DARE UK

‘Virtual’ TREs

Technical Annex

Table of Contents

UK Research and Innovation HDRUK Health Data Research UK ADR UK

1. Section 1: Legal & Governance

1.1. DARE UK: Snowflake TRE Contracts

For a typical academic collaborative research consortium use, the TRE collaboration environment would need the following agreements:

A. To procure the platform and tools that will comprise the TRE collaboration environment (e.g. Snowflake and other software or tools that connect to the Snowflake TRE):
    - The companies' respective terms and conditions of service
    - Entered between the Crick and the company
    - The terms of use of the platform will need to permit sharing access with designated third parties (i.e. the external collaborators)

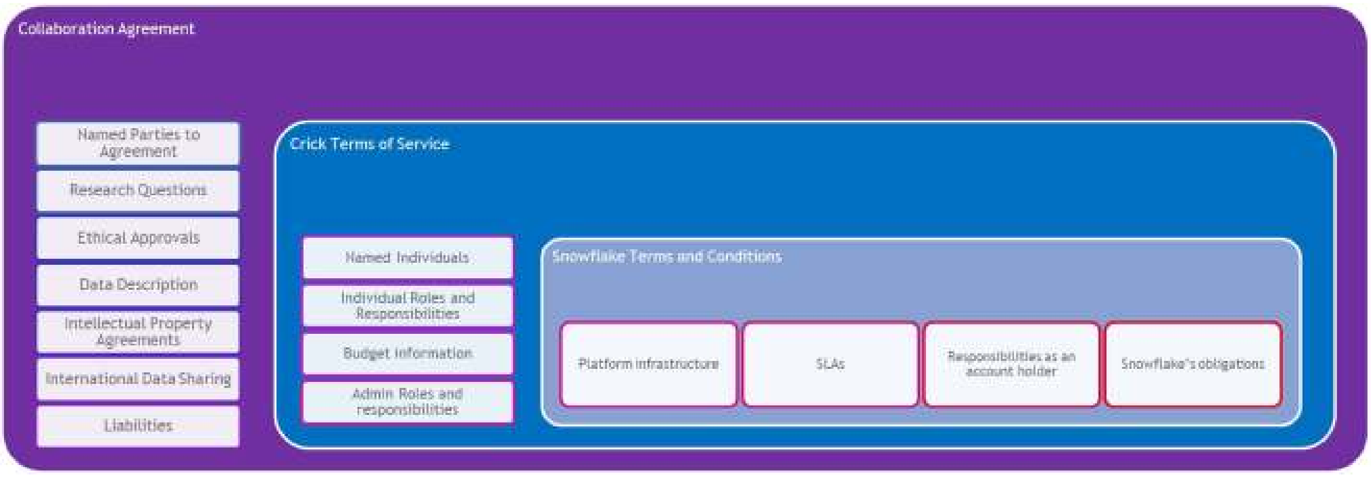
B. To give each participating institution access to the TRE collaboration:
    - The Crick's TRE terms and conditions
    - One copy signed per institution
    - The terms would include, for example:
        - an overview of the key rules of the TRE
        - an order form capturing information needed to configure the TRE
        - the terms and conditions of TRE access, including terms of the third parties' terms (from A above) that need to be flowed down to all user institutions
    - The terms of use will be directed by the upstream service provider's terms and conditions (in this case Snowflake)

C. To govern the participating institutions collaborative research:
    - Collaboration agreement
    - One copy signed by all the participating institutions who wish to undertake research on the shared data
        i. Can include terms to ascent new parties to the agreement, if the original institutions wish to expand their collaboration
    - The form could be a long-form Brunswick collaboration agreement (which is well recognised between UK universities). Terms regarding ownership or use of arising IP and publications may vary, depending on what the collaboration.

D. Optional, if there are 'upload only' institutions who do not access any of the shared data (i.e. not undertaking the research, just data providers):
    - Data sharing agreement
    - Shorter version of agreement (C)

1.2. Overview diagram

The elements in blue are designed to be standard for every agreement, leaving only the elements which have to be unique to every collaboration in purple.



Collaboration Agreement
Named Parties to Agreement
Research Questions
Ethical Approvals
Data Description
Intellectual Property Agreements
International Data Sharing
Liabilities
Crick Terms of Service
Named Individuals
Individual Roles and Responsibilities
Budget Information
Admin Roles and responsibilities
Snowflake Terms and Conditions
Platform infrastructure
SLAs
Responsibilities as an account holder
Snowflake's obligations

Figure 1.2.1: Schematic Representation of Legal Agreement structure

1.3. Data protection and regulatory considerations

Each of the above agreements needs to contain appropriate data protection and data governance terms.

Each participating institution should involve their Data Protection Officer (DPO) in reviewing the proposed arrangement. Each institution is responsible for advising if the data set they propose to share is governed by specific data protection or governance requirements, and ensuring prior to any sharing that such sharing is compliant with all such requirements.

We drafted the agreements on the basis that each upload designated account is located to the region (country) requested by the uploading institution (we'd expect the country their source study participants are located in, or if not available on TRE regions list, a suitable alternative selected by the institution's DPO), and the collaboration designated account is located in the EU, for example Dublin, Ireland.

Some of the aspects we covered in the above agreements were:

- Clearly stating restrictions on users, e.g., for this TRE, we included for example (not exhaustive list):
    - Access and use the data for the research project only
    - No uploading of HIPAA data (meaning patient, medical or other protected health information regulated by HIPAA or any similar U.S. federal or state laws, rules or regulations) without prior Crick approval (because under the TRE terms, TRE designated accounts need to be configured to hold HIPAA data).

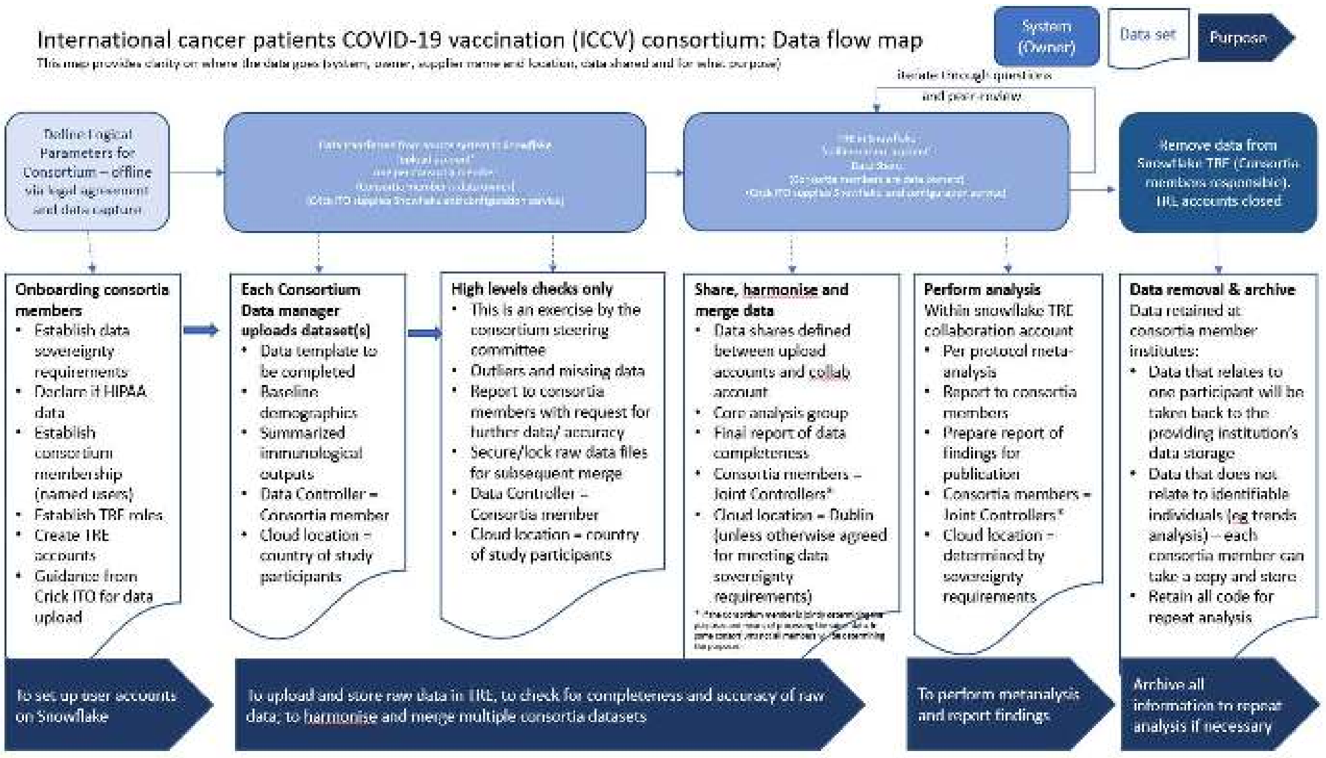
- Stating responsibility on each institution to ensure: data anonymisation or pseudonymisation, data transformation (i.e. transforming format so that analysis can be performed), data screening and quality control steps in accordance with a protocol.

- Including a standard for international data transfer, in this case the EU SCCs and UK Addendum, because if any of the participating institutions are located ex-EU / UK, their looking at data is a transfer ex-EU/ UK)

- Aiming to keep the data protection wording similar across the agreements for quicker review by the institutions DPO/ contract office.

- Including a responsibility on each institution to remove all data from TRE at the end of the project (or early termination). The plan will need to take into consideration the nature of the data and meeting legal and regulatory requirements.

It would also be helpful to share, with the suite of agreements, a data flow diagram to assist the institutions' contracts office / DPO to understand the TRE collaboration. An example is shown here:



International cancer patients COVID-19 vaccination (ICCV) consortium: Data flow map
This map provides clarity on where the data goes (system, owner, supplier name and location, data shared and for what purpose)
System (Owner)
Data set
Purpose
iterate through questions and peer-review
Define Logical Parameters for Consortium – offline via legal agreement and data capture
Remove data from Snowflake TRE (Consortia members responsible). TRE accounts closed
Onboarding consortia members
Establish data sovereignty requirements
Declare if HIPAA data
Establish consortium membership (named users)
Establish TRE roles
Create TRE accounts
Guidance from Crick ITO for data upload
Each Consortium Data manager uploads dataset(s)
Data template to be completed
Baseline demographics
Summarized immunological outputs
Data Controller = Consortia member
Cloud location = country of study participants
High levels checks only
This is an exercise by the consortium steering committee
Outliers and missing data
Report to consortia members with request for further data/ accuracy
Secure/lock raw data files for subsequent merge
Data Controller – Consortia member
Cloud location = country of study participants
Share, harmonise and merge data
Data shares defined between upload accounts and collab account
Core analysis group
Final report of data completeness
Consortia members – Joint Controllers*
Cloud location = Dublin (unless otherwise agreed for meeting data sovereignty requirements)
Perform analysis
Within snowflake TRE collaboration account
Per protocol meta-analysis
Report to consortia members
Prepare report of findings for publication
Consortia members = Joint Controllers*
Cloud location = determined by sovereignty requirements
Data removal & archive
Data retained at consortia member institutes:
Data that relates to one participant will be taken back to the providing institution's data storage
Data that does not relate to identifiable individuals (eg trends analysis) – each consortia member can take a copy and store
Retain all code for repeat analysis
To set up user accounts on Snowflake
To upload and store raw data in TRE, to check for completeness and accuracy of raw data; to harmonise and merge multiple consortia datasets
To perform metanalysis and report findings
Archive all information to repeat analysis if necessary

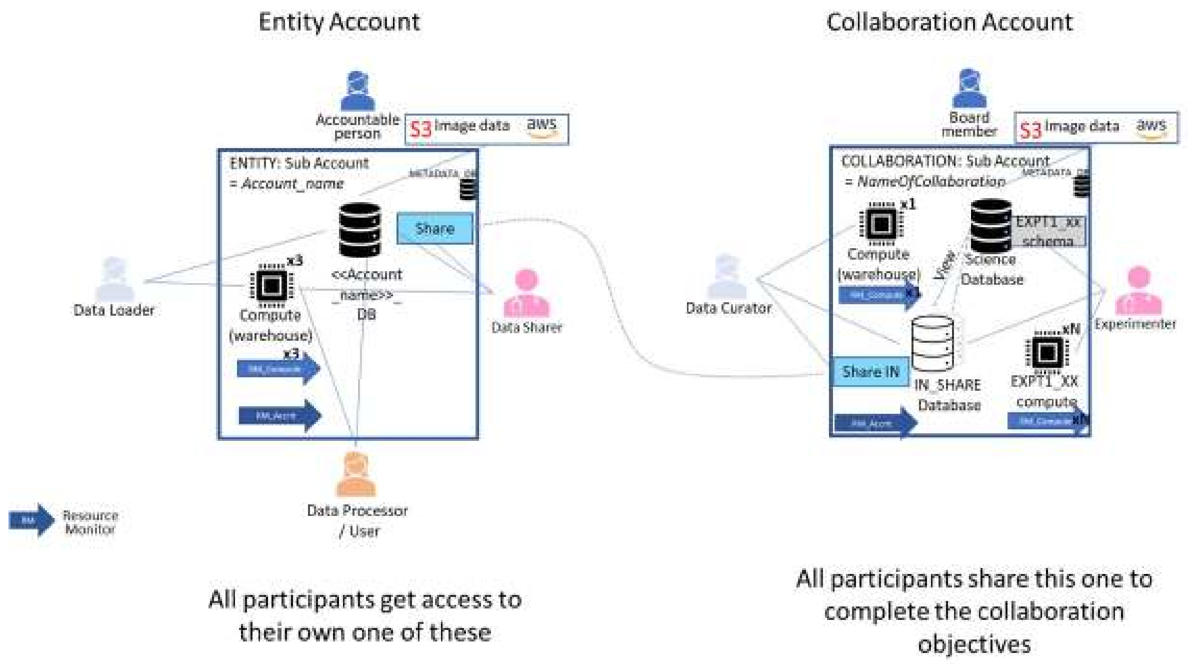
Figure 1.3.1: Example Data Flow Diagram to communicate process and roles for using the TRE in a collaboration

2. Section 2 : Snowflake TRE Data design, architecture and tooling

2.1. Data Process

The TRE collaboration environment is built around a standard process flow, see Figure 2.1.1.

1) ELT tools (see section xxxx) enable data to be LOADED into the Database for each ENTITY account. This loading activity is only done by people holding the DATA LOADER role for that account.
2) Once Loaded the data can be PROCESSED ready for sharing to the collaboration account and experimentation. This processing is only able to be done by people holding the DATA PROCESSING role.
3) Once prepared the data can be SHARED to a collaboration account. This sharing can only be done by people holding the DATA SHARER role.



2.1.1 Process Flow of Data within a Collaboration TRE

4) Once the data has been shared, then the data can be made available to the users within the COLLABORATION account. To make this happen the data needs to be CURATED into sets in a Science Database ready for use. This curating can only be done by people holding the DATA CURATOR role. Granting access to experimentation areas is also done by the DATA CURATOR role, and so it is this role that conforms to instructions coming from the collaboration steering committee.
5) The final EXPLOITATION stage is where the analysis of the combined data is done. This can only be done by people holding the EXPERIMENTER role and done within a specific SCHEMA set up specifically for the authorised experiment. This is where tools such as RStudio may be deployed and connected to the datasets. This is where data created by the Collaboration will be stored (in SCIENCE database) and where material for Journal publications will be based.

2.2. Data Sharing

2.2.1. Sharing within Region

Sharing of data means something very specific in Snowflake (see https://docs.snowflake.com/en/user-guide/data-sharing-intro.html). It is only possible between Accounts in the same Snowflake Region.

A data share is where one team Account allows access (through a view) to another team Account. It is like a "grant", and means the "grantee" is not in control of the data. A share can be made with very fine granularity selecting only specific columns or rows of data from a single table or a combination of tables.

Only COLLABORATION accounts have the ability to consume a Shared database. Entity Accounts are set up to not be able therefore to share data between themselves (as they cannot ingest the share).

The ACCOUNT can share three generic sources of data:

1. Sharing Data from <<Account_name>>_DB: This is the structured and semi-structured data
2. Sharing S3 internal stage: This is the binary data files (such as images)
3. Sharing S3 External Stage: This is the binary data files (such as images) and any data they have already stored in their own S3 bucket areas.

2.2.2. Sharing between Regions

If the Accounts are not in the same region then the data files need to be replicated. This means that the account data is copied. Replication also maintains synch of the data so scheduled updates are made to keep the "copy" identical. Once copied the data is no longer 100% controlled by the owning ACCOUNT. The removal of the replication does means no updates to the data already shared occur, but the is no "remote delete" option.

Data Share replication: https://docs.snowflake.com/en/user-guide/data-share-replication.html

Replicating Databases: https://docs.snowflake.com/en/user-guide/database-replication-intro.html

2.2.3. Utilising a Share

Once a share is made the receiving account gets notification of the Share and the DATA CURATOR role can then instantiate the shared database (and the shared tables, columns and rows). This is very simply done: https://docs.snowflake.com/en/user-guide/data-share-consumers.html#

```
CREATE DATABASE <name> FROM SHARE <provider_account>.<share_name>
```

Once the database is created, then a view of that data is created within the SCIENCE_DB and then providing grant access to the different EXPERIMENTER Roles using it is normal database administration.

2.2.4. Types of Data

The Snowflake platform supports all data types and can hold and enable querying of the data in JSON format. Structured data and unstructured data are held in databases, or as external tables. ELT / ETL tools will map such data into the snowflake type automatically, and database DDL SQL syntax can be deployed to set up and modify table formats if or when specialist tailoring is needed.

Image and other binary files are held in staging areas. Two options exist, 1) internal S3 AWS containers or ii) External mounted S3 AWS containers. Both of these are provided for in the build of the Account. The difference is that the Internal S3 stage is inherently enabled at account creation time. The external S3 needs to be configured manually through support of the *admin team*.

Snowflake also provides the opportunity to define data types. And a variety of knowledge content is available to support these features (a few are listed here):

- Data Types: https://docs.snowflake.com/en/sql-reference/intro-summary-data-types.html
- Semi-structured data : https://docs.snowflake.com/en/user-guide/semistructured-concepts.html like JSON/XML/Parquet/ORC/AVRO
- Binary data: https://docs.snowflake.com/en/user-guide/binary.html (image files etc)

2.3. Data Governance and Control

Governance and control of the data is done in a variety of ways. No restrictions are placed on the types of data held nor the table formats. Instead, the principles embedded in the collaboration agreement, and legal regulations provide an operating framework.

The platform is governed through three key functions:

1) Approvals
2) Roles
3) Facilities

2.3.1. Approvals

No person can be created without an auditable approval trail replicated in two places - the workflow ServiceNow platform component and in the logs held on the implementation template code and the snowflake platform.

No native snowflake capability is available to users to create other users or change passwords or set up resources. The ACCOUNTABLE PERSON role is the authorisation, and can see the data held in the <<Account_name>>_DB and their accounts METADATA set so they can interrogate activity being done on their ACCOUNT. For Collaboration Accounts the BOARD MEMBER role has equivalents.

2.3.2. Roles

The flow of data shown in Figure 2.1.1 is enabled through five different roles:

ENTITY ACCOUNT

1) DATA LOADER
2) DATA PROCESSOR
3) DATA SHARER

COLLABORATION ACCOUNT

4) DATA CURATOR
5) EXPERIMENTER

Each stage requires the specific role to used. If the same person is given all roles, then to do anything malicious they will have to consciously change their role for each stage. The full audit metadata inherent in the platform will mean this will be identified and action can be taken.

