

Targeted Index Builds

## Benefit to Queries

Partition Elimination

# Partitioning Guidance

## Partition for data management

Lesser benefit had on partition elimination for faster performance

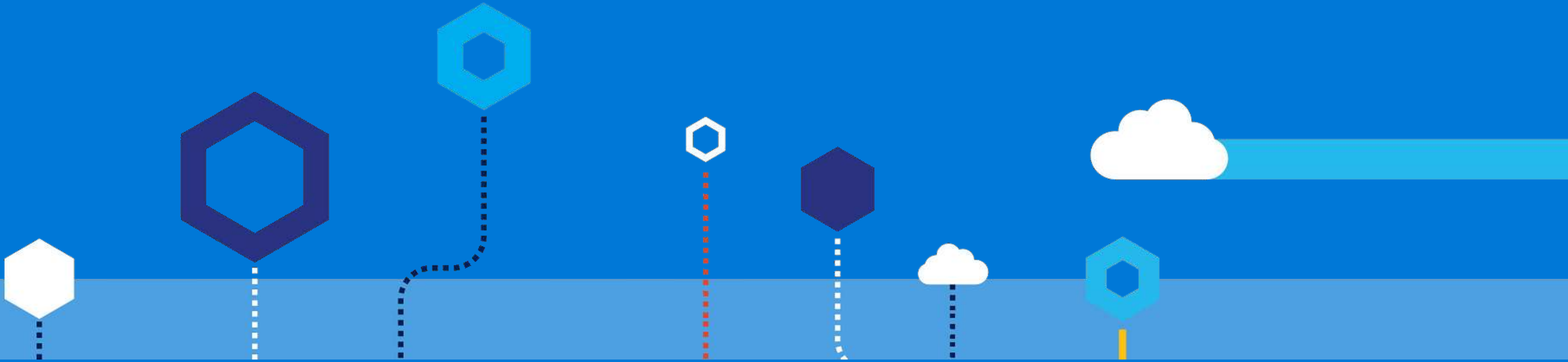
## Don’t over partition!

Partitioning granularity likely to differ to SQL Server
- Data is already spread across 60 distributions

Columnstore index row groups give ideal performance with 1 million rows each

Need at least 60 million per partition!

## Loading patterns



# Loading patterns

## Pattern 1: Batch loads (PolyBase)

Load large volumes of data in parallel

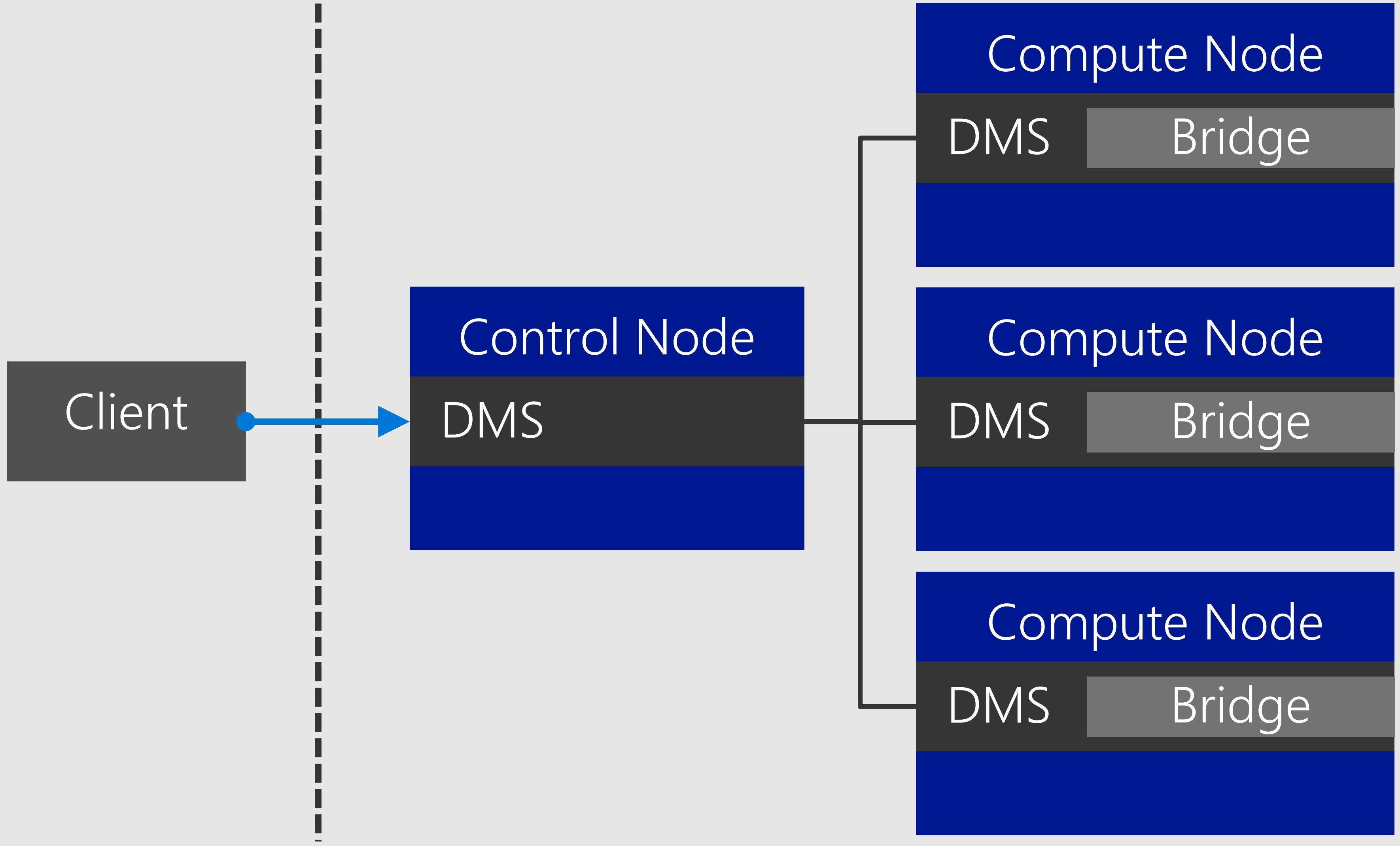
Move data from Azure storage to SQL DW in parallel to Compute nodes.

