Table of Contents

Arcus Migration Project

This document provides an overview of the data migration project from the CDL Hadoop platform to the Arcus platform, detailing the migration process, data layers, core functionalities, tools, testing approaches, and access requirements.

- **Migration Purpose and Platforms:** The data is migrated from CDL, built on Hadoop Cloudera, to Arcus to reduce maintenance costs and improve data editing flexibility. CDL stores data in Parquet and Apache HUDI, whereas Arcus uses MinIO storage with Apache Iceberg format for curated and presentation layers, offering ACID guarantees, schema evolution, and versioning.
- **Data Layers in Arcus:** Arcus organizes data into three layers: Raw (landing zone with unprocessed data stored as received in MinIO raw bucket), Curated (cleaned and standardized data optimized for processing in Iceberg format), and Presentation (final business consumption layer also in Iceberg format).
- **Core Data Transformations**: Key transformations during migration include schema alignment by adding new columns, merging columns based on business logic, and dropping unnecessary columns such as source_id. Data from multiple streams can be migrated in staged phases.
- **Historical Data Migration (HDM):** Data migration follows a bottom-up approach starting from the latest ingestion date folder in CDL, migrating folders in descending order to avoid duplicates. Migration intervals typically move one folder per iteration but can be increased to optimize throughput. Date filters enable selective migration.
- **Transformation Details**: During migration, schema transformations include dropping unnecessary columns, adding relevant columns for Arcus, and combining date-related fields into a single ingestion date column to reduce redundancy.
- **Tools and Access Requirements:** Essential tools include Jupyter for coding, Rancher for log checks, Superset for data viewing, Tiga for access requests, Airflow for workflow testing, and MinIO for data storage. Development environments use VS Code or IntelliJ, PuTTY and WinSCP for file transfers, and additional tools like Scala and PySpark for transformations.

- **Testing Methodologies:** Testing includes unit testing in development environments, sanity and data validation checks in test environments, and monitoring in production. Great Expectations library is used for automated data validation with minimum checks on raw data and extended checks on curated data. Tox environment supports unit testing of Airflow DAGs.
- **Access and Support Tools**: Tiga is used to raise access tickets for resources like buckets and VDI; Plaza provides IT and access-related support; and MFT tickets provision robot account access for file transfers on UNIX GW nodes. Procedures for onboarding streams and requesting bucket access are outlined.
- **Testing Approaches for HDM and Scala/DAG:** HDM testing involves unit tests, sanity checks, data validation across layers, and consistency verification between CDL and Arcus. Scala and DAG testing includes unit tests, manual verification of data migration across layers, and monitoring pipeline health in test and production environments.

Project Overview

The scope of the project involves migrating the following CDL to arcus platform:

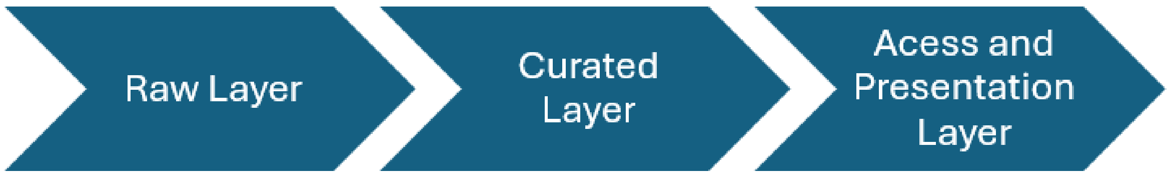
Data – The data is stored in Hadoop platform. In Historical Data Migration the data is migrated in partitions. Only the required source is migrated to Access Layer of Arcus.

Data pipelines – The PySpark/scala based data pipelines that are present in CDL are migrated to Arcus. It is made sure that the business logic is not changed to ensure the data integrity.

Reasons:

- To reduce the maintenance cost paid to the Cloudera platform.
- In CDL, the data is stored in parquet and Apache HUDI. It is more restricted, as it was difficult to edit the data.

Movement of Data in Arcus



Raw Layer

It is the landing area for raw data without any processing.

- Storage: The raw data is stored in a raw bucket on MinIO.
- Format: Data is stored exactly as received from the source.

Curated Layer

The data gets cleaned, standardized and processed. Data here is optimised for further processing or exploration.

- Storage: Access bucket on MinIO so that it can be accessed by Trino.
- Format: The data is stored in Apache Iceberg format by default. It may be stored in parquet format when specified. Iceberg gives strong ACID guarantees and easy schema changes without full rewrites. It is efficient for both batch and streaming reads. Also, it allows versioning and rollbacks.

Presentation Layer

It is the final layer where data is published for business consumption.

- Storage: Access bucket on MinIO so that it can be accessed by Trino.
- Format: The data is stored in Apache Iceberg format by default. It may be stored in parquet format when specified. Iceberg gives strong ACID guarantees and easy schema changes without full rewrites. It is efficient for both batch and streaming reads. Also, it allows versioning and rollbacks.

Core Functionality

The core functionality refers to the business logic and data transformation operations that are applied to the data to make it suitable for Arcus platform

1. Schema Alignment: New columns are added to the dataset as per the target schema. Ensures consistency across different data sources and makes the data compatible with Arcus.

2. Column Merging: Specific columns are combined based on business logic to form new columns or consolidated fields. Example: Concatenating first and last names into a full name.

3. Dropping Unnecessary Columns: Columns that are not required for analysis or further processing are removed to reduce data volume and enhance efficiency. Example: the source_id in CDL is dropped when migrated to Arcus.

Stage-wise Data Movement:

In certain scenarios, the data of multiple streams from a single source is moved in phases or stages. For example, if a source has 9 streams, there may be a requirement to move only 5 streams in Stage 1. Move the remaining 4 streams in Stage 2, possibly in parallel or after validation of Stage 1.

Arcus Entry point

It is the platform where we find most of the required URL platform. An entry platform to access different tool/platform. It has the following options:

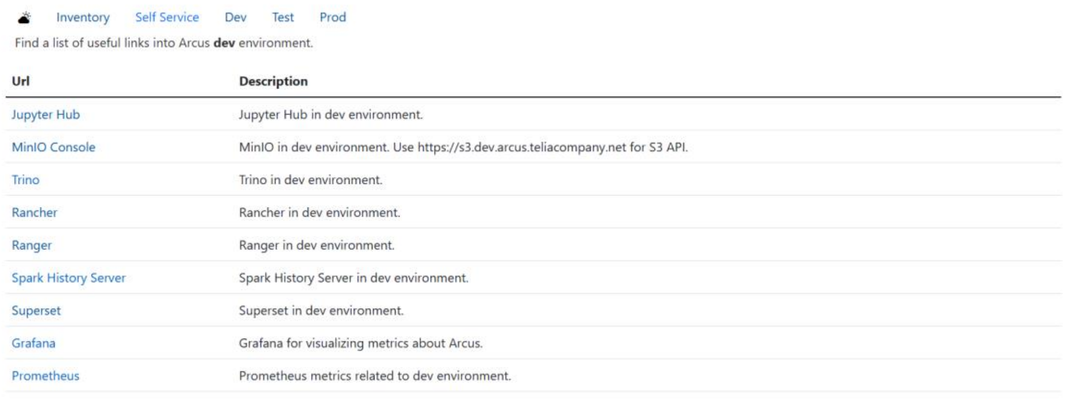
Inventory: In inventory we can see list of squads that are in Arcus. We can go to each squad and see the members of that squad.

Self – Service: Here on can onboard themselves to the team. Also onboard a stream, import a stream required to work on the migration.

Dev – Tools/ Platform URLs that are common for Development Environment. Also, we can find the Airflow links for the squads.

Test - Tools/ Platform URLs that are common for Test Environment. Also, we can find the Airflow links for the squads.

Prod - Tools/ Platform URLs that are common for Production Environment. Also, we can find the Airflow links for the squads.



Find a list of useful links into Arcus **dev** environment.

Url	Description
Jupyter Hub	Jupyter Hub in dev environment.
MinIO Console	MinIO in dev environment. Use https://s3.dev.arcus.teliacompany.net for S3 API.
Trino	Trino in dev environment.
Rancher	Rancher in dev environment.
Ranger	Ranger in dev environment.
Spark History Server	Spark History Server in dev environment.
Superset	Superset in dev environment.
Grafana	Grafana for visualizing metrics about Arcus.
Prometheus	Prometheus metrics related to dev environment.