2.3.3. Facilities

The Snowflake platform offers two different types of account: 1) Enterprise and 2) Business Critical. If ISO27001 or HIPPA standards need to be met then the account is created with the Business Critical type, which has additional and necessary controls and security facilities to support this.

Snowflake also comes with extensive metadata capture which is transparent and available in a standard format across all databases. This means that all logins, table access, sql statements run etc is timestamp recorded and exposed to SQL just as other data would be. So, querying of this metadata will allow the development of algorithms to identify and report activity that is deemed “risky”.

Snowflake also offers on-platform running of python and java routines. This enables more advanced data processing algorithms to be deployed connecting other facilities and approaches not native to SQL. These have been used to bring all the metadata from all the information tables together in one place and for this to be kept for longer than the snowflake “out-of-the-box” holding time (normally 365 days). As such all the audit and compliance metadata can be assured into a repository to help with Research Data Management (RDM) and similar activities.

2.4. Snowflake Data Tools

The TRE platform is designed to have the same set of specific facilities for each account type.

2.4.1. ENTITY Accounts

For the ENTITY Account type there is always:

A database called <<Account Name>>_DB. This is where the loaded data resides in tables and where the processed data would also sit. This is the database that would have shares made from it to other accounts.

A database called METADATA_DB. This is an administrative database and not accessible by any of the users of the account. It is used by system resources to maintain a full copy of all the native snowflake metadata on usage and object information. This is the base source of the data that feeds the master Organisation wide equivalent database that supports Reporting.

To process any data in the TRE a compute facility is needed. Each role has its own. These compute tools are not needing to be known by the users, they are automatically connected by the platform. However, if remote tools are used then the connectors often require the specification of the compute “warehouse” to be used. These are named as

DATA LOADER:	<<ACCOUNT_NAME>>_XS _DL _WH
DATA PROCESSOR:	<<ACCOUNT_NAME>>_XS _DP _WH
DATA SHARER:	<<ACCOUNT_NAME>>_XS _DS _WH

ACCOUNTABLE_PERSON <<ACCOUNT_NAME>>_XS _AP_WH

2.4.2. COLLABORATION Accounts

For every share made to the Collaboration an INSHARE database is required. This INSHARE database will be used, by the DATA CURATOR role holding person, to provide access to the shared data. It acts a little like a transparent pipe enabling the COLLABORATION data to see the shared data. It is not a copy of the data, the master of which remains in the originating ENTITY account.

The naming convention for these INSHARE databases is: <<ACCOUNT_NAME>>_INSHARE _DB

These IN_SHARE databases are not accessible to the EXPERIMENTER role, only the DATA_CURATOR. There is then a further two databases:

A science database called <<ACCOUNT_NAME>>_SCIENCE_DB. This is the database the EXPERIMENTER role people use to develop the results of the collaboration. This database has specific SCHEMA designs, with a separate schema for each approved experiment.

The second, called METADATA_DB, performs the identical function as for ENTITY accounts, bringing together a full historical record of what was done in the COLLABORATION account. Again, it is not accessible by people, and only used by the system (although the BOARD MEMBER role can access it).

Again, there are compute "warehouses" for each of the three roles:

DATA CURATOR:	<<ACCOUNT_NAME>>_ XS_DC_ WH
EXPERIMENTER :	<<ACCOUNT_NAME>>_ XS_EX_WH
BOARD MEMBER :	<<ACCOUNT_NAME>>_XS_BM_WH

2.5. Naming Conventions

In order to match certain data up strict adherence to naming conventions will be needed. This is because some of the "joins" desired are not explicitly possible within the systems Snowflake.

For a SubAccount called <<ACCOUNT_NAME>> the objects are labelled in Table 1.

Type of Object	NAME	Description
Database:	<<ACCOUNT_NAME>>_INSHARE _DB	For INSHARES form other accounts involved in the collaboration. This is only relevant for COLLABORATION accounts
Database:	<<ACCOUNT_NAME>>_SCIENCE _DB	The Database within which all Science analysis is completed and results stored.
Database	<<ACCOUNT_NAME>>_DB	The database used by the ENTITY to land their loaded data, process this into an transformed form, and this is the database that is the source of the data to be "Shared" or "Replicated" to the collaboration account involved.

Database	METADATA_DB	This holds the Metadata curated for the account
Compute (Warehouses)	<<ACCOUNT_NAME>>_ XS _<<2 letter role> _WH <<ACCOUNT_NAME>>_<<EXPT NAME>> _ XS _ EX _ WH	The two letter roles are **DL**: Data Loader (Entity Account) **DP**: Data processor (Entity Account) **DS**: Data Sharer (Entity Account) **AP**: Accountable Person (Entity Account) **DC**: Data Curator (Collaboration Account) **EX**: Experimenter (Collaboration Account) **BM**: Board member (Collaboration Account) For the experimenter roles the specific EXPERIMENT name will be added to the end. The WH refers to Warehouse The XS signifies Extra -Small and is the initial default size of the compute on start-up.
Compute (Warehouses)	METADATA_WH CORE_WH	The Metadata Compute that is used to curate the metadata tasks. No specific RM - uses ACCOUNT_RM CORE_WH does the object creation and manage the internal systems activity (uses ACCOUNT_RM)
Resource Monitor	<<ACCOUNT_NAME>>_XS _<<2 letter role>> _RM <<ACCOUNT_NAME>>_<<EXPT NAME>>_ XS _ EX _ RM	The resource Monitor associate with the COMPUTE resources for the role. The two-alpha shorthand for the roles are the same as those used for the Compute objects. RM stands for Resource Monitor
Resource Monitor	<<ACCOUNT_NAME>>_ACCOUNT _RM	The Resource Monitor associated with the ACCOUNT overall (including all compute and storage).

Table 1 Naming Convention for different Snowflake objects

The key is to ensure transparency to the user and to enable derivation code, using named objects, to relate them to ROLES and therefore People who have access to them. By aligning all the objects names with the roles more control is provided.

Additional naming conventions, within the processes that move data from the different parts of the system eco-system (Service Now, AWS flows, Snowflake, GITHUB etc), also helps to ensure that the readability and traceability is maximised.

2.6. Audit and Compliance Data

2.6.1. Metadata

As already discussed, the TRE platform provides a transparent full set of usage and information metadata. These are all brought together for each account within a METADATA_DB and then collected at the platform level in copies within the ORG_ADMIN_DB. The refresh Schedule for the METADATA (both at Account level and ORG_ADMIN level) sets the latency of the data. A similar co-ordinated refresh schedule will be needed at the reporting end to ensure the latest “real time” facts are being used. At present a 1 hr refresh at METADATA level is used and a 2 hr Refresh at PowerBI[1].

The overarching process for the report creation is shown in Figure 1. The Org Admin account holds all METADATA from all the individual accounts across the Snowflake platform. Views are created in that database to present the objects shown in Figure . These Views are “copied” into the powerBi world to replicate the equivalent data model as shown. The permissions table uses the Microsoft UserPrincipalName() – obtained from the identity management stage and the Azure Active Directory, see section XXX (identity) - to filter the data to just those required by that person based on the roles they have been granted to have. The screens have filter tools which on selection act as filters. The charts and visuals are themselves active filters allowing “click” selection filtering.

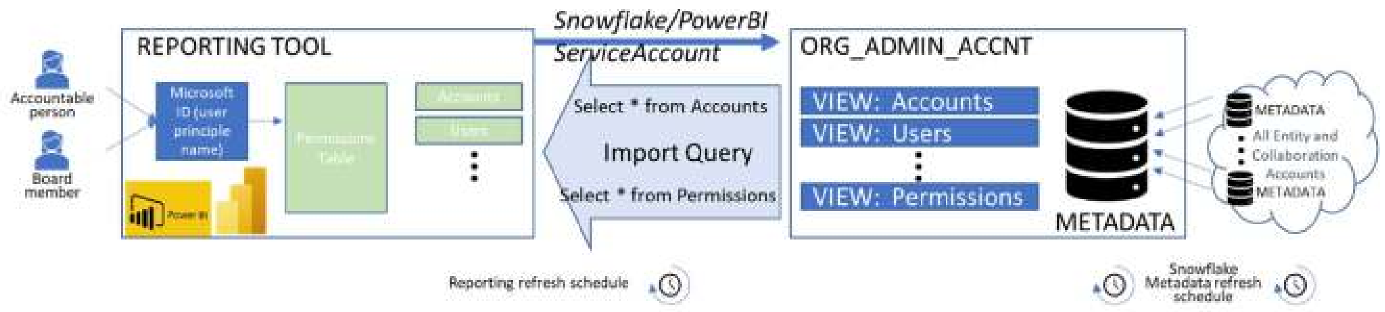


Figure 1.6.1.1: Top-level components and flows for Reporting Audit and Compliance

These separate information and usage metadata tables have been used to provide an audit and compliance data model, see Figure . It shows Billing information in yellow and usage information is green. Data objects are shown in Grey and cover Columns in Tables in Schemas in Databases. In addition, in grey, are all the users, their login and the SQL they instructed the platform to run (be it through remote tools or native on the platform).

In Blue are tables that are additions to the native Snowflake facts. A set of people is derived from the combined users, the set of accounts is held, and the budgetary information entered in the configuration and setup phase (and any in life changes) is held. The platform also offers the ability to TAG data - see Tagging Data (shown in red in Figure 2.6.1.2).

[1] Refresh rates dictate costs as each refresh involves compute and storage

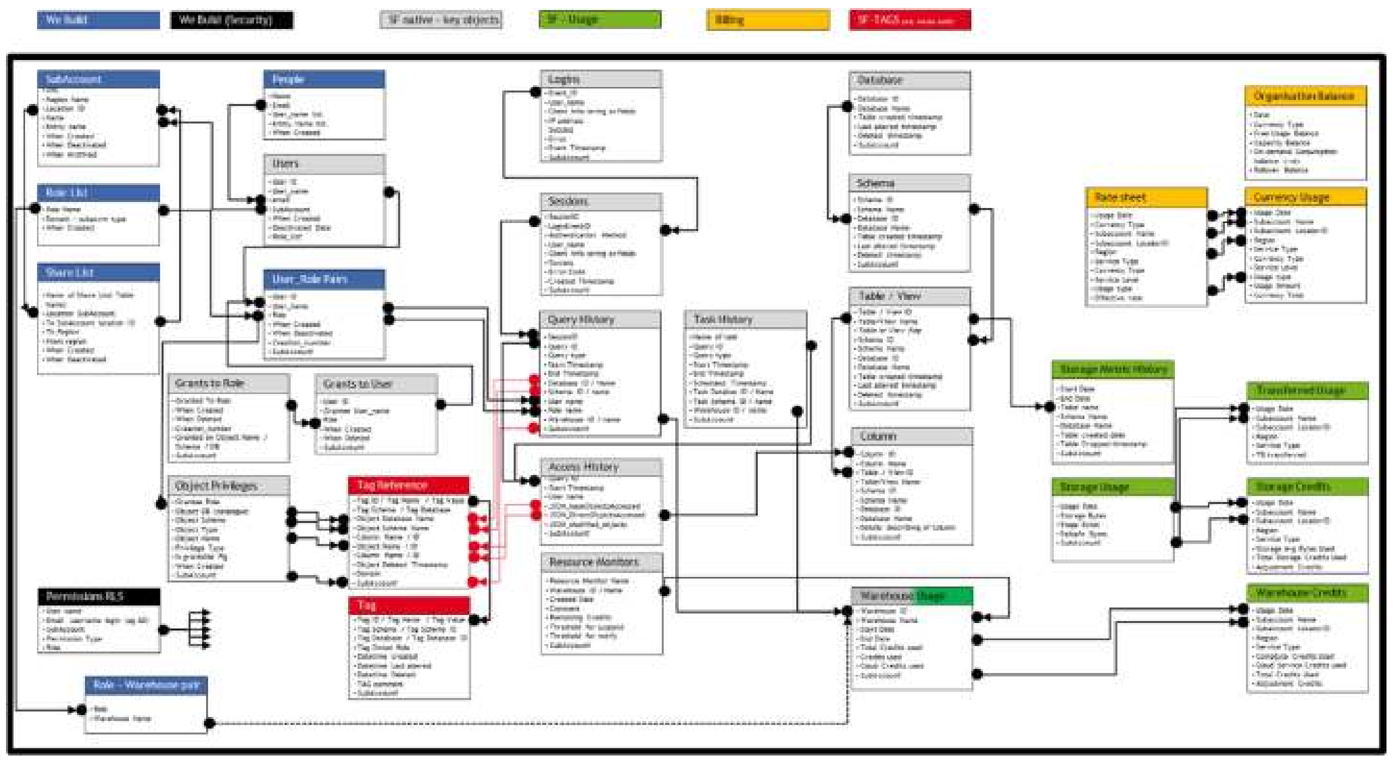


Figure 2.6.1.2: Metadata model of the Platform usage and information data

2.6.2. Data Flows

To facilitate the collection of the metadata to support reporting and to record who did what when and what objects existed in the TRE accounts, two additional roles are enabled. These are not accessible to users and are used by the system.

The roles are:

a) ACCOUNT_REPORTER
b) ORG_DATA_CURATOR

The ACCOUNT REPORTER role is used to access the METADATA_DB in each account. It has read, but not write, privileges on all the Information and Usage metadata collected there for the account (ENTITY or COLLABORATION).

The ORG_DATA_CURATOR role is used by a central system platform account (The ORG_ADMIN_ACCOUNT) to access the METADATA_DB through shares or replications to it.

The computing associated with collecting and curating this key metadata is done using a hidden system compute "warehouse". The costs of this activity are covered by a low level % "tax" on the credit budgets provided in the agreement.

2.7. Reporting

In normal operation a TRE will not see the usage information except through a standard cloud-based web reporting tool. At present this is implemented within PowerBI. No account will be needed for PowerBI but the costs associated with providing a licence through the Crick may be charged back.

This single reporting tool that implements a Row Level Security model, offers access to audit and compliance information. Only people with ACCOUNTABLE PERSON or BOARD MEMBER roles will see a set of reporting tool interfaces that help with monitoring activity and discharging audit and compliance roles outlined in the legal collaboration agreement, as well as good governance methods.

This tool is to be found at this URL: https://app.powerbi.com/groups/d7b42544-803b-4db3-99d5-a32f89d7e440/reports/a655e6a7-1151-40b2-94fc-d6b7a1a59d20/ReportSection02cc206c8cebce6b82dd

The areas provided are:

1) What any individual in a collaboration organisation can see: Based on the account the person is responsible for they will see the users approved and the roles and platform components they have access to.
2) Who is involved in a collaboration: What roles people have, what logins have been made? Who has logged in when, how many times have they failed, how did they login?
3) Usage levels: What usage has been made of the compute and storage facilities. What the storage capacity is by table name and database?
4) What money has been spent: How much cost has been incurred?

Additional reports can be provided as they are developed and will be provided through the same application / tool.

The data model within the reporting tool is shown in Figure 2.7.1.

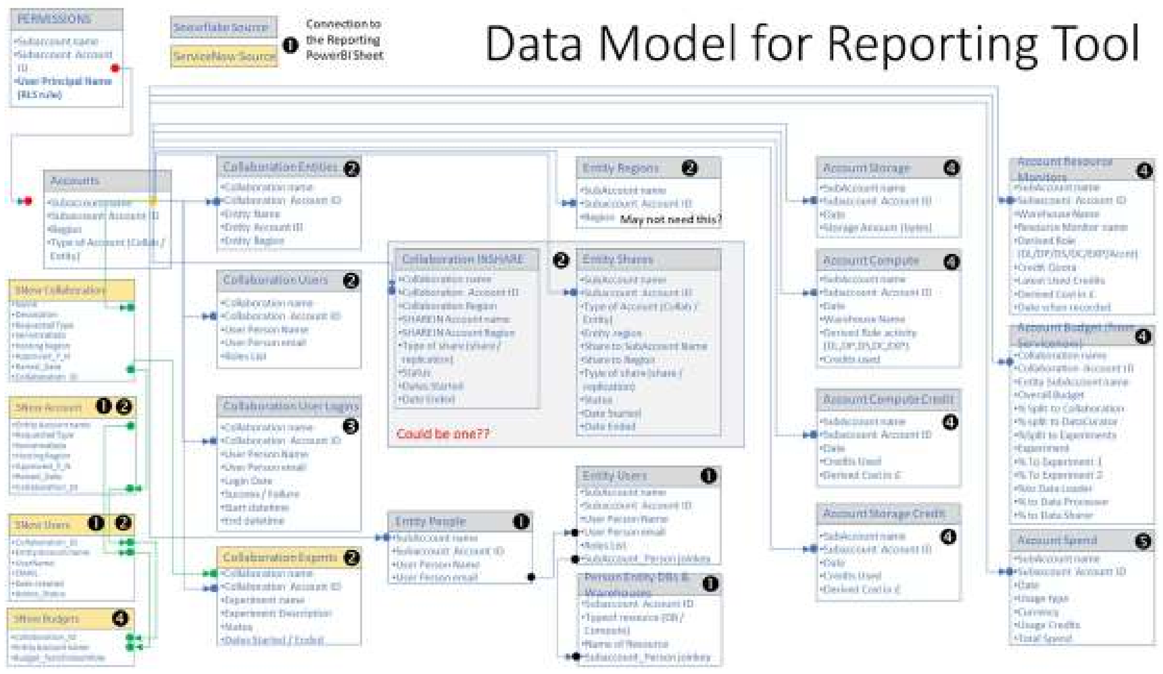
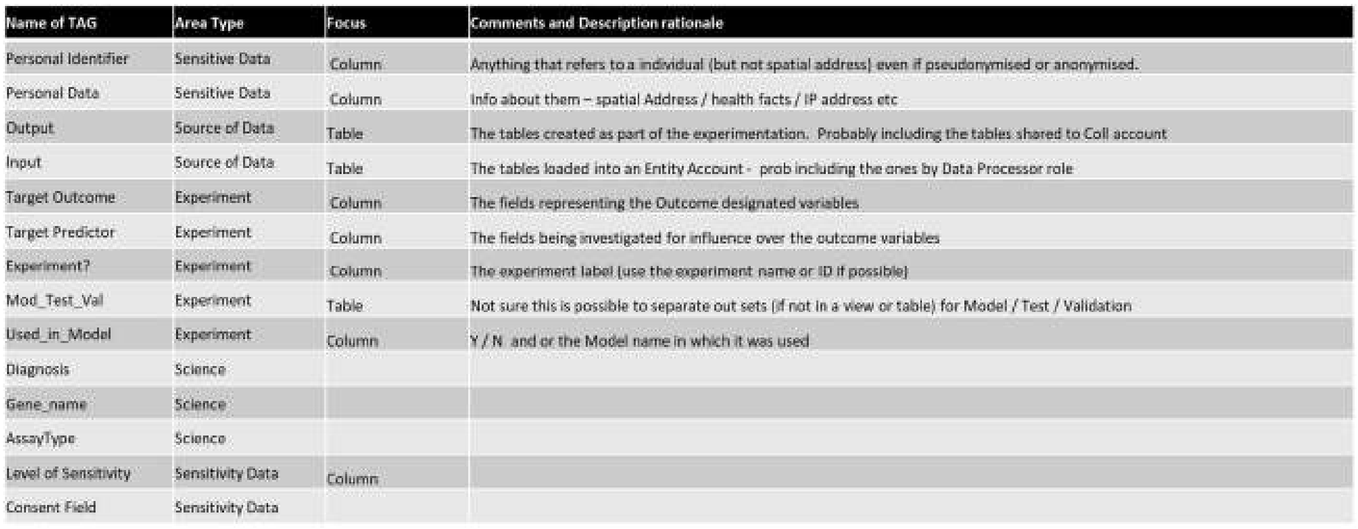


Figure 2.7.1: Data model for reporting

2.8. Tagging Data

The underlying Snowflake technology supporting the TRE platform offers tagging of data objects at a variety of levels. See this knowledge article: https://docs.snowflake.com/en/user-guide/object-tagging.html

However, a set of reserved tags have been defined for within the platform developed, see Table 2.8.1.



