O'REILLY®

Continuous Data Pipelines

Tomas Sobotik

Batch x Continuous

Batch processing

Ingestion at given time

Latency in hours or days

User specified WH

Always single transaction

Load Metadata stored for 64 days

Continuous processing

Ingestion triggered by event

Latency in minutes

Serverless – Snowflake provide WH

Single or multiple transactions based on the number and size of the rows

Load Metadata stored for 14 days

Continuous processing

Benefits

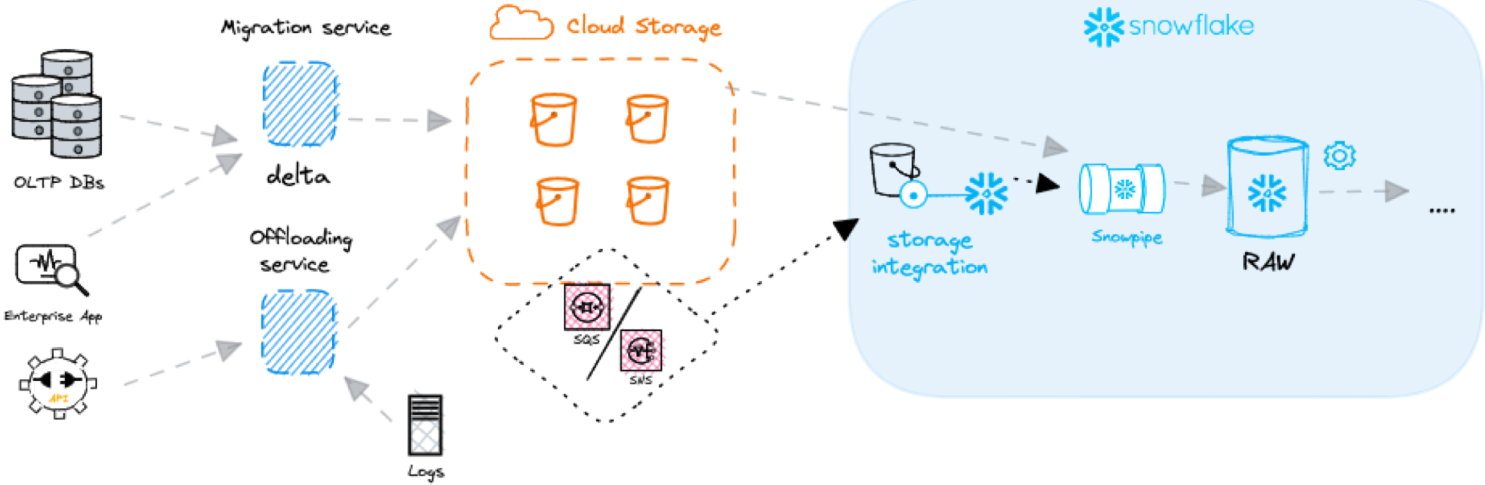
+ continuous ingestion
+ ingestion automation
+ better latency
+ serverless

Event driven

- based on notifications from cloud messaging services
- REST API call

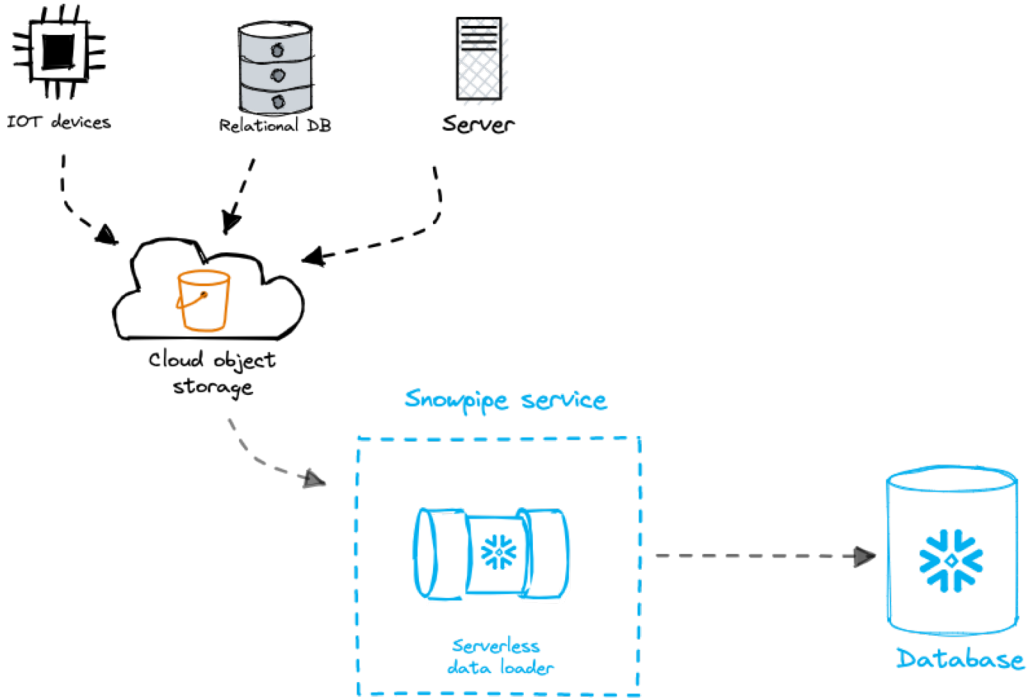
Cons

- latency in mins
- harder reprocessing
- need for logging & monitoring

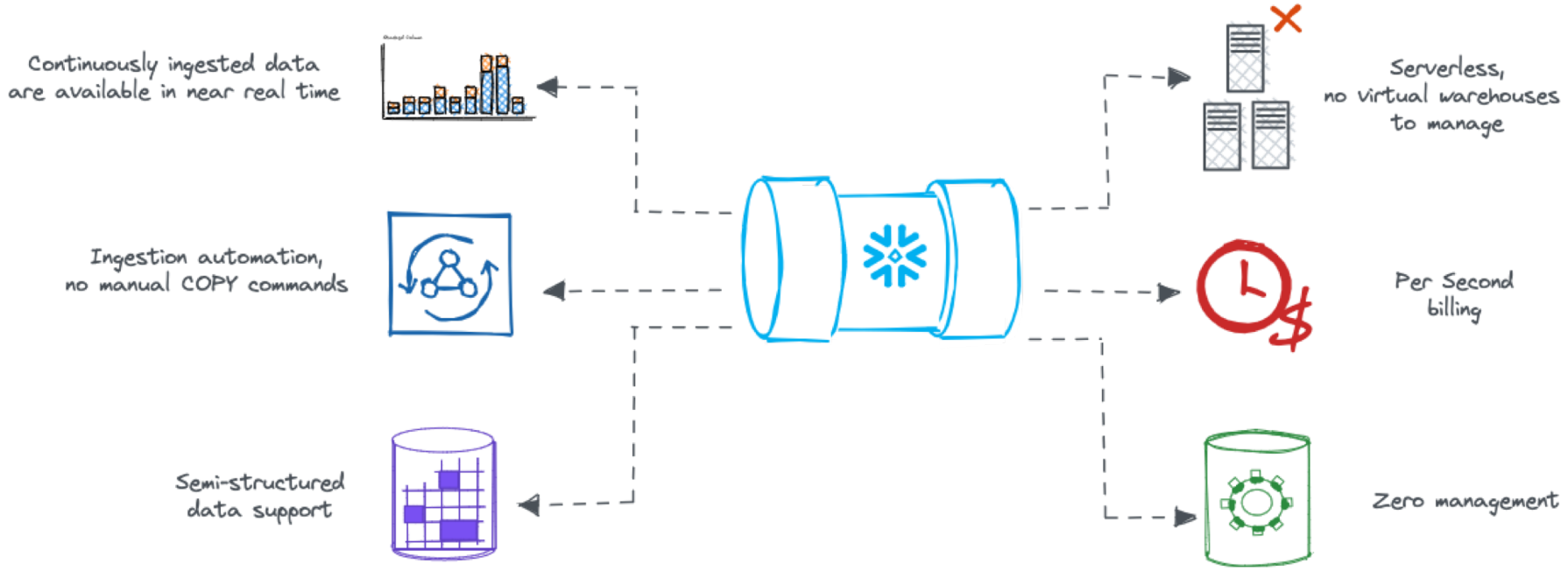


Snowpipes

- Automated data loading feature
- Near real time data streaming
- No need to dedicate and suspend virtual warehouse
- Serverless with per second billing
- File (batch) oriented
- Credit consumption is visible under SNOWPIPE warehouse
- Use cases
  - Automated data ingestion
  - IOT data stream import
  - Data Lake files import