Link - https://entrypoint.arcus.teliacompany.net/env/dev

HDM (Historical Data Migration)

In Historical Data Migration the base and/or access layers are migrated as needed for each source. The data is migrated partition – wise, with partitions typically organized by data.

Folder Structure: In CDL, data is partitioned by date, and each folder's name represents the ingestion date. The migration process involves identifying the folders in CDL and migrating them to Arcus.

Migration Logic:

- Start with the folder that has the maximum date value (i.e., latest available date in CDL). Bottom – up approach is used to migrate the data.
- Migrate that folder first to Arcus.
- Continue migrating in descending order (next latest folder that is not yet present in Arcus).
- This ensures no duplicate migration and a systematic catch-up of historical data.

Migration Intervals

- By default, the migration happens at an interval of 1, meaning one folder per iteration.
- Occasionally, an interval of 3 is used — meaning in a single iteration three folders are migrated.
- This helps in optimizing throughput for large datasets or backlog clearing.

Example:

Folder Dates in CDL: 20231101, 20231102, 20231103, 20231104

Current Max folder in Arcus: 20231101

Next migration (interval = 1): 20231102

Next migration (interval = 3): 20231102, 20231103, 20231104

Filter-Based Migration

Migration can be configured with start and end date filters, allowing selective migration between specific date range and controlled data load.

Transformation during migration

During the migration process from CDL to Arcus, schema transformation is also performed:

- Drop columns Unnecessary fields from CDL are removed.
- Add columns new columns relevant to Arcus are added.

Column transformation

Fields like ing_year, ing_month, ing_day are combined into a single ing_date column. After this the original columns are dropped to avoid redundancy.

Skill Set

- Python
- PySpark
- SQL
- Data warehouse Concepts

Required Access

- Jupiter – To test code
- Rancher - To check the logs if there are any errors
- Superset - To view the data in access layer in Iceberg format.
- Tiga – To raise ticket to access resource
- Airflow – To test
- MinIO – Place where the data is stored

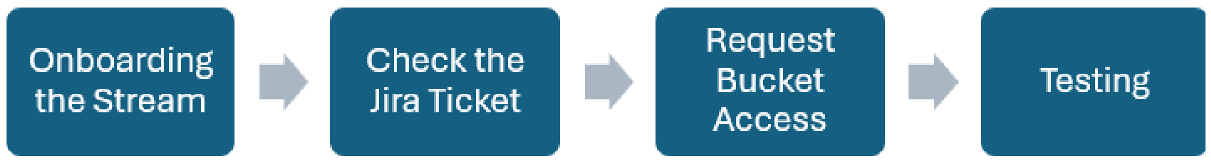
DAG

It is a collection of tasks organized in a way that reflects their dependencies and execution order.

Execution of DAG:

1. The data ingestion is done from edge server/source to raw layer.
2. GX - min check is applied in raw layer.
3. The data is moved from raw to curated layer transformations are applied.
4. Then the data is pushed to access layer.
5. GX - ext check is applied in access layer.

It is developing part of the workflow (DAG) that handles transforming the data using PySpark so that it conforms to the schema required by Arcus during the migration from CDL.



1. **Onboarding the Stream**

- Log in using your TCAD ID and password.

- Click “Onboard Stream”.
- Fill in the required details: Squad Name, Source Name and Stream Name.

2. **Check the Jira Ticket**
   All essential information about the stream is included in the Jira ticket like Old DAG repository, Squad name, Stream name and Sample Data. If any information is missing, reach out to the squad’s point of contact for clarification.

3. **Request Bucket Access**
   Use TIGA to raise a ticket for bucket access permissions. Submit access requests for the entire team, ensuring everyone has access to the required MinIO bucket. After getting all the access work on development.

4. **Testing**
   Testing is done using the sample data. However, the testing against real data is done during the production. Sometimes the sample data is not provided in the Jira ticket, we need raise a ticket for sample data. Sometimes an IP address is provided, with help of PuTTY and WinSCP we can download the data from the given directory.

   If the test fails, we can access the error logs using rancher also we can download the log file.

Required Access:

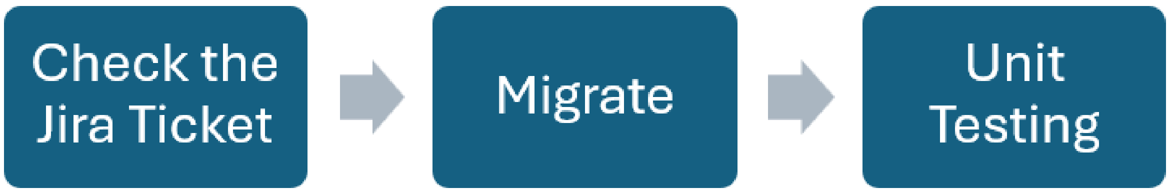
- Jupiter – To write code
- Rancher - To check the logs if there are any errors
- Superset - To view the data in access layer in Iceberg format.
- Repository
- Tiga – To raise ticket to access resource
- Airflow – To test
- Vs code - (local and VDI) - We clone the git repo in VS and work on it.
- Putty and WinSCP – To upload the files

Skill Set

- Python
- PySpark

Scala

The Scala-based transformation logic is primarily located in the core folder. This is where all the core transformation logic is written (e.g., cleaning, renaming, dropping columns). The app folder acts as the entry point and contains only the plain data, transformations are not applied here.



1. **Check the Jira Ticket**
   All essential information about the stream is included in the Jira ticket like Old DAG repository, Squad name, Stream name and Sample Data. If any information is missing, reach out to the squad's point of contact for clarification.
2. **Migrate the data**

The data first arrives in raw layer in Arcus. Basic checks and validation are performed to ensure ingestion integrity. Transformations are applied in Curated Layer like Dropping unnecessary columns, adding new columns where required, Renaming and restructuring as per Arcus schema.
After successful transformation, the cleaned and validated data is pushed into the Access Bucket (Layer). This layer holds the final curated data, ready for downstream consumption.

3. **Unit Testing**
   Unit testing is performed to ensure logic integrity. If test cases are already written in CDL, similar validations are replicated in Arcus.

Required Access:

- Jupiter – To write code
- Rancher - To check the logs to see if there are any errors.
- Superset - To view the data in access layer in Iceberg format.
- Repository
- Tiga – To raise ticket to access resource
- Airflow – To test
- IntelliJ - We clone the git repo and work on it.
- Putty and WinSCP – To upload the files

Skill Set

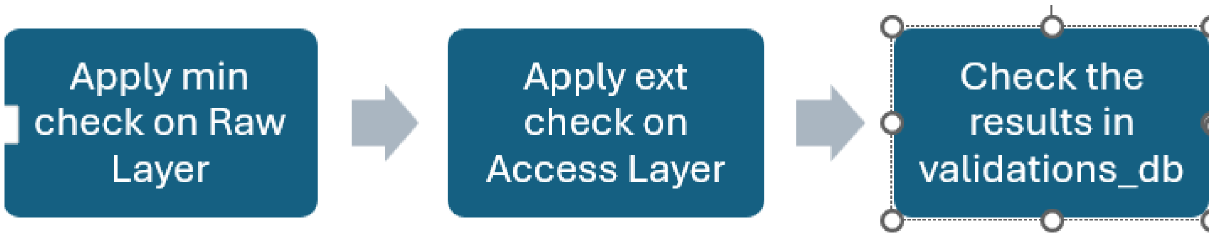
- Scala
- PySpark

Great Expectations

Great Expectations is an open-source Python-based data validation and documentation library developed to help teams maintain data quality and improve data pipeline reliability.

- Check if your data is correct and clean (e.g., no missing values, right data types, etc.)
- Automate data testing during ETL/data pipelines
- Create clear reports of what’s expected in your data

Flow


Apply min check on Raw Layer
Apply ext check on Access Layer
Check the results in validations_db

The task here is to check if the data is migrated from CDL to Arcus in right format.

