C4C to Snowflake Data Integration Solution

Introduction

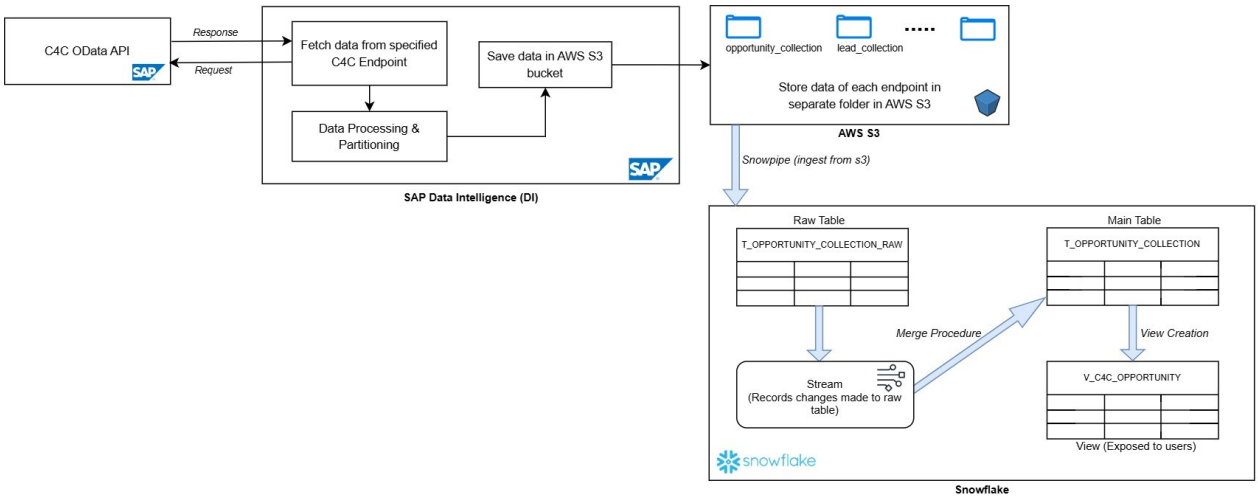
In this project, we are implementing a data integration solution that transfers data from the SAP C4C system into Snowflake, leveraging the C4C OData API for data extraction. The process begins with fetching data from the SAP C4C OData API, which is then staged in Amazon S3 in the form of Parquet files using SAP Data Intelligence.

On the Snowflake side, we have established a raw table designed to append all incoming data from the S3 bucket. This raw table serves as an intermediate storage layer, allowing us to accumulate data in snowflake before further processing. Subsequently, we perform a data merge operation where the raw data is integrated into the main table. This merging process is governed by specific merge conditions (mostly using the primary key of respective API endpoint), ensuring that data integrity and consistency are maintained throughout the integration process.

Following document is primarily divided into 2 sections.

- **SAP DI side development**: We use SAP DI to bring data from the C4C API into AWS S3 which is the staging area for data ingestion into Snowflake.
- **Snowflake side development:** Data from S3 is ingested into Snowflake. Detailed explanation is provided in further sections.

To get a high-level overview of the data integration process please refer to Figure 1 given below.



C4C - Snowflake Overall Flow

Figure 1: *C4C-Snowflake Overall Flow*

1. SAP Data Intelligence Side Development

This section discusses the method in which data from C4C OData API is fetched into S3.

