4 - DataFlow Overview // Onboarding

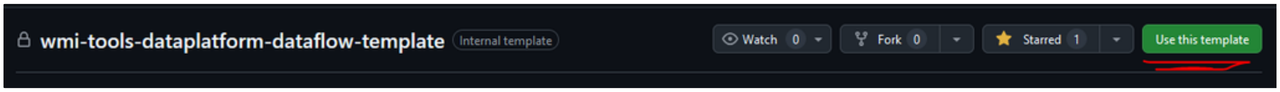
Introduction

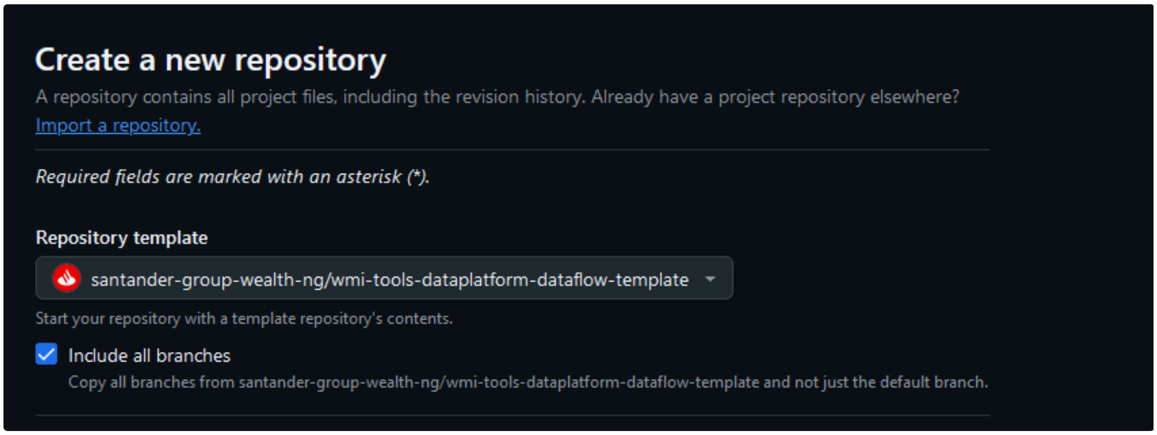
How do I generate the repository?

The repository should be generated using the following repository as a template https://github.com/santander-group-wealth-ng/wmi-tools-dataplatform-dataflow-template Connect your Github account

The steps to follow are as follows:

- Click on the `Use this template` button





Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Repository template

santander-group-wealth-ng/wmi-tools-dataplatform-dataflow-template

Start your repository with a template repository's contents.

