O'REILLY

Monitoring of Data Pipelines

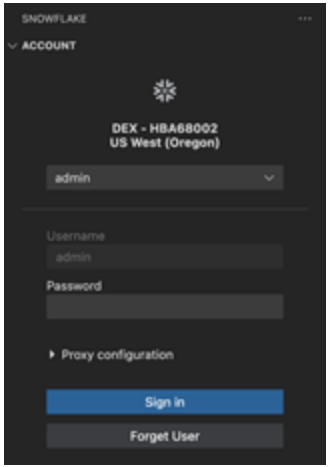
Tomas Sobotik

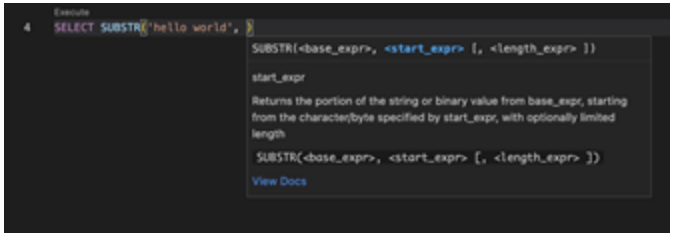
Challenges with Snowflake data pipelines

- Compared to traditional ELT/ETL tools (DBT, Azure Data Factory, Fivetran, etc.) Snowflake is lacking a default functionality in relation to:
  - 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

- Native GIT integration or VS CODE with extensions
- Keeping data pipeline code in GIT is almost necessary
  - Can save a lot of pains
- 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;
```

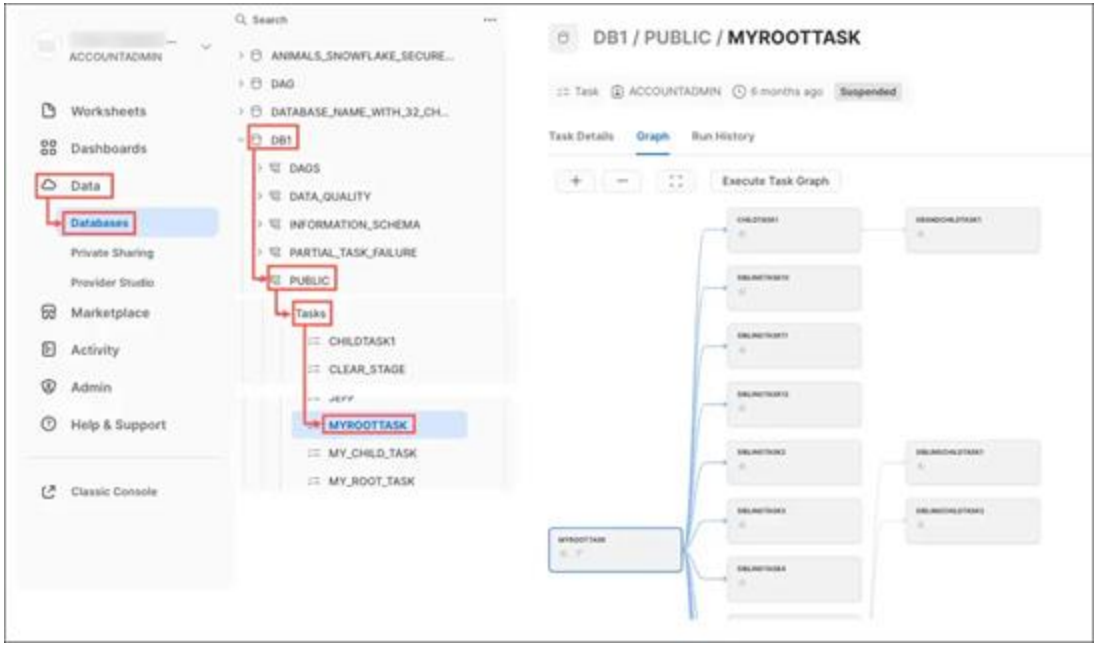
Stream stale & task suspension prevention

- 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.

Detailed description is available on github -> w1_exercise_desc.pdf

O'REILLY

Stored Procedures in Snowflake

Tomas Sobotik

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

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)
```