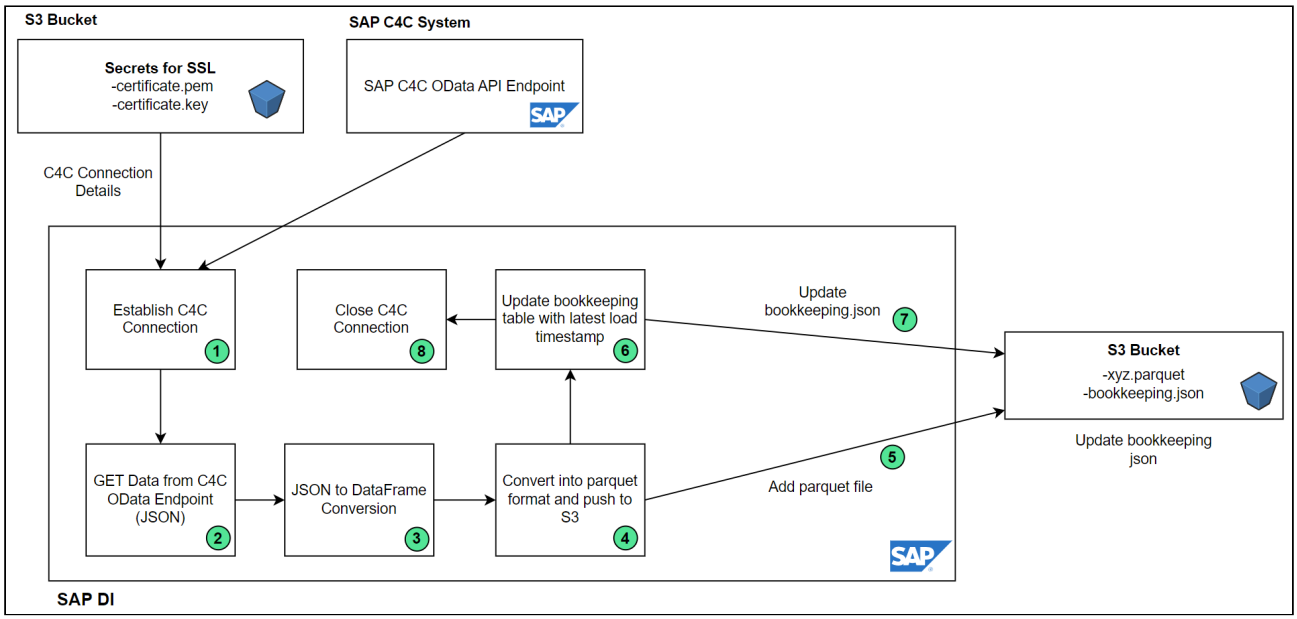


Figure 1: SAP DI *flow*

1.1 C4C OData API Authorization - Certificate Conversion

This part outlines the steps required to generate the necessary certificate files for authenticating with the C4C OData API. You will need a certificate file in either `.p12` or `.pfx` format, which will be used to create the required `.key` and `.pem` files. This below is just a one-time setup and needs to be performed again whenever the existing certificate expires.

Prerequisites

- **OpenSSL**: Ensure that OpenSSL is installed on your local machine. You can download it from OpenSSL's official website[1].
- **Certificate File**: Obtain the `.p12` or `.pfx` certificate file from your C4C OData Administrator.
- **Password**: You will need the passphrase for the certificate, which will be provided by the C4C OData Administrator.

Steps to generate Certificate Files

Follow these steps to convert your certificate file and generate the required authentication files:

1 https://www.openssl.org/

1. Convert the Certificate File to PEM Format

```
openssl pkcs12 -in certificate.p12 -out certificate_dev.pem -nodes
```

2. Extract the Private Key

```
openssl pkcs12 -in certificate.p12 -nocerts -out certificate-dev-key.key
```

3. Convert the Private Key to Standard Format (Decryption of private key)

```
openssl rsa -in certificate-dev-key.key -out certificate_dev.key
```

Resulting Files

After completing the above steps, you should have the following files:

- `certificate_dev.pem` : The PEM file for authentication.
- `certificate_dev.key` : The private key file for authentication

Storage and Access

These files will be stored in a dedicated Amazon S3 bucket for secure access. Below is the directory structure for the S3 bucket:

cloud-for-customer-sales-config/

- **dev/**
  - certificate_dev.key
  - certificate_dev.pem
- **acc/**
  - certificate_acc.key
  - certificate_acc.pem
- **prod/**
  - certificate_prod.key
  - certificate_prod.pem

Fetching Certificates

The certificates will be fetched by SAP DI using Python's Boto3 library for C4C authorization.