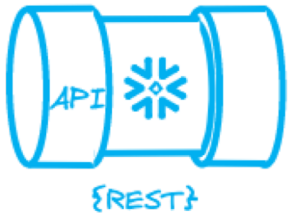


Faster data ingestion with Snowpipes



Snowpipes – how it works

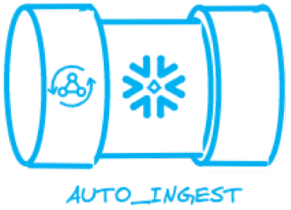
- Pre-defined object that specifies the COPY command for data load
- Contains instructions how and where to load and convert data into records in the target table
- Supports all data types (including semistructured)



Manually call Snowpipe REST API endpoint

Pass a list of stage files in the stage location

Works with internal and external stages



Receive notifications from cloud provider when files arrive

Notification trigger the processing

Only external stages are supported

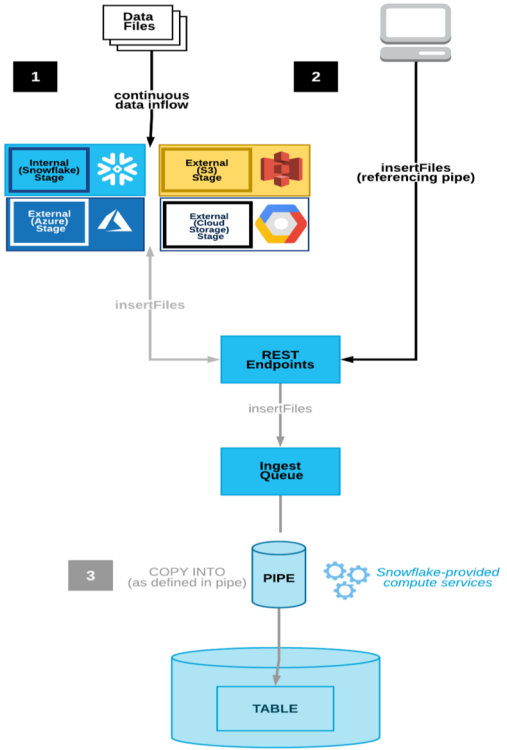
Snowpipe cross cloud support

Snowflake account Host	Amazon S3	Google Cloud Storage	Microsoft Azure Blog Storage	Microsoft Data Lake Storage Gen2	Microsoft Azure General-purpose v2
Amazon Web Services					
Google Cloud Platform					
Microsoft Azure					

Snowpipe REST endpoints to load data

- Uses key-based authentication when calling REST endpoints
- Data apps approach
  - 1. Data files copied to an internal or external stage
  - 2. Client calls `insertFiles` endpoint with list of files to ingest and a defined pipe
  - 3. Snowflake provided WH loads files to table

- Use client app / AWS Lambda to call the REST Endpoint



Preparing to load data using REST API

1. Create Stage
2. Create a Pipe
3. Security configuration
    - Generate Public-private key pair for making calls to REST API
    - REST endpoint uses JSON Web Token (JWT) for authorization
    - Assign the public key to the user
    - Granting privileges to DB objects in use (DB, schema, pipe, target table…)
    - Best practice – create separate user to handle the REST api calls

4. Stage data files in internal/external stage
5. Call REST API endpoint `insertFiles`

Snowpipe REST API endpoints

- `insertFiles`
  - `POST`
  - Informs Snowflake about files to be ingested into a table
  - `https://{account}.snowflakecomputing.com/v1/data/pipes/{pipeName}/insertFiles?requestId={requestId}`

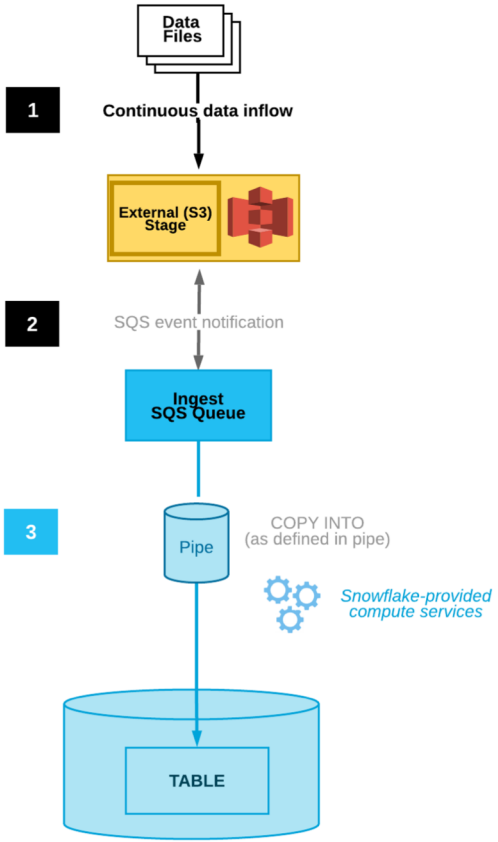
- `insertReport`
  - `GET`
  - Retrieves a report of files submitted via `insertFiles`
  - The 10000 most recent events
  - Events are retained for max 10 mins

- `loadHistoryScan`
  - `GET`
  - Report about ingested files
  - Return history between two points in time
  - Max 10000 items returned
- For more comprehensive view of history without limits use Information Schema table function – COPY_HISTORY

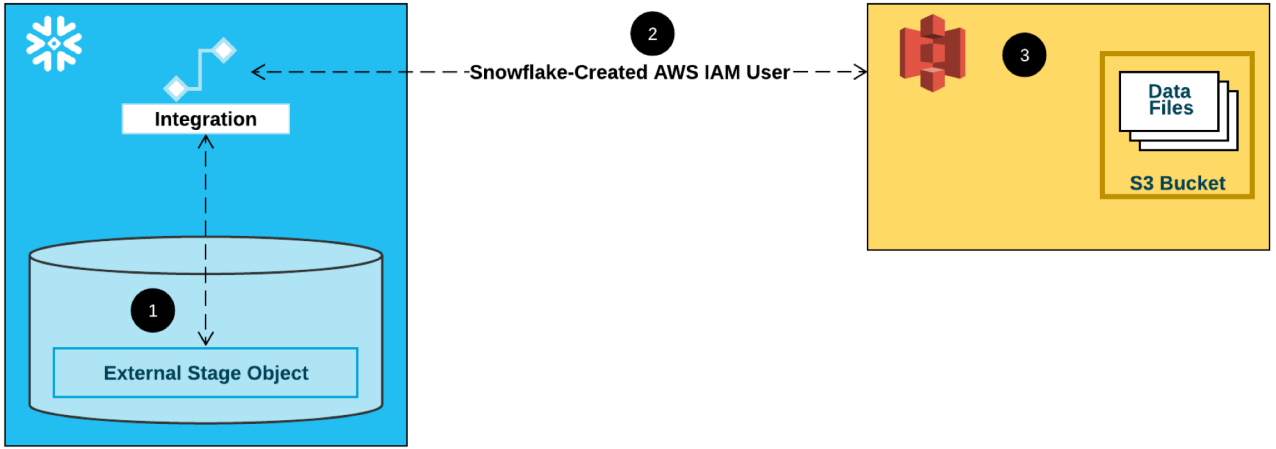
How to create Snowpipe with Auto Ingest

- It requires configuration on cloud provider side
  - Bucket access permission
  - IAM role for Snowflake
  - SQS notification for S3 bucket
- Snowflake configuration
  - Storage integration object - encapsulates the access privileges for accessing the S3 bucket

```
create pipe mypipe auto_ingest=true as
  copy into mytable
  from @my_db.public.mystage
  file_format = (type = 'JSON');
```



Configure secure Access to cloud Storage



1. External stage references a storage integration object
2. Snowflake associate storage integration with S3 IAM user created for your account
3. AWS admin grants permission to the IAM user to access the bucket referenced in the stage definition.