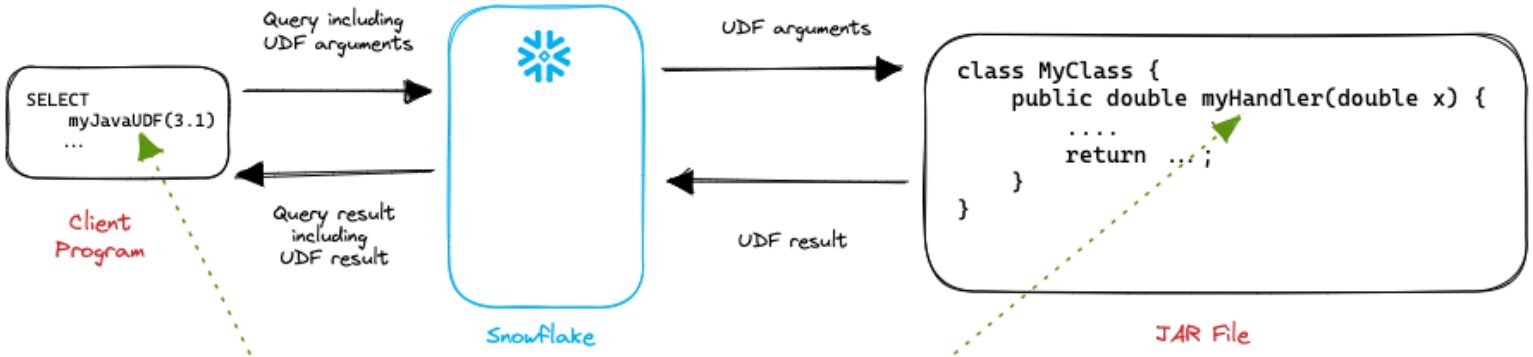
* in-line handler

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

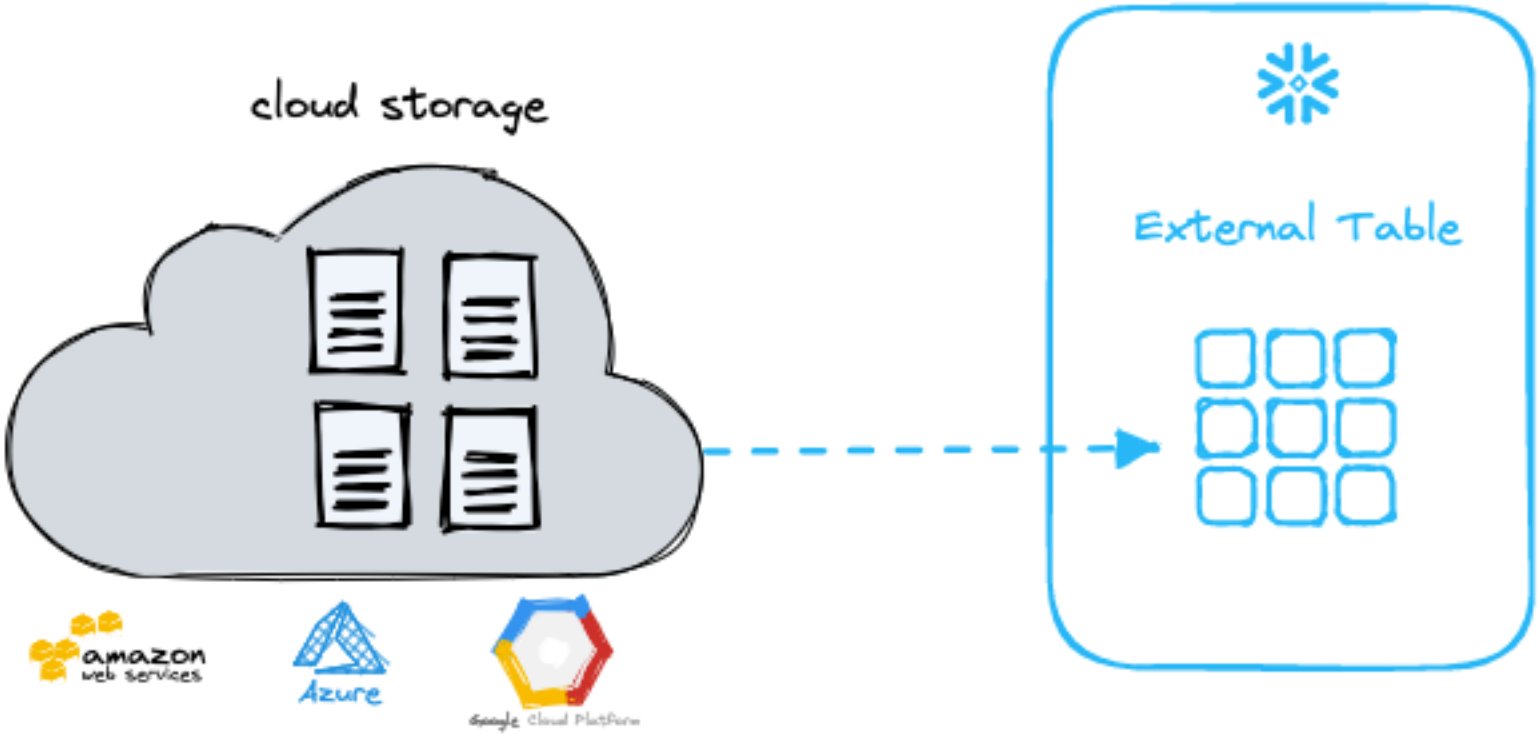
External tables

- 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

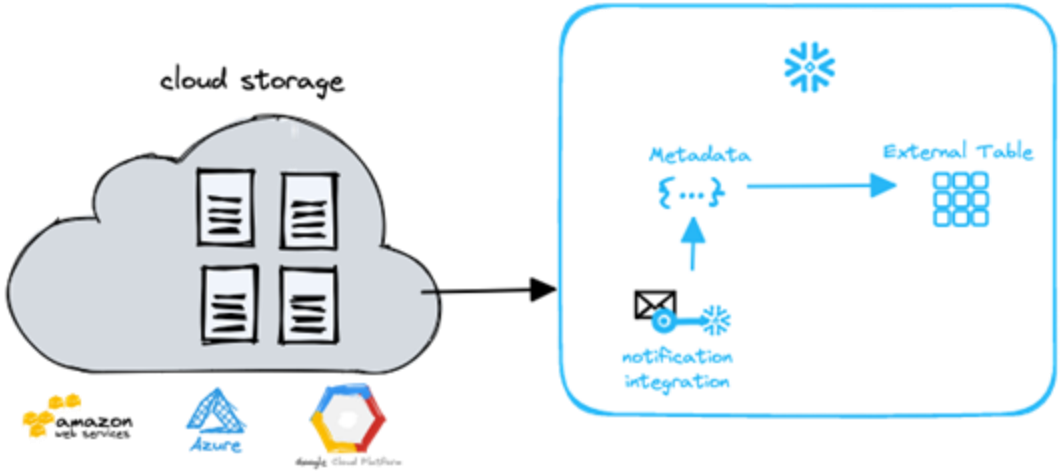
- Use it when you want to add/remove partitions selectively
- Often used when you want to sync external tables with other metastore (AWS Glue, Apache Hive)
- `PARTITION_TYPE = USER_SPECIFIED`
- Adding a new partitions with `ALTER EXTERNAL TABLE` command
- Can’t be automatically refreshed

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®

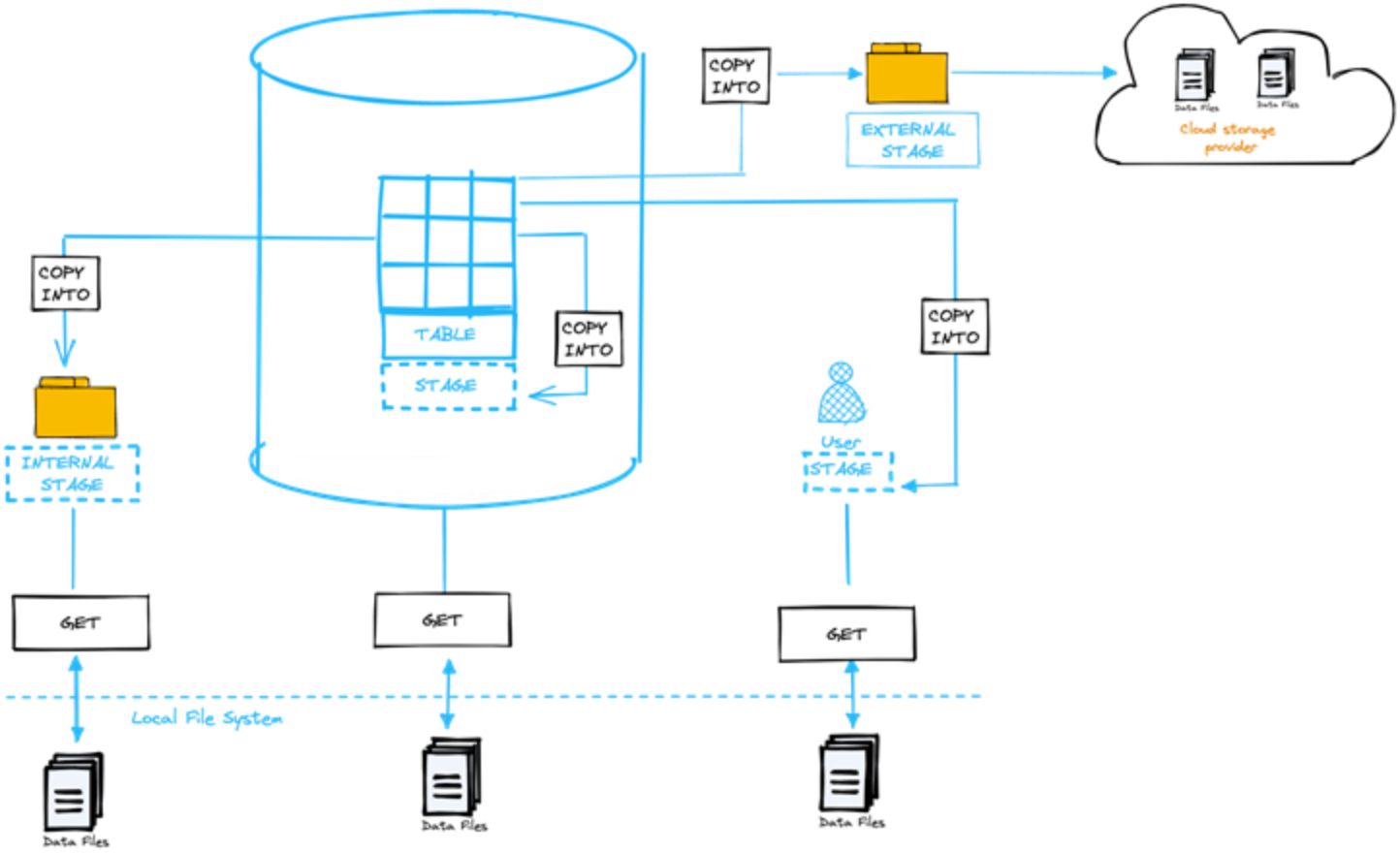
Data Unloading

Tomas Sobotik

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**

O'REILLY®

Intro into SQL API

Tomas Sobotik

SQL API

- A new REST interface for submitting SQL statements
- Supports all standard DDL, DML, and Queries

Why SQL API?

- Drivers cannot always be loaded/used
- Developer preference
- Ease of migration

What can you do with SQL API:

- Build custom REST based applications
- Easily migrate existing applications built for APIs
- Integrate with applications that provide a REST Interface (ServiceNow, SalesForce, etc.)
- Integrate with resource constrained environment

Billing?