1.2 Understanding the directory structure in S3

S3 is the stage location for the data ingestion into snowflake. Data from the API lands up as parquet file in S3 initially.

eda-snowflake-prod/

```
├── cloud_for_customer_sales/
    ├── bookkeeping/
            ├──opportunity_collection.json
            └── employee_collection.json
    ├── data/
            ├──opportunity_collection/
                ├──1__2024-09-02T08-29-24Z__5000.parquet
            └── employee_collection/
                ├──1__2024-09-02T09-30-00Z__10000.parquet
```

This is how the directory structure looks like in S3. We mainly have 2 folders i.e. bookkeeping and data.

Before proceeding, in the context of this project, an **endpoint** refers to a specific OData API provided by the C4C system that allows access to particular data. For example:

- The **OpportunityCollection** endpoint retrieves data related to opportunities.
- Similarly, other endpoints are designated for accessing various types of data, such as accounts, leads, visits etc.

bookkeeping: This folder has separate json files for each endpoint. Suppose 5 endpoints are there, then 5 json files would be there. JSON files are named after the endpoint (in lowercase).

data: This folder stores C4C API data that SAP DI fetches from respective endpoints in the form of parquet files. Each endpoint has its own subfolder, where the corresponding Parquet files are stored.

JSON Structure for an Endpoint (Initial)


```
[{
  "BASE_URL":"https://my325055.crm.ondemand.com/sap/c4c/odata/v1/c4codataapi/
OpportunityCollection[2]",
  "TOP_VAL":"5000",
  "TABLE_NAME":"opportunity_collection",
  "TIMESTAMP":"None",
  "UPDATE_IDENTIFIER":"EntityLastChangedOn"
}]
```


What is Endpoint: Here "**OpportunityCollection** " is the name of the endpoint.

TOP_VAL: Maximum number of records to be fetched during one API call. This is limited to prevent any memory issue.

TABLE_NAME: Name of the folder in S3 where the data of this API will reside in parquet format.

TIMESTAMP: First its set to None. This is done to get an initial full load of all data in the given endpoint. Once the first load is done, the loading timestamp will change and a new record will be added to this JSON list.

UPDATE_IDENTIFIER: This is that field in the API, which denotes the last changed time of each record. Since we don't intend to do full loads each time. This will help us in doing delta loads.

JSON Structure for an Endpoint after one load.

2 https://my325055.crm.ondemand.com/sap/c4c/odata/v1/c4codataapi/ServiceRequestCollection

Assuming the load was started at 1pm. Notice that one record was automatically added. Everything remains same, except for the TIMESTAMP. Now next time when data is loaded, we fetch only those records updated/added after this TIMESTAMP.


```
[{
   "BASE_URL":"https://my325055.crm.ondemand.com/sap/c4c/odata/v1/c4codataapi/
EmployeeCollection[3]",
   "TOP_VAL":"5000",
   "TABLE_NAME":"employee_collection",
   "TIMESTAMP":"None",
   "UPDATE_IDENTIFIER":"EntityLastChangedOn"
},

{
   "BASE_URL":"https://my325055.crm.ondemand.com/sap/c4c/odata/v1/c4codataapi/
EmployeeCollection[4]",
   "TOP_VAL":"5000",
   "TABLE_NAME":"employee_collection",
   "TIMESTAMP":"2024-08-21T13:00:00Z",
   "UPDATE_IDENTIFIER":"EntityLastChangedOn"
}]
```


1.3 Creating an SAP DI Graph

An **SAP Data Intelligence (DI) Graph** is a visual workflow or pipeline in SAP Data Intelligence that defines a sequence of data operations for processing, transforming, or integrating data. It allows you to design and automate end-to-end data workflows by connecting various operators (nodes) that perform specific tasks like reading, transforming, or writing data.