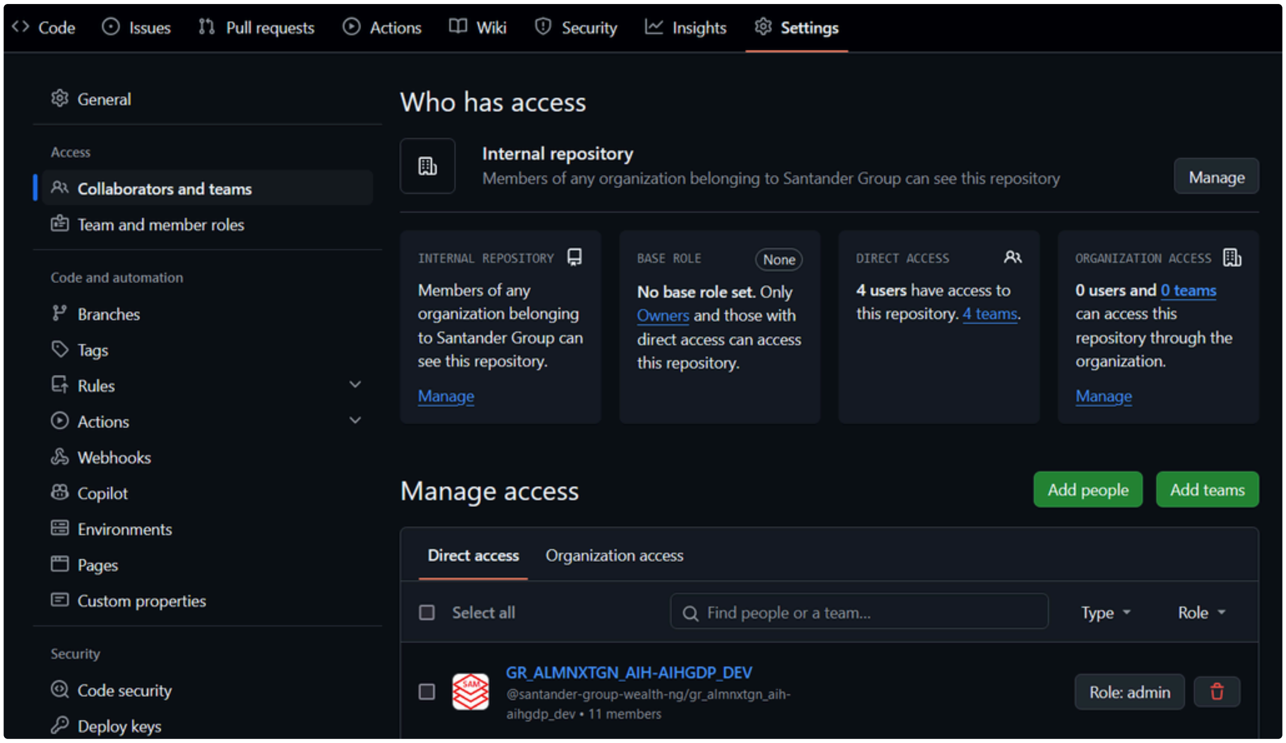
☑ Include all branches

Copy all branches from santander-group-wealth-ng/wmi-tools-dataplatform-dataflow-template and not just the default branch.

- Important enable `include all branches`
- The name that should be given to the repository must follow the following format
  - aih-dataplatform-python-dataflow-{project name}_{data flow name} → aih-dataplatform-python-dataflow-morningstar_classes

> ⚠ The user which done this task should remove his nominative permission and should add the GR_ALMNXTGN_AIH-AIHGDP_DEV team with admin role.

- The way for changing the permissions of the repository is going to “Settings” → “Collaborators and teams” → “Manage access”. First, the technician should add the GR_ALMNXTGN_AIH-AIHGDP_DEV team by clicking on the button “Add teams” and give the admin role. Then, he should remove his self's nominative permission.



Who has access

Internal repository
Members of any organization belonging to Santander Group can see this repository
Manage

INTERNAL REPOSITORY
Members of any organization belonging to Santander Group can see this repository.
Manage

BASE ROLE None
No base role set. Only Owners and those with direct access can access this repository.

DIRECT ACCESS
4 users have access to this repository. 4 teams.

ORGANIZATION ACCESS
0 users and 0 teams can access this repository through the organization.
Manage

Manage access
Add people
Add teams

Direct access | Organization access

□ Select all | Find people or a team... | Type | Role

□ GR_ALMNXTGN_AIH-AIHGDP_DEV
@santander-group-wealth-ng/gr_almnxtgn_aih-aihgdp_dev • 11 members
Role: admin

Repository Structure

Once the **dataflow repository** is created and configured we will have a directory structure like this one:

```
Proyect/
├── .github/
├── data_catalog/
│   └── data_catalog.json
├── data_quality/
│   ├── rule1.gdq
│   └── rule2.gdq
├── test/
│   ├── files/
│   │   └── file1.txt
│   └── test_1.py
├── transformation/
```

```
│   ├── src/
│   │   ├── dataflow/
│   │   │──── default  ├── raw_to_staging.py
│   │   │               ├── staging_to_common.py
│   │   ├── utils/
│   │   └── tests/

│   ├── transformation.json
```

The .**github** folder in a newly created repository typically contains configuration files related to repository management and automation. Here's what it may include:

- `workflows/`:
  - Contains YAML files that define GitHub Actions workflows for CI/CD or other automation processes.
  - You can configure action folders to organize required and auxiliary actions more effectively.