It is part of cloud service layer

Capabilities and limitations

Can do

- Submit SQL statements for execution
- Check the status of the execution of a statement
- Cancel the execution of a statement
- Fetch query results concurrently
- Manage deployment (e.g. provision users, roles, create tables, etc.)
- Data are returned in partitions

Can’t do

Not supported:

- PUT command
- GET command
- CALL command returning a table
- Python stored procedures

Following statement works only within request that specifies multiple statements

- ALTER SESSION
- USE <object>
- BEGIN, COMMIT, ROLLBACK

Endpoints

http://<account_identifier>.snowflakecomputing.com/api

- `/api/v2/statements`
  - Submit SQL statements

- `/api/v2/statements/<statementHandle>`
  - Check the status of the execution of a statement
  - `<statementHandle>` = unique identifier for statement submitted for execution

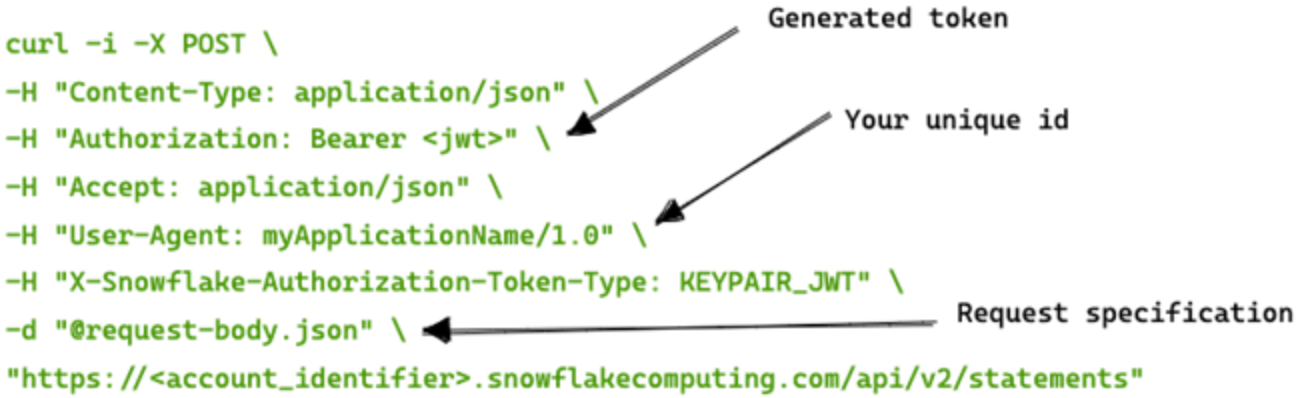
- `/api/v2/statements/<statementHandle>/cancel`
  - Cancel the execution of a statement

Authentication

- Using Oauth
- Using key-pair authentication
    1. Generate public-private key-pair
    2. Assign the public key to the Snowflake user
    3. Generate the fingerprint of the public key in the application code
    4. Generate JSON Web Token (JWT) with following fields in the payload
        - Iss – Issuer of the JWT
        - Sub – subject for the JWT
        - Iat – Issue time for the JWT in UTC
        - Exp – expiration time for the JWT in UTC
    5. Include generated JWT token into each API request

How to submit request

- Specify the request ID – it distinguishes this request from others
- Construct the body of the request with following fields
  - Statement – SQL statement you want to execute
  - In case of bindings variables use ? Placeholder in the statement and include `bindings` field
  - Use `warehouse, database, schema` and `role` fields for specifying corresponding values
  - Values are case-sensitive



Request body example

```
{
   "statement": "select * from T where c1=?",
   "timeout": 60,
   "database": "TESTDB",
   "schema": "TESTSCHEMA",
   "warehouse": "TESTWH",
   "role": "TESTROLE",
   "bindings": {
      "1": {
         "type": "FIXED",
         "value": "123"
      }
   }
}
```

Processing the responses

- When you submit a request and you receive a response with 202 which contains Query status object with multiple fields:


```
{
"code": "090001",
"sqlState": "00000",
"message": "successfully executed",
"statementHandle": "e4ce975e-f7ff-4b5e-b15e-bf25f59371ae",
"statementStatusUrl": "/api/v2/statements/e4ce975e-f7ff-4b5e-b15e-bf25f59371ae"
}
```


- To check if the statement has finished executing, you must send a request to check the status of the statement
- If the statement has finished successfully, Snowflake returns the HTTP reponse code 200 and rows from the results in a `ResultSet` Object

Response body


```
{
  ...
  ...
"resultSetMetaData" : {
   "numRows" : 50000,
   "format" : "jsonv2",
   "partitionInfo" : [ {
      "rowCount" : 12288,
      "uncompressedSize" : 124067,
      "compressedSize" : 29591
    }, {
      "rowCount" : 37712,
      "uncompressedSize" : 414841,
      "compressedSize" : 84469
    }],
},
"data": [
   ["customer1", "1234 A Avenue", "98765", "2021-01-20 12:34:56.03459878"],
   ["customer2", "987 B Street", "98765", "2020-05-31 01:15:43.765432134"],
   ["customer3", "8777 C Blvd", "98765", "2019-07-01 23:12:55.123467865"],
   ["customer4", "64646 D Circle", "98765", "2021-08-03 13:43:23.0"]
  ]
}
```

SQL API demo & exercise

Exercise

This exercise is dedicated to using the SQL API. We will go through authorization and sending the API request to Snowflake. For sending the API requests we are going to use Postman.

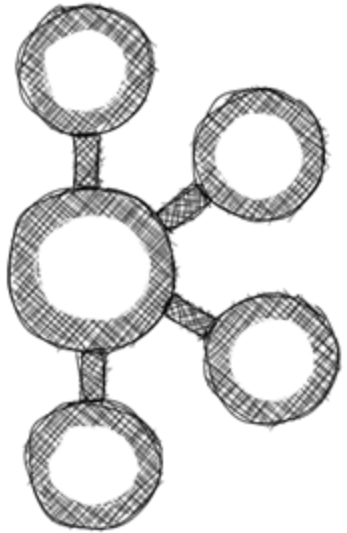
O'REILLY

Kafka Connector overview

Tomas Sobotik

What is Apache Kafka?

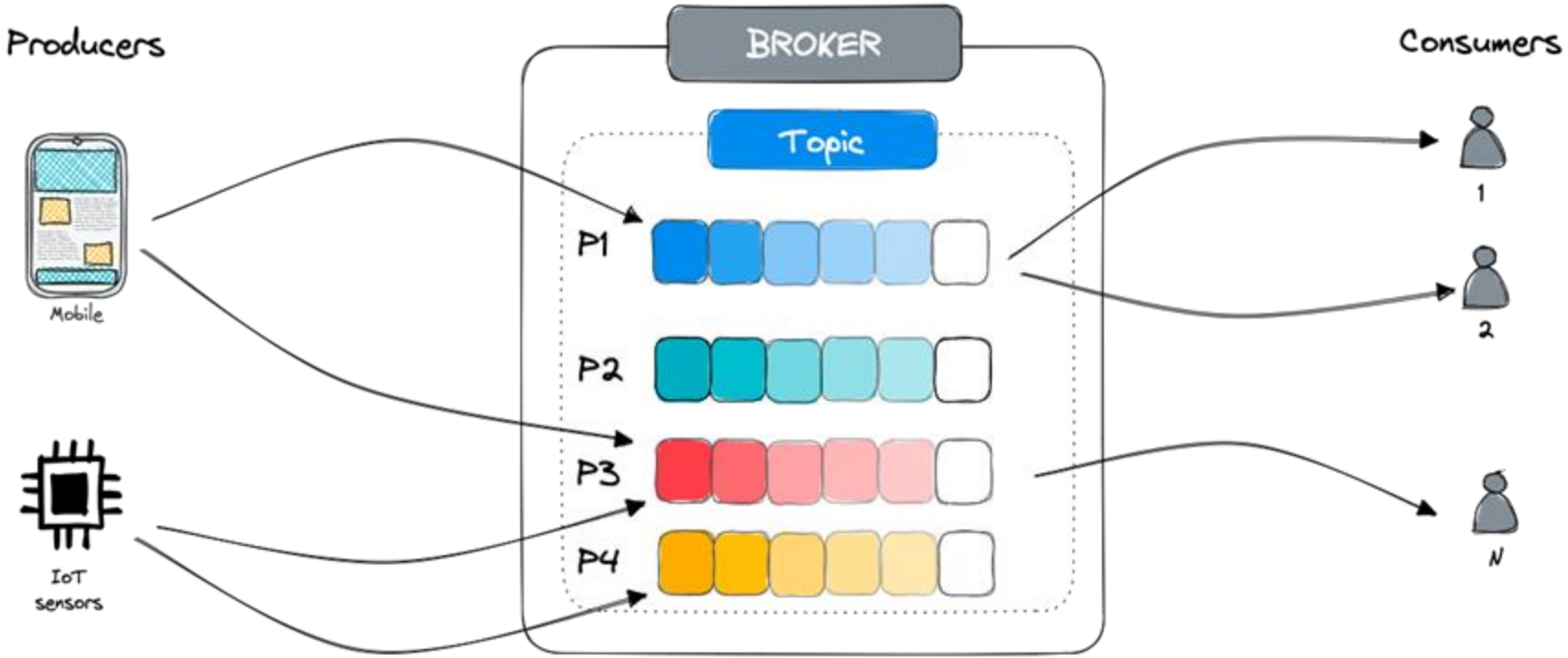
- Publish – subscribe messaging system
- Used for event streaming
- Asynchronous communication
- Messages are organized in topics (category)
- Publisher send messages/record to topics
- It can include any kind of information
- Message attributes
  - Key & value (mandatory)
  - Timestamp, headers (optional)
- Value could be whatever needs to be sent. Many times it is a JSON