Name of TAG	Area Type	Focus	Comments and Description rationale
Personal Identifier	Sensitive Data	Column	Anything that refers to a individual (but not spatial address) even if pseudonymised or anonymised.
Personal Data	Sensitive Data	Column	Info about them – spatial Address / health facts / IP address etc
Output	Source of Data	Table	The tables created as part of the experimentation. Probably including the tables shared to Coll account
Input	Source of Data	Table	The tables loaded into an Entity Account - prob including the ones by Data Processor role
Target Outcome	Experiment	Column	The fields representing the Outcome designated variables
Target Predictor	Experiment	Column	The fields being investigated for influence over the outcome variables
Experiment?	Experiment	Column	The experiment label (use the experiment name or ID if possible)
Mod_Test_Val	Experiment	Table	Not sure this is possible to separate out sets (if not in a view or table) for Model / Test / Validation
Used_in_Model	Experiment	Column	Y / N and or the Model name in which it was used
Diagnosis	Science		
Gene_name	Science		
AssayType	Science		
Level of Sensitivity	Sensitivity Data	Column	
Consent Field	Sensitivity Data		

Table 2.8.1: Tagging architecture for TRE platform - reserved models to be applied by participating entities

The application of tags can be done within native Snowflake SQL or through the use of commands passed by functions.

3. Section 3: ServiceNow Processes

3.1. ServiceNow Portal

The Crick uses a custom ServiceNow Portal to direct users to the relevant forms that they need to carry out admin functions for their collaboration. These include:

1. Create Consortium
2. Add Entities
3. Create Budgets
4. Add Users to Collaboration account
5. Add Users to Entity accounts
6. Modify Users
7. Add Experiment
8. Add Warehouse
9. Archive Request

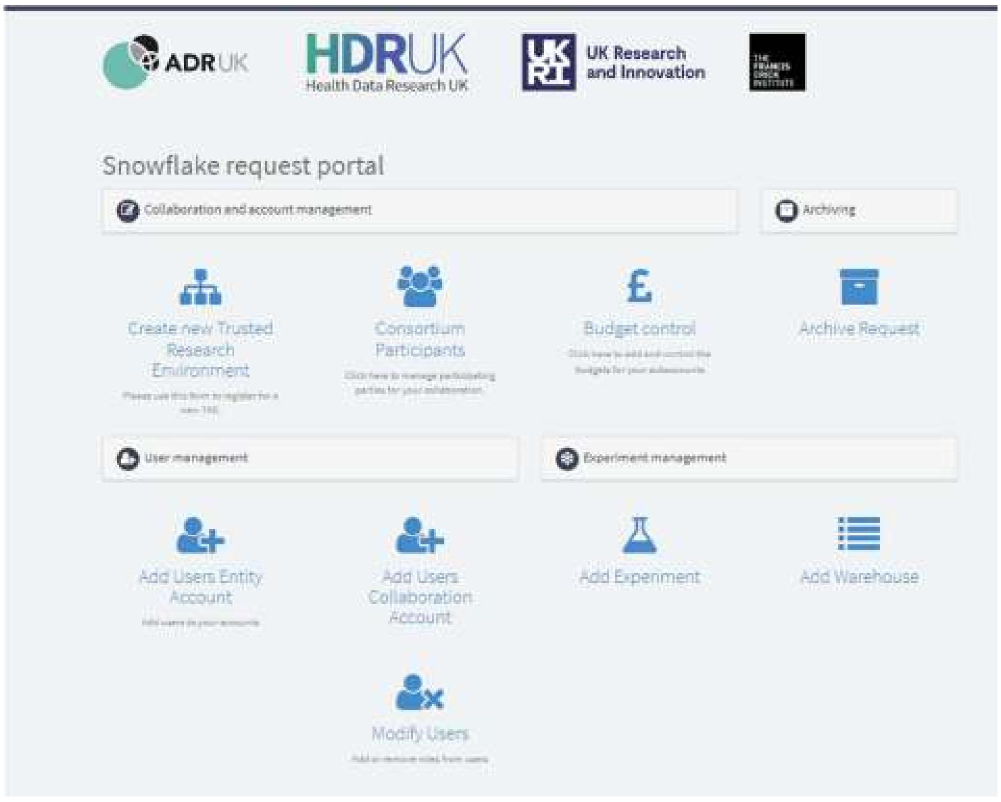

ADRUK
HDRUK Health Data Research UK
UK Research and Innovation
THE FRANCIS CRICK INSTITUTE
Snowflake request portal
Collaboration and account management
Archiving
Create new Trusted Research Environment
Please use this form to register for a new TRE
Consortium Participants
Click here to manage participating parties for your collaboration
Budget control
Click here to add and control the budgets for your subaccounts
Archive Request
User management
Experiment management
Add Users Entity Account
Add users to your accounts
Add Users Collaboration Account
Add Experiment
Add Warehouse
Modify Users
Add or remove roles from users

Figure 3.1.1: The Service Now Portal Homepage

3.2. Creating a TRE

1. Users fills out ServiceNow portal form with their consortium details and attaches any agreement documentation.
2. ServiceNow creates a record on the consortium table (x_tfci_snowflake_consortium) and a collaboration account record on the account table (x_tfci_snowflake_account_records) and a budget record on the budget table (x_tfci_snowflake_budget)
3. The details will be checked by the snowflake administrator and approved.
4. Once approved ServiceNow will then run a Flow to take the details from the account ticket and create a JSON file with all the configuration details.
5. The JSON file is then uploaded to the Amazon S3 bucket (tre-setup-account-prod) *See Section 4.3.1*

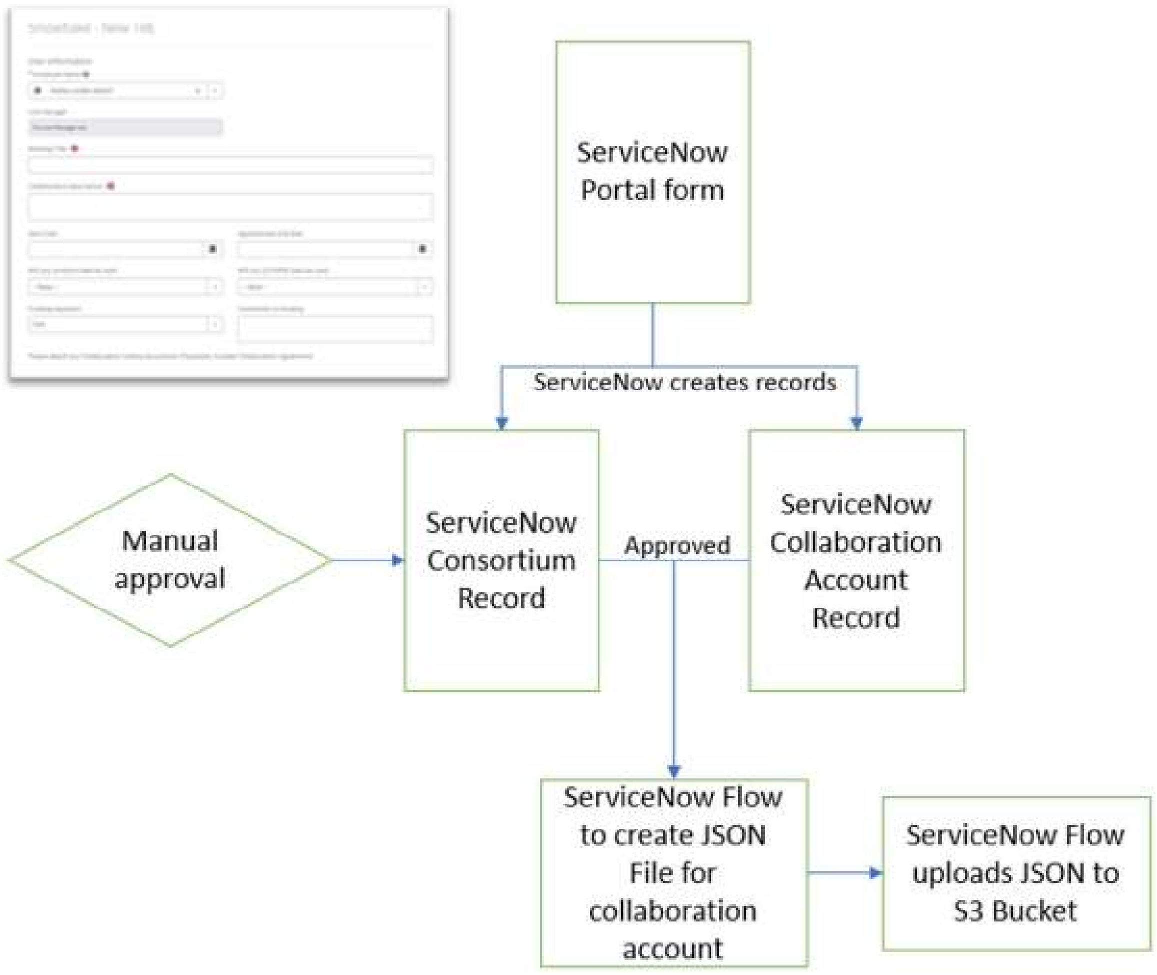

ServiceNow Portal form
ServiceNow creates records
Manual approval
ServiceNow Consortium Record
Approved
ServiceNow Collaboration Account Record
ServiceNow Flow to create JSON File for collaboration account
ServiceNow Flow uploads JSON to S3 Bucket

Figure 3.2.1: Service Now – The 'Create TRE' form and flow for the form.

3.3. Add Consortium Entities

1. User fills out a ServiceNow Portal form They can add up to six LABs, six STP's and 4 external partners. If they need more then the form can be filled out again.
2. On the form they can specify if the account will use anu Personal data that conforms to ISO27001 or HIPAA
3. A ServiceNow Flow then turns each account into a separate account record and connects it to the consortium record selected on the form.
4. When the accounts are approved on the record a ServiceNow flow creates individual JSON files with all the configuration details.
5. A ServiceNow flow then uploads these files to the Amazon S3 Bucket (tre-setup-account-prod). *See Section 4.3.3*

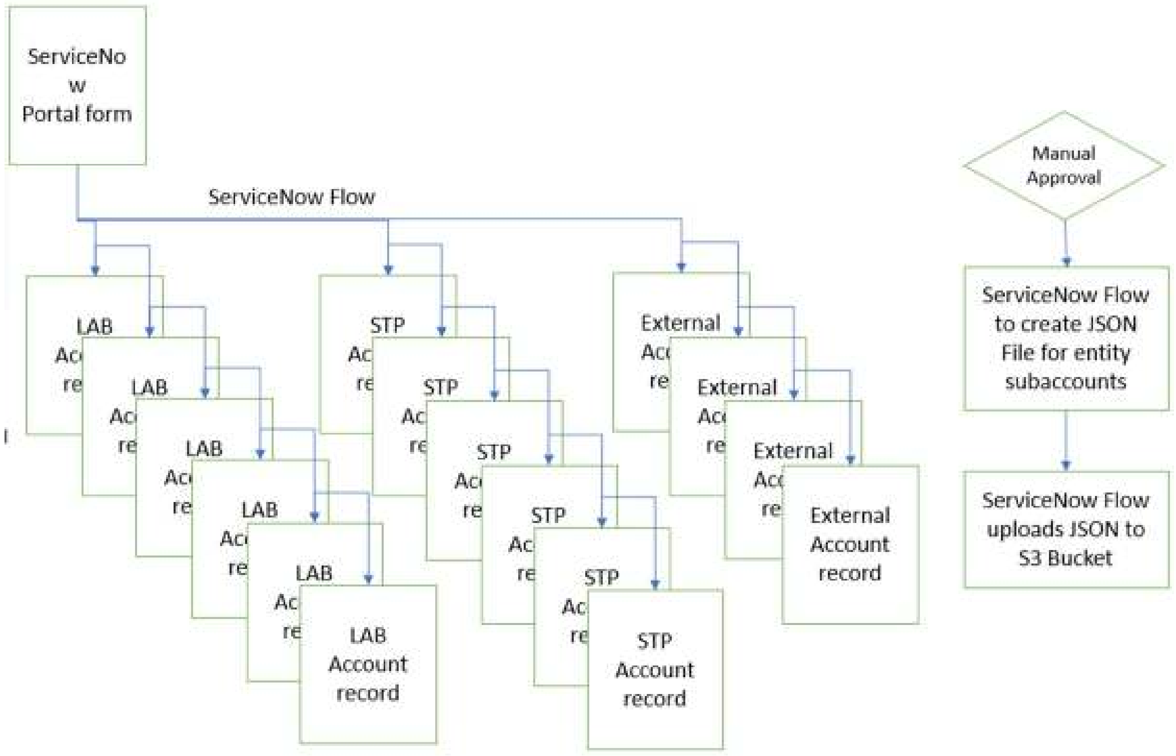


Figure 3.3.1: Service Now – Creation of accounts for TRE.

3.4. Add Users (Collaboration and Entity)

There are 2 separate forms for adding users to Collaboration accounts or entity accounts. The only difference is the Roles that can be chosen.

1. The user chooses Consortium and Subaccount from the dropdown list. If adding users to the collaboration account, they only select the Consortium.
2. Using the pop-up section of the form they can add users and choose specific roles. If the user has a Crick email, then it automatically selects OKTA enabled so they can sign in with their Single Sign On details.
3. A ServiceNow Flow then creates a user record (x_tfci_snowflake_user_records)
4. A confirmation email will be sent to the user to check they are real and need to be added.
5. When the user replies to the confirmation email it will be recorded in the record.
6. User account is then approved.
7. Once approved a ServiceNow flow creates a JSON file and uploads it to an Amazon S3 Bucket (tre-setup-user-prod) *(See Section 4.3.5)*

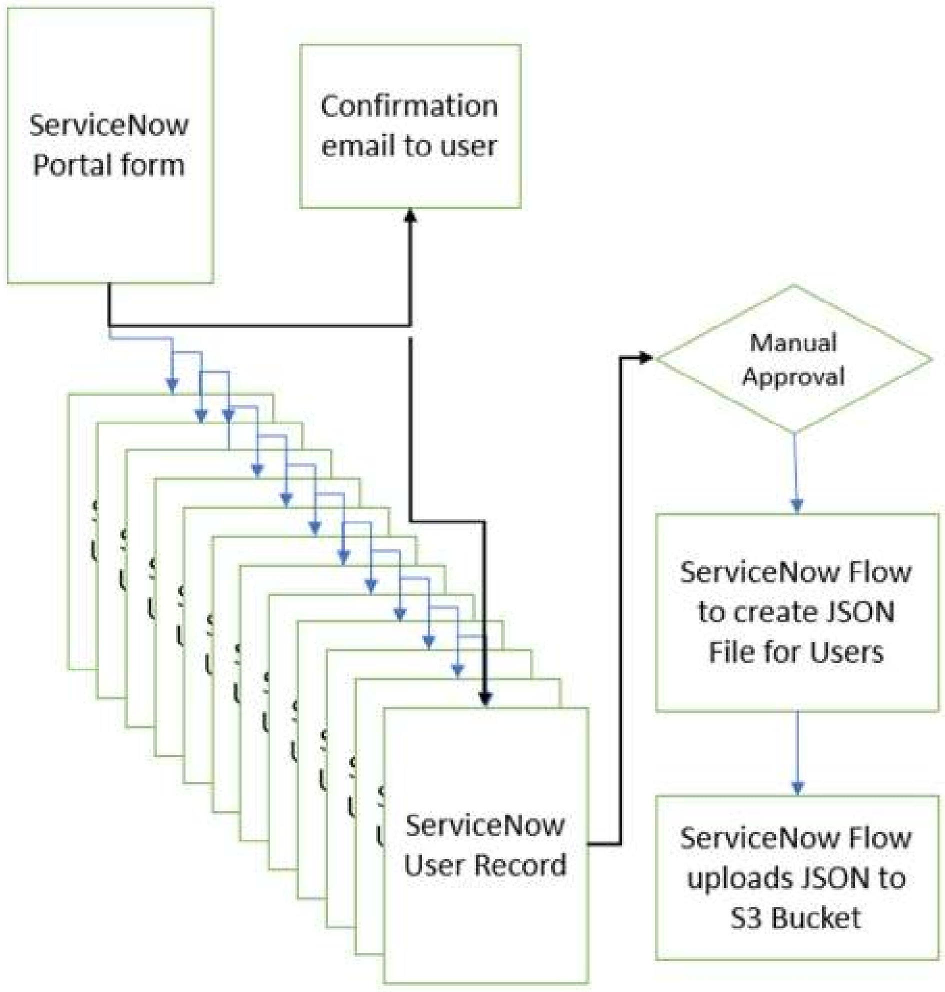

ServiceNow Portal form
Confirmation email to user
Manual Approval
ServiceNow Flow to create JSON File for Users
ServiceNow User Record
ServiceNow Flow uploads JSON to S3 Bucket

Figure 3.4.1: Service Now – Creation of new users for TRE

3.5. Add Experiment

1. User selects Consortium ID and Sub Account and budget record from the drop-down lists.
2. User enters experiment name, description, start and end dates then submits.
3. Experiment record is created in ServiceNow.
4. Experiment is then approved.
5. A ServiceNow flow then creates a JSON file and uploads it to an Amazon S3 Bucket. (tre-setup-experiment-prod) (*See Section 4.3.4)*

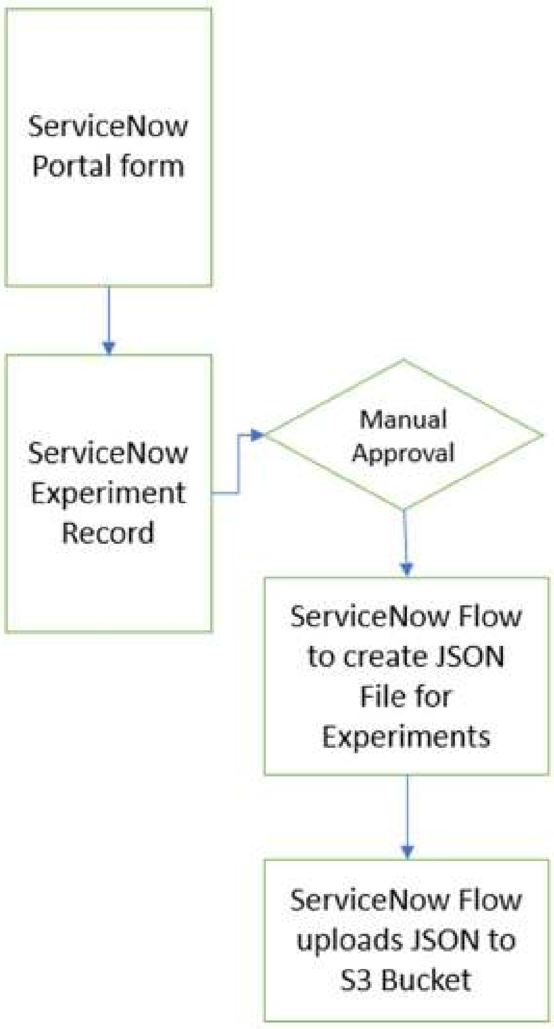

ServiceNow Portal form
ServiceNow Experiment Record
Manual Approval
ServiceNow Flow to create JSON File for Experiments
ServiceNow Flow uploads JSON to S3 Bucket

Figure 3.5.1: Service Now – Creation of new experiments in collaboration account for TRE.