Configure Access permissions for the S3 bucket

- You need to grant following permissions to Snowflake
  - s3:GetBucketLocation
  - s3:GetObject
  - s3:GetObjectVersion
  - s3:ListBucket
- Best practise: Create an IAM policy for Snowflake access to the S3 Bucket
  - Assign the policy to IAM role

Storage integration object and integration between Snowflake and AWS

1. Create Storage Integration object
    - Use IAM role created in previous step
2. Retrieve AWS IAM user for your Snowflake account
    - `DESC Integration my_integration`
    - Record the STORAGE_AWS_IAM_USER_ARN and STORAGE_AWS_EXTERNAL_ID
3. Grant the IAM user Permissions to Access Bucket Objects
    - Configure a Trust relationship in IAM role we created earlier

Create a S3 Event Notification to Automate Snowpipe

Let’s use SQS for that

1. Execute show pipes to get the notification channel associated with snowpipe
2. Configure event notification for S3 bucket associated with the stage
    - All events related to object creation
    - Send a notification to SQS Queue we got in the first step

Poll

1. How big warehouse is recommended to assign to snowpipe

a) MEDIUM
b) X-LARGE
c) no warehouse should be assigned

2. What are snowpipe types? Select all of them

A) Notification based
B) REST endpoint
C) Manually triggered
D) AUTO INGEST

3. What billing model does snowpipe use?

a) Per file billing
b) Per load billing
c) Per second billing

4. What objects are used by snowpipe? Select all of them

a) Virtual warehouse
b) COPY command
c) FILE_FORMAT
d) Stream
e) Stage

Snowpipe demo & exercise

Exercise

We are going to practise the continuous data ingestion into Snowflake in this exercise. We will use snowpipe feature for this. It again requires configuration on the Cloud provider which is similar to settings which we have done for the batch data ingestion. There are required the bucket access permission and IAM role in order to access Snowflake. Now we also need some kind of notification to be sent to snowpipe when a new file will be landed in cloud storage.

We will do all the needed setup on both ends (Snowflake & AWS) in this exercise and then we will try to ingest some data through snowpipe.

In the next session we will be talking more how to automate it further and automatically process newly loaded data via snowpipe.

Detailed description of this exercise is available on github:

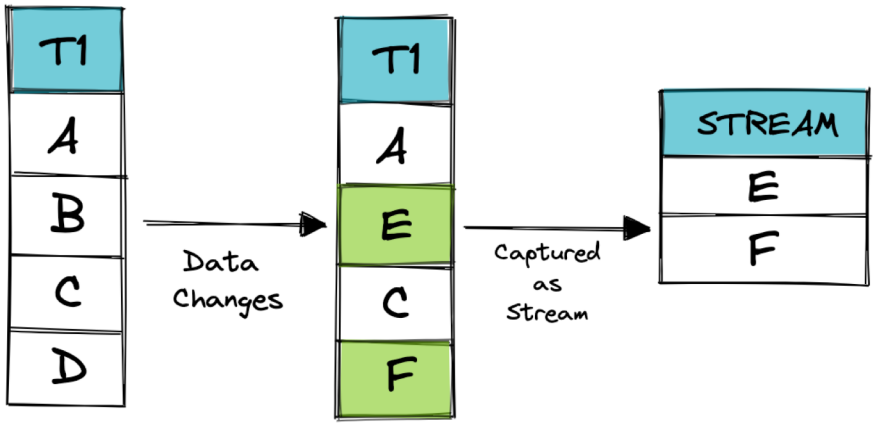
sf_data_engineering_bootcamp/week1/exercise_desc.pdf

O'REILLY®

Streams and Tasks

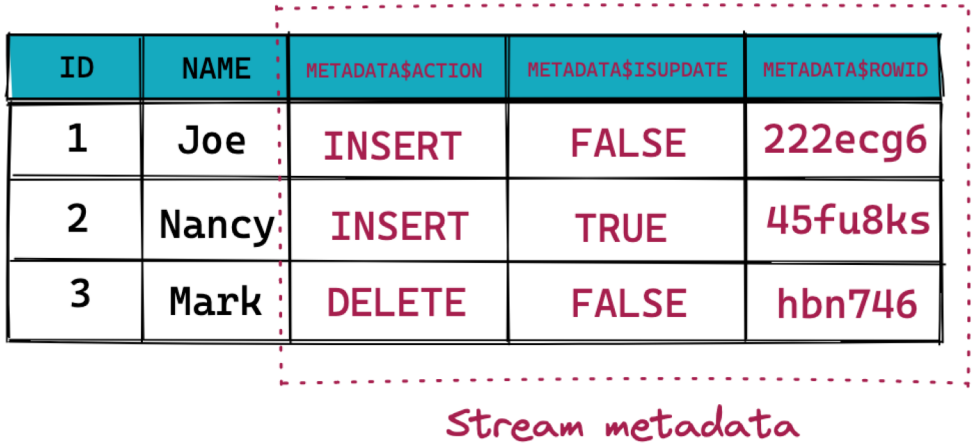
Tomas Sobotik, 9.6.2023

Streams



- Changed data capture (CDC)
- Identify and act on changed records
- Stream append 3 metadata columns to the table
  - Tracks the changed data
  - Little storage required
- Can be queried (consumed) as standard table/view
- When consumed as part of DML operation, the stream is cleared
- Stream itself does not contain any data
- Offset = point in time

Streams



- Stream metadata columns
  - METADATA$ACTION
  - METADATA$ISUPDATE
  - METADATA$ROWID
- Type of streams
  - Standard (delta)
  - Append-only
  - Insert-only (external tables)
- Supported objects to track changes
  - tables
  - directory tables
  - external tables
  - views (in preview)

Data retention and staleness

- If stream is not consumed within defined period of time it becomes stale and data inaccessible
- Default value: 14 days
- Controlled by two parameters
  - DATA_RETENTION_TIME_IN_DAYS
  - MAX_DATA_EXTENSION_TIME_IN_DAYS

- How to check?
  - DESCRIBE STREAM or SHOW STREAMS
    - STALE and STALE_AFTER columns

- When stale => recreate the stream
- Multiple consumers? -> multiple streams

Staleness

- Retention period for table < 14 days -> Snowflake temporarily extend the period up to a maximum of 14 days by default
- Maximum number of extension days is controlled by MAX_DATA_EXTENSION_TIME_IN_DAYS

DATA_RETENTION_TIME_IN_DAYS	MAX_DATA_EXTENSION_TIME_IN_DAYS	Consume Stream in X Days
14	0	14
1	14	14
0	90	90

Streams on Views

- Views, secure views but not MVs
- Views in Data Sharing!
- Requirements
  - All of underlaying tables must be native tables
  - Not yet supported operations: GROUP BY, QUALIFY, LIMIT, correlated subqueries, subqueries not in FROM clause
  - Only native scalar functions
  - Enabled change tracking on underlaying tables
- Be careful about Join Behavior
  - Changes that have occurred on the left table since the stream offset are being joined with the right table, changes on the right table since the stream offset are being joined with the left table, and changes on both tables since the stream offset are being joined with each other.
  - https://docs.snowflake.com/en/user-guide/streams-intro#join-results-behavior

CHANGES – alternative to Streams

- On table, views – not for directory and external tables
- Enables querying the change tracking metadata without having to create a stream with explicit transactional offset.
- Multiple queries can retrieve the change tracking metadata between different start and endpoints
- You must enable change tracking on source object or have as stream
  - `ALTER TABLE t1 SET CHANGE_TRACKING = TRUE`