Kafka components

- **Producer**
  - Client apps that publish (write) events to Kafka
- **Consumers**
  - They subscribe to topics to read and process the events
  - Producers & Consumers are fully decoupled – high scalability.
  - Producer never needs to wait for consumers
- **Broker**
  - One or more servers (clusters) running the Kafka
- **Topic**
  - Category/feed name to which records are stored and published
- **Topic partition**
  - Topics are divided into partitions
  - Each record in a partition is identified by its unique offset.
  - Allows parallelization by splitting the data into particular topic partition across multiple brokers

High level Kafka architecture



Snowflake & Kafka

- Two versions of the connector
  - Confluent Kafka
  - Open source Apache Kafka package
  - Connector needs to be installed/configured on Kafka server

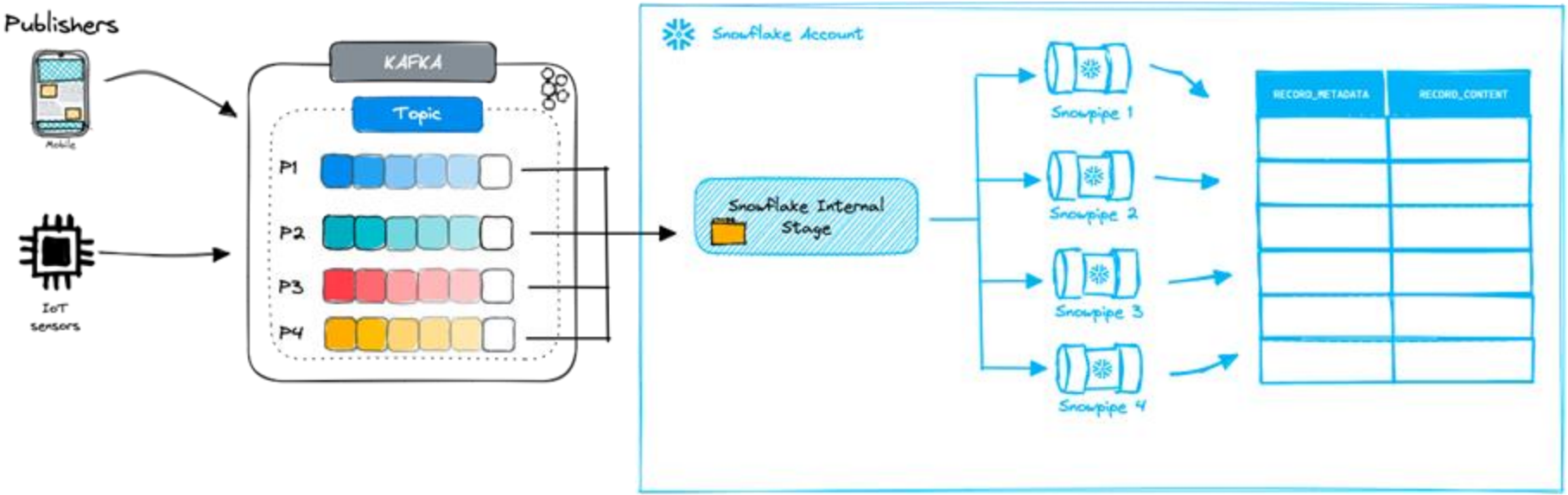
- Topic produce stream of rows which should be inserted into Snowflake table
- Each Kafka message = one row
- One Topic supplies messages (rows) to one table
- Topic could be mapped to existing table in Kafka configuration
- If not mapped connector creates a new table for each topic
  - Topic name – snowflake table name (UPPER CASE)

Kafka topic table schema

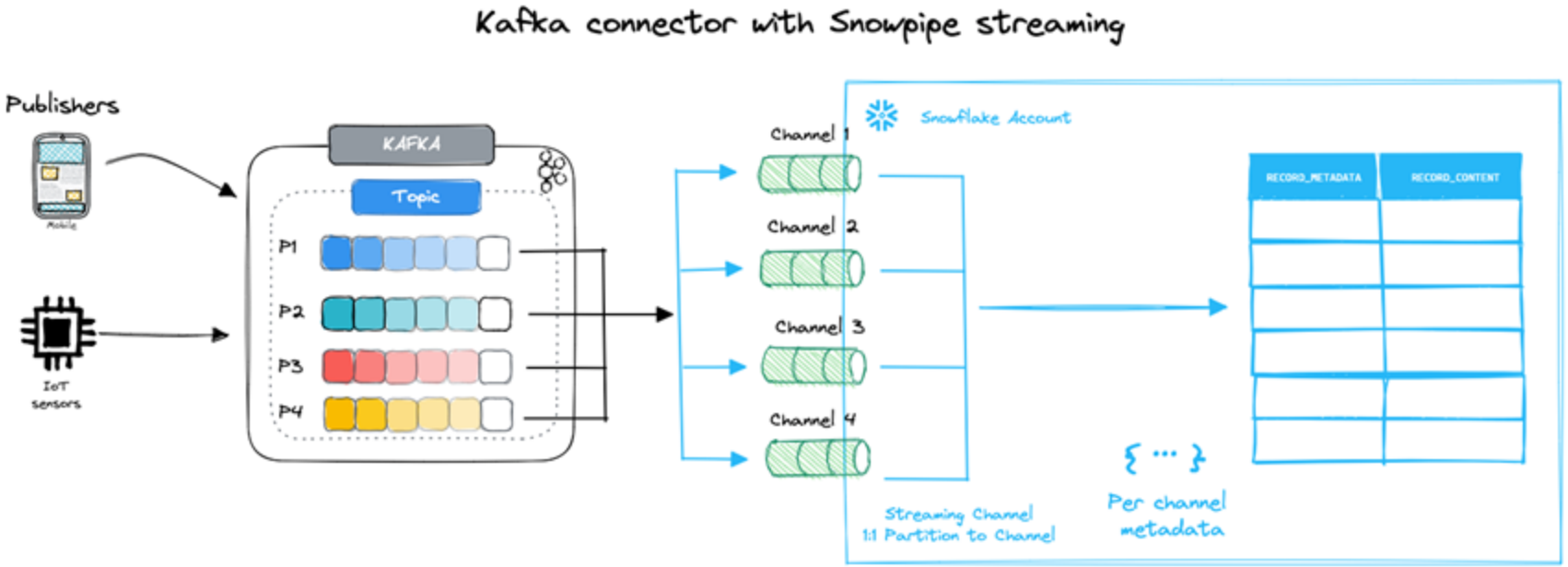
- Two mandatory columns
- VARIANT columns holding the data in JSON structure
- **RECORD_CONTENT** – this contains the Kafka message
- **RECORD_METADATA** – metadata related to the messageDefault metadata fields:
    - topic
    - partition
    - offset
    - CreateTime
    - key
    - schema_id
    - headers