3.6. Create Budgets

1. User selects Consortium ID and Sub Account from the drop down lists and enters project/grant code.
2. Choosing the various options on the form the form will calculate the cost and amount of credits for each user role, collaboration account and entity account
3. This then creates a ServiceNow record in the budgets table (x_tfci_snowflake_budget) and connects it to the subaccount record and the consortium record.
4. Data is pulled from the budget record into the JSON files to create accounts and set-up users. *See Section 4.3.2*

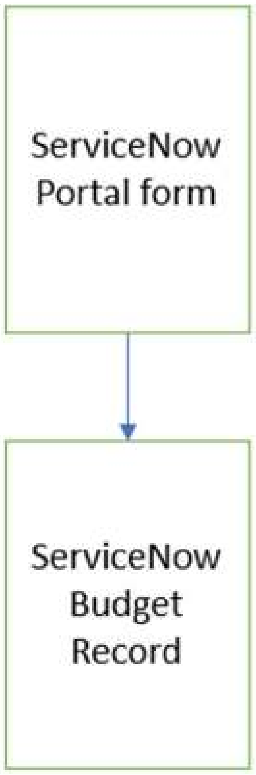
Calculations and splits.

Inputs:

- Total Budget
- Number of months for project
- Percentage split for Collaboration account
- Percentage split for Entity Accounts
- Percentage split for Collaboration Roles
- Percentage split for Entity Roles

Outputs:

- Cost per month
- Cost per month for Collaboration account
- Cost per month for Entity Accounts
- Cost per month for Board Member role
- Cost per month for Data loader role
- Cost per month for Data Curator role
- Cost per month for Data Sharer role
- Cost per month for Data Processor role
- Cost per month for Accountable person role
- Cost per month for Experimenter role



3.7. ServiceNow records view

Six main tables are used for storing the records in ServiceNow.

1. Consortium Records – Stores details of the TRE consortium, contact details, any legal agreements can be attached to the record. This acts as the master record for the TRE and links to all the tables below.
2. Account Records – Details of the entity and collaboration accounts. Linked to the consortium record
3. Budget Records – Budget records for each entity and collaboration account. Linked to the consortium and account records
4. User Records – User details for each account including roles. Linked to the account and consortium records.
5. Experiment Records – Details of the experiments, linked to account and consortium records
6. Warehouse Records – Details of the Warehouses, linked to account and consortium records.

3.7.1. Consortium Records

Fields	**Description**
Number	The ServiceNow record number
Created	Date the record was created
Created by	Who created the record
Working Title	The title of the consortium , script turns it to capitals and turns spaces into underscores
Collaboration description	Discription of the collaboration
Sensitive Data	
ISO27001	Will any sensitive data be used that is classed as ISO27001
HIPAA	Will any sensitive data be used that is classed as HIPAA
Details of Sensitive Data	Details of sensitive data
Users	
Consortium Participant's Lead	Name of the consortium lead
Leads email	Their email
Consortium Agreement	
Consortium agreement signed	Name of the person that signed the agreement
Start date	
End Date	
Hosting details	
Hosting location of upload account	

Hosting location of collaboration account	
Financial Information	
Funding expected	Core funding / Grant / Mix
Funding comments	
Fees budget	Metadata and admin charge permonth
Entity budget	Overall budget for Entity accounts
Collaboration budget	Overall budget for collaboration account
Invoicing Information	
Address	Adress to send invoice to
Via	Email to send invoice to
With a copy to	
Quoting reference	
Any other information	
Archive	
Archive requested	Checkbox is true if requested
Archive date	
Automatic or manual	Archiving process
Name of user	User who will perform the archiving
Full path to location of data	
Username	To access archive
Password	To access archive
To be archived	All or specific
Items to be archived.	List specific items to be archive if above is Specific
I confirm no meta data will be archived	True or false
Signed	
Signature (francis crick)	Details of person who has signed the legal agreement at the Crick
Name	
Title	
Date	
Signature (Consortium)	Details of the person who has signed the legal agreement on behalf of the consortium
Approved	
Approved	Yes or no
Approved by	Name of Snowflake admin
Approved date	

3.7.2. Account records

Fields	**Description**
Active	True or false
Consortium ID	Reference to the Consortium record
Name of Body	Name of the account
URL	The snowflake url
Type	Type of account. (Collaboration, Lab, STP, External)
Domain	Domain of external entities
Region	AWS region of account
ISO27001	Any sensitive data being used of type ISO27001
HIPAA	Any sensitive data being used of type HIPAA
Approved	
Data Sharing only account	Only data sharing rolls can be assigned
Archive	
Archive requested	Checkbox is true if requested
Archive date	
Automatic or manual	Archiving process
Name of user	User who will perform the archiving
Full path to location of data	
Username	To access archive
Password	To access archive
To be archived	All or specific
Items to be archived.	List specific items to be archive if above is Specific
I confirm no meta data will be archived	True or false

3.7.3. Budget Records

Fields	**Descriptions**
Consortium ID	Reference to consortium record
Sub account	Reference to Account record
Number	ServiceNow number of the budget record
Project Code	Internal project code
Grant Code	External grant code
Overall budget	Overall budget for the account
Contract period	How long in months the contract is for
Value per month	
Notify percent	The percentage at which the account will notify the user that their credit is nearly up
Collaboration / Entity split	

Percentage for entity account	
Value for entity account	
Credits for entity account	
Percentage for collaboration account	
Value for collaboration account	
Credits for collaboration account	
Roles	
Data loader percent	% of budget for data loader role
Data loader value	Value per month for data loader role
Credits for data loader	Credits per month for data loader role
Data Processor percent	
Data processor value	
Credits for data loader	
Data sharer percent	
Data sharer value	
Credits for data sharer	
Data curator percent	
Data curator value	
Credits for data curator	
Experimenter percent	
Experimenter value	
Credits for experimenter	
Board member percent	Default is 5%
Board member value	
Credits for the board member	
Accountable person percent	
Accountable person value	
Credits for the accountable person	

3.7.4. User records

Fields	**Description**
Consortium ID	Reference to consortium record
Subaccount	Reference to account record
Okta enabled	SSO enabled for user login
Role	
User	Users email
Active	True or false, when approved will trigger the json upload flow
Created date	
Firstname	
Lastname	
LOGIN	Login name for SSO

3.7.5. Warehouse records

Fields	Description
Active	True or false
Warehouse	The warehouse name
Consortium ID	Reference to the consortium record
Name of Body	Reference to the account record
Archive	
Archive requested	Checkbox is true if requested
Archive date	
Automatic or manual	Archiving process
Name of user	User who will perform the archiving
Full path to location of data	
Username	To access archive
Password	To access archive
To be archived	All or specific
Items to be archived.	List specific items to be archive if above is Specific
I confirm no meta data will be archived	True or false

3.7.6. Experiment record

Field	Description
Number	ServiceNow record number
Active	True or false
Consortium Id	Reference to the consortium record
Experiment name	
Experiment description	
Budget account	Reference to the budget account record
Experimenter percent	Percentage of the budget for the experimenter
Experimenter value	
Data curator percent	
Data curator value	
Start date	
End date	
Notify percent	
Approved	True or false
Approval date	
Approved by	
Archive	
Archive requested	Checkbox is true if requested
Archive date	
Automatic or manual	Archiving process

Name of user	User who will perform the archiving
Full path to location of data	
Username	To access archive
Password	To access archive
To be archived	All or specific
Items to be archived.	List specific items to be archive if above is Specific
I confirm no meta data will be archived	True or false

4. Section 4: TRE Off-Snowflake Processing

4.1. Overview

The 'TRE Off-Snowflake Processing' refers to the workflows that build the cloud infrastructure for the TRE using a cloud platform called Snowflake. For full details about Snowflake, please refer to their online documentation here: https://www.snowflake.com/en/

The aim is to create an environment that can automatically create specific snowflake accounts used for data sharing between external and internal users. The data needs to be protected and transferred safely. The snowflake environment allows to use of custom roles to control data access. For more information on types of snowflake account. *(See Section 2.3)*

Figure 4.1.1 below, shows the high-level diagram for the setup to create the process to build snowflake objects. As in the diagram, the process moves from ServiceNow, AWS and finally to snowflake.

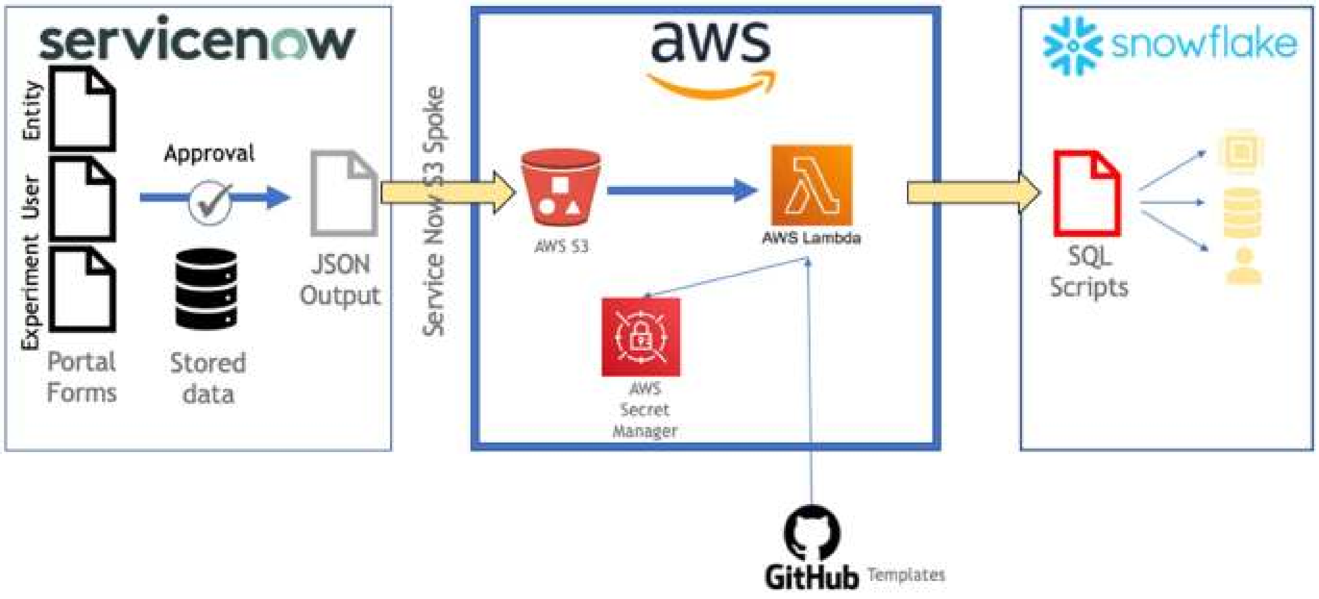


Figure 4.1.1: Overview off-snowflake processing in the cloud. From ServiceNow to AWS to Snowflake.

A form in ServiceNow is populated by a user and approved - for more detail, see section 3. The output from the form is dropped into a pre-built S3 bucket in AWS. This triggers a Lambda function; the event is controlled by logging on to the S3 bucket. This lambda function uses a bespoke tool "Flows", built by our technical partner Infinite Lambda. This could also be custom built using Terraform, or manually copied and run in the snowflake client. To build the SQL script to run and create compute, storage and users necessary for the account to run.

Other tools could be used for processing the GitHub queries, one being GitHub Actions. However, the reason we selected AWS Lambda was the physical size of the run: GitHub Actions was tested and found to be too slow for the needs of the Crick instance. AWS Lambda functions gave us more flexibility to split up all the parts of the code and run individually. AWS Secret Manager could be replaced with other options such as Azure Vault or CELO.

4.2. Crick Snowflake Organisation

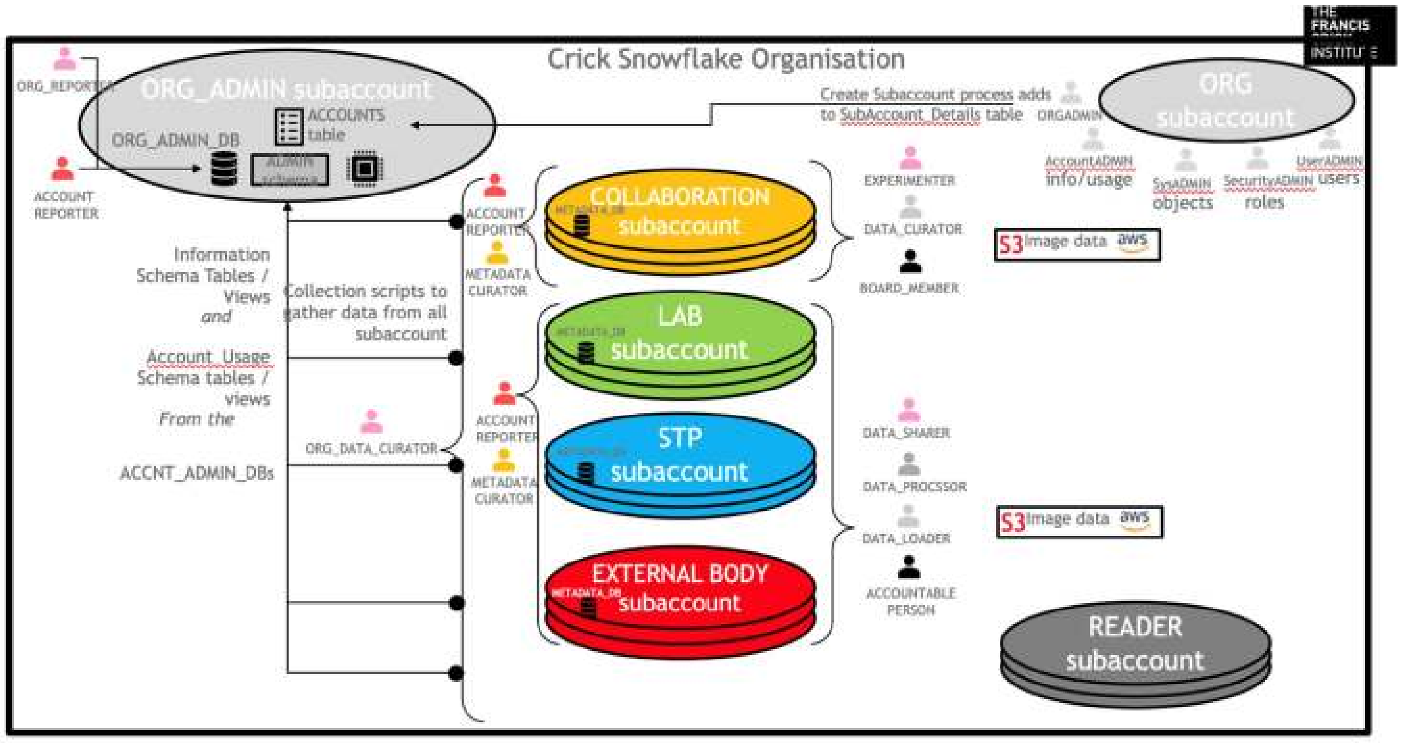


Figure 4.2.1: Overview off-snowflake processing in the cloud. From ServiceNow to AWS to Snowflake.

Figure 4.2.1 allows us to see our design of the organisation, showing the links between different types of accounts and the roles available on the account. The Crick Snowflake Organisation has specific rules and naming conventions to control security and to have uniformity across the TRE. See Section 2 for more information. From Figure 4.2.1, there are 4 different types of accounts: COLLABORATION, LAB, STP, and EXTERNAL BODY. 'LAB' and 'STP' are specific to the organisational structure of the Crick – these could easily be renamed to match a different naming convention at a given institution, eg 'GROUP' and 'FACILITY'. For each of these accounts there are specific roles. *(See 2.4.1 for more detail on entity accounts and 2.4.2 for collaboration accounts).* With every account we will create at least 2 databases one to capture metadata called 'METADATA_DB', which we use for auditing purposes, and a database to carry out the work that needs to be done as part of the collaboration.
In the Diagram above the metadata is visible as a small database on the left of the subaccount.

While the ORG account that creates all the subaccount cannot see any data within the subaccounts, we actively will share metadata to allow us to audit the subaccounts with minimal intrusion. This also allows us to build rich reports on activities on the subaccounts ensuring we identify any red flags as soon as possible

For more details see section 2.6.1 on Metadata Processing.

The ORG subaccount on the right-hand side in Figure 12 is the master organisation account. This is used to create new subaccounts within the organisation.

The TRE environment build is kicked off by json files being dropped in pre-made S3 Buckets from ServiceNow *(See Section3).* There is an AWS Spoke built into ServiceNow, this allows easy connection to the S3 Buckets and subsequently the lambda function build.

4.3. AWS Setup

The AWS console contains S3 buckets and lambda functions. These are created and remade through Terraform.[1] For the crick solution we have a dev and prod account to allow changes. All S3 buckets and lambda functions are tested before being rebuilt by the GitHub code. This also a thorough check to make sure our objects are built correctly in AWS. As mentioned above, there are other cloud providers which can be used for this.

Github link: [1]FrancisCrickInstitute/TRE_off_snowflake_computing/terraform/

4.3.1. JSON output from ServiceNow

The json files that are received from service now are a basic form of:

```
{
VARIABLE_NAME1: “VARIABLE_VALUE1”,
VARIABLE_NAME2: “VARIABLE_VALUE2”,
…..
}
```

The JSON files holds various information needed to create the SQL queries to build different objects of the TRE. More about the specific files for each lambda function in the next section.