This includes Databricks integrations!

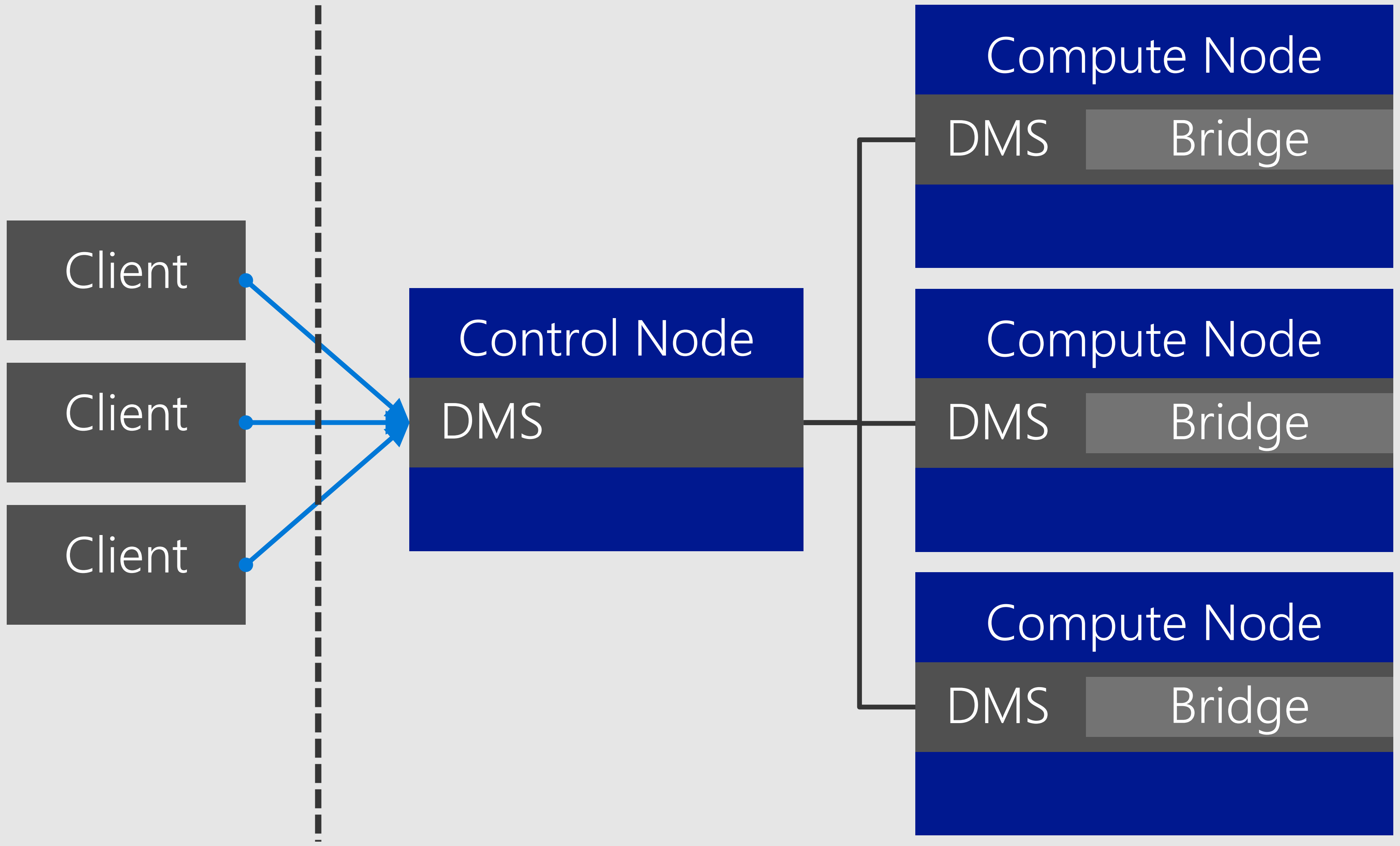
## Pattern 2: Streaming loads (BCP)