- Amount of provided metadata is configurable using configuration properties

Kafka connector workflow



Kafka connector & Snowpipe streaming



Installing the Kafka connector

- Needs to be done by Kafka cluster admins
- It differs per environment
- From Snowflake perspective
  - Create a user with key-pair authentication
  - Provide a configuration file

```
connector.class=com.snowflake.kafka.connector.SnowflakeSinkConnector
tasks.max=8
topics=topic1,topic2
snowflake.topic2table.map= topic1:table1,topic2:table2
buffer.count.records=10000
buffer.flush.time=60
buffer.size.bytes=5000000
snowflake.url.name=myorganization-myaccount.snowflakecomputing.com:443
snowflake.user.name=jane.smith
snowflake.private.key=xyz123
snowflake.private.key.passphrase=jkladu098jfd089adsq4r
snowflake.database.name=mydb
snowflake.schema.name=myschema
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=com.snowflake.kafka.connector.records.SnowflakeAvroConverter
value.converter.schema.registry.url=http://localhost:8081
value.converter.basic.auth.credentials.source=USER_INFO
value.converter.basic.auth.user.info=jane.smith:MyStrongPassword
```

Poll

1. Kafka Connector creates the landing table
   a) TRUE
   b) FALSE

2. What are two columns Kafka connector populates
   a) MESSAGE_BODY, MESSAGE_METADATA
   b) CONTENT_BODY, CONTENT_METADATA
   c) RECORD_CONTENT, RECORD_METADATA

3.) Which data type does hold the Kafka message
   a) VARCHAR
   b) VARIANT
   c) TEXT

O'REILLY®

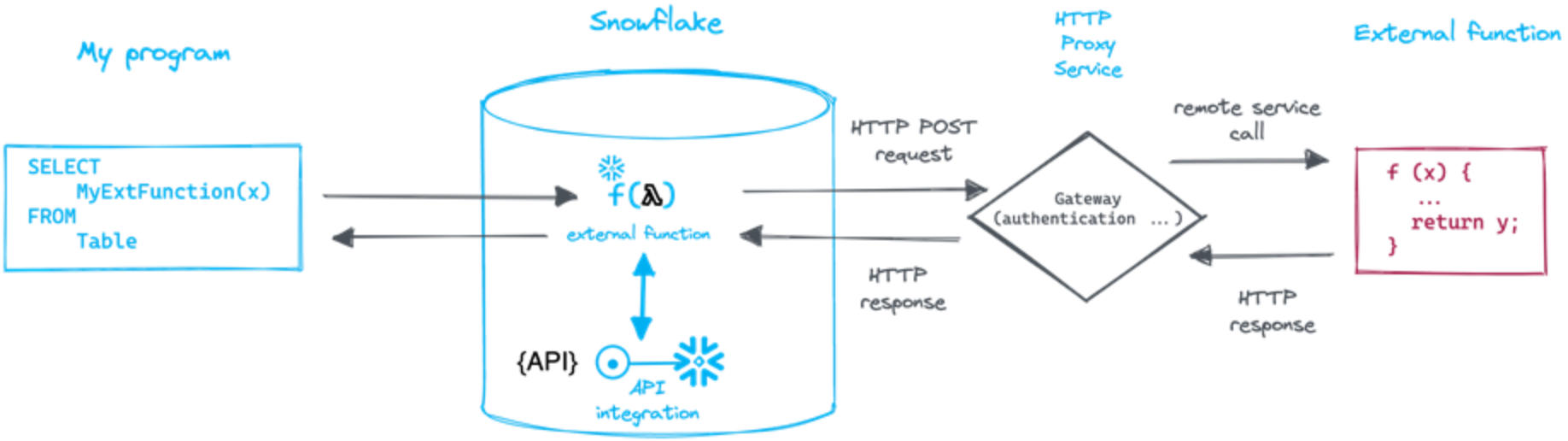
External Functions

Tomas Sobotik

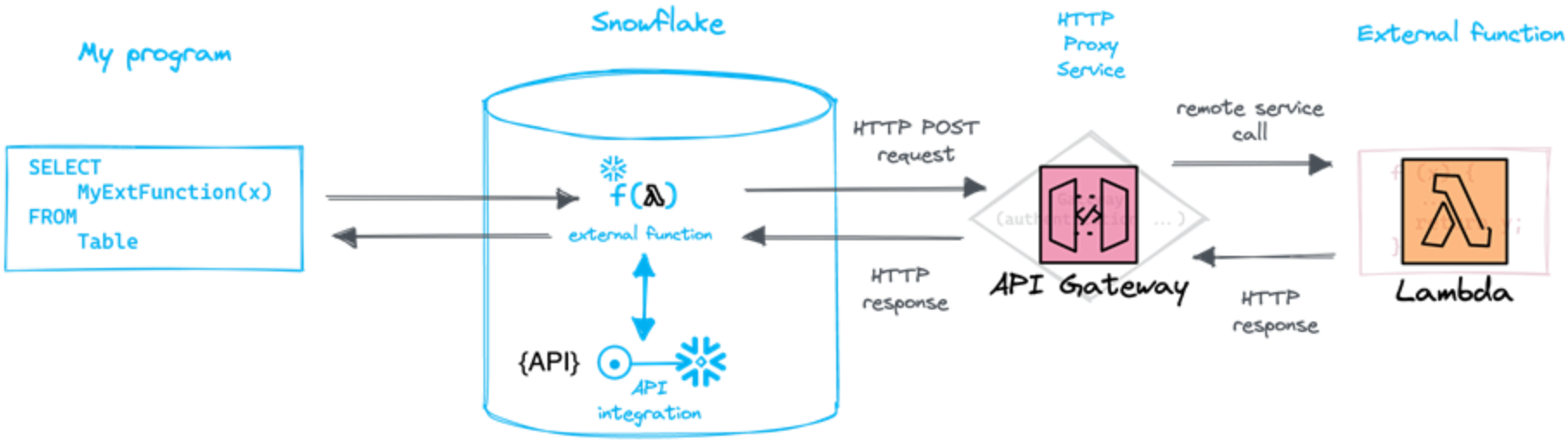
External Functions

- Call executable code which has been developed and runs outside of Snowflake
- Call external APIs, process data in Snowflake
- No need to export and reimport data
- Simplify your data pipelines
- Communication with external service is done by API integration
- Requires configuration on external cloud platform
- Use cases
  - ML models training
  - Translate service
  - Integration with other tools (Jira, etc.)
  - Custom Lambda
  - Geocoding

External Function - High-Level Overview



External Function - High-Level Overview



Benefits and limitations

Benefits

- Code could be in any language
- Remote service can use functions and libraries that can’t be accessed by UDF
- Remote service could be called from Snowflake and other software

Limitations

- Cloud platform security knowledge required
- Complex configuration on cloud platform side
- Only scalar functions are supported
- Can’t be shared via Data Sharing
- More overhead than UDF
- Max response size is 10 MB

External function creation workflow

- Cloud platform setup
  - Create a remote service (Lambda)
  - Create a proxy Service (API Gateway)
  - Create IAM Role
  - Setup the trust relationship between cloud platform and Snowflake
- Snowflake setup
  - Create the API integration
  - Create external functions
- Message format
  - JSON
  - Lambda accepts JSON and returns a JSON

Message body example

- One item in the object with a key ”data”
- “Data” item’s value is a JSON array
  - Each element is one row of data
  - Each row of data is a JSON array of one or more columns
  - First column is row number
  - Remaining columns contain the arguments for the function

```
{
    "data": [
                [0, ”Warsaw"],
                [1, ”Toronto"]
            ]
}
```