4.3.2. Lambda Function Triggers

All lambda functions are stored on the GitHub within a code repository.[1]

There are 20 Lambda functions:

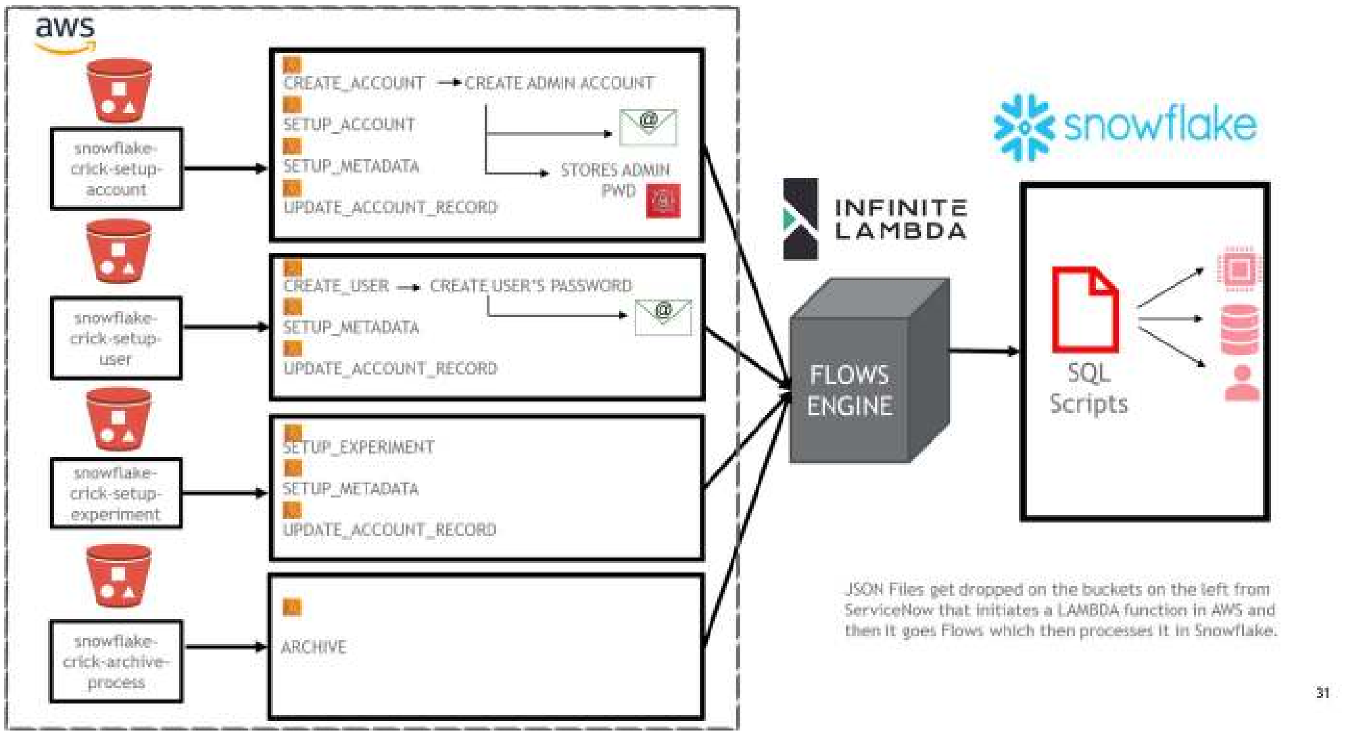
1. Create_account
2. Setup_account
3. Setup_user
4. Setup_experiment

5. Setup_okta_integration
6. Setup_okta_user
7. Create_external_stage
8. Setup_metadata
9. Alter_metadata
10. Update_snowflake_account_records
11. Update_snowflake_budget
12. Update_snowflake_consortium
13. Update_snowflake_experiment
14. Update_snowflake_experiment
15. Update_snowflake_user_records
16. Update_svn_account_records
17. Update_svn_experiment_records
18. Update_svn_objects_records
19. Update_svn_user_records
20. Update_svn_warehouse_records

Lambda function 1- 7 are used to set up the TRE. 8-9 are for the metadata. 10-15 are used to update snowflake metadata tables from ServiceNow. 16-20 are used to update ServiceNow metadata tables from snowflake.

There are various JSON files depending on the forms filled in:

1) Setup Accounts including separate forms for a collaboration or entity.
2) Add Experiment
3) Create/Modify User
4) Archive Process.



JSON Files get dropped on the buckets on the left from ServiceNow that initiates a LAMBDA function in AWS and then it goes Flows which then processes it in Snowflake.

Figure 4.3.2.1: File push into an S3 bucket trigger lambda function on the right.

Github links:

[1] FrancisCrickInstitute/TRE_off_snowflake_computing/lambda/

4.3.3. Create and Setup Accounts

When a user once to create a new account within the crick organisation, a form is filled in see SN docs. The form is dropped in S3 bucket called snowflake-crick-setup-account. This will trigger create_account lambda function$_1$. (*See Figure 13)* The JSON form will need to have the following variables to complete:

This includes information about the Account:

CONSORTIUM_ID

ACCOUNT_NAME

REGION

MONTHLY

NOTIFY_PERCENT

TYPE

START_TIMESTAMP

END_TIMESTAMP

CONTRACT_PERIOD

ACCOUNTABLE_PERSON

CREDITS_BUDGET_ACCOUNT

CREDITS_SUBACCOUNT_DATA_SHARER

CREDITS_SUBACCOUNT_DATA_LOADER

CREDITS_SUBACCOUNT_DATA_PROCESSOR

CREDITS_SUBACCOUNT_ACCOUNTABLE_PERSON

CREDITS_BUDGET_COLLABORATION

CREDITS_BUDGET_DATA_CURATOR

CREDITS_BUDGET_BOARD_MEMBER

Both collaboration & entity accounts

Just for entity account

Just for collaboration account.

4.3.3.1. Create Account

Based on the json input mentioned above from ServiceNow *(See Section 3.7.2)*, this lambda function generates template 03_setup account$_3$ and using bespoke "Infinite Lambda Flows Tool" creates a new account in the Snowflake organisation account based on 03_setup_account template$_3$.

The password for the ACCOUNTADMIN_MAIN is generated, saved to AWS secrets and sent to snowflake-notification-admin@crick.ac.uk.

Provisioning the URL for the new account can take 5-10 minutes and then will trigger setup account lambda function

After create_account lambda function has run. This will trigger setup_account.

GitHub Code

$_1$ *FrancisCrickInstitute/TRE_off_snowflake_computing/lambda/create_account/lambda.py*

$_2$ *FrancisCrickInstitute/TRE_off_snowflake_computing/templates/01_create_account.j2*

$_3$*FrancisCrickInstitute/TRE_off_snowflake_computing/templates/02_setup_account.j2*

4.3.3.2. Setup Account

Setup account will create the ACCOUNTADMIN_MAIN and stores the password in AWS Secret Manager. There is a secondary account ACCOUNTADMIN_BACKUP for an extra layer of processing if the main account is locked out. Both passwords are also stored in Azure Vault. The lambda function will trigger either 04_entity_subaccount.j2$_2$ or 06_collaboration_subaccount.json$_3$ depending on the type of account being setup.

Type of subaccount	Template used.
STP	04_entity_subaccount.j2$_2$
External	
Lab	
Collaboration	06_collaboration_subaccount.j2$_3$

To connect to the subaccount, the process uses the ACCOUNTADMIN_MAIN, created in the previous step. The password would be taken from the AWS secret manager. It uses the flow built in tool to create the desired yml file from the j2 template. The YML file will contain sql code that can run directly in the newly created subaccount to create new objects.

GitHub Code

$_1$ *FrancisCrickInstitute/TRE_off_snowflake_computing/lambda/setup_account/lambda.py*

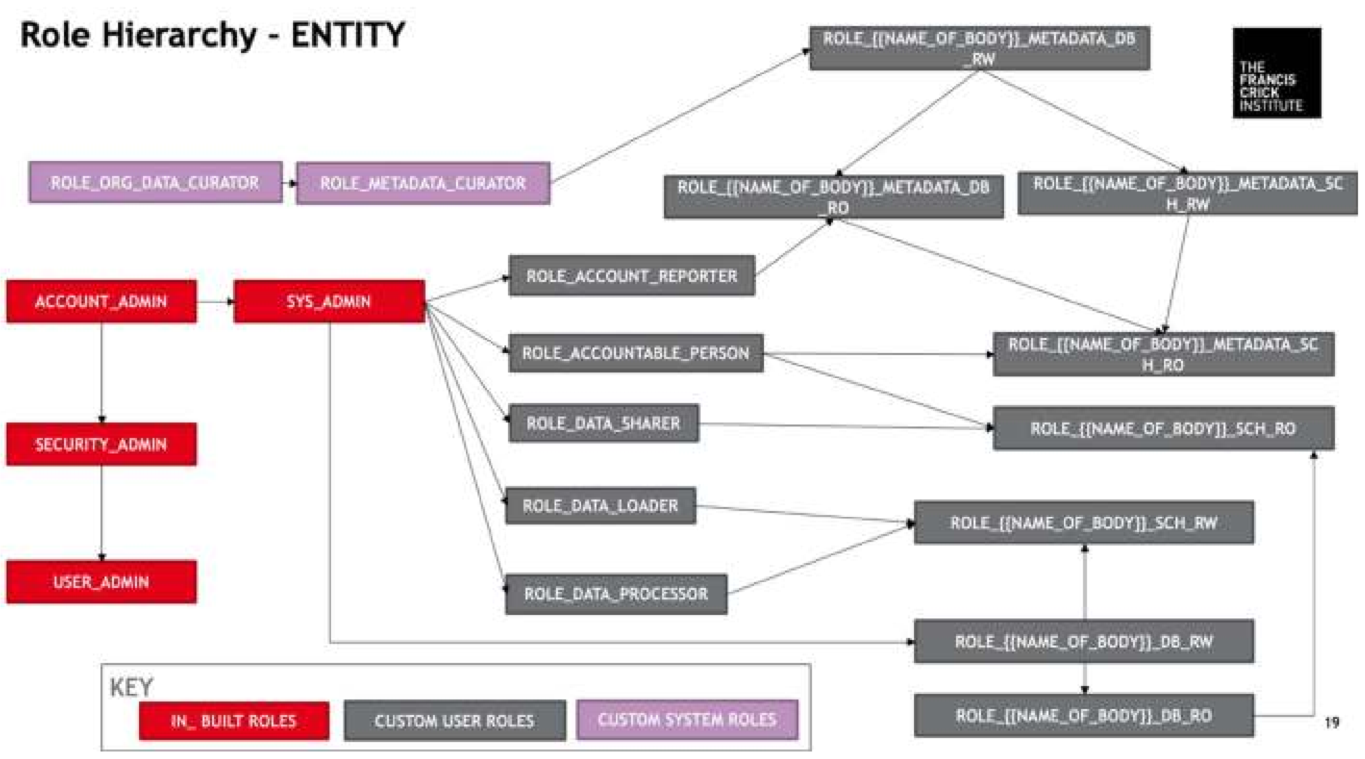
$_2$ *FrancisCrickInstitute/TRE_off_snowflake_computing/templates/04_entity_subaccount.j2*

$_3$*FrancisCrickInstitute/TRE_off_snowflake_computing/templates/06_collaboration_subaccount.j2*

The SQL commands will create the following objects depending on the type of account. For more info on the naming conventions and objects. *See section 2.5.* The below resources are for an account name "ACCOUNT_NAME"

STP, LAB & ENTITY

1) Resource Monitors for roles
    a. <<ACCOUNT_NAME>>_ACCOUNT_RM – Account Level Resource Monitor
    b. <<ACCOUNT_NAME>>_XS_DS_RM – Data Sharer Resource Monitor
    c. <<ACCOUNT_NAME>>_XS_DP_RM – Data Processor Resource Monitor
    d. <<ACCOUNT_NAME>>_XS_DL_RM – Data Loader Resource Monitor
    e. <<ACCOUNT_NAME>>_XS_AP_RM – Accountable Person Resource Monitor
2) 2 Databases
    a. <<ACCOUNT_NAME>>_DB
    b. METADATA_DB
3) 1 schema for each database
    a. <<ACCOUNT_NAME>>_SCH for <<ACCOUNT_NAME>>_DB
    b. METADATA_SCH for METADATA_DB
4) Warehouse
    a. CORE_WH
    b. <<ACCOUNT_NAME>>_XS_DL_WH – Warehouse for Data Loader
    c. <<ACCOUNT_NAME>>_XS_DP_WH – Warehouse for Data Processor
    d. <<ACCOUNT_NAME>>_XS_DS_WH – Warehouse for Data Sharer
    e. <<ACCOUNT_NAME>>_XS_AP_WH – Warehouse for Accountable Person
5) Internal Stage to store data
6) Grant access to metadata tables etc. to for real time updates
7) Create roles for subaccount
    a. ROLE_ACCOUNT_REPORTER
    b. ROLE_METADATA_CURATOR
    c. ROLE_ORG_DATA_CURATOR
    d. ROLE_DATA_SHARER
    e. ROLE_DATA_LOADER
    f. ROLE_DATA_PROCESSOR
    g. ROLE_ACCOUNTABLE_PERSON
8) Create share privileges
9) Grant roles to top level roles see Figure 14 below to see inherited roles for collaboration



Role Hierarchy - ENTITY
THE FRANCIS CRICK INSTITUTE
ROLE_{{NAME_OF_BODY}}_METADATA_DB_RW
ROLE_ORG_DATA_CURATOR
ROLE_METADATA_CURATOR
ROLE_{{NAME_OF_BODY}}_METADATA_DB_RO
ROLE_{{NAME_OF_BODY}}_METADATA_SCH_RW
ACCOUNT_ADMIN
SYS_ADMIN
ROLE_ACCOUNT_REPORTER
ROLE_ACCOUNTABLE_PERSON
ROLE_{{NAME_OF_BODY}}_METADATA_SCH_RO
SECURITY_ADMIN
ROLE_DATA_SHARER
ROLE_{{NAME_OF_BODY}}_SCH_RO
ROLE_DATA_LOADER
ROLE_{{NAME_OF_BODY}}_SCH_RW
USER_ADMIN
ROLE_DATA_PROCESSOR
ROLE_{{NAME_OF_BODY}}_DB_RW
KEY
IN_ BUILT ROLES
CUSTOM USER ROLES
CUSTOM SYSTEM ROLES
ROLE_{{NAME_OF_BODY}}_DB_RO
19

Figure 14: Role Hierarchy for entity

See Section 2 and HLD for more information on Roles & Naming convention. The below resources are for an account name "ACCOUNT_NAME"

COLLABORATION

1) Resource Monitors
   a. <<ACCOUNT_NAME>>_ACCOUNT_RM
   b. <<ACCOUNT_NAME>>_XS_BM_RM
   c. <<ACCOUNT_NAME>>_XS_DC_RM
2) 2 Databases
   a. <<ACCOUNT_NAME>>_SCIENCE_DB
   b. METADATA_ DB
3) 1 schema for METADATA DB
   a. METADATA_SCH for metadata DB
4) Warehouse
   a. CORE_WH
   b. METADATA_WH
   c. MAIN_WH
   d. <<ACCOUNT_NAME>>_XS_BM_WH
   e. <<ACCOUNT_NAME>>_XS_DC_WH
5) Grant access to metadata tables etc. to for real time updates
6) Create roles for subaccount
   a. ROLE_ACCOUNT_REPORTER

b. ROLE_METADATA_CURATOR
c. ROLE_DATA_CURATOR
d. ROLE_ORG_DATA_CURATOR
e. ROLE_BOARD_MEMBER
f. ROLE_ACCOUNT_REPORTER
g. ROLE_SHARE_ADMIN

7) Create share privileges
   a. Assigns ROLE_ORG_DATA_CURATOR with create share privileges
8) Grant roles to top level roles see Figure 6 below to see inherited roles for collaboration

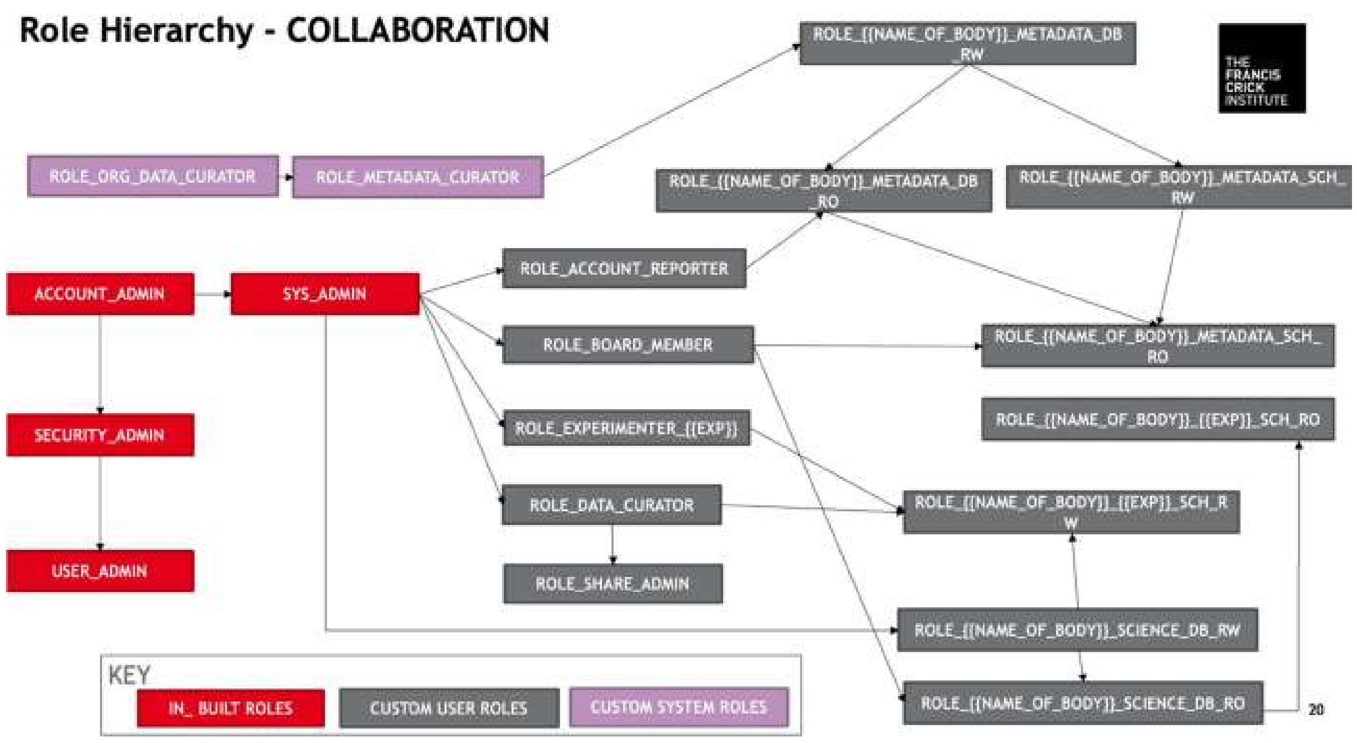

Role Hierarchy - COLLABORATION
THE FRANCIS CRICK INSTITUTE
ROLE_{{NAME_OF_BODY}}_METADATA_DB_RW
ROLE_ORG_DATA_CURATOR
ROLE_METADATA_CURATOR
ROLE_{{NAME_OF_BODY}}_METADATA_DB_RO
ROLE_{{NAME_OF_BODY}}_METADATA_SCH_RW
ACCOUNT_ADMIN
SYS_ADMIN
ROLE_ACCOUNT_REPORTER
ROLE_BOARD_MEMBER
ROLE_{{NAME_OF_BODY}}_METADATA_SCH_RO
ROLE_{{NAME_OF_BODY}}_{{EXP}}_SCH_RO
SECURITY_ADMIN
ROLE_EXPERIMENTER_{{EXP}}
ROLE_DATA_CURATOR
ROLE_{{NAME_OF_BODY}}_{{EXP}}_SCH_RW
USER_ADMIN
ROLE_SHARE_ADMIN
ROLE_{{NAME_OF_BODY}}_SCIENCE_DB_RW
ROLE_{{NAME_OF_BODY}}_SCIENCE_DB_RO
KEY
IN_ BUILT ROLES
CUSTOM USER ROLES
CUSTOM SYSTEM ROLES
20

Figure 15: Role Hierarchy for collaboration

See Section 2 for more information on Roles & Naming convention.

4.3.4. New Experiment for Collaboration

The input values required are:

ACCOUNT_NAME

NAME_OF_EXPERIMENT

CONTRACT_PERIOD

START_TIMESTAMP

END_TIMESTAMP

NOTIFY_PERCENT

CREDITS_COLLAB_EXPERIMENTER

Once the new experiment json output from ServiceNow *(see Section 3.5)* is dropped in the S3 bucket setup_new_experiment. It will trigger the lambda function setup_experiment.[1] This will create a new experiment in a pre-chosen collaboration account, based on the input from ServiceNow. The template 12_new_experiments[2] is transformed using flows applies to run SQL code in pre-selected Snowflake account.