- Parameters
  - `INFORMATION => DEFAULT | APPEND_ONLY`
  - `OFFSET =>` difference in seconds from the current time
  - `STATEMENT => <id>` – query ID as a reference point for Time Travel
    - DML, SELECT, BEGIN, COMMIT transaction
  - `STREAM => <stream name>`, stream offset is used as the AT point in time

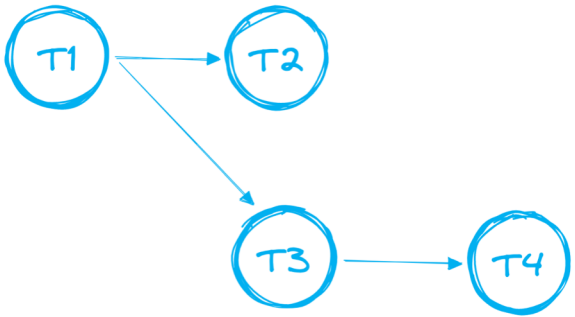
Exercise 1 - Practise the streams

Let’s follow-up on previous session dedicated to snowpipes. Now we try to automatically process the data which have been ingested by snowpipe. We need two additional database objects for it - streams and task.

In this first exercise we will create stream object on top of our landing table from previous session dedicated to snowpipe and we will be slowly building up the complete integration for continuous integration. Detailed instructions are again available on github.

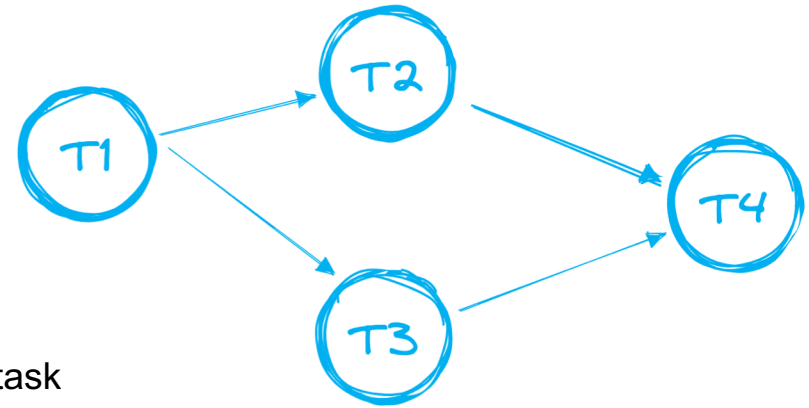
Tasks

- Scheduled execution of SQL operation
  - Single SQL statement
  - Call to a stored procedure
- Often combine with streams for ELT pipelines to process changed table rows
- Trigger
  - Schedule (CRON)
  - Interval (minutes, seconds ..)
  - Predecessor - child task starts when parent task completes
  - Condition = stream contains a new data
  - Manually
- Tasks can be chained together to create complex data pipelines (DAGs) - tree of tasks



Tree of tasks details

- One root task
- Only single direction - no loop support
- Total numbers
  - 1000 tasks in total for single DAG
  - 100 predecessors tasks
  - 100 child tasks
- If any task in the tree needs to be updated, the root task needs to be always suspended



- How to check the dependencies between tasks
  - TASK_DEPENDENTS table function
  - New UI

Task creation

- Key parameters
  - WAREHOUSE
  - SCHEDULE
  - WHEN
  - AFTER
  - SUSPEND_TASK_AFTER_NUM_FAILURES

```
create or replace task my_task
   warehouse = xsmall_vwh
   schedule = '1 minute'
   as
    insert into dim_customers (
        select customer_id, customer_name
        from s_src_customer_stream
    where metadata$action = 'INSERT');
```

- Newly created task is suspended
  - Needs to be resumed by ALTER command

Task triggered by new data in stream

```
create or replace task my_task
   warehouse = xsmall_vwh
   schedule = '5 minute'
   when SYSTEM$STREAM_HAS_DATA('s_src_customer_stream')
   as
    insert into dim_customers (
        select customer_id, customer_name
        from s_src_customer_stream
    where metadata$action = 'INSERT');
```

Serverless Task creation

- Key parameters
  - ~~WAREHOUSE~~
  - SCHEDULE
  - WHEN
  - AFTER
  - SUSPEND_TASK_AFTER_NUM_FAILURES

```
create or replace task my_task
   warehouse = xsmall_vwh
   schedule = '1 minute'
   as
    insert into dim_customers (
        select customer_id, customer_name
        from s_src_customer_stream
    where metadata$action = 'INSERT');
```

- Newly created task is suspended
  - Needs to be resumed by ALTER command
- Simpler and more cost efficient
- Existing task can be modified to be serverless
- It uses last few executions to decide WH size

Task triggered by new data in stream

```
create or replace task my_task
   warehouse = xsmall_vwh
   schedule = '5 minute'
   when SYSTEM$STREAM_HAS_DATA('s_src_customer_stream')
   as
    insert into dim_customers (
        select customer_id, customer_name
        from s_src_customer_stream
    where metadata$action = 'INSERT');
```

Exercise 2 – task creation

And last part for our continuous data pipeline. We need to somehow process the loaded data which are now available in the stream. For that we need task which will be running only when the stream has some fresh data. Thanks to that we will achieve complete integration, starting by uploading fresh data into external stage. Such integration will be incremental, we will be always processing only the freshly loaded data.

O'REILLY®

Monitoring of Data Pipelines

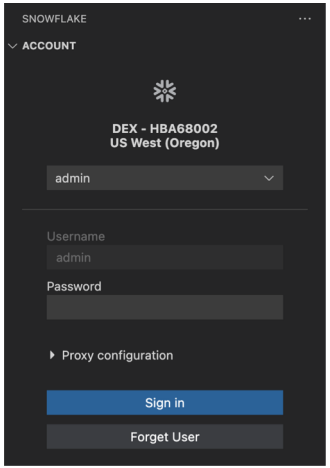
Tomas Sobotik

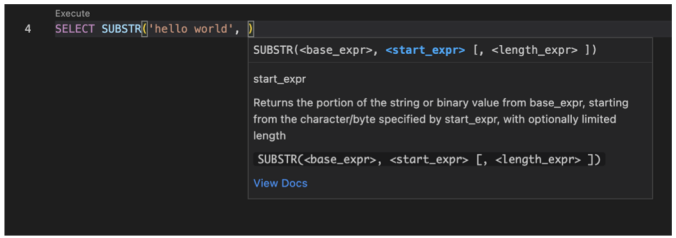
Challenges with Snowflake data pipelines

- Compared to traditional ELT/ETL tools (DBT, Azure Data Factory, Fivetran, etc.) Snowflake is lacking a default functionality in relation to:
  - Version control
  - Monitoring (failures, not running pipelines, test cases, etc.)
  - Dataset management (tables/views creation or updates)
- Snowflake offers features how for building own monitoring solution around available metadata
- Third party integrations
  - VS Code (GIT support)
  - SNS + Lambda (error notifications)
  - Streamlit (monitoring dashboards)

Version control

- No built-in support for GIT
- Keeping data pipeline code in GIT is almost necessary
  - Can save a lot of pains
- Workaround – VS CODE and Snowflake Official extension
  - VS CODE has native GIT support
  - Snowflake extension can trigger the code in Snowflake
    - Intellisense support
    - DB explorer
    - Query results and history
    - Connect to multiple accounts and easily switch
  - Write code in VS Code, run it in SF and version it in GIT