Each endpoint has its corresponding DI Graph. Everything remains the same expect for a parameter in the Python Script which denotes the name of the endpoint to fetch. This is done by specifying the name of JSON file stored in the bookkeeping folder. The script reads this file to get endpoint details like BASE_URL, UPDATE_IDENTIFIER etc. (as explained in section 1.2)

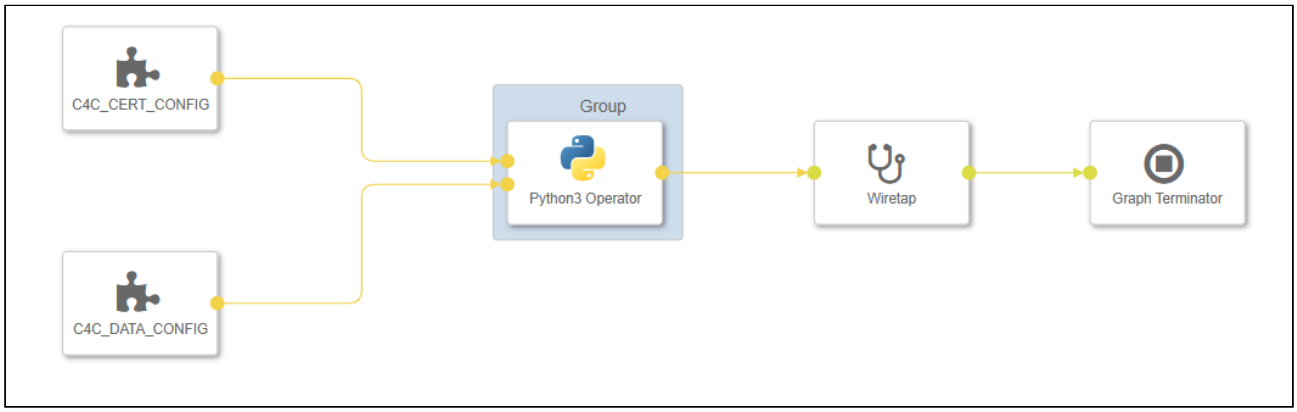


Fig 3: SAP DI Graph Structure

C4C_CERT_CONFIG: This configuration object contains the credentials required to access the S3 bucket where the C4C certificate is securely stored. As explained in previous sections, we use a certificate-based authorization to access the C4C OData API's.

C4C_DATA_CONFIG: This configuration object provides the access and secret keys for the S3 bucket that stores the C4C data.

3 https://my325055.crm.ondemand.com/sap/c4c/odata/v1/c4codataapi/ServiceRequestCollection
4 https://my325055.crm.ondemand.com/sap/c4c/odata/v1/c4codataapi/ServiceRequestCollection

Python3 Operator: The python operator in SAP Data Intelligence enables custom scripting in Python programming language for data workflows. It uses credentials provided to connect to the S3 bucket, fetch raw data, process it and save the transformed data back to the S3 bucket in parquet format.

Wiretap: This is an operator which is used to view the logs generated by the pipeline.

Graph Terminator: This operator is used to mark the end of a pipeline, ensuring proper termination of the data flow in the graph.

To access the script used in the python operator please follow this link: <To be updated soon>

1.4 Scheduling the DI Graph

Once the pipeline is ready, the graph needs to be scheduled to run automatically and dump the data retrieved from the C4C API into respective S3 bucket.

2. Snowflake Side Development

Assuming raw data is present in S3, given below are some of the next steps.

DEV_DB: DEV_TR_DEVOPS

ACC_DB & PROD_DB: PROD_TR_DEVOPS

It is advised to use the above roles while developing in Snowflake. This ensures your objects are accessible to someone else working on the same endpoint.