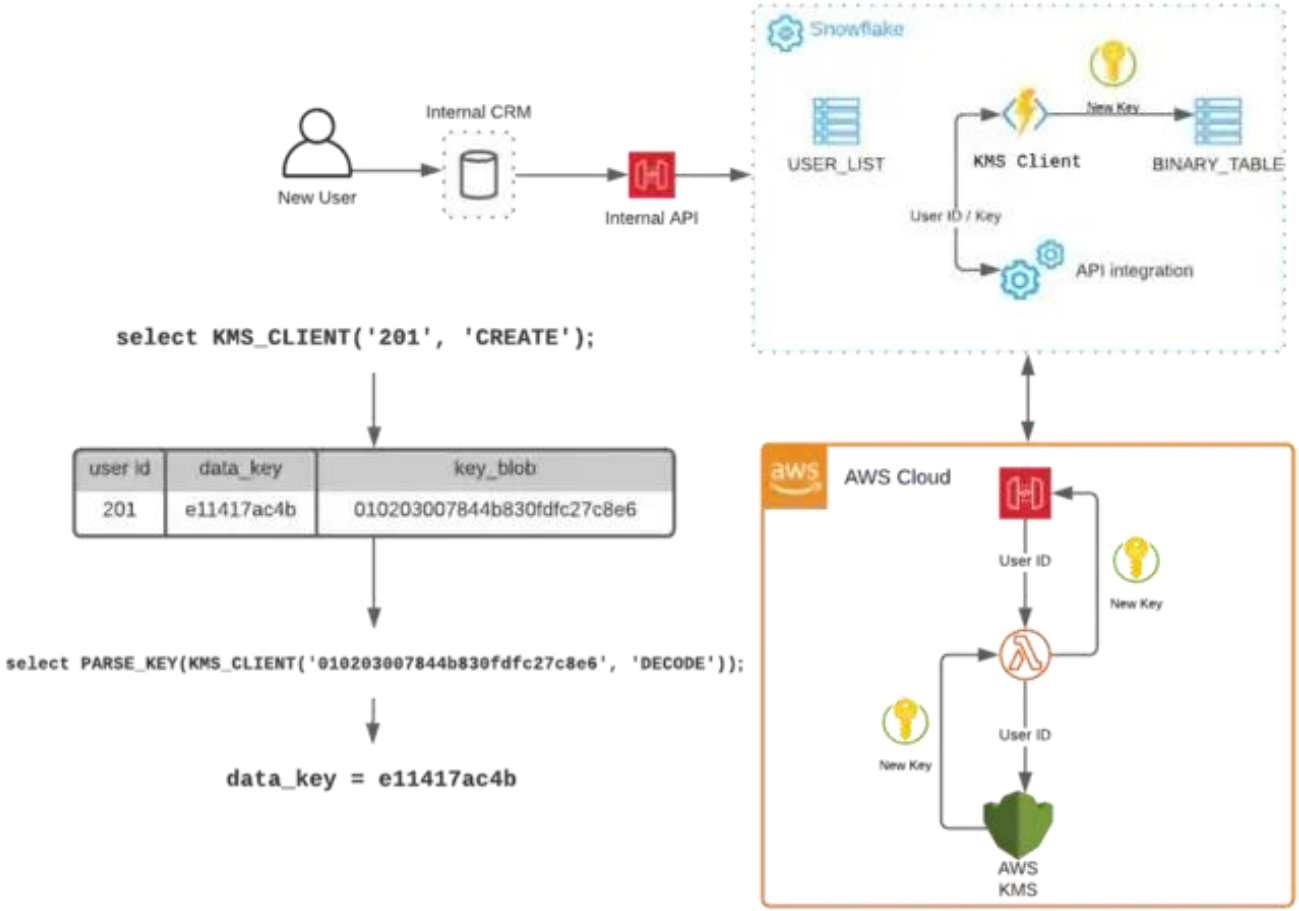
Message response

- Similar JSON format as the request
- One row for each row sent by Snowflake
- Each row has two values
  - The row number – correspond to row numbers in the request body
  - Value returned from the function for that row – it could be compound value (an OBJECT), but it must be exactly one value -> it is scalar function!

```
{
    "data":
    [
        [ 0, { "City" : "Warsaw",  "latitude" : 52.23, "longitude" :  21.01 } ],
        [ 1, { "City" : "Toronto", "latitude" : 43.65, "longitude" : -79.38 } ]
    ]
}
```

```
select val:city, val:latitude, val:longitude
    from (select ext_func_city_lat_long(city_name) as val
            from table_of_city_names);
```

External Function - custom data encryption



Exercise

We are going to focus on creation the external function in this exercise. We will try to leverage Amazon Translate API provided by Boto AWS SDK and translate and return German translation for input strings in English.

Detailed instructions are on GitHub

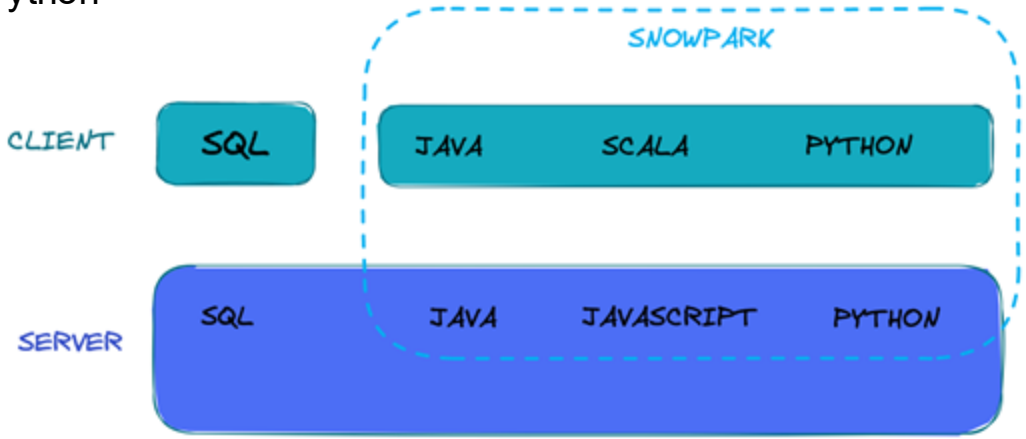
O'REILLY

Snowpark Data Transformations

Tomas Sobotik

What is Snowpark

- New development framework for Snowflake
- Developers can code in their familiar way with language of their choice
- Create ML model, data pipeline, data apps in single platform
  - Faster
  - More secure
- Data frame style programming
- Code runs in Snowflake using the elastic platform
- Supports Scala, Java & Python