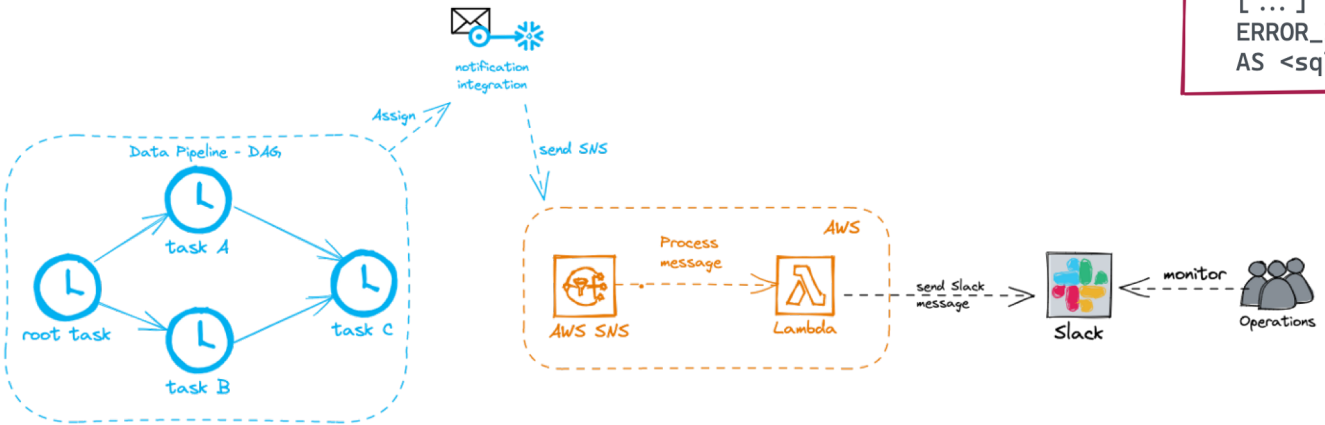


VS CODE & Snowflake demo

Error notifications for tasks

- Receive a notification when task fails
- Snowflake and Cloud Provider integration (AWS, Azure, GCP)
- new Snowflake object **NOTIFICATION INTEGRATION**

```
CREATE TASK <name>
  [ ... ]
  ERROR_INTEGRATION = <integration_name>
  AS <sql>
```



Complete step by step guide & in detail overview:
https://medium.com/snowflake/error-notifications-for-snowflake-tasks-ca5798884e67

Error Notification configuration for AWS SNS

1. Create an SNS Topic
    - Use same region as your Snowflake account - better latency, avoid egress cost
2. Create IAM policy
    - Allows publish to the SNS topic
3. Create IAM Role
    - We assign the privileges on the SNS topic
    - This will be granted to third party (Snowflake)
4. Create Notification Integration object in Snowflake
5. Grant Snowflake Access to the SNS Topic
6. Enable Error Notification in Tasks
    - In task DAG it is enough to assign it to root task

Error Notification integration demo & exercise

Exercise 1 – Create error Notification integration

In the first exercise in this session we are going to build an error notification for task. When task fails, an SNS message will be sent and we can react on that message somehow - send a slack/teams notifications for instance. As we do not have a Slack environment available we will build everything up to that part when it should be send to slack.

Detailed description is available on github -> w1_exercise_desc.pdf

Task failures troubleshooting

1. Did Task run or not ?
   - Query the TASK_HISTORY table function
   - Task may have run but the SQL failed
   - Verify if the predecessor task has run
2. Verify if task was resumed
   - DESCRIBE TASK or SHOW TASK
3. Verify the permissions of the task owner
4. Verify the condition
   - Are there any data in the stream
5. Check the task timeout
   - 60 min default value
   - Could be changed with USER_TASK_TIMEOUT_MS parameter
   - Increase the WH size
   - Rewrite the SQL statement

Monitoring the pipelines

- Have a robust operation system around the pipelines
  - Stream stale prevention
  - Task failure notifications
  - Task suspension notifications
  - Pipeline overview

Monitoring the pipelines

- Have a robust operation system around the pipelines
  - Stream stale prevention
  - Task failure notifications
  - Task suspension notifications
  - Pipeline overview

Where to get data for it?

Monitoring the pipelines

- Have a robust operation system around the pipelines
  - Stream stale prevention
  - Task failure notifications
  - Task suspension notifications
  - Not running tasks
  - Pipeline overview

Where to get data for it?

Snowflake metadata

Task history retrieval

- TASK_HISTORY view or table function
- Table function = last 7 days history
- View = last year history

- Key columns
  - NAME
  - STATE
  - QUERY_TEXT
  - ERROR_MESSAGE
  - ROOT_TASK_ID

```
select *
  from table(information_schema.task_history(
    scheduled_time_range_start=>dateadd('hour',-1,current_timestamp()),
    result_limit => 10,
    task_name=>'MYTASK'));
```

```
select *
from snowflake.account_usage.task_history
limit 10;
```

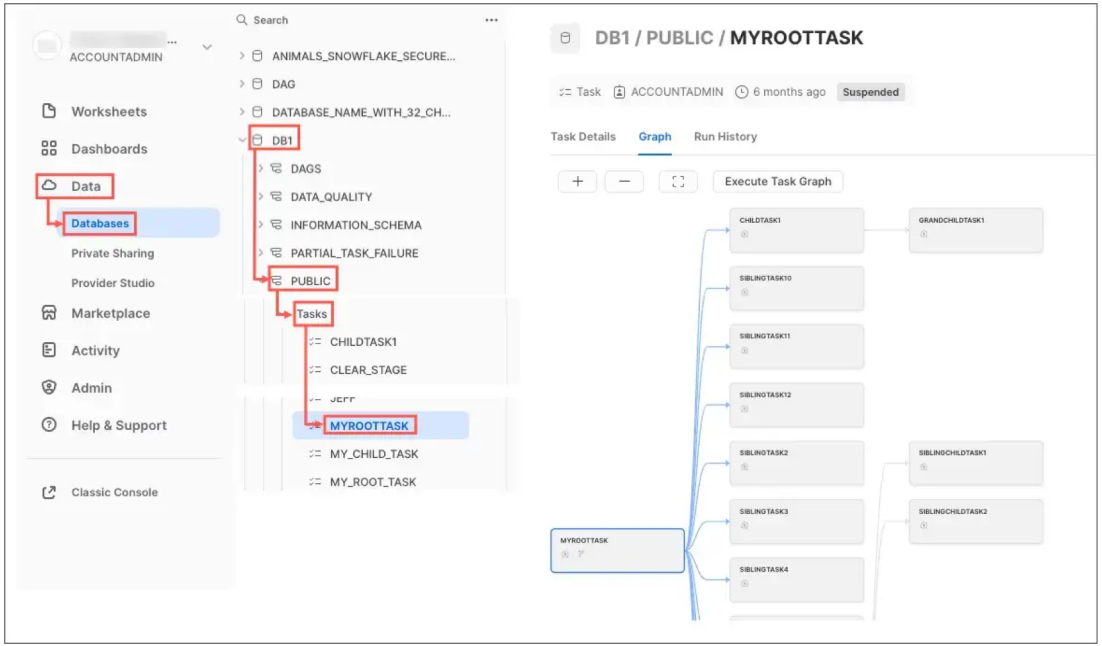
Stream stale & task suspension prevention

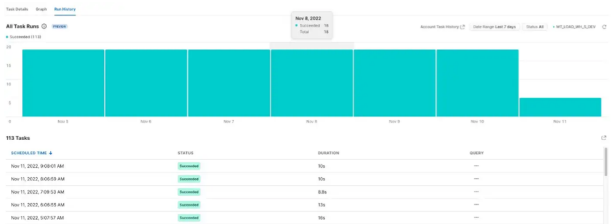
- SHOW STREAM & SHOW TASK commands to get details about both features and their current state
- Process them with result_scan function to filter relevant entries (stale stream, suspended task)
- Have a process to notify a team about such situation
  - Lambda function + Slack/Teams API
  - Email Notification integration object + SYSTEM$SEND_EMAIL() stored procedure

```
CREATE NOTIFICATION INTEGRATION my_email_int
TYPE=EMAIL
ENABLED=TRUE
ALLOWED_RECIPIENTS=('first.last@example.com','first2.last2@example.com');
```

```
CALL SYSTEM$SEND_EMAIL(
'my_email_int',
'person1@example.com, person2@example.com',
'Email Alert: Task A has been suspended. ', 'Last Run Time: XXXXX' );
```

New UI for viewing DAG and task history





Exercise 2 – email notification - task state check

We are going to try improve our pipeline little bit and built also email notification for monitoring the task state. We will build a stored procedure which will be checking the state of the task. In case of the suspended state, an email will be sent to defined email addresses.