1. First the minimum check is applied on the raw data layer.
2. Then the extended check is applied on access layer.
3. These checks are stored in a database and the dark triggers the validations results are stored validations DB

The checks are defined as expectations and stored in a database in "ge_expextation_store" table, where the table includes columns like suitename and value. The suitename is derived based on the type of check and the data layer it applies to, while the value column holds the corresponding expectation logic.

For all the squads except tfi the dbname is network_non_sensitive.

For **tfi** dbname is tfi_non_sensitive

Template for creating the suitename:

arcus_squadName_sourceName_streamName_layer_check_quality_0_1

There 2 types of checks are applied:

1. min check on Raw Layer
2. ext check on Access Layer

1. min check

minimum check is applied at the raw layer to perform basic validations, such as verifying that the columns are in the correct order and ensuring the row count falls within the expected range.

To check if the number of rows are between the specified range
validator.expect_table_row_count_to_be_between(min_value=min_value, max_value=max_value)

To check if the column order is correct
validator.expect_table_columns_to_match_ordered_list(column_list = column_list)

2. ext check

Extended checks are applied at the access layer to validate specific conditions, such as ensuring that the injection_date column contains exactly 8 digits.

To check if ingestaion data has 8 digits

validator.expect_column_values_to_match_regex("ingestion_date", r"^\d{8}$")

Required Access

- Jupiter notebook – To upload data.
- Superset - To view the data in access layer in Iceberg format.
- Hue- To see the columns present in CDL.
- Repository (GitHub)
- Bucket access (optional)

Skill Set

- Python GX module
- Apache Airflow

Manually adding suitenames and Expectation

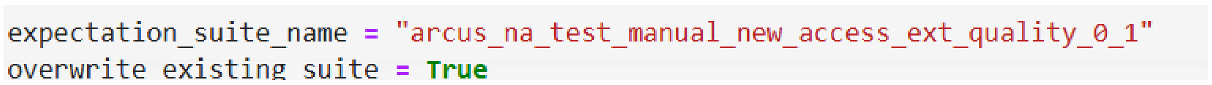
While using automation to add suitenames the suitenames, columns, min value and max value are directly picked from the excel sheet. But while adding the suitenames manually all these values including the schema are hard coded.

DB details

- username = "network_non_sensitive"

- password = "<password>"
- database = "network_non_sensitive"
- host = "cp100pgsql01.ddc.teliasonera.net"
- port =5432

The suitename is hard coded.

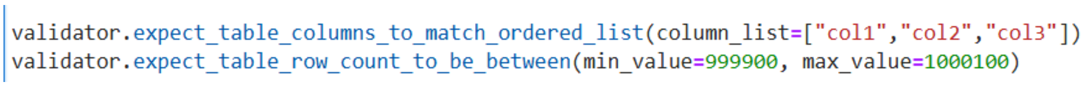


```
expectation_suite_name = "arcus_na_test_manual_new_access_ext_quality_0_1"
overwrite_existing_suite = True
```

The schema is defined.

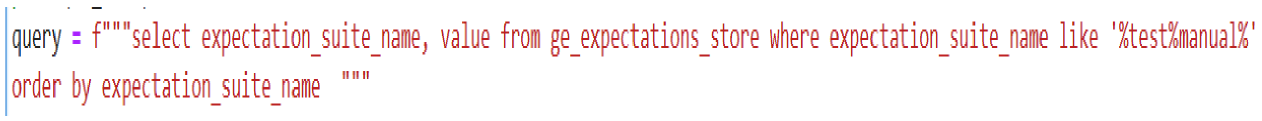
```
# Define schema with new columns
schema = StructType([

    StructField("col1", StringType(), True),
    StructField("col2", StringType(), True),
    StructField("col3", StringType(), True)
])
```

While adding the expectations we need to pass the column names and min and max values



```
validator.expect_table_columns_to_match_ordered_list(column_list=["col1","col2","col3"])
validator.expect_table_row_count_to_be_between(min_value=999900, max_value=1000100)
```

We can check the suitenames and expectations added in ge_expectation_store



```
query = f"""select expectation_suite_name, value from ge_expectations_store where expectation_suite_name like '%test%manual%'
order by expectation_suite_name  """
```

Can check the logs in ge_validations_store

```
select * from ge_validations_store where
expectation_suite_name='arcus_test_manual_cs_access_ext_quality_0_1'  order by
run_time desc
```

Tox Environment

The Tox environment is a virtualized testing setup primarily used for writing and executing unit tests for Directed Acyclic Graphs (DAGs). It ensures that the DAG logic, task dependencies, and configurations are syntactically and functionally correct before integration with any actual code or data pipelines.

Characteristics

- Used for unit testing only: Focuses on testing the structure, logic, and airflow-specific constructs within the DAG.
- No integration with actual business logic or external code: Does not interact with Scala code, external data sources, or production configurations.
- Fast feedback loop: Ideal for quick validation of changes without spinning up the full pipeline.

How it is different from Development environment?

Tox environment is used for isolated unit testing of Airflow DAGs, ensuring basic correctness. The development environment, on the other hand, is designed for full workflow development and testing where the DAG is connected to processing code (like Scala), and data flow is tested end-to-end.

Setup

Local Environment Setup for Python Development in VSCode and tox setup

Tox Environment setup

Before starting install pip
Local Environment Setup for Python Development in VSCode and tox setup
Step 1: Open VSCode
Open VSCode on your local machine.

Step 2: Open the Terminal in VSCode
From the menu, go to Terminal → New Terminal.

Step 3: Create and Activate a Virtual Environment

Create a virtual environment (replace myenv with your desired name):

```
pip install virtualenv

    python -m venv mein
```

Step 3: Install Dependencies
pip install pytest

Step 4: pip install tox. type tox and enter

Note: when we clone git make sure these commands are executed.

```
git config --global user.name "<github-username>"

    git config --global user.email "<teliaemail>"
```

Tiga

It is a tool used to raise access related tickets for various resources within the organization. It allows users to request access to buckets, VDI, Jira and other internal tools or platforms. Through, Tiga users can submit manage and track access requests ensuring that necessary permissions are granted securely and efficiently.

When to use it?

We use it whenever we need to request access to specific tools, platforms or environments required for our work. This ensures the access is granted.

For example, when we need to read or write data from bucket that we can't currently access.

Steps to Raise the ticket:

1. Open Tiga.
2. Add details:
- Submit request for - for whom we are requesting the access for. Usually when we are raising the request for bucket we request for the whole team.
- Search for the resource/tool that needs to be accessed.
3. Submit the ticket.

Tiga link - https://tiga.teliacompany.net/dashboard.aspx

Plaza

Plaza is a centralized internal service centre designed to support employees in resolving common IT and access-related issues. It acts as a single point of contact for technical assistance and service requests within the organization.

When to use it?

- When you face problems logging into or using your Virtual Desktop Infrastructure (VDI).
- If you're unable to access or use platforms like Jira, Citrix, MobilePass you can raise a support ticket via Plaza.
- Plaza allows you to directly connect with the internal support team through chat to resolve technical or access-related issues in real time.

Steps to Raise the ticket:

1. Login to Plaza.
2. Select People Hub Support.
3. Select the category that is related to the issue.
4. Add Subject.
5. Add a short description about the issue.

Plaza link - Plaza Login - Employee Center

MFT

We initiate an MFT (Managed File Transfer) ticket to provision the necessary access for the robot account to the required folders on the GW node, which operates in a UNIX

environment.
Each squad is assigned a dedicated folder within the GW node, and incoming data is routed directly into their respective folders.

Robot account - connects with squad machine and ensure that the data is placed in the right folder

Folder Structure

ra → Data is present in 1 folder

sa and na → For each stream/source a folder is created.

- Usually, the flow in Arcus prod is stopped as the files were also incoming with the flow. Raise a ticket to start the flow in Arcus Production Environment.
- In case the is not ingested in the Raw Layer during the data pipeline is up in Production, check the folder is GW node. Go to the squad specific folder and check of the files are present there.

Pre-requisite:

- In Arcus entry point platform, the stream is onboarded, and it is imported.
- Ensure TIGA request is raised and approved to get MFT robot account access to the specific folder in GW node.
- Ensure folders are created at the GW node for that source/stream.