2.1 S3 to raw table

Automated Method (Recommended):

Run the below procedure to get data from S3 stage to Snowflake raw table.

C4C_INIT_PROC('<BASE_NAME>');

BASE_NAME: Name of respective endpoint folder in S3.

Example: CALL C4C_INIT_PROC('opportunity_collection');

This procedure will create the raw table, main table, stream, pipe and refresh the pipe. All done in one go.

Manual Method:

You can perform all the steps which the procedure (C4C_INIT_PROC) performs. All the commands are given below:

1. Create raw and main table for each endpoint

```
CREATE OR REPLACE TABLE T_OPPORTUNITY_COLLECTION_RAW
  USING TEMPLATE (
    SELECT ARRAY_AGG(OBJECT_CONSTRUCT(*))
      FROM TABLE(
        INFER_SCHEMA(
          LOCATION=>'@CLOUD_FOR_CUSTOMER_SALES_RAW_EXT_STG/data/
opportunity_collection',
```

```
          FILE_FORMAT=>'c4c_parquet_format'
        )
      ));

-- Creating the main table
CREATE OR REPLACE TABLE T_OPPORTUNITY_COLLECTION
  USING TEMPLATE (
    SELECT ARRAY_AGG(OBJECT_CONSTRUCT(*))
      FROM TABLE(
        INFER_SCHEMA(
          LOCATION=>'@CLOUD_FOR_CUSTOMER_SALES_RAW_EXT_STG/data/
opportunity_collection/',
          FILE_FORMAT=>'c4c_parquet_format'
        )
      ));
```

Raw table naming format: T_**<NAME_OF_ODATA_ENDPOINT>**_RAW

Main table naming format: T_**<NAME_OF_ODATA_ENDPOINT>**

Create a stream on top of the raw table. This keeps a track of the records being inserted into the raw table. Eventually the merge procedure uses this stream data to merge data from raw to main table.

2. Create a stream on top of the raw table.

```
create or replace stream OPPORTUNITY_COLLECTION_STREAM
on table T_OPPORTUNITY_COLLECTION_RAW
append_only = true;
```

Stream naming format: **<NAME_OF_ODATA_ENDPOINT>**_STREAM

Create a pipe on the external stage. This is done to get the data from s3 bucket (parquet files) into the snowflake raw table

3. Create pipe to ingest data from S3

```
create or replace pipe
PROD_DB.CLOUD_FOR_CUSTOMER_SALES_RAW.OPPORTUNITY_COLLECTION_PIPE auto_ingest=true as
 COPY INTO T_OPPORTUNITY_COLLECTION_RAW
FROM @CLOUD_FOR_CUSTOMER_SALES_RAW_EXT_STG/data/opportunity_collection/
MATCH_BY_COLUMN_NAME=CASE_INSENSITIVE
FILE_FORMAT = (TYPE = 'PARQUET');z
```

Pipe naming format: **<NAME_OF_ODATA_ENDPOINT>**_PIPE

Refresh the pipe that has been created, in order to get the initial load from the existing files.

4. Refresh the Pipe

```
ALTER PIPE OPPORTUNITY_COLLECTION_PIPE REFRESH
```

So now that these steps have been completed, you should be able to see the initial load in the raw table, sometime after refreshing the pipe.

2.2 Raw table to Main Table Merge

Use this procedure to merge the data from raw table to the main table.

The merge happens on the basis of the primary key of the respective endpoint.