For the experiment the objects created are :

1. Resource Monitor for experimenter for experiment - <<ACCOUNT_NAME>>_<<NAME_OF_EXPERIMENT>>_XS_EX_RM
2. Warehouse for experimenter for experiment <<ACCOUNT_NAME>>_<<NAME_OF_EXPERIMENT>>_XS_EX_WH
3. Schema for experiment in <<ACCOUNT_NAME>>_SCIENCE_DB <<NAME_OF_EXPERIMENT>>_SCH
4. Create Role for experiment ROLE_EXPERIMENTER_{{NAME_OF_EXPERIMENT}}
5. Grant Role ROLE_<<NAME_OF_EXPERIMENT>>_SCH_RW to ROLE_EXPERIMENTER_<<NAME_OF_EXPERIMENT>> & ROLE_DATA_CURATOR
6. Grant access to metadata tables etc. to for real time updates

Github code:

[1] *FrancisCrickInstitute/TRE_off_snowflake_computing/lambda/setup_experiment/lambda.py*

[2] *FrancisCrickInstitute/TRE_off_snowflake_computing/templates/12_new_experiment.j2*

4.3.5. Create User

Once the user json from ServiceNow dropped in the S3 bucket tre_setup_user, it will trigger the lambda function setup_user[1]

The input values required are:

ACTION

ACCOUNT_NAME

LASTNAME

FIRSTNAME

CONSORTIUM_ID

ROLE_NAME

EMAIL

OKTA_ENABLED

LOGIN

REGION

There is a field in this file called ACTION. This can be one of 4 options; ADD_USER, DELETE_USER, ADD_ROLE, or REMOVE_ROLE. Based on action type, it will generate either 08_new_users or 09_new_role.

Action TYPE	Template generated	Tasks used
ADD_USER	08_new_users[2]	Create_users.j2 [4]
DELETE_USER	08_new_users[2]	Drop_users.j2[5]
ADD_ROLE	09_new_role[3]	Create_roles.j2 [6]
REMOVE_ROLE	09_new_role[3]	Revoke_roles.j2 [7]

ADD_USER:

Generates a new user in the chosen subaccount using the field <<ACCOUNT_NAME>> using the template 08__new_users[2]. It will generate a temporary password for the new user within the lambda function. Login into the entity/collaboration using the ACCOUNTADMIN_MAIN password. It will create a new user, with the details given in the input file. The only difference for each account is whether it is OKTA_ENABLED. This will be true for all crick employees (*See Section 4.4)*. It sends an email to the user with login details and the account link. In a separate email, it sends the temporary password, this needs to be changed within a certain time frame.

DELETE_USER:

Generates using the template 08_new_users[2] in comparison to the above one it reverts the code and removes user. It sends and email to the "accountable person/board member" with username, and account that this has been actioned on.

ADD_ROLE:

Generates using template 09_new_role[3] for existing users within the chosen subaccount. It will grant an additional role specified in the file. Roles are restricted base on the type of account (see Petes doc). It sends and email to "accountable person/board member" with role, username, and account that this has been actioned on.

REMOVE_ROLE:

Generates using 09_new_role[3] for existing users within the chosen subaccount. In comparison to the above one it reverts the code and removes the role from the user. An email is sent to "accountable person/board member" with role, username, and account that this has been actioned on.

Github code

1 FrancisCrickInstitute/TRE_off_snowflake_computing/lambda/create_user/lambda.py

2 FrancisCrickInstitute/TRE_off_snowflake_computing/templates/08_new_users.j2

3 FrancisCrickInstitute/TRE_off_snowflake_computing/templates/09_new_role.j2

4 FrancisCrickInstitute/TRE_off_snowflake_computing/tasks/09_users/create_user.j2

5 FrancisCrickInstitute/TRE_off_snowflake_computing/tasks/09_users/drop_user.j2

6 FrancisCrickInstitute/TRE_off_snowflake_computing/tasks/08custom_roles/revoke_role.j2

7 FrancisCrickInstitute/TRE_off_snowflake_computing/tasks/08_custom_roles /create_roles.j2

4.4. Okta integration with Snowflake

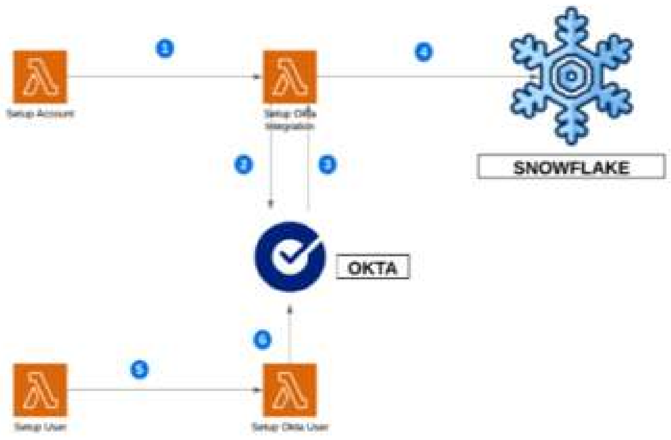


Figure 16: Lambda function flow for OKTA integration

1. After successful setup of Lab, STP or Collaboration account, lambda function Setup Okta Integration is triggered
2. Lambda creates new Okta SAML application, new Okta group, and assign the app to the group
3. Lambda queries saml2 issuer, SSO URL and x509 certificate from the new Okta app
4. Using saml2 issuer, SSO URL and x509 certificate, lambda creates Security integration in the Snowflake account
5. After successful setup of new user (or removal of existing user) in Snowflake account, lambda function Setup okta user is triggered
6. Lambda adds (or removes) user from the okta group, thus enabling (disabling) user's access to the corresponding Snowflake account via Okta

GitHub Code

1 FrancisCrickInstitute/TRE_off_snowflake_computing/lambda/setup_okta_integration/lambda.py

2 FrancisCrickInstitute/TRE_off_snowflake_computing/templates/setup_okta_user.j2

4.5. Metadata Processing

There are 3 types of Snowflake accounts we use for metadata processing:

Organisation account: This account is the main organisation account, as mentioned above this is the account to create all accounts. It contains **organisation_usage** data in SNOWFLAKE database.

Snowflake documentation: https://docs.snowflake.com/en/sql-reference/organization-usage.html

Organisation admin: This account contains the reporting account with databases named METADATA_DB and {ACCOUNT_NAME}_IN_SHARE_DB. The inshare database is replicated into the account from the main account. It contains *Views* for audit reporting (visualized in Power BI) and custom *tables* from Service Now.

The URL for current Crick platform:

https://wjljpih-org_admin.snowflakecomputing.com/

Sub-accounts: These are collaborations, entities accounts created within a consortium. They are environment for data analysis tools and scientific research. These accounts are mentioned above.

4.5.1. Data collected and details

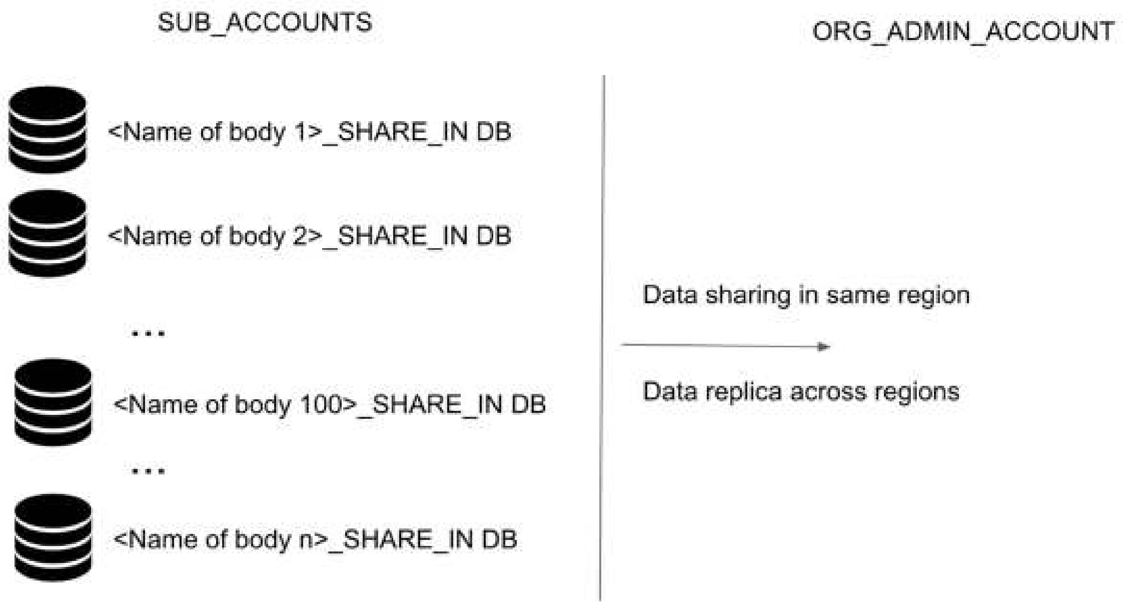

SUB_ACCOUNTS
ORG_ADMIN_ACCOUNT
<Name of body 1>_SHARE_IN DB
<Name of body 2>_SHARE_IN DB
...
<Name of body 100>_SHARE_IN DB
...
<Name of body n>_SHARE_IN DB
Data sharing in same region
Data replica across regions

Figure 4.5.1: Overall Architecture

4.5.2. Account usage tables

Here is data we can see for each SNOWFLAKE account in `SNOWFLAKE.ACCOUNT_USAGE`.

Snowflake documentation: https://docs.snowflake.com/en/sql-reference/account-usage.html

With a set period (60 minutes for current platform), a SNOWFLAKE TASK is run to trigger stored procedure. This captures new changes from `SNOWFLAKE.ACCOUNT_USAGE` to `METADATA_DB.METADATA_SCH` in each account and marked with current time on field `UPDATED_AT`

```
- name: ACCESS_HISTORY
- name: AUTOMATIC_CLUSTERING_HISTORY
- name: COMPLETE_TASK_GRAPHS
- name: COPY_HISTORY
- name: DATABASES
- name: DATABASE_STORAGE_USAGE_HISTORY
- name: DATA_TRANSFER_HISTORY
- name: GRANTS_TO_ROLES
- name: GRANTS_TO_USERS
- name: LOAD_HISTORY
- name: LOGIN_HISTORY
- name: METERING_DAILY_HISTORY
- name: METERING_HISTORY
- name: PIPE_USAGE_HISTORY
- name: REPLICATION_USAGE_HISTORY
- name: ROLES
- name: SCHEMATA
- name: STAGES
- name: STAGE_STORAGE_USAGE_HISTORY
- name: TAGS
- name: TAG_REFERENCES
- name: USERS
- name: STORAGE_USAGE
- name: QUERY_HISTORY
- name: WAREHOUSE_METERING_HISTORY
- name: SESSIONS
- name: TASK_HISTORY
- name: COLUMNS
- name: TABLES
```

4.5.3. Information schemas table

Here is data we can see for each SNOWFLAKE account in `SNOWFLAKE.INFORMATION_SCHEMA`.

Snowflake Documentation: https://docs.snowflake.com/en/sql-reference/info-schema.html

With a set period (60 minutes for current platform), a SNOWFLAKE TASK is run to trigger stored procedure to capture new changes from `SNOWFLAKE.INFORMATION_SCHEMA` to `METADATA_DB.METADATA_SCH` in each account and marked with current time on field `UPDATED_AT`

```
- name: USAGE_PRIVILEGES
- name: TABLE_STORAGE_METRICS
- name: PROCEDURES
- name: OBJECT_PRIVILEGES
- name: TABLE_PRIVILEGES
- name: REPLICATION_DATABASES
```

3. Show commands tables:

Here is data we can see for each SNOWFLAKE account by running SQL query `SHOW {Objects}`

Snowflake Documentation: https://docs.snowflake.com/en/sql-reference/sql/show.html

With a set period (60 minutes for current platform), a SNOWFLAKE TASK is run to trigger stored procedure to capture new change to `METADATA_DB.METADATA_SCH` in each account and marked with current time on field `UPDATED_AT` . Since SHOW command only has live data, we have another field `IS_DELETED` to differentiate history.

```
- name: SHARES
  key_identifier: name
- name: WAREHOUSES
  key_identifier: name
- name: RESOURCE_MONITORS
  key_identifier: name
```

4. Live tables:

These are custom tables created for service now update. All these data is sent back to Service Now as reference for creating new forms.

`EXPERIMENT_LIVE`

This table is created during set up account as empty. An INSERT SQL script is trigger when a new experiment is created.

`ORGANIZATION_ACCOUNT_MASTERSHARE_DB.ORG_SCH.UPDATE_ACCOUNT_RECORD`

This table is used to keep track live consortium_id with account_locator, account_name. Once an account is created, a stored procedure in organization account is trigger.

`ORGANIZATION_ACCOUNTS_LIVE`

This table shared to org admin account from organization accounts. By running `SHOW ORGANZATION ACCOUNTS` with ORGADMIN role, we have data of existing accounts being used in platform. We also know when it is created, url, locator …

`GRANTS_TO_USER_LIVE`

This table is managed by a javascript stored procedure. It has live data of USERS and ROLES in a sub account. Once there is a user or role modified, the lambda function is trigger to update this table.

`WAREHOUSES_LIVE`

Keep track on current warehouses in platform.

4.5.4. Organization data sharing:

A special replica database from organization account to include organization usage data. For current platform database and schema is: `ORGANIZATION_ACCOUNT_MASTERSHARE_DB.ORG_SCH`

4.5.5. Views in ORG_ADMIN_DB

Metadata views having convention: `{objects_type}_VIEW`. For example: `GRANTS_TO_USERS_LIVE`

The live tables are updated straight away. All other SNOWFLAKE account usage or information schema tables have expected 0-120 minutes delay on update.

There is stored procedure to automatically update a view with + `UNION ALL SELECT * FROM {new_sub_account}_SHARE_IN_DB` when an account is created.

4.5.6. ADMIN_SCH.X_TFCI_SNOWFLAKE_ tables

Custom data from Service Now on active CONSORTIUM, USERS, EXPERIMENTS, WAREHOUSES, BUDGET, ACCOUNTS.

4.6. Update ServiceNow Records and Snowflake Metadata Tables

To allow consistency throughout the process, there is an API in place to connect the metadata for the TRE in the two locations, the data is stored. ServiceNow generates some of the information and so does Snowflake during the process. After, any of the lambda function have been run it will update the metadata records to have an up-to-date view on all accounts, users, warehouse, experiments & tables within the system. See Figure ?? below.

The API is built into ServiceNow. The API pushes data to snowflake ORG_ADMIN account and then pull data from ServiceNow. The tables mentioned in Section 3.7 are some of the data that is moved.

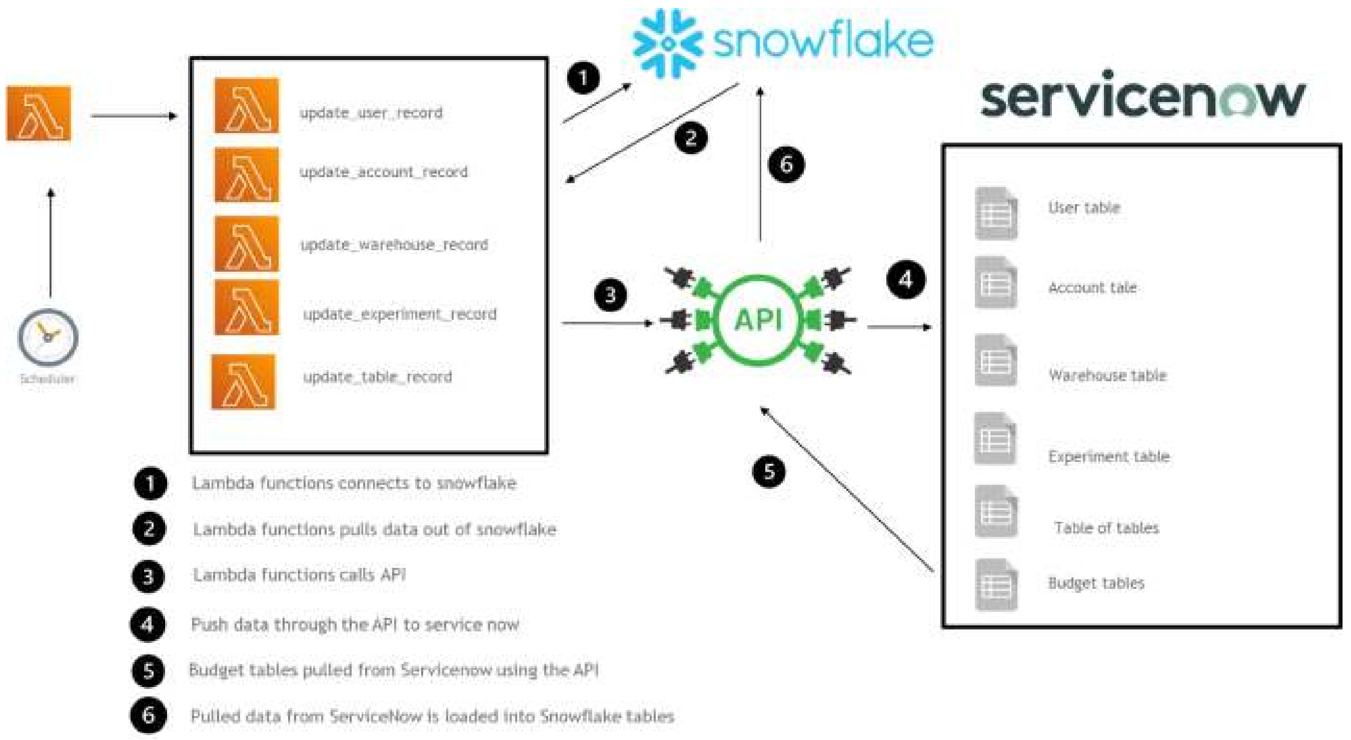

Scheduler
update_user_record
update_account_record
update_warehouse_record
update_experiment_record
update_table_record
snowflake
servicenow
API
User table
Account tale
Warehouse table
Experiment table
Table of tables
Budget tables
1 Lambda functions connects to snowflake
2 Lambda functions pulls data out of snowflake
3 Lambda functions calls API
4 Push data through the API to service now
5 Budget tables pulled from Servicenow using the API
6 Pulled data from ServiceNow is loaded into Snowflake tables

Figure 4.6.1: ServiceNow to Snowflake integration using API

4.7. Archive Process

Once the collaboration project ends, there needs to be an efficient and safe way to remove the data from the account, and return it to the relevant data owners, as specified in the research collaboration agreement. If we were to suspend and delete the account, all data would be lost. The alternative is to continue paying for the storage of the data indefinitely.

To initiate an archive process, the user is required to fill in an archive form in the ServiceNow Portal. Once this has been completed, there is a field which states when the archive needs to happen and what objects are required to be taken off the platform. The following steps will then be triggered, after approval by accountable person.

- There is an email 5 days before the archive sent out to accountable person informing all work needs to be stopped on the account. (Email sent from service now).
- Another email is sent out 2 days before archive informing the process will begin the following day. The file is then dropped in an S3 bucket named tre_archive. This will trigger lambda function archive$_1$. The lambda function builds a file from 14_archive.j2$_2$ using the input variables from the json.

This lambda function will check if any queries are running on the account (manual or automatic).

If no, it will perform the following tasks:

1. Create internal stage for the data to place into
2. Create an archive role
3. Create an archive resource monitor with the allocation of credits based on the size of the archive.
4. Create a warehouse (compute) for the role archive.
5. Assign a user or service account to the ROLE_ARCHIVE
6. Suspend all warehouses so no queries can be run

If yes:

- The process cannot prepare the account for archive and will send out an email to accountable person & user running query, stating “Archive cannot complete while queries are running”. The process will need to be triggered again.

After this has completed, we are ready to archive data. There will be two choices, manual or automatic.