Steps to raise request

1. Find the MFT flow id for the existing cdl flow.
2. Create a new request, refer any of the existing MFT flow request to create new one
3. Need to provide following information to raise new request
   a. MFT flow id
   b. Gateway node, specific to squad
   c. Target Path on GW node where files would arrive
   d. In case of multiple streams under source, explain which file should come to which folder.
   e. MFT Robot account for the squad
   f. Add watchers so that stakeholders are informed about the progress.

Steps to get access to Buckets

1. Open Telia inventory page, click on self – service.
2. Select Onboard a stream.
3. Enter details – squad name, source name, stream name, and stream type
   The stream type can be:
   - Push – raw, curated and access buckets
   - Pull – curated and access buckets
   - Aggregation - access bucket only
   - Custom – Select buckets of our choice.
4. Click on submit.
5. Then go to Onboard an Import.
6. Enter details – squad name, source name, stream name, and tool
7. Click submit.
8. In Inventory, click on the squad’s name and then you will be able to see the buckets.
9. Copy the bucket names and go to Tiga.
10. Raise the ticket for Bucket access for the team.

There are times when even though your name specified for that bucket access you would not be able to access it. At that time posted in the team’s channel.

Steps to Onboard a Squad (stream)

1. Open Telia inventory page, click on self – service.
2. Select Onboard a stream.
3. Enter details – squad name, source name, stream name, and stream type
4. The stream type can be:
    - Push – raw, curated and access buckets
    - Pull – curated and access buckets
    - Aggregation - access bucket only
    - Custom – Select buckets of our choice.
5. Click on submit.

Tools/ platforms access required to work on the squad

- Jupiter – To write code
- Repository
- Rancher - To check the logs if there are any errors.
- Superset - To view the data in access layer in Iceberg format.
- Tiga – To raise ticket to access resource
- Airflow – To test
- IntelliJ - We clone the git repo and work on it.
- Putty and WinSCP – To upload the files.
- Source buckets.

Steps to create a new repo in GitHub

Before creating a repo make sure you have access to the following links

1. Check for access in the link below

   Link: https://github.com/telia-company/arcus-cicd-workflows-and-actions

   If there is no access, make sure the respective Tiga roles for GitHub have been raised and approved.

2. Then check access

   Link: https://github.com/orgs/telia-company/teams/cdl-developers

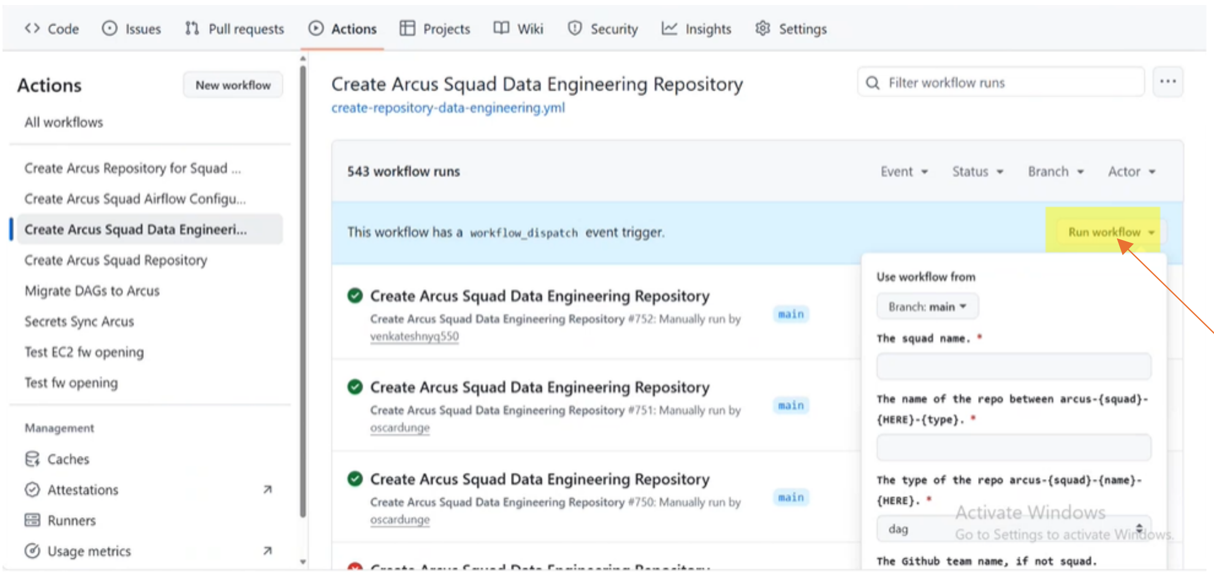
   If access is not available request one of the maintainers for access.

Steps to create new repo

1. Once there is access for above links

   https://github.com/telia-company/arcus-cicd-workflows-and-actions/actions/workflows/create-repository-data-engineering.yml

   Click on above link which opens to repo creation landing page.



Create Arcus Squad Data Engineering Repository
create-repository-data-engineering.yml

543 workflow runs

This workflow has a `workflow_dispatch` event trigger.

Create Arcus Squad Data Engineering Repository
Create Arcus Squad Data Engineering Repository #752: Manually run by venkateshnyq550 main

Create Arcus Squad Data Engineering Repository
Create Arcus Squad Data Engineering Repository #751: Manually run by oscardunge main

Create Arcus Squad Data Engineering Repository
Create Arcus Squad Data Engineering Repository #750: Manually run by oscardunge main

Use workflow from
Branch: main

```
The squad name. *

The name of the repo between arcus-{squad}-{HERE}-{type}. *

The type of the repo arcus-{squad}-{name}-{HERE}. *
dag
The Github team name, if not squad.
```

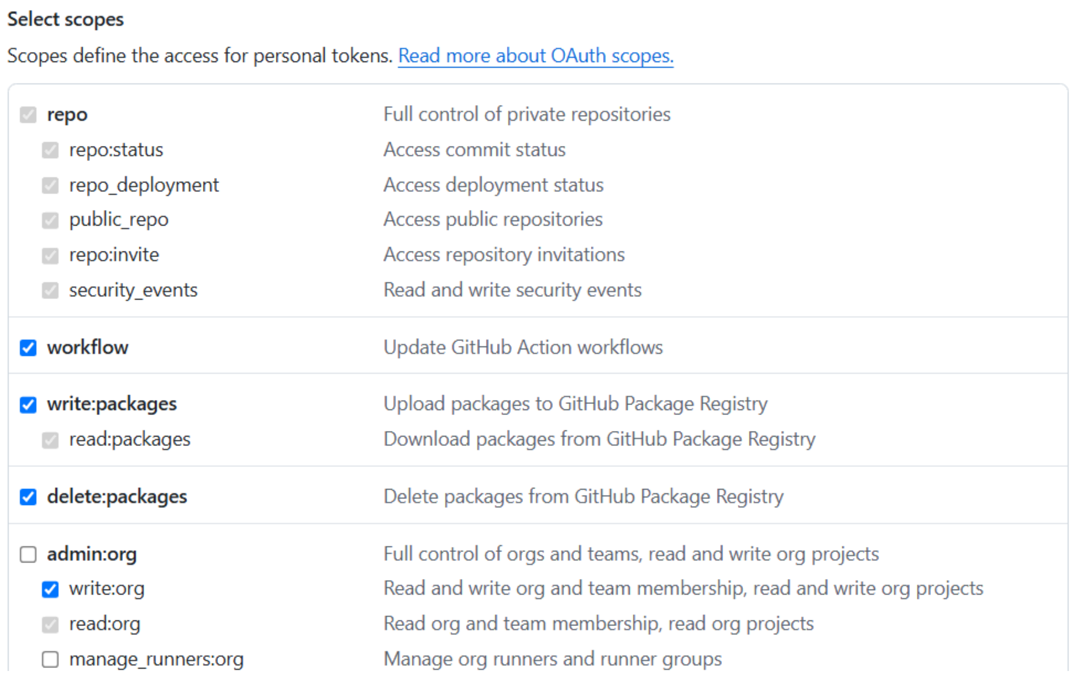
1a) Click on "Run Workflow" and enter the details.

