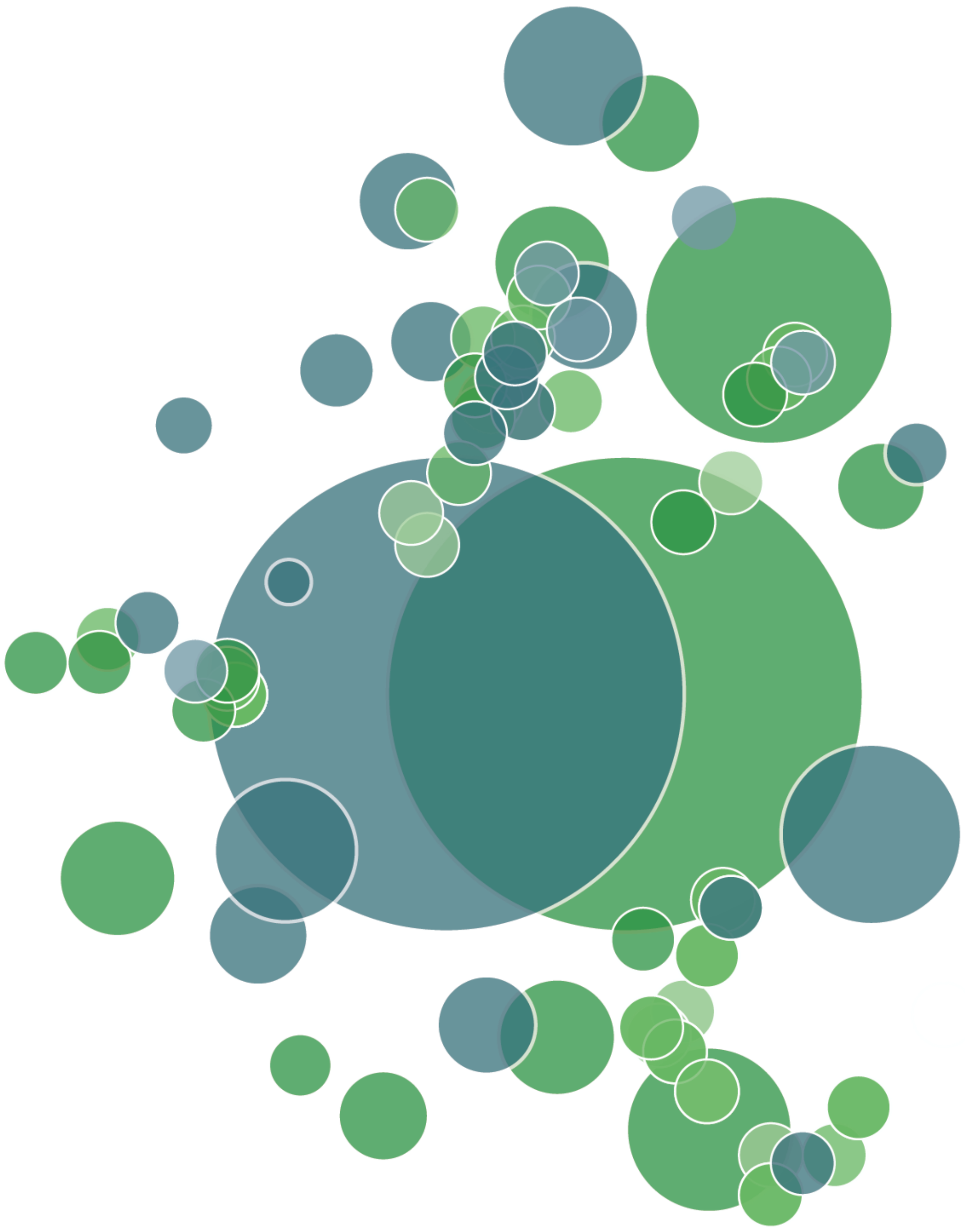

Prepared By: Madeleine Corneli

Microsoft Azure SQL Data Warehouse

Getting started with Tableau and Azure SQL Data Warehouse

What this is: a self-service guide to uploading flat files into Azure SQL data warehouse and operationalizing the warehouse for external connections, specifically Tableau.

Who is this for: data engineers & people responsible for setting up the Azure SQL DW pipeline or people interested in testing out Tableau and Azure SQL DW

What this isn't: a production guide to deploying & tuning Azure SQL DW. This is primarily intended for proof-of-concept purposes.