Why Snowpark

Effective Architecture

single platform

easier collaboration

support for different languages

Build scalable solutions

unique Data Cloud platform

great price/performance ratio

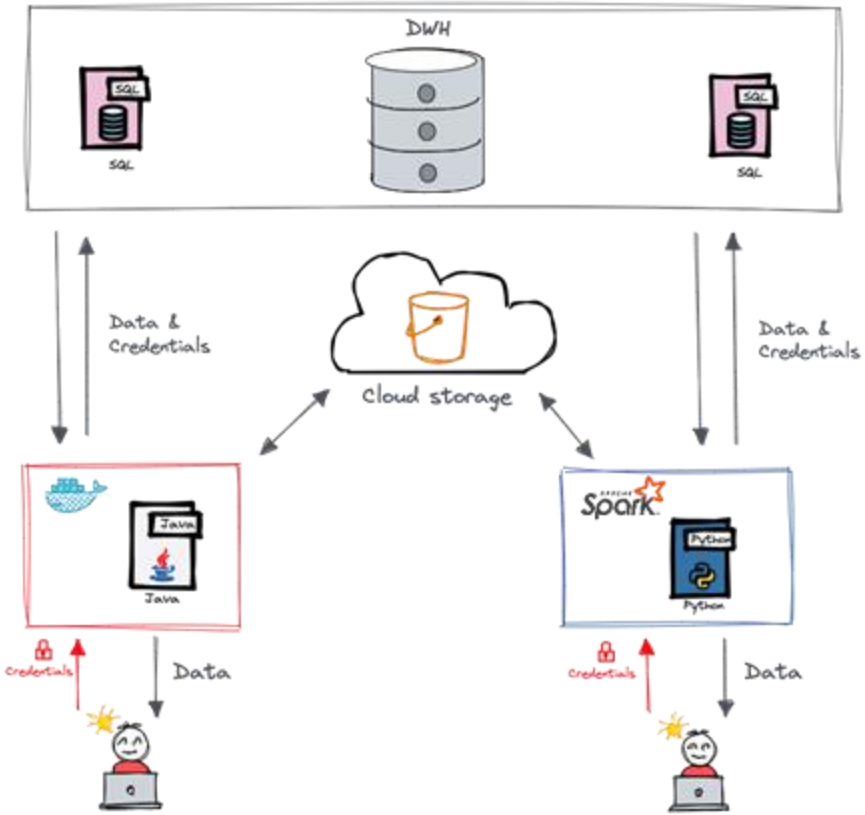
almost zero maintenance

Confidence

build-in governance

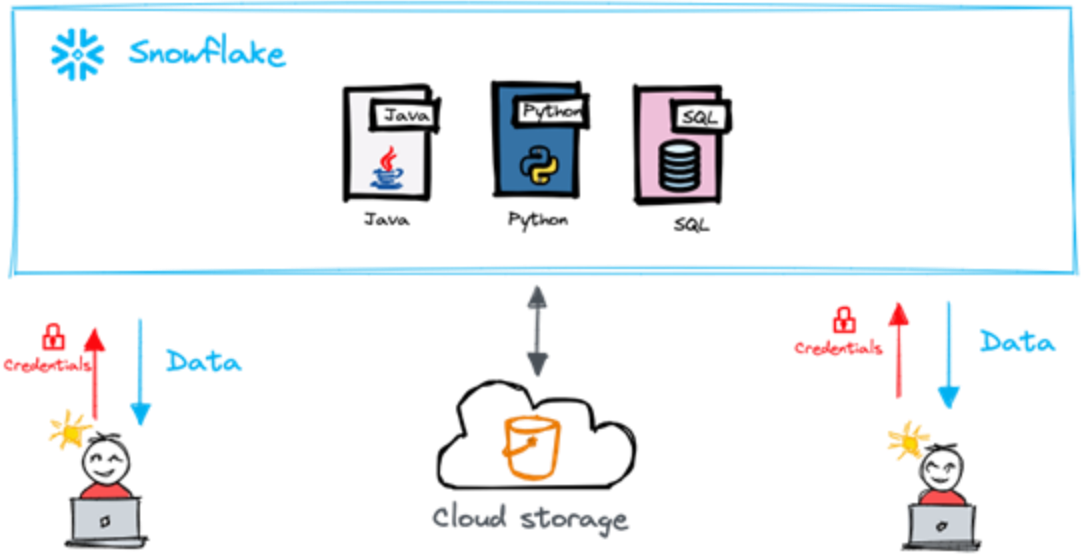
same security across all workloads

Traditional approach



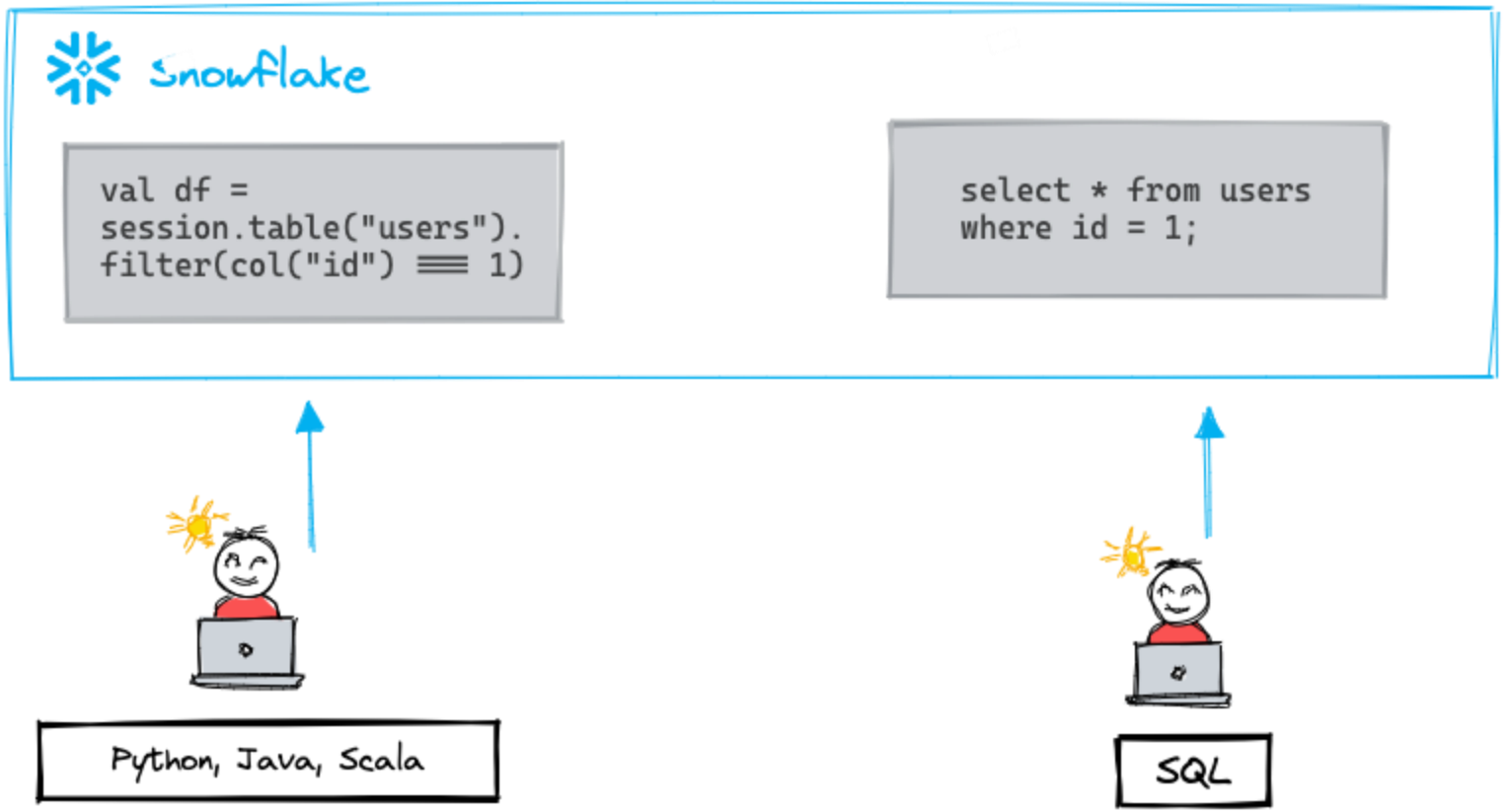
- Separate processing clusters per language
- Data Silos
- Data Movement between various systems and components
- Complex governance and security
- Infrastructure management

Snowpark way of working



- One platform with native support for Java, Python, Scala, SQL ..
- No Data Silos
- Easier data collaboration
- Simplified governance and security
- Utilizing Snowflake infrastructure - easier resizing

Snowpark x SQL



How to start

1. Prepare your own, local development environment
   - Snowpark requires Python 3.8 - 3.11 – you can use Anaconda, Miniconda or virtualenv
   - Python virtual environment
   - Jupyter Notebooks
   - Prepare your local IDE
   - Establish a session to interact with the Snowflake Database
2. Write Snowpark Code in Python Worksheets
   - Anaconda packages are available out of the box
   - Custom packages could be imported from stages to be used in scripts
   - No support for breakpoint or running only portions of the Python code
   - No support for images or webpages
   - It uses Python 3.8
   - No need to connect to Snowflake – you just setup the worksheet context