- Squad name – the name of the squad the repo needs to be created.
- Name of the repo – The source for which repo needs to be created.
- Type of the repo – The 3 options are

- dag – for the dag code.
- spark – for pyspark code
- maven – for scala code

1b) Create GitHub token

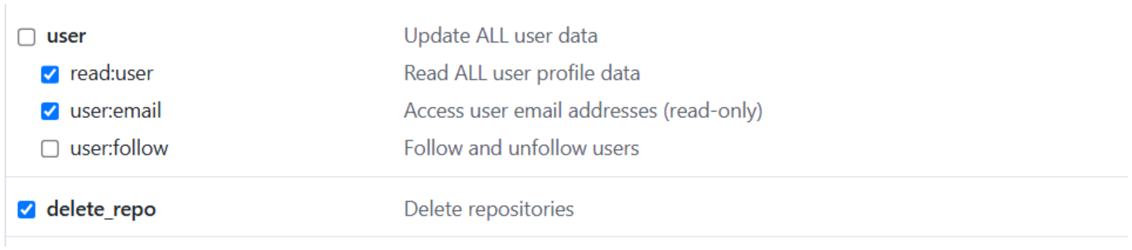
- In GitHub open settings, in the left panel scroll down to developer settings.
- Personal access tokens → Token (classics)
- In the right corner there is "Generate new token"→ Generate new token(classic). Make sure you select these scopes below



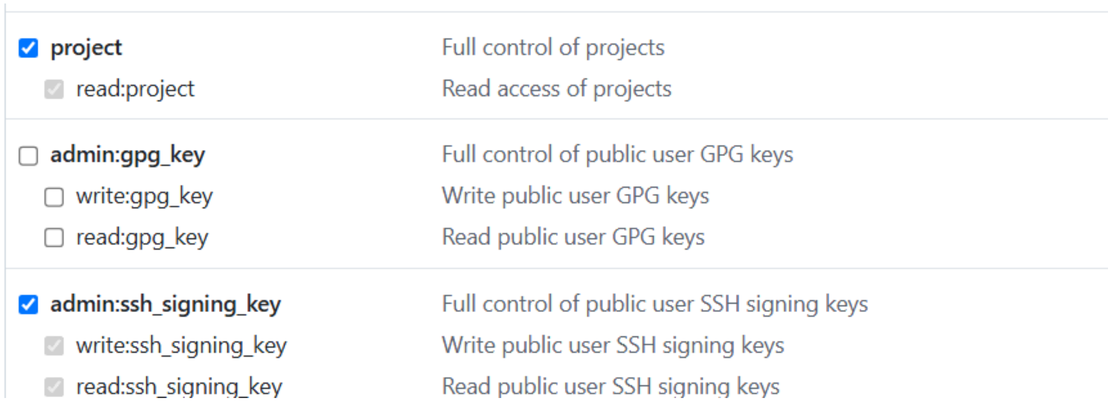
Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

✓ **repo**	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations
✓ security_events	Read and write security events
✓ **workflow**	Update GitHub Action workflows
✓ **write:packages**	Upload packages to GitHub Package Registry
✓ read:packages	Download packages from GitHub Package Registry
✓ **delete:packages**	Delete packages from GitHub Package Registry
☐ **admin:org**	Full control of orgs and teams, read and write org projects
✓ write:org	Read and write org and team membership, read and write org projects
✓ read:org	Read org and team membership, read org projects
☐ manage_runners:org	Manage org runners and runner groups



☐ **user**	Update ALL user data
✓ read:user	Read ALL user profile data
✓ user:email	Access user email addresses (read-only)
☐ user:follow	Follow and unfollow users
✓ **delete_repo**	Delete repositories



✓ **project**	Full control of projects
✓ read:project	Read access of projects
☐ **admin:gpg_key**	Full control of public user GPG keys
☐ write:gpg_key	Write public user GPG keys
☐ read:gpg_key	Read public user GPG keys
✓ **admin:ssh_signing_key**	Full control of public user SSH signing keys
✓ write:ssh_signing_key	Write public user SSH signing keys
✓ read:ssh_signing_key	Read public user SSH signing keys

Note this token as it used to create repos every time.

Once the token is generated paste it in “Your GitHub Token” and click “Run Workflow”. Once executed, it will appear green.

2. Devx-secret rotation

   After successfully creating the repo, click on the link below to add the repo created.

   Link - https://github.com/apps/devx-secret-rotation

   - Click on select repositories and select the new repo created.
   - Click on update access
3. Devx-release-app

   After successfully creating the repo, click on the link below to add the repo created.

   Link - https://github.com/apps/devx-release-app

   - Click on select repositories and select the new repo created.
   - Click on update access
4. Perform secret sync
   This is squad specific, click on link for which repo is been created.

   ra - https://github.com/telia-company/arcus-ra-configuration

   na – https://github.com/telia-company/arcus-na-configuration

   sa – https://github.com/telia-company/arcus-sa-configuration

   tfi – https://github.com/telia-company/arcus-tfi-configuration

   sweb2x – https://github.com/telia-company/arcus-sweb2x-configuration

   crowd - https://github.com/telia-company/arcus-crowd-configuration

   - After clicking on link, create a new branch.
   - Go to secret-sync.yml
   - Add the repo - ${{ github.repository_owner }}/<repo name>

   Make sure you create a new branch to make changes to in secret-sync.yml. DO NOT MAKE CHANGES IN MAIN BRANCH.

After secret sync if you are not able to commit changes check development-workflow.yml

You can self-invite yourself by opening below links:

- **https://github.com/orgs/telia-company/sso**
- **https://github.com/orgs/telia-actions/sso**
- **https://github.com/orgs/telia-oss/sso**

- **https://github.com/orgs/TeliaSoneraNorge/sso**

After self-invite make sure you can make changes in the git repository. If not ask one of the maintainers to add your git username in the below link.

https://github.com/orgs/telia-company/teams/cdl-arcus-migration

Testing Approach – HDM

Testing in Dev environment

- Gather table level information for the migration from CDL.
- Ensure necessary access is granted to perform development task
- Perform unit testing on the implemented logic
- Raise and get approval for the pull request from Capgemini
- Document and attach unit test results to the respective jira ticket

Testing in Test Environment

Sanity check is performed which ensures the air flow dag is running successfully.

- Validate that a small subset of data can be successfully copied from CDL to Arcus.
- Confirm the details return to minio buckets according to the defined partition strategy. across both curated and presentation layer
- Verify the data is visible in superset for both the years

Document the results

If any defects it needs to be fixed.

Testing in Prod Environment

The flow is monitored in prod environment:

- The table names in curated and access layer in both CDL and Arcus
- data copied from data and data copied to date
- Row counts in the curated and presentation layers are verified to be identical across both platforms.
- Schema structures are compared to ensure they match.
- Data values are validated to confirm consistency.

Testing Approach – Scala and DAG

Testing in Dev environment

The unit test cases are performed here. Only the unit cases present in CDL environment are tested here.

The results of unit tests are documented.

<u>Manual Testing</u>

If the data has been migrated properly to each layer from raw→cirated→access.

The data can be accessed via MinIO and Superset.

<u>Testing in Test Environment</u>

In testing environment, sanity testing is performed to ensure:

- The DAG is running successfully, if there is a failure it is shown in Airflow UI.
- The data is ingested properly in Raw Layer.
- The data is present in MinIO raw, curated and access layer.

GX checks are performed – min and ext. Min checks ensures the row count is between the specified range and the columns are in the correct order. Ext checks ensure that the ingestion date is in proper format.

If there is any difference the code needs to be fixed by the Development team.

<u>Testing in Prod Environment</u>

The data pipeline is monitored over a 7-day period, during which the following validations are performed between Arcus and CDL:

- Row counts in the curated and presentation layers are verified to be identical across both platforms.
- Schema structures are compared to ensure they match.
- Data values are validated to confirm consistency.

Testing Approach – Great Expectations

<u>Testing in Dev environment</u>

The expectations are added to the ge_expectations_db along with the suitename.

<u>Testing in Test environment</u>

The DAG triggers expectations. The logs are stored in ge_validation_db. We need to validate if the column order matches with schema.