Format:

Call the Merge procedure

```
C4C_MERGE("SRC_DB_NAME" VARCHAR(16777216), "SRC_SCHEMA_NAME" VARCHAR(16777216),
"TGT_DB_NAME" VARCHAR(16777216), "TGT_SCHEMA_NAME" VARCHAR(16777216), "SRC_TABLE"
 VARCHAR(16777216), "TGT_TABLE" VARCHAR(16777216),"PRIMARY_KEY" VARCHAR(16777216))
```

Example of calling:

Call the Merge procedure

```
CALL C4C_MERGE('DEV_DB', 'CLOUD_FOR_CUSTOMER_SALES_RAW','DEV_DB','CLOUD_FOR_CUSTOMER_
SALES_RAW', 'T_OPPORTUNITY_SALES_TEAM_PARTY_COLLECTION_RAW','T_OPPORTUNITY_SALES_TEAM
_PARTY_COLLECTION','ID');
```

After calling this procedure, main table will also have the data.

2.3 Create Task to automate merging

The task will automate the merging process, ensuring that all data captured by the stream is merged into the main table.

Create task for an endpoint

```
create or replace task
PROD_DB.CLOUD_FOR_CUSTOMER_SALES_RAW.OPPORTUNITY_COLLECTION_MERGE_TASK
warehouse=PROD_WH_ELT
schedule='USING CRON 30 2 * * * CET'
as CALL C4C_RUN_MERGE_IF_STREAM_HAS_DATA_PROC('PROD_DB',
'CLOUD_FOR_CUSTOMER_SALES_RAW','PROD_DB', 'CLOUD_FOR_CUSTOMER_SALES_RAW',
'T_OPPORTUNITY_COLLECTION_RAW','T_OPPORTUNITY_COLLECTION','ObjectID');
```

The schedule is based on requirements. The procedure `C4C_RUN_MERGE_IF_STREAM_HAS_DATA_PROC` is triggered by the task and is designed to check whether the stream contains data. If data is present, this procedure invokes the `C4C_MERGE` procedure to process the data accordingly.

The logs for tasks can be viewed in **T_C4C_TASK_LOG** table given below:

	API_ENDPOINT	TIMESTAMP	ROWS_AFFEC
1	EXCHANGE_RATE_COLLECTION	2025-01-28 01:55:00.812	No Data in Stre
2	VISIT_COLLECTION	2025-01-28 01:50:00.843	No Data in Stre
3	SERVICE_REQUEST_COLLECTION	2025-01-28 01:45:22.369	719
4	OPPORTUNITY_SALES_TEAM_PARTY_COLLECTION	2025-01-28 01:40:22.535	136726
5	OPPORTUNITY_COMPETITOR_PARTY_COLLECTION	2025-01-28 01:35:05.769	0
6	OPPORTUNITY_COLLECTION	2025-01-28 01:31:29.202	135980
7	OPPORTUNITY_BUSINESS_TRANSACTION_DOCUMENT_REFERENCE_COLLECTION	2025-01-28 01:25:46.824	2032

Fig 4: Few records from T_C4C_TASK_LOG table

2.3 Create Views

Once the main tables have data, views can be made on top of it. An example can be seen below.

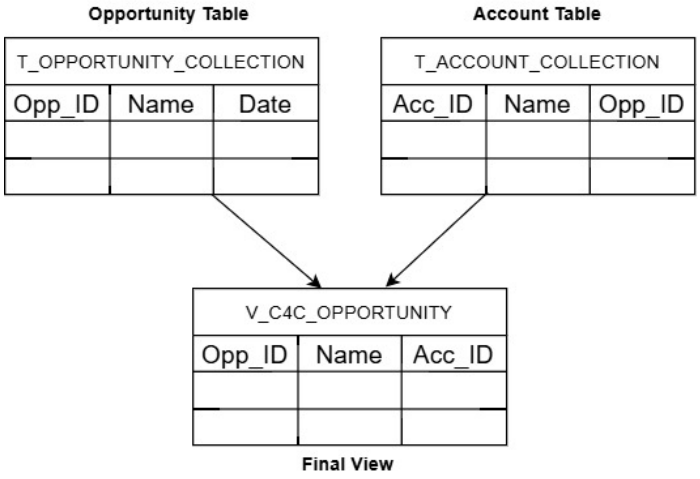


Fig 5: Example of a view