Local environment basic setup

- Main classes for the Snowpark API are in the `snowflake.snowpark` module
- Create a session and connect to Snowflake

```
>>> connection_parameters = {
...     "account": "<your snowflake account>",
...     "user": "<your snowflake user>",
...     "password": "<your snowflake password>",
...     "role": "<your snowflake role>",  # optional
...     "warehouse": "<your snowflake warehouse>",  # optional
...     "database": "<your snowflake database>",  # optional
...     "schema": "<your snowflake schema>",  # optional
... }

>>> new_session = Session.builder.configs(connection_parameters).create()
```

- Do not forget to close the session when you are done

```
new_session.close()
```

Basics operations

- Run SQL command: `session.sql(SELECT * FROM Table).collect()`
- Create a DataFrame: `df_table = session.table("sample_product_data")`
- Create a dataframe with SQL result: `df_sql = session.sql("SELECT name from sample_product_data")`
  - Print out first 10 rows: `df_table.show()`
- Data frames operations
  - Filters - equivalent of SQL WHERE: `df.filtered = df.filter(col("id") == 20)`
  - Joins: `df_joined = df_lhs.join(df_rhs, df_lhs.col("id") == df_rhs.col("parent_id"))`
  - …
- Chaining method calls
  - Query the `sample_product_data`
  - Return the row with `id = 1`
  - Select the `name` and `serial_number` columns
  - `df_product_info = session.table("sample_product_data").filter(col("id") == 1).select(col("name"), col("serial_number"))`

Python worksheets demo & exercise

Exercise

We are going to practise basic data frame transformations related to Snowpark for Python. We will test out the new Python worksheets which are in public preview to find out its limitations and then we will continue with local Python environment and doing the development in Jupyter notebook.

Please find detailed instructions on GitHub

O'REILLY®

Deploying UDFs

Tomas Sobotik

Intro

- Snowpark API allows to create UDFs or Stored Procedures
- When you call UDF, Snowpark executes your function on the server
  - Data does not need to be transferred to the client in order to process data

- Multiple ways how to create UDFs with Python handler
  - Inline Python code
  - Python worksheets
  - Locally and then stage the handler in internal stage (Staged handler)

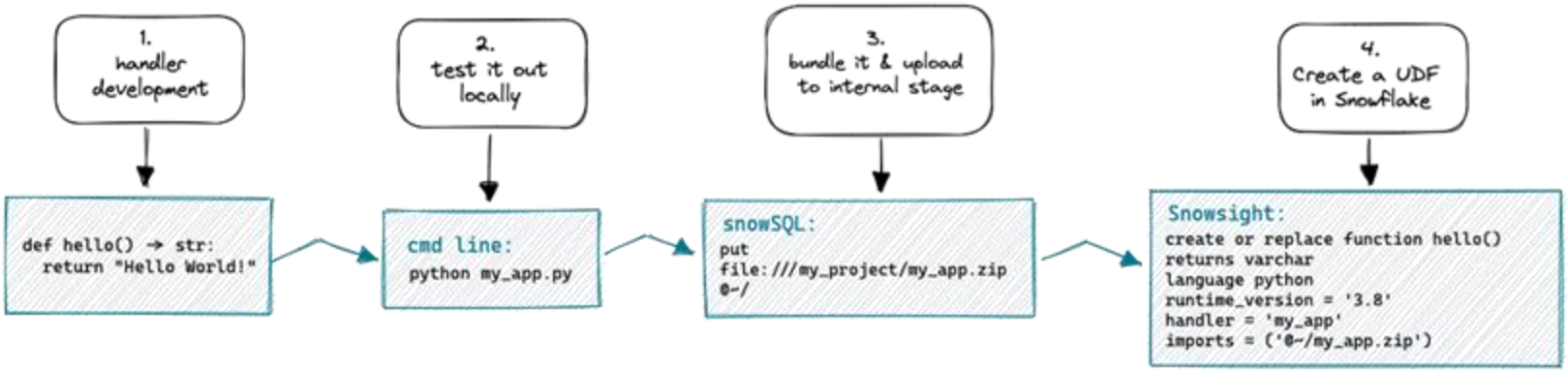
- Why could it be better to develop the code locally and stage it?
  - code might be too large for in-line handler
  - handler can be reused by multiple UDFs/SPs
  - easier to debug / test in existing external tools – especially for complex and large code

Other benefits of local development of Python UDFs

- Use your favourite IDE
- IDE based development might be more user friendly for users familiar with Python who may not be familiar with Snowflake UI
- More freedom
  - You can leverage GIT and other tools to deploy Python functions
  - Same functions might be used somewhere else in your technology stack
- Leverage Python functions from within wider Python scripts outside Snowflake but still pushing the compute down to Snowflake

Local UDF development and the workflow

1. Create a Python handler – UDF logic
2. Test it out locally
3. Bundle it – create a zip file and upload it into Snowflake internal stage
4. Create a Snowflake UDF which refers the handler code in stage and imports all needed packages
5. Call the UDF



Local UDF development and the workflow

1. Create a Python handler – UDF logic
2. Test it out locally
3. Bundle it – create a zip file and upload it into Snowflake internal stage
4. Create a Snowflake UDF which refers the handler code in stage and imports all needed packages
5. Call the UDF

How to simplify it?

SnowCLI

Snowflake Developer CLI

- Open source and community supported tool – not an official offering
  - Some patterns will be incorporated into Snowflake CLI (SnowSQL) in the future

- It allows you locally run and debug Snowflake apps
- Support for Snowpark Python user defined functions and stored procedures, warehouses, and Stremlit apps
- Define packages using `requirements.txt`, with dependencies automatically added via integration with Anaconda at deploy time.
- Deployment artifacts are automatically managed and uploaded to Snowflake stages

- Limitations
  - You must have the SnowSQL configuration file to authenticate to SnowCLI
  - To run Streamlit you must have access to Streamlit private preview

Installation and basic usage – UDF creation

- Pip or homebrew

1. Use empty directory to create your function – I would also recommend the python virtual env
2. Run snow function init
    - SnowCLI populates the directory with the files for a basic function. You can open app.py to see the files.
3. Test the code by running the `app.py` script -> `python app.py`
4. Package the function -> `snow function package`
    - It creates `app.zip` file that has your files in it
5. Login to Snowflake -> `snow login`
6. Configure the environment -> `snow configure`
7. Create a function -> `snow function create`
8. Run the function -> `snow function execute -f "helloFunction()"`

SnowCLI demo & exercise

Exercise

We are going to practise different methods of deploying Python UDFs. Firstly we will try to deploy the UDF via the Snowpark session. As a second exercise we are going to try the community tool called SnowCLI which can help with deploying UDFs into the stage, same like their creation in Snowflake.

Detailed instructions are available on Github again.

Wrap up

- We have covered almost all data engineering features of Snowflake
- Great knowledge foundation for Snowpro Core Certification or Data Engineering specialization
- What we have learnt
  - Data loading strategies
  - Data ingestion workflow and semi-structured data ingestion
  - Continuous data pipelines
  - Building and monitoring data pipelines
  - Using SQL API and external functions
  - Using Snowpark for data transformation

Wrap up

- We have covered all data engineering features of Snowflake
- Great knowledge foundation for Snowpro Core Certification or Data Engineering specialization
- What we have learnt
  - Data loading strategies
  - Data ingestion workflow and semi-structured data ingestion
  - Continuous data pipelines
  - Building and monitoring data pipelines
  - Using SQL API and external functions
  - Using Snowpark for data transformation

Good Luck!

Wrap up

- We have covered all data engineering features of Snowflake
- Great knowledge foundation for Snowpro Core Certification or Data Engineering specialization
- What we have learnt
  - Data loading strategies
  - Data ingestion workflow and semi-structured data ingestion
  - Continuous data pipelines
  - Building and monitoring data pipelines
  - Using SQL API and external functions

Good Luck!

Please, share your feedback.

Thank you!

Check out my other Snowflake courses!

https://www.linkedin.com/in/tomas-sobotik/