If it matches it will display as success, if not then it will show failure. The email will be triggered to the mentioned person/ distribution list.

For a suite, even if one expectation fails then the result is displayed as false for that suite. For example, we have created a suite with 3 expectations – row count, column order and ingestion_date format. If the column order and ingestion_date are in correct format but row count is not within the specified range, then the suite fails.

<u>Testing in Prod Environment</u>

A CDLM ticket is created where we provide the suitenames that need to be added to the prod environment.

Updating the Jira Ticket

link - https://jira.atlassian.teliacompany.net/login.jsp?permissionViolation=true&os_destination=%2Fsecure%2FStructureBoard.jspa%3Fs%3D5408&page_caps=&user_role=

The above link is to the Arcus Migration dashboard. It has tasks that are completed and yet to be completed for the project.

For a main task there are multiple tasks sub tasks assigned to the members of the project. They are CDLM tickets which contain information about squads, sources and streams.
When you open a sub task you can see its status, comments added if any.

We can set the status of a task as:

- In review
- Cancel
- Done
- In progress
- Reopen
- Merged

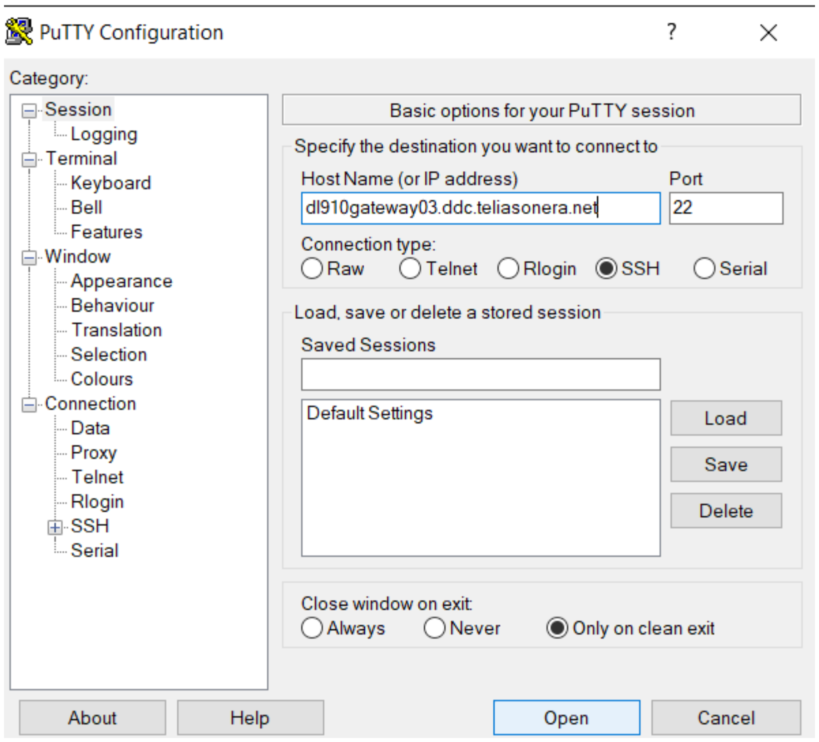
Also, one can assign it to others or themselves.

Deployment Process – UAT production

1. The development work should not be done directly on the main branch. Instead, it must be carried out in a separate feature branch. Once the changes are complete and have passed code review, they can be merged into the main branch.
2. In GitHub Actions, there is an option called "Release Time". Select the appropriate branch and run the workflow. The workflow execution should show a green status, indicating success.
3. After the workflow is completed successfully, a release branch will be created automatically along with a corresponding pull request for review and merging.
4. In case you need permission to merge, add the reviewers for approval.
5. It will deploy the code in dev, test and prod.

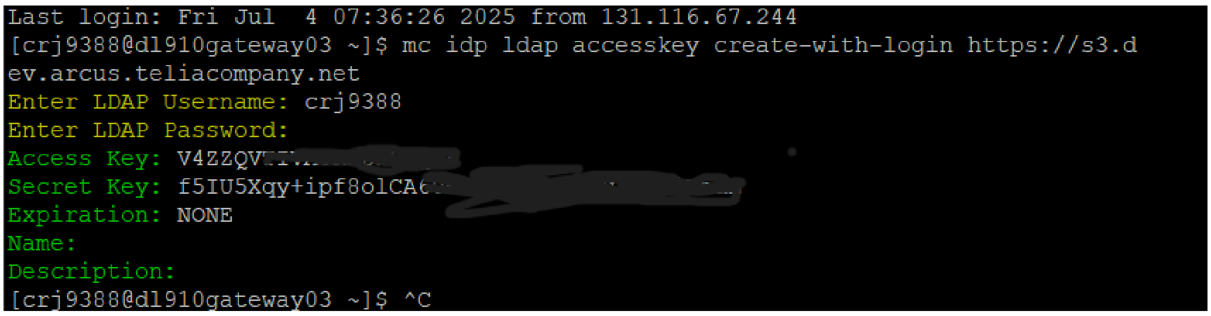
Generation of Access and Secret Key for Jhub

1. Open putty. Use this as hostname - dl910gateway03.ddc.teliasonera.net
2. Leave the port number and others as default. Click on open



Specify the destination you want to connect to
Host Name (or IP address): dl910gateway03.ddc.teliasonera.net
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
Close window on exit: Always Never [✓ Only on clean exit]

3. Enter your TCAD ID.
4. Paste this command -
   mc idp ldap accesskey create-with-login https://s3.dev.arcus.teliacompany.net
   and enter.
5. Enter your TCAD and password.
6. The access key and Secret key is displayed as follows. Save these keys for further reference. Every time when we login to JHub use the same keys.

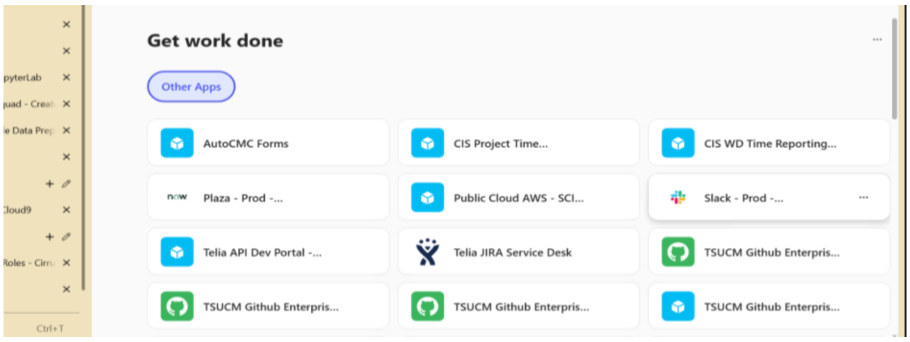


```
Last login: Fri Jul  4 07:36:26 2025 from 131.116.67.244
[crj9388@dl910gateway03 ~]$ mc idp ldap accesskey create-with-login https://s3.d
ev.arcus.teliacompany.net
Enter LDAP Username: crj9388
Enter LDAP Password:
Access Key: V4ZZQV[redacted]
Secret Key: f5IU5Xqy+ipf8olCA6[redacted]
Expiration: NONE
Name:
Description:
[crj9388@dl910gateway03 ~]$ ^C
```

Slack Channel

It is a platform where the users post issues related to tools/platforms. The issue is addressed by other users with the solution for it. It has different channels to address the issue for a platform/tool. The channel's name can be searched.

One can login to slack channel from O365 and click on Slack or use the link provided to login.


Get work done
Other Apps
AutoCMC Forms
CIS Project Time...
CIS WD Time Reporting...
Plaza - Prod -...
Public Cloud AWS - SCI...
Slack - Prod -...
Telia API Dev Portal -...
Telia JIRA Service Desk
TSUCM Github Enterpris...
TSUCM Github Enterpris...
TSUCM Github Enterpris...
TSUCM Github Enterpris...

Link - https://grid-teliafin.enterprise.slack.com/

The different channels available:

#github – The channel where one can post issues related to GitHub.

#data-platform-community – A channel where one can post issues related to Jhub, superset and other platforms.

#slack-faq – In this channel we post issues or query regarding slack.