- Single record or small batches in each load.
- Data moves from source to the Control node and then to the Compute nodes.

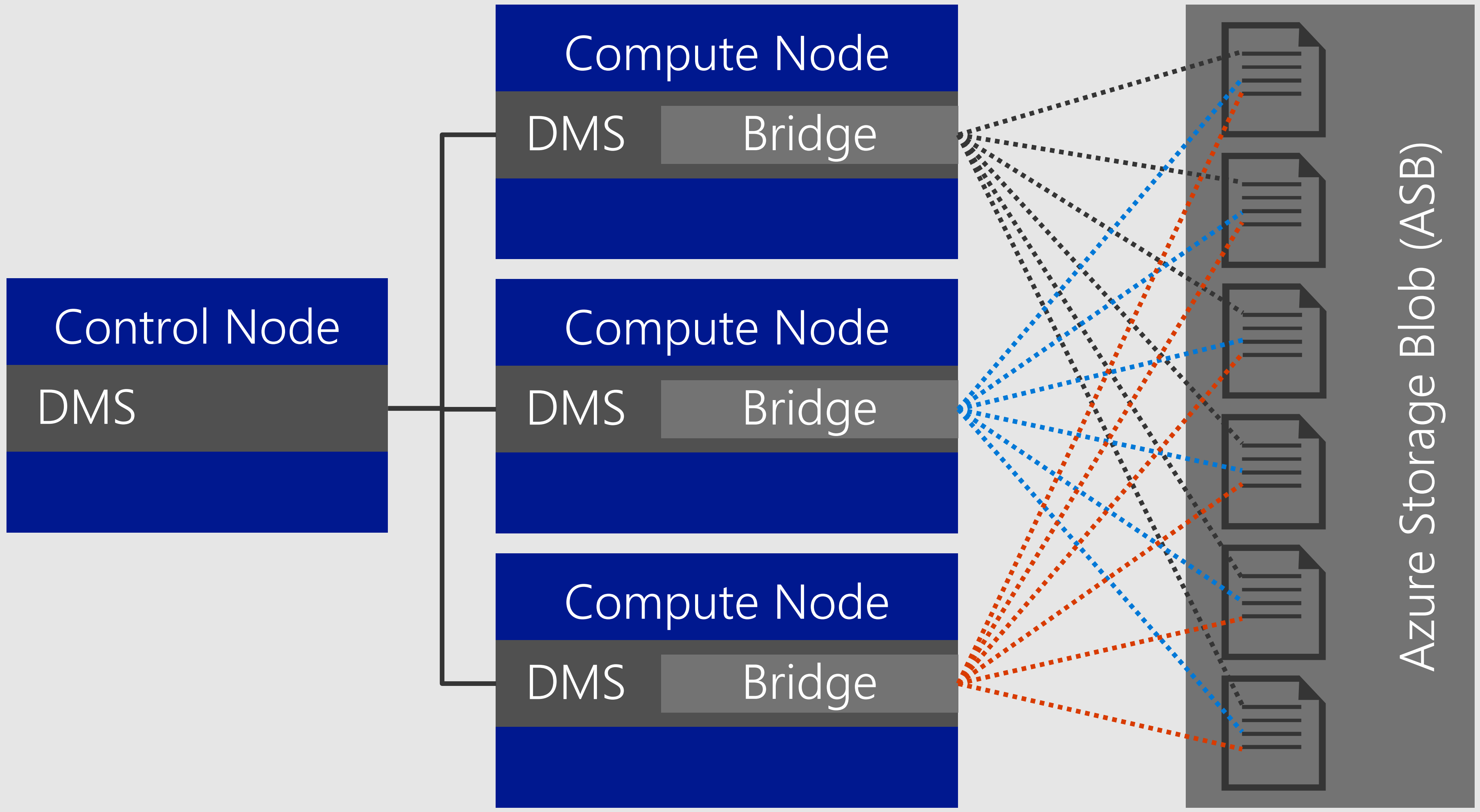
# Single gated client



# Single gated client parallelized



# Polybase parallel load to Azure Storage Blob



## Loading tools

# Create external tables

```
CREATE EXTERNAL DATA SOURCE WASBStor
WITH (TYPE = Hadoop,
      LOCATION = ‘wasbs://<container>@<account_name>.blob.core.windows.net',
          Credential = <Database scoped credential>);
```

Once per WASB container

```
CREATE EXTERNAL FILE FORMAT TextFile
WITH ( FORMAT_TYPE = DELIMITEDTEXT,
          DATA_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec',
          FORMAT_OPTIONS (FIELD_TERMINATOR ='|', USE_TYPE_DEFAULT = TRUE));
```

Once per file format

```