This solution will use another notification integration but this time the type of it will be email. In this case we do not need to create any integration towards cloud provider as the email is sent directly by Snowflake. But this feature currently works only for Snowflake accounts hosted on AWS. If your account uses another cloud provider this will not work for you.

Detailed description is available on github -> w1_exercise_desc.pdf

O'REILLY®

Stored Procedures in Snowflake

Tomas Sobotik, 9.6.2023

Stored procedures

- Extend the system with procedural code executing SQL
- Support for SQL, JavaScript, Python, Java
- Bundle multiple SQL commands into single callable script
- Variables, loops, conditions, cursors
- Transaction support
- Error handling
- Dynamic creation of SQL
- Called as independent statements
  - CALL myStoredProcedure(argument);
- One SP can call another one
- Can return a value
- Cannot return a set of rows

Use cases

- Task automation
  - Requires combination of multiple sql statements or additional logic
- DB clean up
  - Remove data from DB older than XY days
  - multiple DELETE statements bundled into single SP, pass the date as parameter
  - Could be scheduled as a task or run separately
- DB backup
- ML models calculations
- Data compliance verification
- ETL pipeline

Stored procedure example

```
Create or replace procedure myProcedure()
returns varchar
language sql
as $$
        --Snowflake scripting code
        declare
          r float;
          area_of_circle float;
        begin
          radius_of_circle := 3;
          area_of_circle := pi() * r
          return area_of_circle;
        end;
$$
;
```

How to choose the right language

- Own preference
- You already have some other code in that language – consistency
- Language has capabilities which other does not have
- Language has libraries which can help you with data processing
- Do you want to keep your stored procedure code in-line or externally (file on stage)?

Language	Handler Location
Java	In-line or staged
JavaScript	In-line
Python	In-line or staged
Scala	In-line or staged
SQL	In-line

In-line handler example

```
create or replace procedure my_proc(from_table string, to_table string, count int)
returns string
language python
runtime_version = '3.8'
packages = ('snowflake-snowpark-python')
handler = 'run'
as
$$
def run(session, from_table, to_table, count):
        session.table(from_table).limit(count).write.save_as_table(to_table)
return "SUCCESS"
$$;
```

Way of working:

1. Develop & test handler code locally
2. Compile it if necessary (Java, Scala)
3. Copy to Snowsight
4. Create a Stored Procedure

Stage handler example

```
CREATE OR REPLACE PROCEDURE MYPROC(value INT, fromTable STRING, toTable STRING)
RETURNS INT
LANGUAGE JAVA
RUNTIME_VERSION = '11'
PACKAGES = ('com.snowflake:snowpark:latest')
IMPORTS = ('@mystage/MyCompiledJavaCode.jar')
HANDLER = 'MyJavaClass.run';
```

Way of working:

1. Develop & test handler code locally
2. Compile it if necessary (Java, Scala)
3. Upload to internal stage
4. Create a Stored Procedure

Keeping Handler code In-line or on a Stage?

In-line Handler Advantages

- Easier to implement
- Update the code with ALTER command
- Maintain the code directly in Snowsight
- If code needs to be compiled (Java, Scala) it is possible to define a location for output path with TARGET_PATH
- Then code is not compiled with each SP call – faster execution of repeated calls

Stage Handler Advantages

- Can use code which might be too large for in-line handler
- Handler can be reused by multiple stored procedures
- Easier to debug / test in existing external tools – especially for complex and large code

Caller's vs Owner's Rights

Caller's rights

- Runs with privileges of the caller
- Knows caller's current session
- Nothing what caller can't do outside SP can't be even done in SP
- Changes in session persist after end of SP call
- Can view, set, unset caller's session variables and parameters
- Use it when
  - SP operates only on objects owns by caller
  - You need to use caller's environment (session vars)

Owner's rights (default option)

- Runs with privileges of the SP owner
- If SP owner have some privilege (delete data), SP can delete them even if the caller can't
- Can't change session state
- Can't view, set, unset caller's session variables and parameters
- SP does not have access to variables created outside the stored procedure
- Use it when
  - You want to delegate some task to another role without granting such privilege to that role (delete data)
  - You want to prevent callers from viewing the source code of SP

SP walkthrough

Exercise

We are going to improve the monitoring of our data pipeline based on snowpipe, stream and task. We will build another stored procedure which will be monitoring the stale state of our STR_LANDING stream.

O'REILLY®

User Defined Functions in Snowflake

Tomas Sobotik, 9.6.2023

User Defined functions

- Support for SQL, JavaScript, Python, Java
- Develop a reusable logic which is not part of Snowflake
  - Area of circle
  - Profit per department
  - Concatenate names
  - Get bigger number
- Only SQL and JavaScript UDFs can be shared
- Secure version (data sharing)
- Called from SQL query
- DML and DDL is not permitted
- MUST return value

User defined function example

```
Create or replace function addone(i int)
returns int
language python
runtime_version = '3,8'
handler = '3,8'
as $$
  def addone_py(i):
     return i+1
$$;
```

```
SELECT addone(3)
```

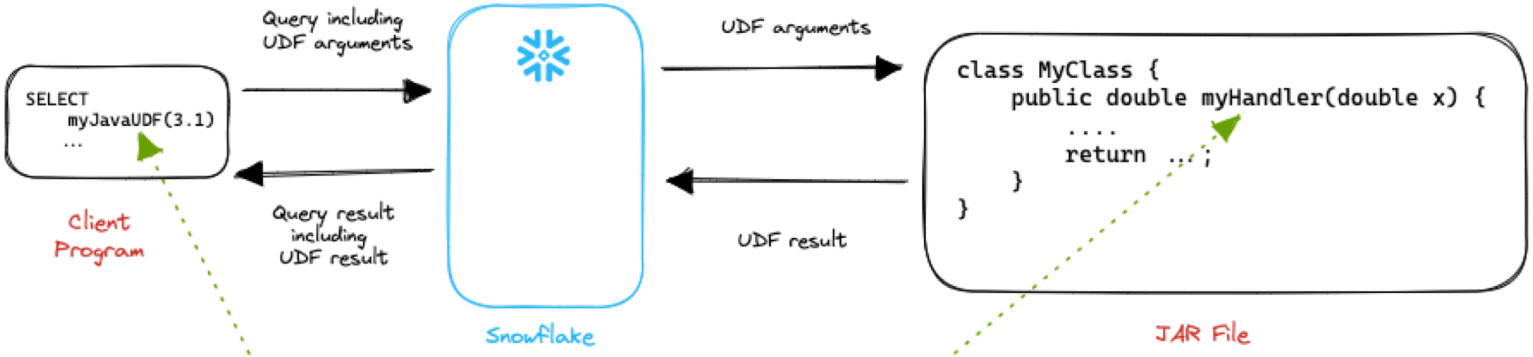
* in-line handler

How to choose the right language

- Own preference
- You already have some other code in that language – consistency
- Language has capabilities which other does not have
- Language has libraries which can help you with data processing
- Do you want to keep your udf code in-line or externally (file on stage)?

