**Data Orchestration:**

These are scenario questions, we are looking for a written response which can include pseudocode if needed. No sample data is associated with these questions.

1. **Data is written into Azure Blob Storage on a daily cadence in a csv file. Describe a process how you would create a new table in a Snowflake database off of the Azure Blob Storage source. We want the Snowflake table to be updated on a weekly cadence. Provide a written explanation.**

**Set up Snowflake External Stage:** Create an external stage in Snowflake to access files stored in Azure Blob Storage. This involves defining a stage object that points to the location of the CSV files in Azure Blob Storage.

**Pseudocode:**

CREATE OR REPLACE STAGE azure\_blob\_stage

URL = 'azure://<storage\_account\_name>.blob.core.windows.net/<container\_name>'

CREDENTIALS = (

AZURE\_SAS\_TOKEN='<your\_azure\_sas\_token>'

);

**Define Snowflake Table Schema:** Define the schema for the Snowflake table that will hold the data from the CSV files. This includes specifying column names, data types, and any constraints necessary to accurately represent the data.

**Create an External Table**: Create an external table in Snowflake that references the files stored in Azure Blob Storage. Specify the file format, location of the CSV files, and schema mapping to map the CSV file columns to Snowflake table columns.

**Schedule Data Loading**: Use Snowflake's scheduled tasks or an external orchestration tool (e.g., Airflow, Azure Data Factory) or you can even write code in python script to schedule the data loading process from Azure Blob Storage to Snowflake. Configure the schedule to run on a weekly cadence, aligning with the frequency of data updates in Azure Blob Storage.

1. **SNOWFLAKE SCHEDULED TASKS:**

**Creating Scheduled Tasks without Cron Expression:**

-- Create a Scheduled Task to load data from Azure Blob Storage to a Snowflake table

CREATE OR REPLACE TASK weekly\_data\_load

WAREHOUSE = your\_warehouse

SCHEDULE = 'WEEKLY'

TIMESTAMP = '2024-02-11 02:00:00'

COMMENT = 'Load data from Azure Blob Storage';

-- Define the SQL script to execute in the task

ALTER TASK weekly\_data\_load

WAREHOUSE = your\_warehouse

STATEMENT = 'COPY INTO your\_snowflake\_table FROM @azure\_blob\_stage FILE\_FORMAT = (TYPE = ''CSV'' FIELD\_OPTIONALLY\_ENCLOSED\_BY=''""'' SKIP\_HEADER = 1);';

-- Enable and start the Scheduled Task

ALTER TASK weekly\_data\_load

RESUME;

ALTER TASK weekly\_data\_load

SET STATE = 'ENABLED';

**SCHEDULE = 'WEEKLY' specifies that the task should run weekly.**

**TIMESTAMP specifies the exact date and time when the task should start.**

**STATEMENT defines the SQL script that the task should execute**.

**Creating Scheduled Tasks with Cron Expression:**

-- Create a Scheduled Task with a cron expression

CREATE OR REPLACE TASK weekly\_data\_load\_cron

WAREHOUSE = your\_warehouse

SCHEDULE = 'USING CRON 0 2 \* \* 0'

COMMENT = 'Load data from Azure Blob Storage';

-- Define the SQL script to execute in the task

ALTER TASK weekly\_data\_load\_cron

WAREHOUSE = your\_warehouse

STATEMENT = 'COPY INTO your\_snowflake\_table FROM @azure\_blob\_stage FILE\_FORMAT = (TYPE = ''CSV'' FIELD\_OPTIONALLY\_ENCLOSED\_BY=''""'' SKIP\_HEADER = 1);';

-- Enable and start the Scheduled Task

ALTER TASK weekly\_data\_load\_cron

RESUME;

ALTER TASK weekly\_data\_load\_cron

SET STATE = 'ENABLED';

**Notes:**

**SCHEDULE = 'USING CRON 0 2 \* \* 0' specifies a cron expression that runs the task every Sunday at 2:00 AM.(zero refers to Sunday and 2 refers 2 AM(it follows 24 hr format)).**

**CRON syntax:**

# \_\_\_\_\_\_\_\_\_\_ minute (0-59)

# | \_\_\_\_\_\_\_\_ hour (0-23)

# | | \_\_\_\_\_\_ day of month (1-31, or L)

# | | | \_\_\_\_ month (1-12, JAN-DEC)

# | | | | \_ day of week (0-6, SUN-SAT, or L)

# | | | | |

# | | | | |

\* \* \* \* \*

1. **USING PYTHON SCRIPTS:**

**import snowflake.connector**

**import datetime**

**import time**

# Snowflake credentials and connection parameters

snowflake\_account = 'your\_snowflake\_account'

snowflake\_user = 'your\_snowflake\_user'

snowflake\_password = 'your\_snowflake\_password'

snowflake\_database = 'your\_snowflake\_database'

snowflake\_schema = 'your\_snowflake\_schema'

snowflake\_table = 'your\_snowflake\_table'