How long this takes: ~ 30 minutes

** Unless otherwise indicated - use the default settings in all GUIs when creating resources, as indicated in the linked Azure documentation

1) Create new Azure SQL DW [link]
    a) For this tutorial we recommend starting with Gen2: DW500c for good performance. The data warehouse can be scaled to accommodate larger datasets [link]
    b) Note: Azure SQL database is simply SQL Server in the cloud, Azure SQL data warehouse is a cloud version of SQL Server optimized for big data - with high performance and storage capabilities
    c) Make sure to add your client IP(s) to SQL server firewall rules as indicated in documentation
2) Choose a dataset to upload
    a) To follow along with the commands in this tutorial please use the clean Superstore CSV included with the post which can be downloaded here [link]. Depending on the origin of your data it may require cleaning or pre-processing which can be accomplished with tools like Tableau Prep
3) Upload dataset to Azure blob storage
    a) Create an Azure storage account [link] in the same resource group as your new Azure SQL DW
        i) Technically this can be anywhere as long as it's in the same subscription
        ii) We strongly recommend putting the storage account into the same region as the SQL DW (collocating) to avoid network egress costs
    b) For this tutorial we're using Azure Storage Explorer to upload data [link]
        i) Depending on the structure, origin or size of your data you may choose to use another tool [link]

c) Open Azure Storage Explorer and connect to the appropriate subscription (we're connecting using Azure environment log in - there are other options, like SAS keys if you don't own the storage account)
   i) Expand the storage account you just created
   ii) Create a new Blob Container by right-clicking on *Blob Containers*
   iii) Upload the file(s) you want to move to SQL DW to the blob container

4) Set up SQL DW for data transfer
   a) We are using SQL Server Management Studio (SSMS) [link] to connect to your SQL DW and transfer data.
   b) Connect to your SQL Server from SSMS
      i) Server name can be found on the SQL data warehouse resource page in your Azure web portal
      ii) Use *SQL Server Authentication* and the username and password you specified upon creation of the SQL Server
      iii) If you get a connection error make sure you have added your IP to the firewall rules (step **1c**)
   c) Expand the object explorer and right click on the name of the SQL DW you created to open a new query editor. To execute individual queries, paste them into the editor and choose *Execute* or hit Ctrl+E
   d) Create a staging schema - we will create our temporary external tables in this schema to separate them from production tables.

```
CREATE SCHEMA [staging];
```

   e) Create a Master key - this encrypts the key for your storage account so that people can't browse and access the storage key.

```
CREATE master key;
```

   f) Create a Credential object to access the storage account
      i) Navigate to the Storage Account you created in step **3a** in the Azure web portal. Copy the first key from the Access Keys tab.
      ii) Note - this is the master key to your storage account - only use this key for your own data uploads, do NOT share this key.

```
CREATE DATABASE SCOPED CREDENTIAL AppCred WITH IDENTITY = '<Account
Username>', SECRET = '<Storage Access Key>' ;
```

   g) Create an External data source pointing at the storage account and blob container you created
      i) This allows you to query any files in the referenced storage account. We can keep adding additional files and easily batch-query them, allowing for easy maintenance

```
CREATE EXTERNAL DATA SOURCE SecureBlobStorage
```

```
WITH
(
	TYPE = Hadoop,
	LOCATION = 'wasbs://<blob
container>@<storageaccount>.blob.core.windows.net',
	CREDENTIAL = AppCred
);
```

h) Create an External File Format object

```
CREATE EXTERNAL FILE FORMAT [CsvFormatWithHeaderUTF8date] WITH (
	FORMAT_TYPE = DELIMITEDTEXT,
	FORMAT_OPTIONS (
	FIELD_TERMINATOR = ',',
	FIRST_ROW  = 2,
	STRING_DELIMITER = '"',
	USE_TYPE_DEFAULT = False,
ENCODING = 'UTF8',
DATE_FORMAT = 'MM/dd/yyyy'
	)
);
```

i) You need a new object for each unique file format you'll be uploading
ii) This may need to be customized to reflect your specific file format if you are uploading something other than the Superstore CSV file
   (1) FIRST_ROW = 2 indicates the presence of a header row
   (2) FIELD_TERMINATOR = , indicates the type of delimiter
   (3) Specify ENCODING of your file
   (4) Specify the DATE_FORMAT of your data if different from the default YYYY-MM-DD [link]