CREATE EXTERNAL TABLE [dbo].[Customer_import] (
    [SensorKey] int NOT NULL,
    [CustomerKey] int NOT NULL,
    [Speed] float NOT NULL
)
WITH (LOCATION=‘/Dimensions/customer',
      DATA_SOURCE = WASBStor,
      FILE_FORMAT = TextFile
)
```

File path

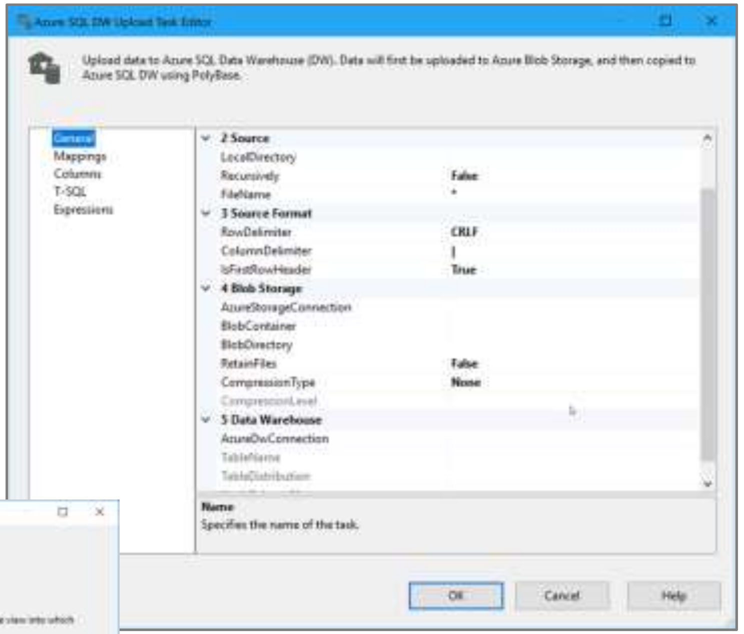
# SQL Server Integration Services (SSIS)

## Overview

SQL Server Integration Services is used to extract, transform data and load data from a variety of sources into Azure SQL Data Warehouse.

There are  two options for loading data into SQL Data Warehouse with SSIS:

- **Azure SQL Data Warehouse Upload Task:** provides best performance but assumes source data is in delimited text file format.
- **Data Flow Task:** slower than SQL Data Warehouse Upload Task but supports a wider range of data sources.