# Connect to Snowflake

snowflake\_conn = snowflake.connector.connect(

user=snowflake\_user,

password=snowflake\_password,

account=snowflake\_account,

warehouse='your\_snowflake\_warehouse',

database=snowflake\_database,

schema=snowflake\_schema

)

# Create a Snowflake cursor

snowflake\_cursor = snowflake\_conn.cursor()

# Schedule the data loading process to run weekly

while True:

# Get the current date

current\_date = datetime.datetime.now()

# Check if it's the start of the week (e.g., Sunday)

if current\_date.weekday() == 6:

# Perform data loading process

load\_data\_query = f"""

COPY INTO {snowflake\_table}

FROM 'azure\_blob\_stage'

FILE\_FORMAT = (TYPE = CSV)

"""

# Execute the load data query

snowflake\_cursor.execute(load\_data\_query)

snowflake\_conn.commit()

print("Data loaded successfully into Snowflake.")

# Wait for the next week's scheduling

time.sleep(604800) # 604800 seconds = 1 week

else:

# Sleep for 24 hours and check again

time.sleep(86400) # 86400 seconds = 24 hours

# Close Snowflake connection

snowflake\_cursor.close()

snowflake\_conn.close()

**Notes:**

We use the snowflake.connector library to connect to Snowflake and execute SQL queries.

The script checks if the current day is Sunday (the start of the week). If it is, it executes the data loading process into Snowflake.

After loading the data, the script sleeps for one week before checking again. This effectively schedules the data loading process on a weekly cadence.

**Transform and Load Data**: During the scheduled data loading process, extract data from the CSV files in Azure Blob Storage, transform it if necessary (e.g., data type conversion, cleansing), and load it into the Snowflake table. You can use Snowflake's COPY command or Snowpipe for efficient bulk loading of data.

1. **Let's say the above data is in our Snowflake database and the end users are asking for delivery of insights. How could we make aggregated data available for an external user to consume? Give consideration to whether the client wants the data pushed to them versus available for them to pull when needed. Assume the data cannot be passed via an emailed csv.**

**Snowflake Secure Data Sharing:**

Snowflake Secure Data Sharing allows you to securely share live, governed data with other Snowflake accounts.

You can create a secure view or materialized view in your Snowflake database that contains the aggregated data.

Then, you can share this view with external Snowflake accounts. The external users can query the shared data directly from their Snowflake accounts in real-time.

You can share data in Snowflake using one of the following options:

* a **Listing**, in which you offer a share and additional metadata as a data product to one or more accounts,
* a **Direct Share**, in which you directly share specific database objects (a share) to another account in your region,
* a **Data Exchange**, in which you set up and manage a group of accounts and offer a share to that group.

There may be different scenarios for data sharing:

* **Scenario 1**: Sharing data between same cloud and same region – you can achieve it through direct share.

1. Create a secure view or materialized view in Snowflake containing the aggregated data.

2. Identify the external Snowflake account within the same region and cloud.

3. Grant appropriate privileges to the external Snowflake account to access the secure view or materialized view.

4. Share the secure view or materialized view with the external Snowflake account using Snowflake Secure Data Sharing.

5. External Snowflake account can now query the shared data directly from their Snowflake account.

**Example code:**

-- Create a secure view or materialized view in Snowflake containing the aggregated data

**CREATE VIEW secure\_view AS**

**SELECT \* FROM aggregated\_data;**

-- Grant privileges to the external Snowflake account to access the secure view

**GRANT SELECT ON secure\_view TO ACCOUNT 'external\_account\_id';**

-- Share the secure view with the external Snowflake account using Snowflake Secure Data Sharing

**CREATE SHARE secure\_share;**

**ALTER SHARE secure\_share ADD ACCOUNT 'external\_account\_id';**

**GRANT USAGE ON DATABASE my\_database TO SHARE secure\_share;**

**GRANT USAGE ON SCHEMA my\_schema TO SHARE secure\_share;**

**GRANT SELECT ON VIEW secure\_view TO SHARE secure\_share;**

* **Scenario 2**: If you need to share data across different regions and different clouds you must perform replication.

1. Create a secure view or materialized view in Snowflake containing the aggregated data.

2. Replicate the aggregated data to a Snowflake account in the target region or cloud using Snowflake's data replication features.

3. Grant appropriate privileges to the external Snowflake account in the target region or cloud to access the replicated data.

4. Share the secure view or materialized view with the external Snowflake account in the target region or cloud using Snowflake Secure Data Sharing.

5. External Snowflake account can now query the shared data directly from their Snowflake account in the target region or cloud.