Language	Handler Location	Sharing
Java	In-line or staged	No
JavaScript	In-line	Yes
Python	In-line or staged	No
SQL	In-line	Yes

Associate Handler method with the UDF name




```
CREATE FUNCTION myJavaUDF(d DOUBLE)
RETURNS DOUBLE
LANGUAGE JAVA
IMPORTS = (@MyStage/MyJarFile.jar)
HANDLER = 'MyJarFile.MyClass.myHandler'
```

max. 500 input parameters

* Stage stored handler

Tabular UDFs

- Scalar = return single value for each input row
- Tabular = return tabular value for each input row
  - RETURN value specify schema of returned table, including data type
  - BODY of UDTF is SQL expression – it must be SELECT statement
  - Max 500 input parameters
  - Max 500 output columns
- INFORMATION_SCHEMA – full of tabular functions

```
SELECT ……
  FROM TABLE ( udtf_name (udtf_arguments) )
```

User defined table function example

```
Create or replace function orders_for_product(PROD_ID varchar)
returns table (Product_ID varchar, Quantity_Sold numberic(11,2))
as $$
  select product_ID, quantity_sold
  from orders
  where product_ID = PROD_ID
$$;
```

```
select product_id, quantity_sold
from table(orders_for_product('compostable bags'))
order by product_id;
```

```
+------------------- +-------------------------+
| PRODUCT_ID  | QUANTITY_SOLD  |
|------------------- +-------------------------|
| compostable bags | 2000.00                  |
+--------------------+-------------------------+
```

Python UDF/UDTF demo and exercise

Exercise

We are going to practice scalar and table UDFs in this example. Firstly we will try to extract a domain name from user emails in our snowpipe_landing table.

Then we will try to improve the solution and write table function which will extract 3 values from single email and return them as a table. We are going to extract the username, first and second domain. This time we will use Python as language for our UDF and UDTF.

O'REILLY®

External Tables

Tomas Sobotik, 9.6.2023

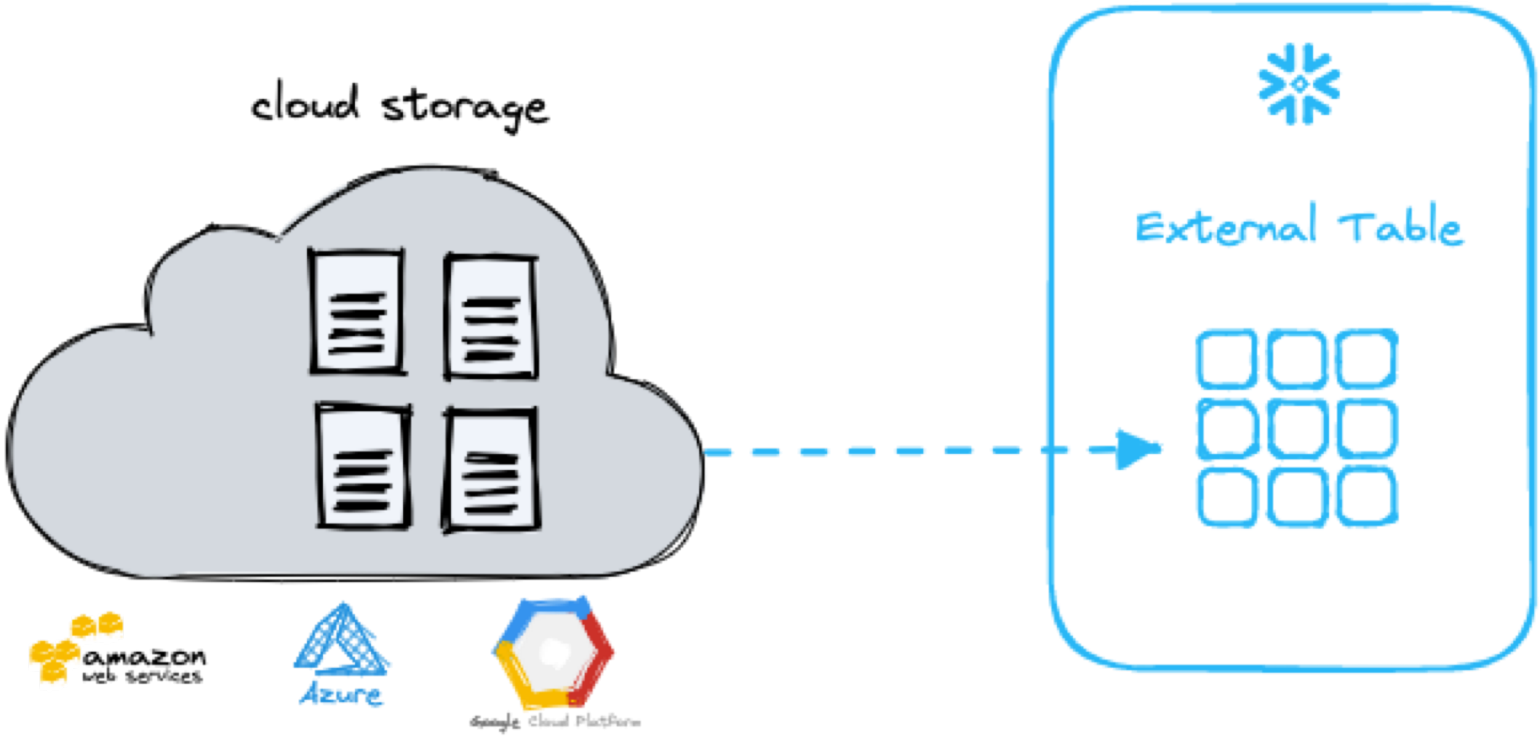
External tables

- Using Snowflake for querying external storage
- Data are stored in data lake in cloud storage
- Read only tables
- Supports any format which is supported by COPY command
- Delta lake support
- Use cases
  - Large data, randomly/not frequently used
  - Existing data Lake used by other services/projects
- Benefits
  - Cost save – save time, storage and processing power
  - Use Snowflake elastic and scalable computing power for external data

File size recommendation

256 – 512 MB for Parquet files
16 – 256 all other supported

External Table



* External table can be combined with internal tables, views

External tables partitioning

- Organize files in logical paths per various dimensions
  - Date, time, country, business unit, etc
- Why? Better performance
- Data are organized in separate slices -> query response time is faster
- Partitions are stored in the external table metadata
- Multiple partition columns are supported
- Partition parse information stored in `METADATA$FILENAME` pseudocolumn -> all files matching the path are part of the partition
- Manual or automatic refresh of the partitions
  - Defined at table creation and can’t be changed later!

Manual X Automatic partitions adding

Manual

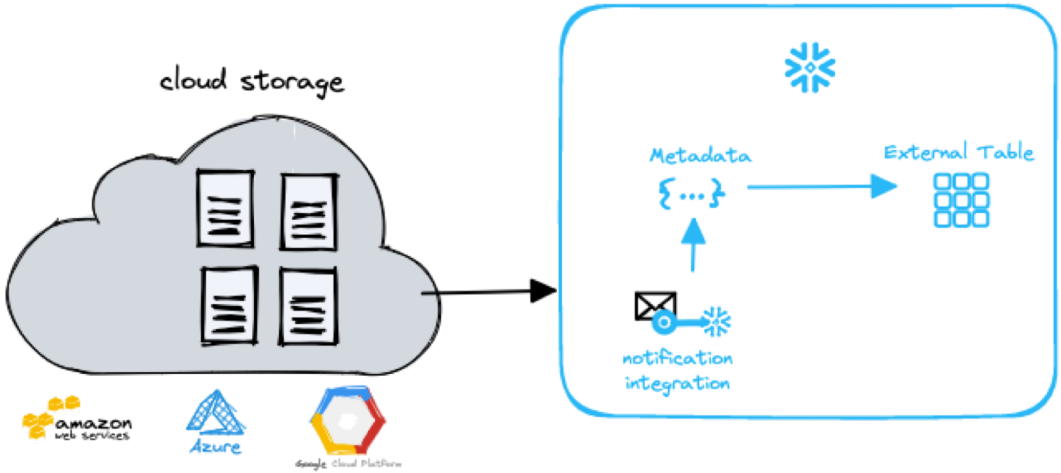
- Use it when you want to add/remove partitions selectively
- Often used when you want to sync external tables with other metastore (AWS Glue, Apache Hive)
- `PARTITION_TYPE = USER_SPECIFIED`
- Adding a new partitions with `ALTER EXTERNAL TABLE` command
- Can’t be automatically refreshed

Automatic

- Define only partitioning column
- During refresh partitions are added automatically based on partitioned column