- Manual - the administrator would assign the role_archive to a specific user to take the data from an internal stage to an external stage using snowflake in built tools or one of their internal tools.
- Automatic – the archive process would take all data from the tables chosen in the archive form and move into an internal stage ready for the user to take off.

Once the data has been removed, we can either resume the warehouses if the account is still going to be used, or the account is finished with, we will suspend all compute resources after archive has completed. Finally, the admin will submit a request to Snowflake to remove the account from the organisation.

All metadata for each account is not archived and kept within the Crick admin account.

GitHub Links

1 *FrancisCrickInstitute/TRE_off_snowflake_computing/lambda/archive/lambda.py*

2 *FrancisCrickInstitute/TRE_off_snowflake_computing/templates/14_archive.j2*

5. Section 5: Snowflake Single Sign-On (SSO)

5.1. Overview

Snowflake TRE users are authenticated via Crick's Okta* SSO solution if they are internal staff. External TRE members are authenticated directly in Snowflake.

Single Sign-On provisioning in Snowflake TRE sub accounts is automated. Data which has been stored from the ServiceNow form for a new TRE is used to create an Okta SAML application, Okta groups to manage assignment and configure the Snowflake sub account to work with the Okta SAML SSO application. The Okta API is used to create the Okta elements see Section 4.4 for more detail and code. External users are created locally in Snowflake and authenticated locally by Snowflake.

5.2. Implementation

5.2.1. Okta Application and Group naming conventions

5.2.1.1. Application Naming

For the app, we use "Snowflake" and then a hyphen and then the project name, so for example the LAB TRE would be "Snowflake – LAB".

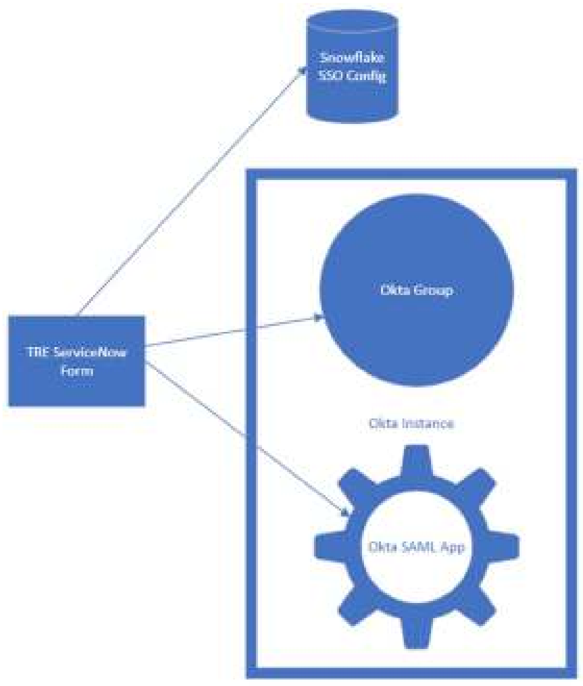
5.2.1.2. Group Naming

For the group naming we have used the format snowflake_TRE_NAME, so LAB for example, we have the group snowflake_LAB.

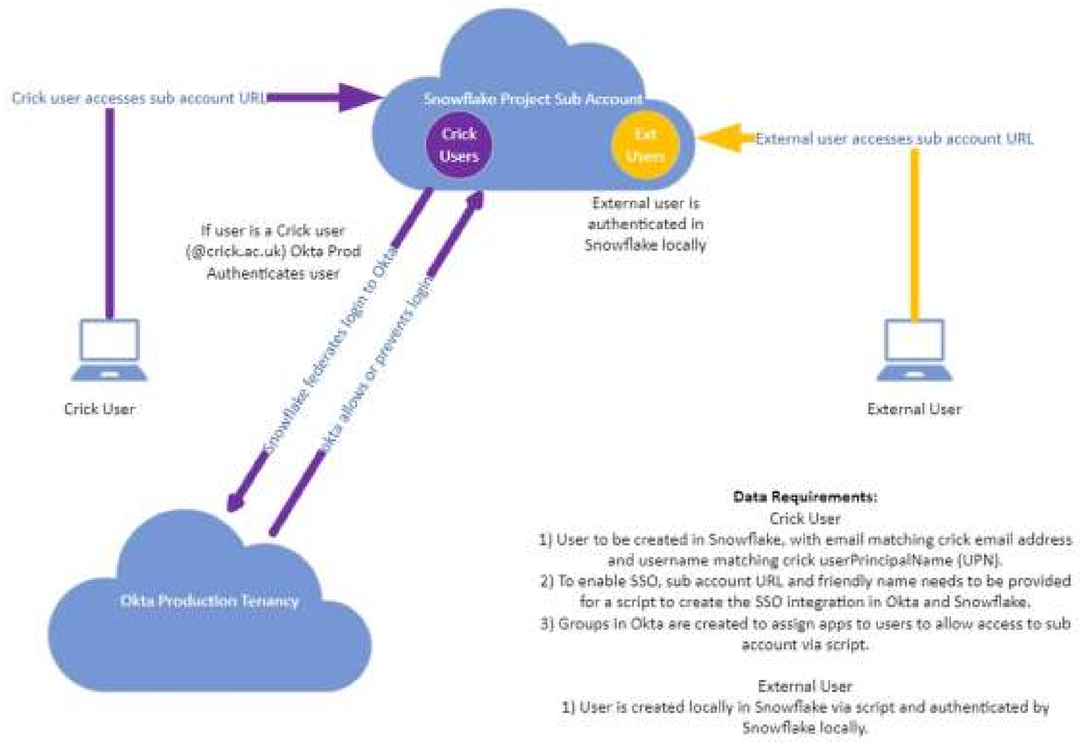
5.3. Process

1) TRE data from ServiceNow form used to create:
    a. Okta SAML App
    b. Okta Group for assignment
    c. Snowflake TRE sub account SSO configuration
2) Crick users added to Okta group for SSO access to TRE
3) External users added to Snowflake TRE and access managed by Snowflake directly

5.3.1. Automated object creation diagram



5.3.2. Overview SSO Flow Diagram


Crick user accesses sub account URL
Snowflake Project Sub Account
Crick Users
Ext Users
External user accesses sub account URL
External user is authenticated in Snowflake locally
If user is a Crick user (@crick.ac.uk) Okta Prod Authenticates user
Snowflake federates login to Okta
Okta allows or prevents login
Crick User
External User
Okta Production Tenancy

Data Requirements:
Crick User
1) User to be created in Snowflake, with email matching crick email address and username matching crick userPrincipalName (UPN).
2) To enable SSO, sub account URL and friendly name needs to be provided for a script to create the SSO integration in Okta and Snowflake.
3) Groups in Okta are created to assign apps to users to allow access to sub account via script.

External User
1) User is created locally in Snowflake via script and authenticated by Snowflake locally.

*Where Okta is noted, this could be any Identity Provider, such as AzureAD for example.

6. Section 6: Reporting Single Sign-On (SSO) and assignment

6.1. Overview

Reports from Snowflake TREs are visualised through PowerBI. PowerBI is already available to Crick staff and authenticated via federated SSO through Okta*. Users External to the Crick can access PowerBI via Microsoft's federated authentication, which allows other organisations using Office 365 or consumer Microsoft accounts to authenticate to our PowerBI instance.

6.2. Implementation

To manage access to the report element of Snowflake we have created two groups in AzureAD, one for internal staff and one for external staff. These groups do not manage data access within the report, but purely if the report can be accessed in the first place. Data level access is managed within the report itself by username.

6.2.1. Groups Created

Snowflake_PowerBI_EXTERNAL_USER_Read_Only

Snowflake_PowerBI_INTERNAL_USER_Read_Only

6.3. Process

This is how a TRE user with access to a report would be provided the access:

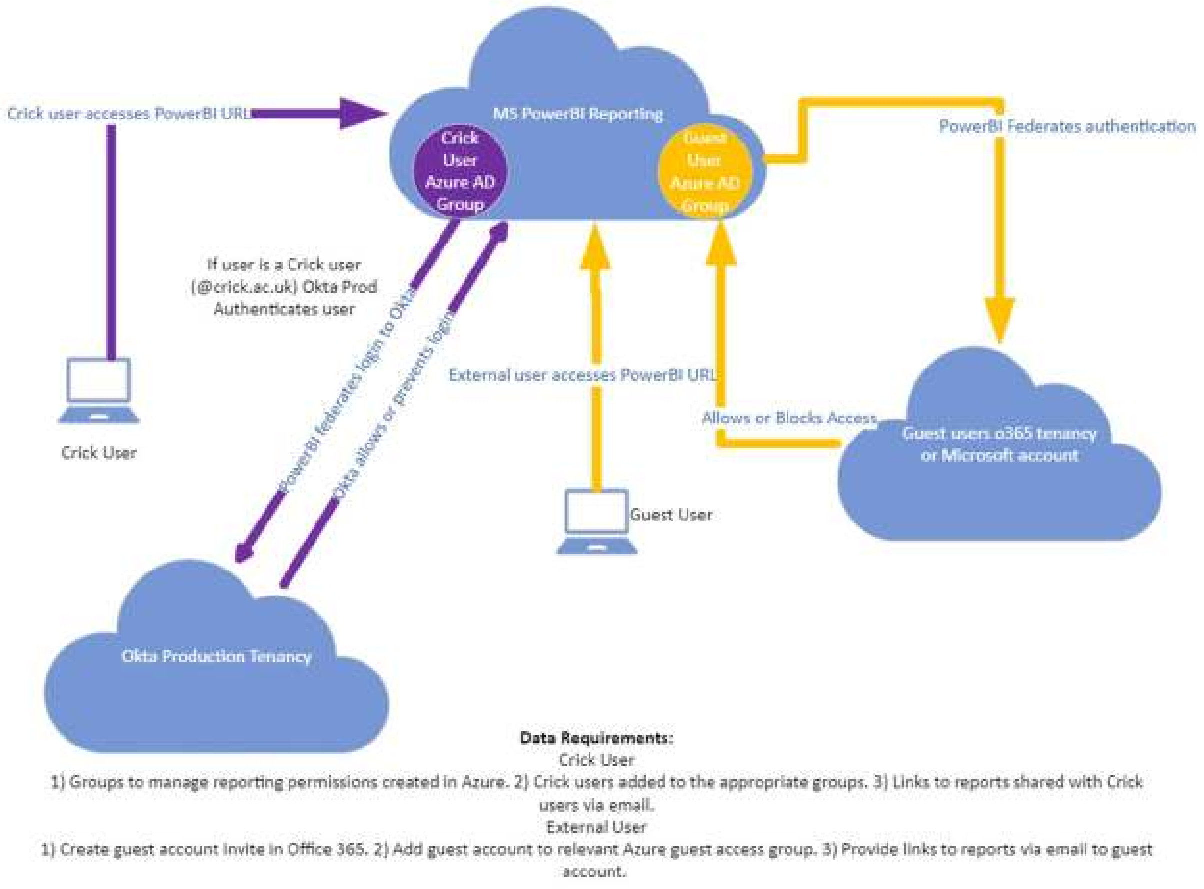
6.3.1. Internal User

1) Internal user in TRE is allowed report access
2) Internal user is added to AzureAD Group Snowflake_PowerBI_INTERNAL_USER_Read_Only
3) Internal user is emailed link to the PowerBI Report

6.3.2. External User

1) External user in TRE is allowed report access
2) External user is invited to Crick Azure AD using their email address noted in the TRE
3) External user accepts invitation to Crick Azure AD
4) External user is added to AzureAD Group Snowflake_PowerBI_EXTERNAL_USER_Read_Only
5) External user is emailed link to the PowerBI Report

6.4. Reporting SSO Overview Diagram


Crick user accesses PowerBI URL
MS PowerBI Reporting
Crick User Azure AD Group
Guest User Azure AD Group
PowerBI Federates authentication
If user is a Crick user (@crick.ac.uk) Okta Prod Authenticates user
PowerBI federates login to Okta
Okta allows or prevents login
External user accesses PowerBI URL
Allows or Blocks Access
Guest users o365 tenancy or Microsoft account
Crick User
Guest User
Okta Production Tenancy

Data Requirements:
Crick User
1) Groups to manage reporting permissions created in Azure. 2) Crick users added to the appropriate groups. 3) Links to reports shared with Crick users via email.
External User
1) Create guest account invite in Office 365. 2) Add guest account to relevant Azure guest access group. 3) Provide links to reports via email to guest account.

*Where Okta is noted, this could be any Identity Provider, such as AzureAD for example.

7. Section 7: Loading data into vTREs - Extract, Load, Transform (ELT) Tools

7.1. Performance Testing of ELT tools

This section provides an overview of the ways one could approach loading data into Snowflake, via a manual tool or via an automated workflow manager, and the methods used to evaluate them.

We looked to understand performance of loading data from a set of sources, namely:

- An S3 Bucket resource (linked to Snowflake via an external stage)
- Local storage (likely via an internal stage)

The Snowflake recommended (and the method found fastest in testing) is to have a file set up in a stage (usually compressed), and leveraging the COPY INTO command, load data from the staged file into a predefined table.

Database interface tools such DBeaver and DataGrip are also investigated here, and they make life very easy from a user perspective. However, since they have no way to stage files in Snowflake, they instead parse the file in question and make INSERT statements line by line (or in batches) into the table, which decreases performance very drastically and will not work efficiently for any larger files.

Here the performance testing was done with the following:

- A CSV data file of raw size 2.2GB and 0.2GB after compression (GZIP). About 30 million rows
- An AWS Bucket in ‘eu-west-2’ region, set up as an external stage in the snowflake instance
- An XS (extra small) snowflake compute warehouse

7.2. Loading files from a persistent storage source (via an internal stage):

Local files would usually be loaded into an internal snowflake stage before copying across to a table. If using Snowflake’s “snowsql“ CLI tool, the file is automatically converted to a GZIP format compressed file and loaded up into an internal stage (either “named” or “user” type).

```
PUT 'file://C:/[filepath]' @~/ PARALLEL = 20;
```

If a table has been set up with the correct column names as the csv file header, one can perform a COPY INTO command to load this data into the table.

```
COPY INTO PAFCSV_PERFORMANCETEST FROM @~/staged/PAF.csv.gz;
```

In testing, the PUT command took about 5 minutes (compression and upload) and the loading into the table from the internally staged compressed file took about 1.5 mins.

7.3. Loading files from a cloud file store (via an external stage):

We may come across a case where a team already has data in a cloud data store (e.g. Amazon S3, Google GCP, Azure Blob). In this case the external resource may be added to the snowflake instance via an external stage. The loading of files was done first from local storage to an S3 bucket that had been provisioned seperately. The upload of this file (compressed 0.2GB) took about 2.5 minutes. After this point the process is the same as for the internal stage, a COPY INTO command to get the file data in S3 into a snowflake table.

In testing, the COPY INTO command from externally staged compressed file took about 1.5 mins.

7.4. Loading files from Database clients (via bulk insert statements):

For self-service database client applications such as DBeaver (and DataGrip) , the approach they take to load data is slightly different. They do the heavy lifting to load data into the database, rather than go by a stage and letting snowflake parse the data file and copy the data into the table. Here the application reads some amount of rows at a time and inserts them via one SnowSQL transaction. This will naturally cause higher load times for large files. Because the loading takes a longer amount of time, this means the compute warehouse will need to be running for a lot longer and therefore use more credits so this is something to bare in mind. A credit breakdown across these methods will be summarised at the end of this document.

7.5. Manual Self-Serve Data Loading options

This section outlines the tools available for loading into a Snowflake account. Data is typically recommended to be loaded into Snowflake via stages. Stages are file repositories that act as a staging area for the contents of files to be moved to a Snowflake table. There exist both **external** and **internal** stages.

7.5.1. What is an external stage?

External stages are ones in which a Cloud file storage resource (outside of the Snowflake instance, hence external) is provisioned and deployed, and used as a Stage in Snowflake. Files can be loaded into this resource using the appropriate tool for that cloud provider. Currently supported Cloud file stores are AWS S3 Buckets, Azure Blob, and Google Cloud Platform files. Note that the service/user that loads data into the external resource will not necessarily be the same person that can access the stage in Snowflake. A user (or the infrastructure code that will set up the components) will be able to set up the named external stage in Snowflake (given they know the access keys for said storage account) at the schema level (although this may be opened to the rest of the database). Once this connection to the file store is made, the owner of that stage and any access they share to other users will be able to list the files currently inside the stage and copy that data across into tables. Note here that a user that has access to an external stage will not be able to PUT files into the stage, this needs to be done externally as part of the resource's native management tools.

7.5.2. What is an internal stage?

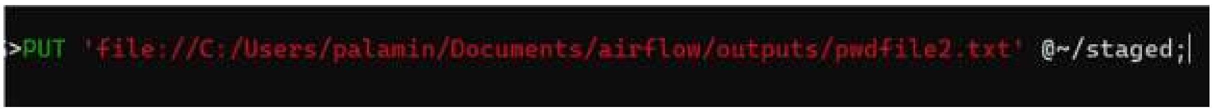
Internal stages are cloud file stores that are Snowflake managed. It is not a cloud resource that is referenceable or configurable outside of Snowflake and is an option that has limited configuration and deployment options. The advantage here is that Snowflake users can create named stages and transfer files from local file systems to these stages. These resources are fluid and very much operate under a pay as you go model as part of Snowflake's own billing. Once these files are put into the stage, they can be COPYed into a table

7.5.3. How may a user load data into a table via an internal stage?

By default, all users in Snowflake have what is referred to as a "user stage". This is a type of internal stage to be used by the user and only by the user and cannot be granted access to another user within the Snowflake instance. Being a default, no set up is needed to provision it and access is out of the box. If the stage is needing to be referenced by other users and be recognised in the design model as a database object, the user will need to set up a "named stage".

The Snowflake out of the box supported and official way to load data from a local file system to an internal stage is via "SnowSQL" and the command line interface (CLI) tool SnowSQL (CLI Client) — Snowflake Documentation , downloadable from the Snowflake website. Once installed and configured, the PUT command allows the user to reference a file path and load that file (and compress it in the process) into an internal stage (be it named or user).