Steps to raise PR

Ensure following before raising a PR:

1. Check if there are code fixes applied on existing code on CDL during last 4 months, we need to apply same on Arcus repo before raising PR (pt mentioned by NA squad)
2. Check that table/field level descriptions are given in DDL statement while creating table in the Superset, this will be used by the DataHub team. TFT team already shared, follow up with NA and RA squad.
3. Unwanted, commented code and SOP are removed used for debugging purposes before raising the PR
4. Code has proper indentation and comments to help someone looking at the code to understand it properly
5. Ensure to attach successful logs for DAG executions for the code and Unit test cases if available.
6. Check data is reaching Minio, accessible and data can be seen in superset, attach all screen shots
7. Ensure that code is not merged with main branch while raising PR, apply branch protection
8. Write a message in CDLM to the reviewer with GitHub repo link and necessary attachments, asking to review the code against the PR
9. Ensure there is no hardcoding in the code, values are read from config file or the way it implemented in existing CDL code
10. Add a readme file and add comments in the code to help the reviewer understand the code logic, approach
11. Check Unit test cases are migrated, if it was available in existing code
12. Add a few unit test cases and attach screen shots for testing results while raising the PR to provide proof of how you tested the pipeline End to end.
13. Ensure that GDPR columns are included in Arcus code if it's applicable
14. Follow the architecture guidelines
15. Ensure all old comments are addressed when re-submitting the PR
16. Understand the existing code properly and ensure to avoid missing any logic, the transformation part applied in existing code
17. Attach the source file used for testing the flow
18. Ensure to match schema with existing tables and attach that comparison in PR

Steps to raise PR

1. Navigate to the Jira ticket to which you want to raise the PR.
2. There is a sub task available for each source with Subject “Development Review & Signoff”. Change its status to In-progress.
3. Add the necessary comments. Attach the logs of the DAG along with required screenshots.
4. Once the review is done there may be changes that need to be made or issues need to be fixed. Make the required changes.
5. After changes mark the task as done.

Airflow variables

For each squad the airflow variables are stored in different git repos. These variables are often used in our code.

Repo links:

na - https://github.com/telia-company/arcus-na-airflow-configuration/blob/main/base.yaml

sa – https://github.com/telia-company/arcus-sa-airflow-configuration/blob/main/base.yaml

ra – https://github.com/telia-company/arcus-ra-airflow-configuration/blob/main/base.yaml

sweb2x – https://github.com/telia-company/arcus-sweb2x-airflow-configuration/blob/main/base.yaml

There are 3 different yml files where we can refer for the variables:

- dev.yml
- test.yml
- prod.yml

Development-workflow.yml

Contains the variable name and path to the variable.

development-workflow.yml

```
{
       "vault": {
         "path": "ra/kv_dev/data/ra_gx_psql_conn_test",
         "name": "db"
       },
       "helm": {
         "path": "global.ra_gx_psql_conn_test.db"
       },
       "envVariable": "RA_GX_POSTGRES_DB_TEST"
     }, {
       "vault": {
         "path": "ra/kv_dev/data/ra_gx_psql_conn_test",
         "name": "user"
       },
       "helm": {
         "path": "global.ra_gx_psql_conn_test.user"
       },
       "envVariable": "RA_GX_POSTGRES_USER_TEST"
     },
```

Dev.yaml

```
  - name: AIRFLOW_CONN_RA_GX_DB
    value: |
     {
       "conn_type": "postgres",
       "description": "Connection to the Great Expectations Store DB for CDL2 to Arcus
Migration",
       "host": "{{ .Values.global.ra_gx_psql_conn_test.db }}",
        "login": "{{ .Values.global.ra_gx_psql_conn_test.user }}",
       "port": "5432",
     }
```

Role names for Bucket Access

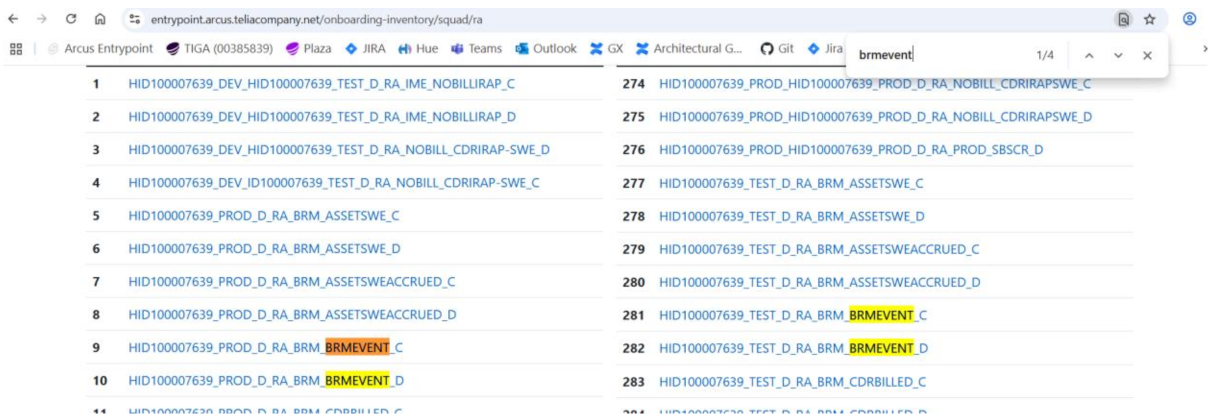
To get bucket access we need to raise the bucket access in Tiga.

ROLE template: D_SQUAD_SOURCE_STREAM_D → for development

D_SQUAD_SOURCE_STREAM_C →For prod
Ex: D_NA_KALIX_TACLIST_D

If you cannot find the role in Tiga.

- Go to Arcus entry point → inventory.
- Click on the squad of the source that you are looking for to raise access.
- Scroll down to see the roles. Ctrl + f and type the source to see the required role. Copy and paste them in Tiga to raise access. For example: HID100007639_TEST_D_RA_BRM_BRMEVENT_D paste this in Tiga.



entrypoint.arcus.teliacompany.net/onboarding-inventory/squad/ra

brmevent 1/4

1	HID100007639_DEV_HID100007639_TEST_D_RA_IME_NOBILLIRAP_C	274	HID100007639_PROD_HID100007639_PROD_D_RA_NOBILL_CDRIRAPSWE_C
2	HID100007639_DEV_HID100007639_TEST_D_RA_IME_NOBILLIRAP_D	275	HID100007639_PROD_HID100007639_PROD_D_RA_NOBILL_CDRIRAPSWE_D
3	HID100007639_DEV_HID100007639_TEST_D_RA_NOBILL_CDRIRAP-SWE_D	276	HID100007639_PROD_HID100007639_PROD_D_RA_PROD_SBSCR_D
4	HID100007639_DEV_ID100007639_TEST_D_RA_NOBILL_CDRIRAP-SWE_C	277	HID100007639_TEST_D_RA_BRM_ASSETSWE_C
5	HID100007639_PROD_D_RA_BRM_ASSETSWE_C	278	HID100007639_TEST_D_RA_BRM_ASSETSWE_D
6	HID100007639_PROD_D_RA_BRM_ASSETSWE_D	279	HID100007639_TEST_D_RA_BRM_ASSETSWEACCRUED_C
7	HID100007639_PROD_D_RA_BRM_ASSETSWEACCRUED_C	280	HID100007639_TEST_D_RA_BRM_ASSETSWEACCRUED_D
8	HID100007639_PROD_D_RA_BRM_ASSETSWEACCRUED_D	281	HID100007639_TEST_D_RA_BRM_BRMEVENT_C
9	HID100007639_PROD_D_RA_BRM_BRMEVENT_C	282	HID100007639_TEST_D_RA_BRM_BRMEVENT_D
10	HID100007639_PROD_D_RA_BRM_BRMEVENT_D	283	HID100007639_TEST_D_RA_BRM_CDRBILLED_C