External tables refresh

- External table metadata needs to be refreshed
- Manually -> `ALTER EXTERNAL TABLE MY_TABLE REFRESH`
  - You can automate it with task or stored procedure
- Automatic refresh based on event notification for cloud storage service
  - New files in the path are added to table metadata
  - Changes to files in the path are updated in the table metadata
  - Files no longer in the path are removed from the table metadata



Automatic refresh of external tables

- Similar approach like Snowpipes use for notification about new files (SNS or SQS)

1. Configure Access Permission for the S3 bucket (IAM Policy + IAM Role)
2. Configure Secure Access to Cloud Storage (Storage integration object)
3. Review IAM User for your Snowflake Account
    - `DESC INTEGRATION`
4. Grant the IAM User Permissions to Access Bucket Objects
    - Trust relationship
5. If needed create a Stage
6. Create an External Table
7. Configure Event Notifications (SQS or SNS)
8. Manually refresh External Table

Good to know in relation to External Tables

- Existence of external table blocks Database replication
- External table can be shared
- External table can’t be cloned
- We can use INFER_SCHEMA, USING TEMPLATE, and GENERATE_COLUMN_DESCRIPTION functions to make it easier

Demo & exercise external table creation

Exercise

We will practice creation of the external tables in multiple ways:

- Create and query the external table without knowing the file schema
- Create and query the external table when you know the file schema
- Create partitioned external table

Detailed instructions and source files are available on GitHub.

O'REILLY®

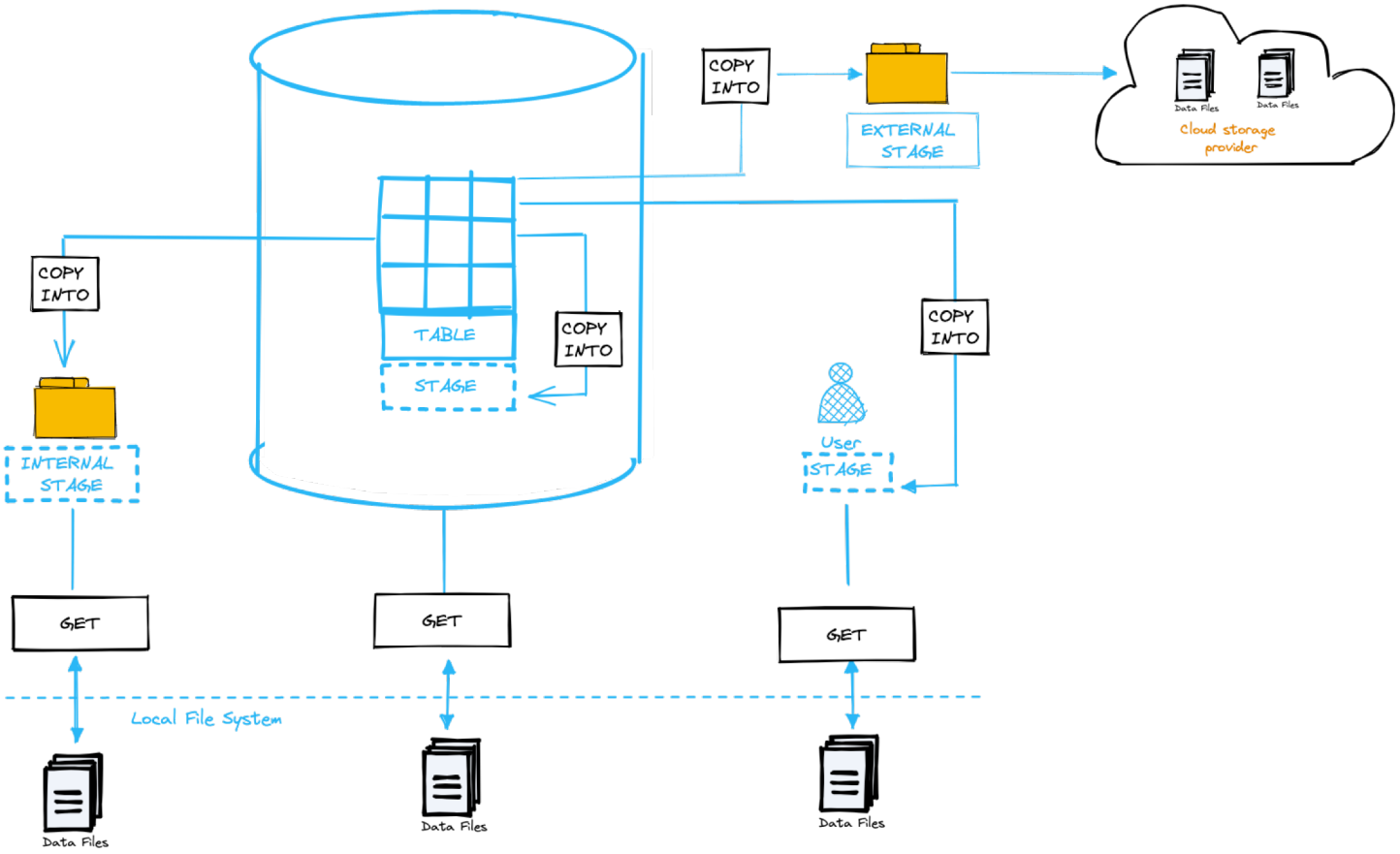
Data Unloading

Tomas Sobotik, 9.6.2023

Data Unloading

- Sharing Data with partners, co workers (different teams)
- Exporting data out
  - Internal stages
  - External stages
- Utilizing same objects like data import
  - Stages + File Format + storage integration
  - COPY command
- Copy command has different restriction than copy command for import
- Unload could be also done into compressed format
- Data unload without export -> sharing via various APIs

Unloading schema



Unload syntax

```
COPY INTO @my_stage
    FROM my_data
    FILE_FORMAT = ( FORMAT_NAME = 'my_format')
```

- Can use any SQL command including joins – no limitations
- COPY options related to unloading
  - OVERWRITE
  - SINGLE
  - INCLUDE_QUERY_ID
  - MAX_FILE_SIZE (5 GB, default 16 MB)
- Supports same formats like import (csv, Parquet, JSON, etc.)
- Handling nulls -> part of file format definitions

File paths and names

- Snowflake appends a suffix to the file name. It includes:
  - The number of virtual machine in the virtual warehouse
  - The unload thread
  - A sequential file number
  - Example: **data_0_0_0.csv**

- You can set the file path for the files. Add it after the stage name
  - `COPY INTO @mystage/Unloading/TableX`

- To set the file name for the files, add the name after the folder name
  - `COPY INTO @mystage/Unloading/TableX/exported_data`

Unloading JSON data

- Data needs to be shared via API

1. Unload data into external stage first
2. Share data via API with consumers

- Multiple ways how to automate whole process:
  - AWS Lambda to perform all the tasks
  - Stored Procedure + Snowflake External function to communicate with third party API
  - *Snowpark has limitation to hit the external APIs
- It must be done from VARIANT data type or you need to convert the data back to JSON
- Constructing the JSON structure -> `OBJECT_CONSTRUCT()` + `ARRAY_AGG()` functions

Constructing JSON

- `OBJECT_CONSTRUCT()`
  - Construct an object from arguments
  - Returns object
  - Arguments = key – value pairs
  - Nested calls are supported
  - It supports expressions and queries to add

```
SELECT OBJECT_CONSTRUCT('a',1,'b','BBBB')
```

```
{
    "a": 1,
    "b": "BBBB"
}
```

Constructing JSON

- `ARRAY_AGG()`
  - Input values are pivoted into an array
  - Returns array type value
  - Arguments – values to be put into the array and values determining the partition into which to group the values
  - Supports DISTINCT

```
SELECT ARRAY_AGG(O_ORDERKEY) WITHIN GROUP (ORDER BY O_ORDERKEY ASC) FROM orders
```

```
[
    3368,
    4790,
    4965,
    5421
]
```

Exercise

This exercise will be dedicated to practising unloading the data into internal and external stages.

Detailed instructions are available on github: **week2/week2_exercise_desc.pdf**