For example, loading a file into my personal user stage:

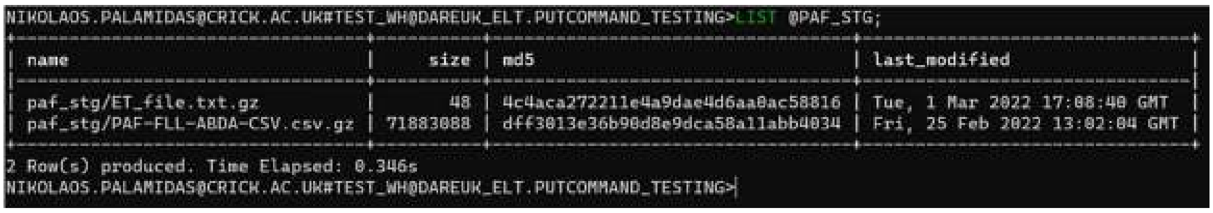


```
>PUT 'file://C:/Users/palamin/Documents/airflow/outputs/pwdfile2.txt' @~/staged;
```

PUTing a file into my personal user stage

Or alternatively you can create a named stage to be used by other users

```
CREATE STAGE PAF_STG;
PUT 'file://C:/Users/palamin/Documents/PAF-FLL-UK-CSV/PAF-FLL-ABDA-CSV.csv' @PAF_STG;
```



```
NIKOLAOS.PALAMIDAS@CRICK.AC.UK#TEST_WH@DAREUK_ELT.PUTCOMMAND_TESTING>LIST @PAF_STG;
+---------------------------------+----------+----------------------------------+-------------------------------+
| name                            |     size | md5                              | last_modified                 |
|---------------------------------+----------+----------------------------------+-------------------------------|
| paf_stg/ET_file.txt.gz          |       48 | 4c4aca272211e4a9dae4d6aa0ac58816 | Tue, 1 Mar 2022 17:08:40 GMT  |
| paf_stg/PAF-FLL-ABDA-CSV.csv.gz | 71883088 | dff3013e36b90d8e9dca58a11abb4034 | Fri, 25 Feb 2022 13:02:04 GMT |
+---------------------------------+----------+----------------------------------+-------------------------------+
2 Row(s) produced. Time Elapsed: 0.346s
NIKOLAOS.PALAMIDAS@CRICK.AC.UK#TEST_WH@DAREUK_ELT.PUTCOMMAND_TESTING>
```

The contents of stages can be displayed with the LIST command

Then the COPY command may be used in SnowSQL to copy the data across into a pre-built table. For example loading data from our @PAF_STG named stage into a pre built table. Notes this does not need to be done in the CLI and may be done via the in-browser snowflake worksheet interface.

```
>COPY INTO POSTCODE_PAF_TEST FROM @PAF_STG FILE_FORMAT=(VALIDATE_UTF8 = TRUE);
```

The table to land the data must be created beforehand for this to work, with the same field names and types pre-configured. This of course requires knowledge on the user's part, not only to understand Snowflake's data types and table creation SQL syntax, but also to know ahead of time what the data structure of the file is. This can be difficult for a non-technical individual to load their data into snowflake.

We can still leverage these methods outlined above, but what we ideally need is a way to streamline this process:

A graphical user interface (GUI) for the tool so that anyone can load data that needs to do so.

A "schema-on-read" approach to loading data into the tables. What is meant here is a way to automatically create the table with the correct names and data types inferred by the contents of the file without needing to be investigated by individuals (who may not feel comfortable doing so).

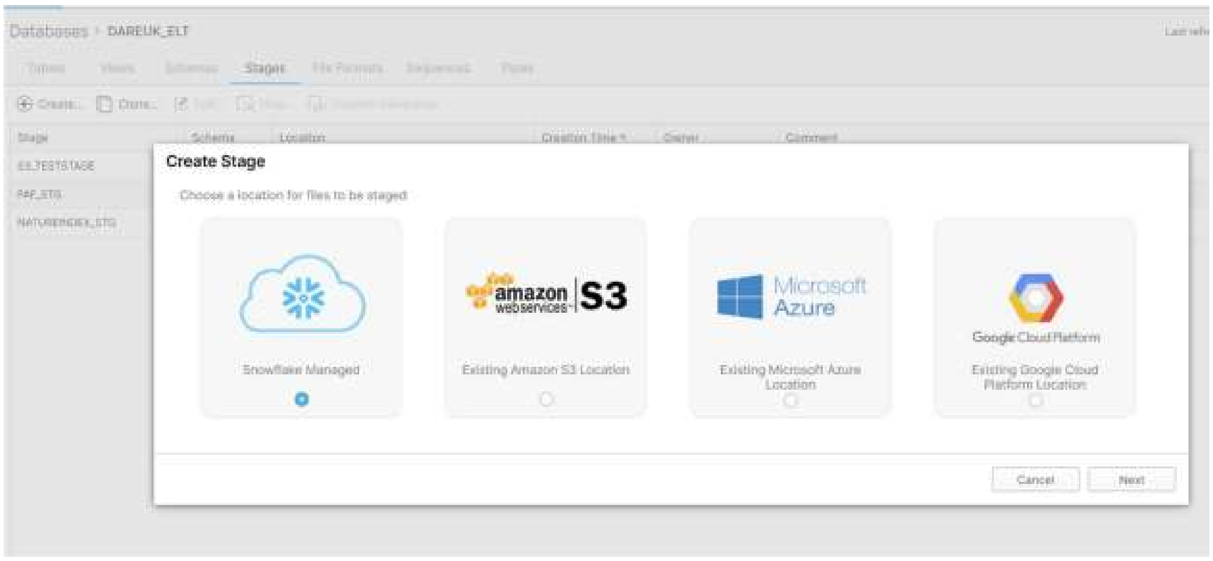
7.5.4. How may a user load data into a table via an external stage?

The external stage approach is different use case to the internal stage. It is more applicable to a set of people who already have a prebuilt cloud-based file storage resource at their disposal or wish to have one. Take for example an external body that already has an AWS S3 bucket that stores genomics data. All that would be required is for the user to set up an external stage referencing the location of the bucket to copy data across to tables, without the need to replicate data across different storage resources.

Another use case would be the need for a variety of users, teams, and services to load data into the data repository that are not necessarily part of the Snowflake instance identity management.

For AWS S3, a file can be loaded into the bucket in a variety of ways. An automated data flow or via their own AWS CLI tool. In testing we loaded using the CLI tool.

To set up the bucket (or other supported) cloud-based storage as an external stage, you can do so graphically in the browser interface portal or via SQL commands.

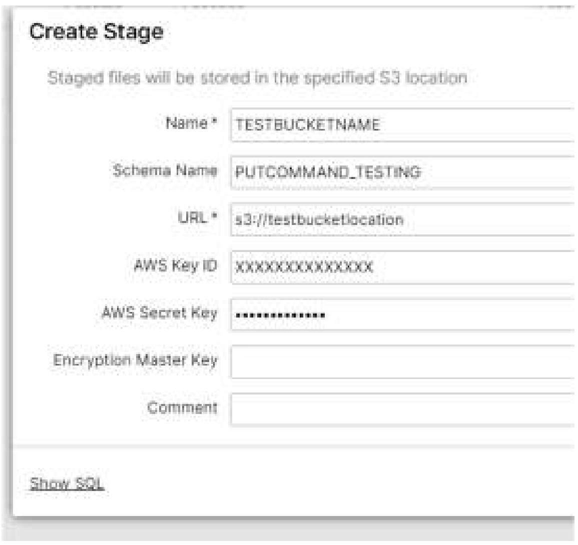


Create Stage

Choose a location for files to be staged

Databases > DAREUK_ELT
Stages
Snowflake Managed
Existing Amazon S3 Location
Existing Microsoft Azure Location
Existing Google Cloud Platform Location
Cancel
Next

Figure 7.5.4.1: Stage Creation Options for Snowflake Managed (internal) and Cloud resources (external)



Create Stage

Staged files will be stored in the specified S3 location

Name *	TESTBUCKETNAME
Schema Name	PUTCOMMAND_TESTING
URL *	s3://testbucketlocation
AWS Key ID	XXXXXXXXXXXXXX
AWS Secret Key	••••••••••••••
Encryption Master Key	
Comment	

Figure 7.5.4.2: Graphical external stage creation

```
--Or Create the stage via SQL which gives you more control of parameters
CREATE STAGE "DAREUK_ELT"."PUTCOMMAND_TESTING".TESTBUCKETNAME
URL = 's3://testbucketlocation'
CREDENTIALS = (AWS_KEY_ID = 'XXXXXXXXXXXXXX' AWS_SECRET_KEY = '*************');
```

Once the external stage is set up there is no difference to how you would use an internal stage, using COPY to copy data into the table from a file.

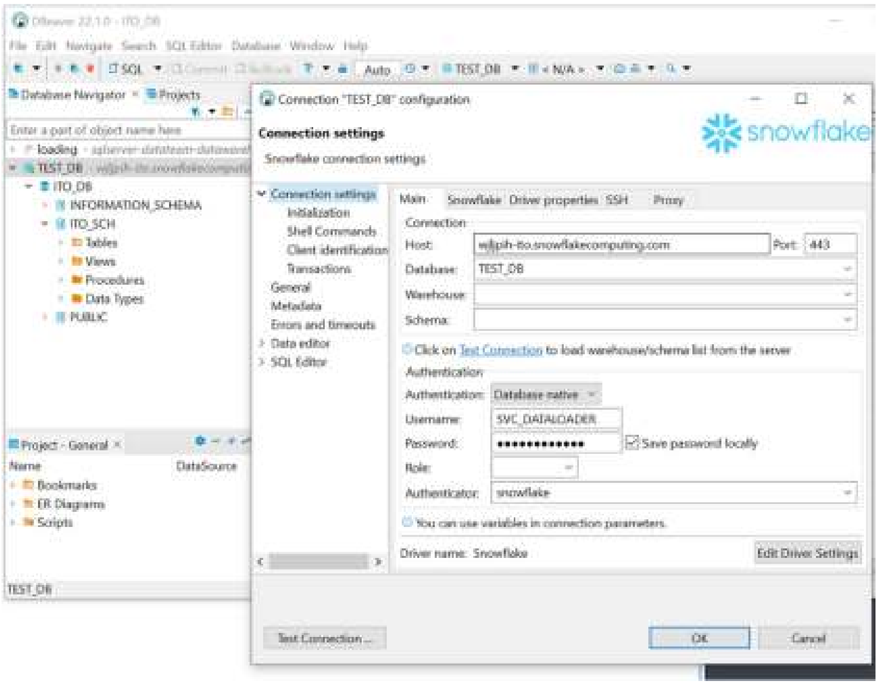
7.5.5. Are there any pre-built software options for loading data graphically?

For many users, the use of the CLI tool and its setup is too technical and fiddly to use. People should have the ability to load data into their Snowflake instance, without knowledge of stages, SnowSQL syntax, data types etc. For this reason we have found a few graphical tools that fulfil this purpose, namely DBeaver and JetBrain's DataGrip.

DBeaver is an open-source and free to use multi-platform database tool (designed mainly for developers but can be used by anyone with the right guidance), that provide an easy-to-use graphical interface for loading data into databases/lakes/warehouses etc. This includes Snowflake out of the box (will manage the drivers necessary with minimal fiddling).

DataGrip is an example of another such database tool, however this required a paid for licence and does not offer any additional functionality to DBeaver. It was found in testing that the free DBeaver community edition licence provided more than enough functionality for this use case.

Once the connection to the appropriate Snowflake subaccount, database, warehouse and schema are configured, a graphical process will guide the user through loading data. The tool reads the raw file (e.g. csv, JSON etc.) and interprets the schema from the contents. This will allow the user to either accept the format, or to adjust according to custom data type and restrictions. Once this is done, the tool builds that table, without needing knowledge of the underlying SQL (although this is visible should you wish to inspect it), with the data types needed, and then loads that data straight from file to table. This method significantly reduces complexity for the user by both inferring data file schema through a graphical interface, and bypassing the requirement to set up and load into stages as an intermediary step.


Connection "TEST_DB" configuration

Connection settings

Snowflake connection settings

Connection

Host: wjljpih-ito.snowflakecomputing.com Port: 443

Database: TEST_DB

Warehouse:

Schema:

Click on Test Connection to load warehouse/schema list from the server

Authentication

Authentication: Database native

Username: SVC_DATALOADER

Password: ••••••••••••• ☑ Save password locally

Role:

Authenticator: snowflake

You can use variables in connection parameters.

Driver name: Snowflake

Figure 7.5.5.1: Setting up the DBeaver connection manager

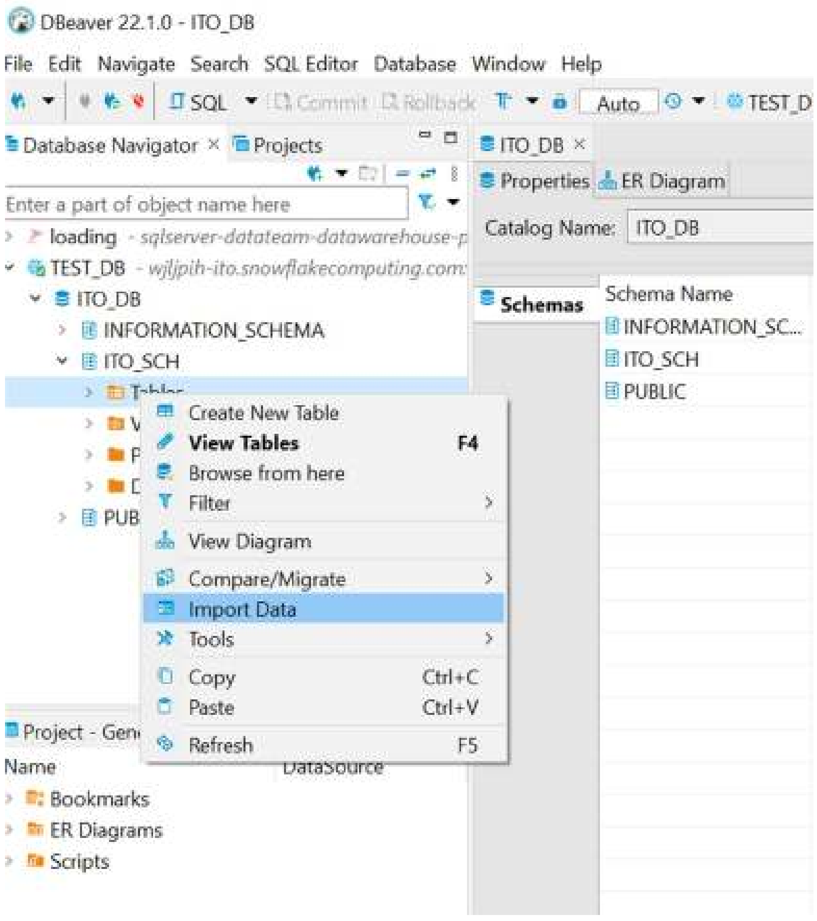
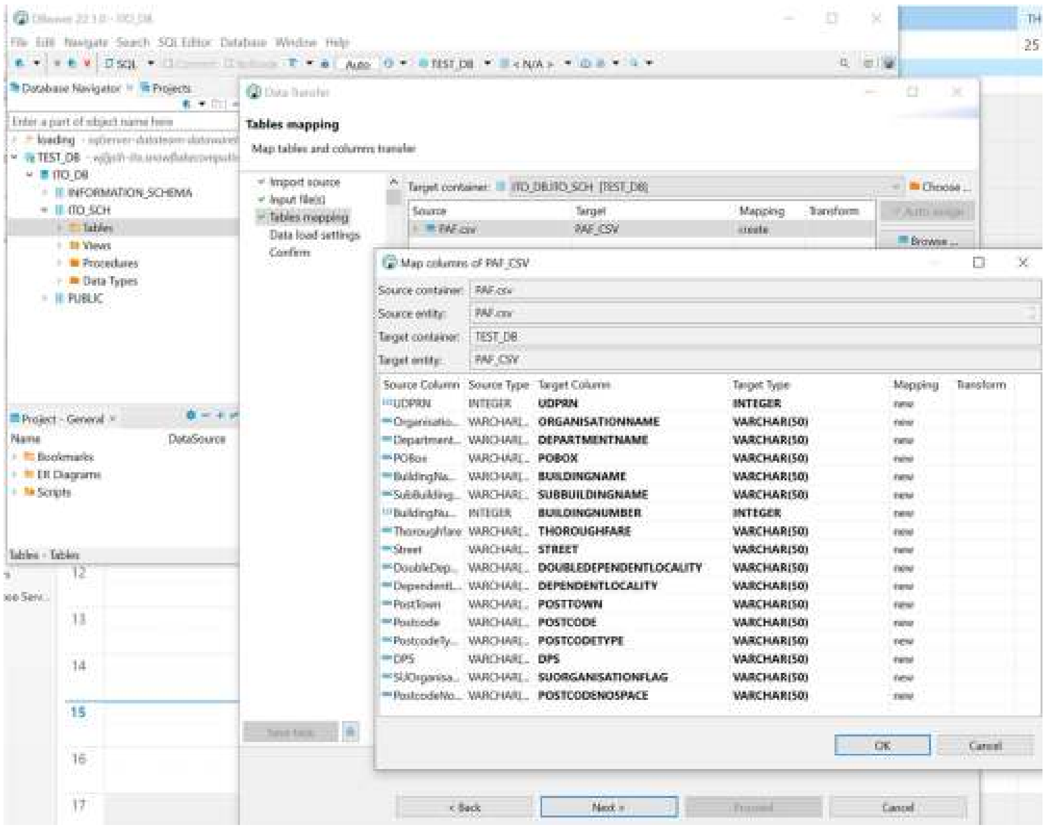


Figure 7.5.5.2: Import data into a schema/database of choice in DBeaver

Once the connection is set up all that is left is to right click onto the database schema the user wishes to load data into, which brings you onto the next screen.


Tables mapping

Map tables and columns transfer

Target container: ITO_DB.ITO_SCH [TEST_DB]

Source	Target	Mapping	Transform
PAF.csv	PAF_CSV	create	

Map columns of PAF_CSV

Source container: PAF.csv

Source entity: PAF.csv

Target container: TEST_DB

Target entity: PAF_CSV

Source Column	Source Type	Target Column	Target Type	Mapping	Transform
UDPRN	INTEGER	UDPRN	INTEGER	new	
Organisatio...	VARCHAR(...	ORGANISATIONNAME	VARCHAR(50)	new	
Department...	VARCHAR(...	DEPARTMENTNAME	VARCHAR(50)	new	
POBox	VARCHAR(...	POBOX	VARCHAR(50)	new	
BuildingNa...	VARCHAR(...	BUILDINGNAME	VARCHAR(50)	new	
SubBuilding...	VARCHAR(...	SUBBUILDINGNAME	VARCHAR(50)	new	
BuildingNu...	INTEGER	BUILDINGNUMBER	INTEGER	new	
Thoroughfare	VARCHAR(...	THOROUGHFARE	VARCHAR(50)	new	
Street	VARCHAR(...	STREET	VARCHAR(50)	new	
DoubleDep...	VARCHAR(...	DOUBLEDEPENDENTLOCALITY	VARCHAR(50)	new	
DependentL...	VARCHAR(...	DEPENDENTLOCALITY	VARCHAR(50)	new	
PostTown	VARCHAR(...	POSTTOWN	VARCHAR(50)	new	
Postcode	VARCHAR(...	POSTCODE	VARCHAR(50)	new	
PostcodeTy...	VARCHAR(...	POSTCODETYPE	VARCHAR(50)	new	
DPS	VARCHAR(...	DPS	VARCHAR(50)	new	
SUOrganisa...	VARCHAR(...	SUORGANISATIONFLAG	VARCHAR(50)	new	
PostcodeNo...	VARCHAR(...	POSTCODENOSPACE	VARCHAR(50)	new	

Figure 7.5.5.3: Import Wizard: Data schema is inferred from the local file which can be adjusted as needed

As seen in the image above, the interface allows you to select a data file of your choice, and then assuming it is clean, will infer the data schema (column names and appropriate data types) to create a possible table definition (DDL) for the data to land in. As is seen here it seems to think that the format of the data is a csv file, and has inferred that there are 4 columns and using the data contained in the scan of the file, has also assumed that the data types. At this point, the user is free to load as is with these assumptions or adjust to other data types if they know their data better. At this point completing the import wizard above will

1) Create the table in the snowflake schema and

2) load the data from that file into said table. You can also use this method to load into existing tables.

Another advantage of DBeaver is that it gives you another graphical tool to perform Snowflake queries on, adding to the CLI and in-browser official services.

NOTE: Despite DBeaver's convenience, under the hood all it is doing is committing insert statements line by line (or in batches) from the raw file. This is fine for smaller files or if load times are not a concern, however for very large files, it does not leverage Snowflake's native Big Data engine to load data from a raw staged file, which means that consequently it is painfully slow for the larger order of GB files. In addition, you get many INSERT statements in the transaction history of snowflake as one row per row insert transaction, which could make auditing harder in the long run.