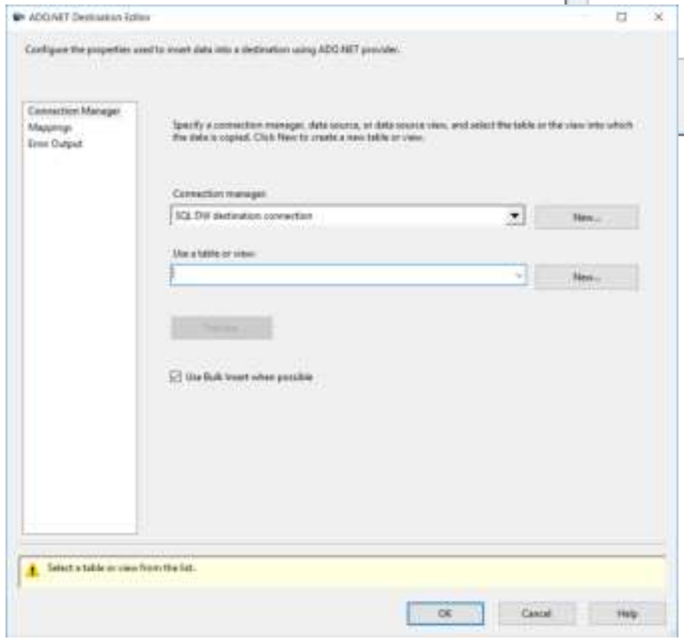


Azure SQL DW Upload Task Editor

Upload data to Azure SQL Data Warehouse (DW). Data will first be uploaded to Azure Blob Storage, and then copied to Azure SQL DW using PolyBase.

| | |
|---|---|
| **2 Source** | |
| LocalDirectory | |
| Recursively | False |
| FileName | * |
| **3 Source Format** | |
| RowDelimiter | CRLF |
| ColumnDelimiter | \| |
| IsFirstRowHeader | True |
| **4 Blob Storage** | |
| AzureStorageConnection | |
| BlobContainer | |
| BlobDirectory | |
| RetainFiles | False |
| CompressionType | None |
| CompressionLevel | |
| **5 Data Warehouse** | |
| AzureDwConnection | |
| TableName | |
| TableDistribution | |

**Name**
Specifies the name of the task.



ADO.NET Destination Editor

Configure the properties used to insert data into a destination using ADO.NET provider.

Specify a connection manager, data source, or data source view, and select the table or the view into which the data is copied. Click New to create a new table or view.

Connection manager: SQL DW destination connection

Use a table or view:

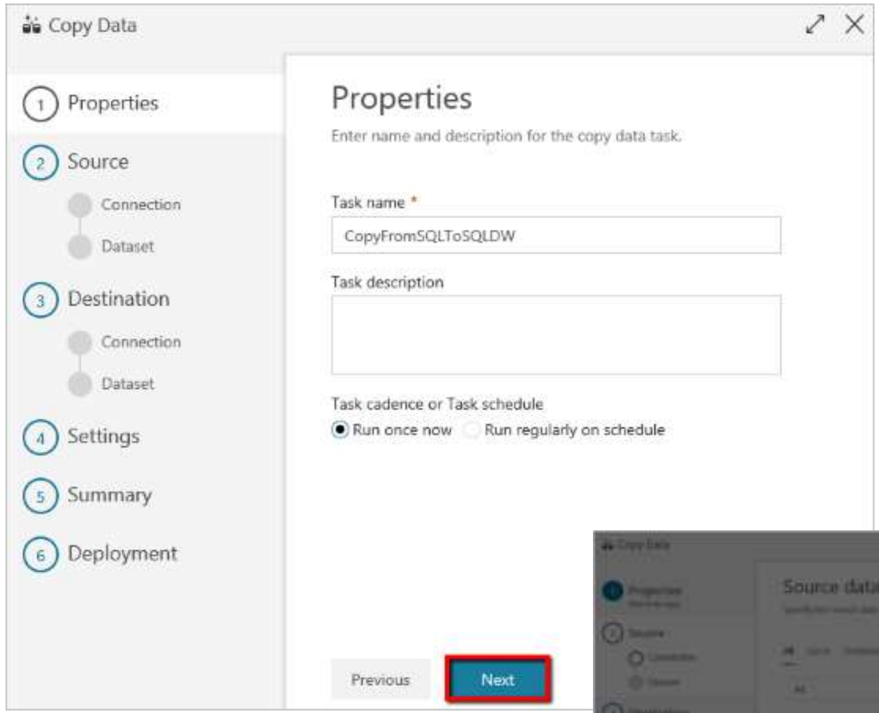
[✓] Use Bulk Insert when possible

Select a table or view from the list.