5) Load data from Azure Blob Storage into external tables
   a) Note -if you are uploading large amounts of data we recommend using a loader user to avoid issues loading commands [link] - the loader user can ingest more rows that the admin and therefore is faster for loading large amounts of data into the warehouse. For this tutorial we are simply using the admin user.
   b) Create external table in staging schema based off file(s) - make a unique schema for each file type

```
CREATE EXTERNAL TABLE [staging].[superstore](
	[rowID] [int] NULL,
	[order_id] [varchar](150) NULL,
	[order_date] [date] NULL,
	[ship_date] [date] NULL,
	[ship_mode] [varchar](150) NULL,
        [customer_id] [varchar](150) NULL,
        [customer_name] [varchar](150) NULL,
        [segment] [varchar](150) NULL,
        [country] [varchar](150) NULL,
        [city] [varchar](150) NULL,
        [state] [varchar](150) NULL,
        [postal_code] [int] NULL,
        [region] [varchar](150) NULL,
        [product_id] [varchar](150) NULL,
        [category] [varchar](150) NULL,
        [sub_category] [varchar](150) NULL,
        [product_name] [varchar](150) NULL,
        [sales] [float] NULL,
        [quantity] [int] NULL,
        [discount] [float] NULL,
        [profit] [float] NULL
 )
 WITH (
 LOCATION='/',
  DATA_SOURCE = SecureBlobStorage,
  FILE_FORMAT = CsvFormatWithHeaderUTF8date,
  REJECT_TYPE = VALUE,
  REJECT_VALUE = 0
);
```

- i) LOCATION: specify csv file name OR say "/" - that will pull all CSVs in the blob container - use this only if all files have the same format. Alternatively you can use folders to separate different file structures
- ii) Customize FILE_FORMAT to match the one created in step 4g
- iii) Customize DATA_SOURCE to match the one created in step 4f

c) Pause here to query the tables (expand Tables and External Tables in the Object Explorer and right click on the external table and choose *Select top 1000 rows*) and check the results
- i) Performance may be poor since this is pulling data from storage blob via semantic layer & the data is not yet stored in the SQL DW

d) Troubleshooting CREATE EXTERNAL TABLE errors
- i) "Row has exceeded maximum polybase row size limit of 1,048,576 bytes"
  - (1) Use fewer, or smaller VARCHAR types
- ii) "… java.util.IllegalFormatConversionException: d != java.lang.String"
  - (1) Make sure you have the correct UTF encoding by opening in a rich text editor like VSCode
  - (2) If your columns have commas in them make sure they have string escape characters as well

iii) "Error converting data type VARCHAR to DATETIME"
   (1) Make sure you specify the correct date format. See step 4g

iv) MalformedInputException: Input length = 1
   (1) Check that there are no unrecognized characters (sometimes happens with UTF8)

6) Create internal table with appropriate distribution [link]
   a) A round robin distributed table distributes table rows evenly across nodes and is appropriate for a smaller dataset with no apparent joining key

```
CREATE TABLE [dbo].[superstore_round_robin]
WITH
(
          DISTRIBUTION = ROUND_ROBIN
,   CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT  *
FROM  [staging].[superstore]
;
```

   b) A replicate distribution adds a copy of the dataset to every compute node and is appropriate for small dimension tables in a star schema [link]

```
CREATE TABLE [dbo].[superstore_replicate]
WITH
(
          DISTRIBUTION = REPLICATE
,   CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT  *
FROM  [staging].[superstore]
;
```

      i) For replication, issue a simple query in SSMS before querying to trigger initial table replication

```
Select Top 1 * from [dbo].[superstore_replicate];
```

   c) A hash-distributed table distributes table rows across the Compute nodes by using a deterministic hash function

```
CREATE TABLE [dbo].[superstore_hash]
```

```
WITH
(
        DISTRIBUTION = HASH([customer_id])
,   CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT  *
FROM  [staging].[superstore]
;
```

7) Operationalize access to data & add necessary users for Tableau access
    a) For production usage don't use a loader user as it will consume all the resources so no other users can query efficiently and don't use the admin user since it is locked into a small resource class
    b) To support high concurrency - put all users in a group and provision with small to medium RC (resource class) [link]
    c) For additional control consider using static resource classes [link]
8) Connect from Tableau and begin your analysis. Use the credentials for the admin user for testing and user credentials created according to step 7 for production usage. [link]
9) When you are finished make sure to clean up your Azure resource to prevent accumulating costs [link]
10) Refer to Azure documentation for additional best practices [link]