**Example code:**

-- Replicate the aggregated data to a Snowflake account in the target region or cloud

**CREATE DATABASE IF NOT EXISTS target\_database;**

**CREATE SCHEMA IF NOT EXISTS target\_schema;**

**CREATE TABLE IF NOT EXISTS target\_database.target\_schema.replicated\_data AS**

**SELECT \* FROM aggregated\_data;**

-- Grant privileges to the external Snowflake account in the target region or cloud to access the replicated data

GRANT SELECT ON TABLE target\_database.target\_schema.replicated\_data TO ACCOUNT 'external\_account\_id';

-- Share the replicated data with the external Snowflake account in the target region or cloud using Snowflake Secure Data Sharing

**CREATE SHARE replicated\_data\_share;**

**ALTER SHARE replicated\_data\_share ADD ACCOUNT 'external\_account\_id';**

**GRANT USAGE ON DATABASE target\_database TO SHARE replicated\_data\_share;**

**GRANT USAGE ON SCHEMA target\_schema TO SHARE replicated\_data\_share;**

**GRANT SELECT ON TABLE target\_database.target\_schema.replicated\_data TO SHARE replicated\_data\_share;**

* **Scenario 3**: If your end client does not have a snowflake account, we can provide reader account for the consumer (end client)

1. Create a secure view or materialized view in Snowflake containing the aggregated data.

2. Export the aggregated data to an external storage location (e.g., Amazon S3, Azure Blob Storage) using Snowflake's data export capabilities.

**COPY INTO '@s3://bucket/aggregated\_data.csv'**

**FROM (SELECT \* FROM aggregated\_data)**

**CREDENTIALS = (AWS\_KEY\_ID='your\_aws\_key\_id' AWS\_SECRET\_KEY='your\_aws\_secret\_key')**

**FILE\_FORMAT = (FORMAT\_NAME='csv' FIELD\_OPTIONALLY\_ENCLOSED\_BY='"' SKIP\_HEADER = 1);**

3. Configure secure access controls and permissions on the external storage location to restrict access to authorized users only.

4. Provide the external users with access credentials or pre-signed URLs to access the exported data files securely.

5. External users can download the data files from the external storage location using HTTPS or SFTP protocols.

6. Optionally, develop a custom API or web service to provide programmatic access to the exported data files for automated consumption.

**Scenario 3: When you have the need to provide access to end client who does not have snowflake account)**

**-- Create a role for the reader account**

**CREATE ROLE reader\_role;**

**-- Grant necessary privileges to the role**

**GRANT USAGE ON DATABASE my\_database TO ROLE reader\_role;**

**GRANT USAGE ON SCHEMA my\_schema TO ROLE reader\_role;**

**GRANT SELECT ON TABLE my\_table TO ROLE reader\_role; -- Grant SELECT access to the exported table**

**-- Create a user and assign the reader role**

**CREATE USER reader\_user PASSWORD = 'password' DEFAULT\_ROLE = reader\_role;**

**Once its done, you can share the URL to the end client, they can access and pull the data whenever needed.**

**Third-Party BI Tools Integration**: Snowflake integrates with various third-party business intelligence (BI) tools such as Tableau, Power BI, and Looker. You can create dashboards and reports using these BI tools based on the aggregated data stored in Snowflake. External users can access the dashboards and reports through web interfaces or dedicated BI applications.

**Data Extraction and Delivery**:

To share data from Snowflake with a client via SFTP (SSH File Transfer Protocol), you would typically follow these general steps:

Extract Data from Snowflake: Extract the data from Snowflake in a format suitable for transfer, such as CSV, JSON, or Parquet.

Transfer Data to SFTP Server: Use an SFTP client or library to transfer the extracted data files to the client's SFTP server.

**Pseudo Code Explanation:**

1. **Extract Data from Snowflake:**

Use Snowflake's COPY INTO command to extract data from a specific table.

Save the extracted data into a staging area within Snowflake.

Ensure the file format and compression settings are suitable for the transfer.

1. **Transfer Data to SFTP Server:**

Use a suitable programming language (like Python) with an SFTP library (like pysftp) to establish a connection to the SFTP server.

Provide the necessary authentication credentials (host, username, password, port).

Define the local file path of the extracted data file and the remote directory on the SFTP server where the file will be uploaded.

Connect to the SFTP server and upload the local file to the specified remote directory.

**Sample Code:**

import pysftp

# Connection details for the SFTP server

sftp\_host = 'sftp.example.com'

sftp\_user = 'username'

sftp\_password = 'password'

sftp\_port = 22

# Path to the local file to be transferred

local\_file\_path = '/path/to/local/data.csv'

# Remote directory on the SFTP server to upload the file

remote\_dir = '/incoming/'

try:

# Connect to the SFTP server

with pysftp.Connection(host=sftp\_host, username=sftp\_user, password=sftp\_password, port=sftp\_port) as sftp:

print("Connection successful")

# Change directory to the remote directory

sftp.cwd(remote\_dir)

# Upload the local file to the SFTP server

sftp.put(local\_file\_path)

print("File uploaded successfully")

except Exception as e:

print("Error:", str(e))