# Azure Data Factory Copy Data tool

## Overview

The Azure Data Factory Copy Data tool provides an intuitive wizard that allows you to copy data from a variety of data sources into Azure SQL Data Warehouse.



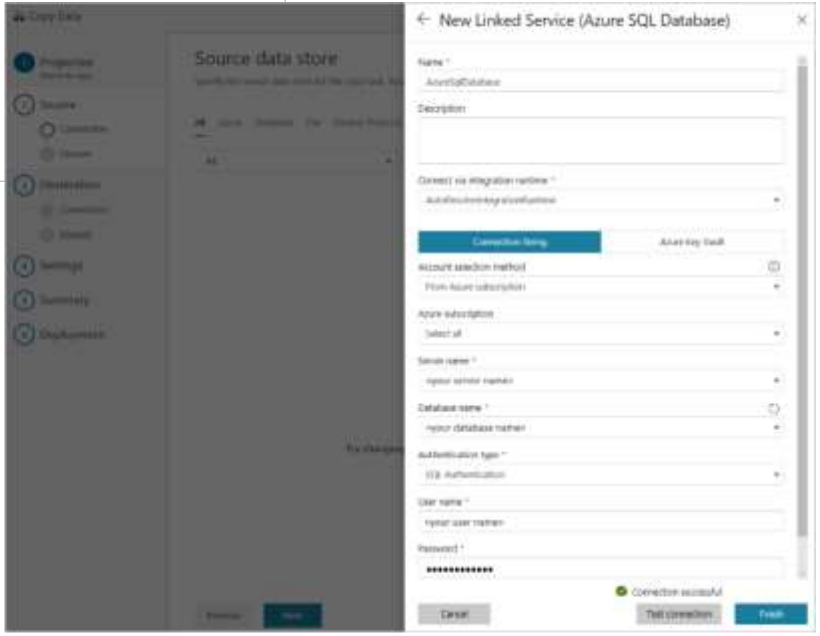
Properties

Enter name and description for the copy data task.

Task name *: CopyFromSQLToSQLDW

Task description:

Task cadence or Task schedule: [✓ Run once now] Run regularly on schedule



New Linked Service (Azure SQL Database)

Name *: AzureSqlDatabase

Description:

Connect via integration runtime *: AutoResolveIntegrationRuntime

Account selection method: From Azure subscription

Azure subscription: Select all

Server name *: <your server name>

Database name *: <your database name>

Authentication type *: SQL Authentication

User name *: <your user name>

Password *: ••••••••••••

Connection successful

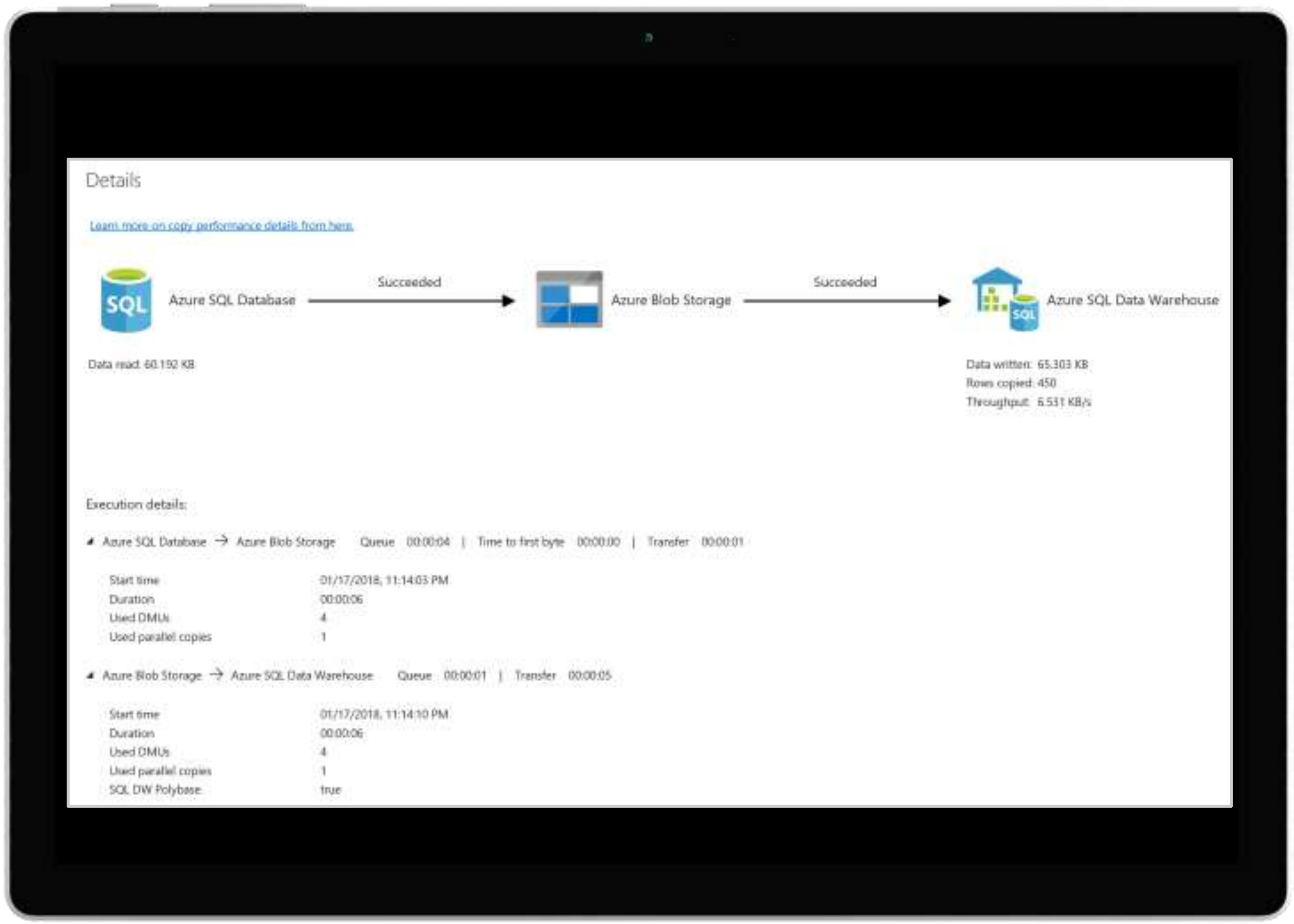
# Azure Data Factory Copy activity

## Overview

The Azure Data Factory Copy activity allows copying to and from Azure SQL Data Warehouse from any supported data store.

The Copy activity also supports retrieving data from a SQL source by using a SQL query or stored procedure. Authentication can be via:

- SQL Authentication
- Service principal token authentication
- Managed identity token authentication


Details

Learn more on copy performance details from here.

Azure SQL Database —Succeeded→ Azure Blob Storage —Succeeded→ Azure SQL Data Warehouse

Data read: 60.192 KB

Data written: 65.303 KB
Rows copied: 450
Throughput: 6.531 KB/s

Execution details:

▲ Azure SQL Database → Azure Blob Storage    Queue 00:00:04 | Time to first byte 00:00:00 | Transfer 00:00:01

| | |
|---|---|
| Start time | 01/17/2018, 11:14:03 PM |
| Duration | 00:00:06 |
| Used DMUs | 4 |
| Used parallel copies | 1 |

▲ Azure Blob Storage → Azure SQL Data Warehouse    Queue 00:00:01 | Transfer 00:00:05

| | |
|---|---|
| Start time | 01/17/2018, 11:14:10 PM |
| Duration | 00:00:06 |
| Used DMUs | 4 |
| Used parallel copies | 1 |
| SQL DW Polybase | true |

# Databricks – structured streaming

## Overview

The Databricks SQL DW connector supports batch and structured streaming support for writing real-time data into Azure SQL Data Warehouse.

It uses Polybase and the Databricks structured streaming API to stream data from Kafka or Kinesis sources directly into SQL Data Warehouse at a user-configurable rate.