Creating a table through Jhub

```
#creating a spark session

from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Create Iceberg Table") \
    .config("spark.sql.catalog.iceberg", "org.apache.iceberg.spark.SparkCatalog") \
    .config("spark.sql.catalog.iceberg.type", "hive") \
    .config("spark.sql.catalog.iceberg.warehouse", "s3a://resource-na") \
    .getOrCreate()



from pyspark.sql.types import StructType, StructField, StringType, DateType

 # Define the schema
 schema = StructType([
    StructField("subscription_id", StringType(), True),
    StructField("customerid", StringType(), True),
    StructField("requested_date", DateType(), True),
    StructField("pur1033", StringType(), True)
 ])


# Create an empty DataFrame
 empty_df = spark.createDataFrame([], schema)
```

```
# Save it as an Iceberg table
 empty_df.write \
    .format("iceberg") \
    .mode("overwrite") \
    .saveAsTable("iceberg.iceberg.t_<table_name>")  # replace with your actual
catalog.db.table
```

Creating a partition table through Jhub

```
#creating a spark session

from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Create Iceberg Table") \
    .config("spark.sql.catalog.iceberg", "org.apache.iceberg.spark.SparkCatalog") \
    .config("spark.sql.catalog.iceberg.type", "hive") \
    .config("spark.sql.catalog.iceberg.warehouse", "s3a://resource-na") \
    .getOrCreate()
```

```
from pyspark.sql.types import StructType, StructField, StringType, DateType

 # Define the schema
 schema = StructType([
    StructField("subscription_id", StringType(), True),
    StructField("customerid", StringType(), True),
    StructField("requested_date", DateType(), True),
    StructField("pur1033", StringType(), True)
 ])
```

```
# Create an empty DataFrame
 empty_df = spark.createDataFrame([], schema)


# Save it as an Iceberg table
 empty_df.write \
   .format("iceberg") \
   .mode("overwrite") \  # Use 'overwrite' to replace an existing table, 'append' to add data
   .partitionBy("<column_name>”) \  # Partition column
   .saveAsTable("iceberg.iceberg.t_<table_name>")
```

NOTE: If a table with data needs to be created make sure the data is passed while creating the data frame

SQL queries

<u>To query the access bucket data in superset</u>

Choose:

- Database Arcus Iceberg
- Schema à Iceberg
- See table schema search for the stream you are looking for

After this, query the required information like the select, desc queries.

<u>GX queries (in Jupiter Notebook)</u>

```
query = f"""select expectation_suite_name, value from ge_expectations_store where expectation_suite_name like '%<source name>%' order by expectation_suite_name  """

df = spark.read.format("jdbc").option("url", source_url).option("query", query).option("user", username).option("password", password).option("driver", "org.postgresql.Driver").load()

df.show(1000,truncate=False)
```

Reconciliation

Here we compare CDL and Arcus data and row count for all the streams in a source

Comparing the data CDL and Arcus

- First identify the corresponding rows between the old and new platforms. This is done through a trial-and-error method, where we attempt to find matching rows from both platforms. SQL queries are written to find the matching data.
- Once we have identified the matching rows, we then proceed to compare each column of the matching row between the two platforms.

Example:

select * from t_sweb2x_rioprag_cdrgprsoutgoing where chdur_s = '000525' and ddv = 39466519

Assuming chdur_s and ddv columns have unique data. We start comparing the query result from both CDL and Arcus.

Do the comparison in excel sheet for better readability

Glossary / Acronym

Squad: Classifies sources into multiple divisions.

Stream: It says how the data flows in from source.

na: Network Assurance

ra: Revenue Assurance

sa: Service Assurance

tfi: Telia Finance

sweb2x: Sweden B2X

Arcus Access Master: Arcus Access Master.xlsx

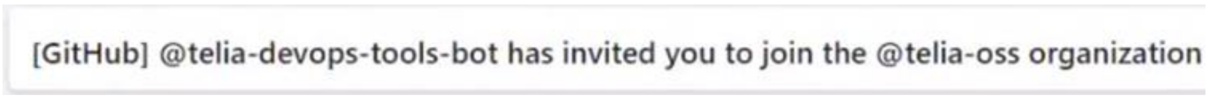
Point of Contact for squads

Sl.no	Squad	POC
1.	sa	suyesh.amatya@teliacompany.com

2.	na	barbara.krolo@teliacompany.com
3.	ra	remya.poulose@teliacompany.com
4.	tfi	viktorija.uzusiene@teliacompany.com
5.	crowd	barbara.krolo@teliacompany.com
6.	sweb2x	remya.poulose@teliacompany.com

Steps to set-up

Step 1: Go to the mail received from GitHub with the subject below.


[GitHub] @telia-devops-tools-bot has invited you to join the @telia-oss organization

- Click on join.
- Enter the credentials.
- Generate github token in settings →developer settings → Generate token.
- Authenticate from the other 2 mails received from GitHub

Step 2:

In developer settings → Personal access. Click on configure SSO and make sure there are 3 accounts authenticate each one of them.

Step 3:

Click on the link below.

https://jfrog.teliacompany.io/ui/user_profile

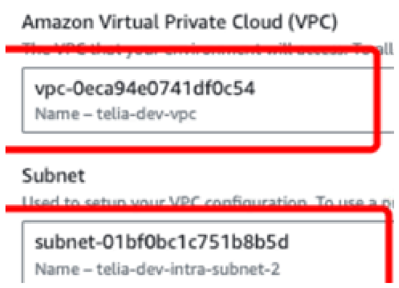
Click on generate API key. Copy and save it.

Step 4: Creating Red Shift credentials (skip it for TFI squad)

- Open AWS apps
- Click on telia-cps-cirrus-dev→ SE_DEV_Developer
- Search for redshift
- On the right plane click on query-editor v2.
- Click on redshift-cluster on the left side. Enter details:
    - Select temporary credentials.
    - Database: se_adw
    - Username: <telia-mail>

Step 5: Create an AWS instance

- Name: <tcad-id>
- Use this VPC and Subnet and create instance.


Amazon Virtual Private Cloud (VPC)

vpc-0eca94e0741df0c54
Name – telia-dev-vpc

Subnet

subnet-01bf0bc1c751b8b5d
Name – telia-dev-intra-subnet-2

Step 6: echo all information to .cirrus_profile file in your home directory.

echo "

export GIT_EMAIL=<put your git email here>

export GIT_USER=<put your GitHub username here>

export GIT_PASSWORD=<put your token here>

export REDSHIFT_HOST=redshift-cluster.cb65mjkgzoqx.eu-north-1.redshift.amazonaws.com

export REDSHIFT_PORT=5439

export REDSHIFT_USER=<put your Redshift username here>

export REDSHIFT_PASSWORD=<put your Redshift password here>

export REDSHIFT_DB=<put your Redshift password here>

export JFROG_USER=<put your jfrog user here>

export JFROG_API_TOKEN=<put your jfrog token here>

export PIP_INDEX_URL=https://\$JFROG_USER:\$JFROG_API_TOKEN@jfrog.teliacompany.io/artifactory/api/pypi/cirrus-python/simple

export CIRRUS_TENANT=<put your tenant name here>

export CIRRUS_ENV=dev

export DBT_PROFILES_DIR=.

export AWS_PROFILE=dev

export COMPOSE_DOCKER_CLI_BUILD=1

export DOCKER_BUILDKIT=1

source ~/venv_airflow/bin/activate

" > ~/.cirrus_profile

Step 7: Install Python and create python virtual environment

sudo dnf update -y

sudo dnf install -y python3.11 python3.11-pip python3.11-devel

python3.11 -m pip install --upgrade pip

python3.11 -m venv ~/venv_airflow/

source ~/venv_airflow/bin/activate

Step 8: Run command to configure your AWS profile. Press Enter when prompting the following questions:

aws configure sso

After the command is executed, we will receive a link and a code. Open the link in browser and enter the code. Chose proper account (dev) and role.

Step 9: Run below command to configure GIT with previously set variables:

git config --global credential.helper '!f() { sleep 1; echo "username=${GIT_USER}"; echo "password=${GIT_PASSWORD}"; }; f'

git config --global user.name $GIT_USER

git config --global user.email $GIT_EMAIL

Confluence page:
https://itwiki.atlassian.teliacompany.net/pages/viewpage.action?pageId=984188143&spaceKey=CIRRUS&title=Cloud9%2Bfor%2BDBT%2Band%2BAirflow%2Bdevelopment