Source: https://docs.azuredatabricks.net/spark/latest/data-sources/azure/sql-data-warehouse.html#streaming-support

```
# Prepare streaming source; this could be Kafka,
Kinesis, or a simple rate stream.
df = spark.readStream \
  .format("rate") \
  .option("rowsPerSecond", "100000") \
  .option("numPartitions", "16") \
  .load()

# Apply some transformations to the data then use
# Structured Streaming API to continuously write the
data to a table in SQL DW.
df.writeStream \
  .format("com.databricks.spark.sqldw") \
  .option("url", <azure-sqldw-jdbc-url>) \
  .option("tempDir",
"wasbs://<containername>@<storageaccount>.blob.core.
windows.net/<directory>") \
  .option("forwardSparkAzureStorageCredentials",
"true") \
  .option("dbTable", <table-name>) \
  .option("checkpointLocation", "/tmp_location") \
  .start()
```

# Mechanism for loading

1. PolyBase
2. SSIS*
3. ADF
4. BCP
5. SQLBulkCopy API
6. Attunity Cloudbeam
7. ASA/Storm**

| | PolyBase | SSIS | ADF | BCP | SqlBulkCopy |
|---|---|---|---|---|---|
| Rate | Fastest | | | | Slowest |
| Rate increase as DWU increases | Yes | Yes | Yes | No | No |
| Rate increases as you add concurrent load | No | No | No | Yes | Yes |

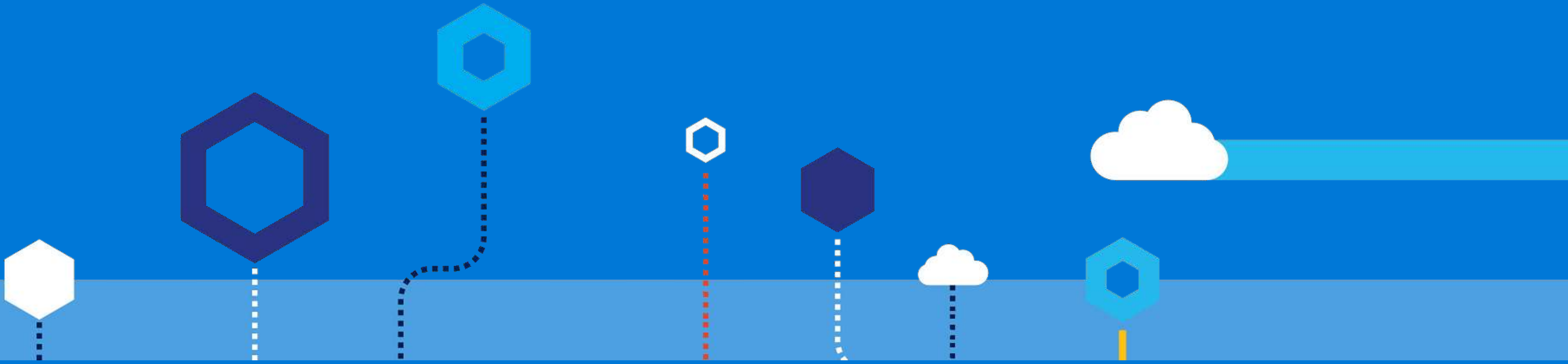
* With SSIS Azure Feature Pack June 2017 or newer

** Not a good idea

# Loading method considerations

| Loading Type | Source | Concerns | Advice |
|---|---|---|---|
| Batch loading | WASB/ADLS | Latency of data | Do it! |
| Micro Batch loading | WASB/ADLS | Potential impact to index health.<br>Impact on machine resources | Do it with caution… Make sure that loads are big enough |
| Streaming Load | Azure Stream Analytics, BCP | Column store Index health.<br>Load Performance | Do it with higher caution… Understand higher latency and impact on segment quality. |

## Best practices



## Dimension tables - best practices

Use **round robin** or **replicated** for small tables

Use clustered index, not clustered columnstore index

Can load directly to production because of small size

Use metadata rename to reload data

## Fact table best practices

Use partitions to reduce the loading impact on the production table

Consider landing data from ADL in a staging table

Take advantage of directory structure to limit loading scope

## More reading…

- Guidance for designing distributed tables
- https://docs.microsoft.com/en-us/azure/sql-data-warehouse/sql-data-warehouse-tables-distribute

- Columnstore indexes
- https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-2017

- Analyze your workload in Azure SQL Data Warehouse
- https://docs.microsoft.com/en-us/azure/sql-data-warehouse/analyze-your-workload
- Adaptive caching powers Azure SQL Data Warehouse performance gains
- https://azure.microsoft.com/en-us/blog/adaptive-caching-powers-azure-sql-data-warehouse-performance-gains/

- Cheat sheet
- https://docs.microsoft.com/en-us/azure/sql-data-warehouse/cheat-sheet

Microsoft

# Q&A