> ⚠ This folder is synchronized with the template and cannot be changed its content.

data_catalog folder

The **data_catalog** folder contains a `data_catalog.json` file that is a structured JSON document used to describe and manage the datasets within a project. It acts as a central repository of metadata, providing detailed information about the datasets, their structure, sources, and relationships. This file is particularly useful for organizing, documenting, and automating data workflows.

Dictionary Schema

`data_catalog.json`

```
note: (1) Appears one time
      (n) Appears n times
[
  (n) {
    (1)"name": "<table_name>",
    (1)"db": "<database_name>",
    (0,1)"external": "<value>",
    (1) "columns": [
      {
       (n)"FieldName": "<type>"
      }
    ],
    (1) "partitions": [
      (n){
         (1) "PartitionKey": "<partition_key>",
         (0,1) "Type": "<data_type>"
      }
    ],
    (0,1) "format": "<format_type>",
    (0,1) "extra_sentences": "<additional_configurations>",
    (1) "table_properties": "<value>",
    (1) "LfTags": [
      (n){
        (1)"TagKey": "<tag>",
        (1) "TagValues": [
           (n)"<value>"

    ]
   }
  ]
 }
]


```

KEY	DEFINITION	OPTIONS	NOTE
name	Table Name		The name should have the words separeted by underscore, for example, “sells_rips_prod”.
external	Table Type	`EXTERNAL`	Not applicable to Iceberg tables, such as those in staging or common layers
db	Database Name	`<db>.<name>`	The name should have the words separeted by underscore, and the name of the layer at the end, for example, “funds_raw”.
columns	Column Definitions	`(column_name column_type, ...)`	It should be a dictionary with the keys with the name of the column and the value with the type of the fields.
partitions	Partition Keys	`PARTITIONED BY (partition_column column_type)`	Iceberg tables do not include the type because it is defined in the column tags. Athena tables, on the other hand, include the type, but the field is not defined within the column tags.
format	Storage Format	`STORED AS <format>` (e.g., `TEXTFILE`, `PARQUET`)	Not applicable to Iceberg tables, such as those in staging or common layers
extra_sentences	Row Format	`ROW FORMAT DELIMITED FIELDS TERMINATED BY '<delimiter>'`	Not applicable to Iceberg tables, such as those in staging or common layers
table_properties	Table Properties	`TBLPROPERTIES ("key"="value")`	
LFTags	Tags (Lake Formation or Glue Catalog tags)	Not part of DDL (applied post-creation by step_function `addLFTagToResource)`	It’s a list of dictionaries with name of the TagKey and a list of TagValues.

Example:

Click here to expand

```
[
 {
  "name": "sells_rips_prod",
  "db": "funds_raw",
  "external": "EXTERNAL",
  "columns": {
   "SecId":"string",
   "PerformanceId":"string",
   "Date":"string",
   "Deleted":"string",
   "Unit_BAS":"string",
   "Unit_USD":"string",
   "Unit_EUR":"string",
   "Unit_GBP":"string",
   "Unit_CHF":"string"
  },
  "partitions":[
   {
     "PartitionKey": "DataDate",
     "Type": "string"
   }
  ],
  "format": "TEXTFILE",
  "extra_sentences": "ROW FORMAT DELIMITED FIELDS TERMINATED BY ';'",
  "table_properties": "\"skip.header.line.count\"=\"1\"",
  "LfTags": [
   {
     "TagKey": "domain",
     "TagValues": [
       "funds"
     ]
   },
   {
     "TagKey": "taxonomy",
     "TagValues": [
       "public"
     ]
   }
  ]
 },
 {
  "name": "sells_rips_correction",
  "external": "EXTERNAL",
  "db": "funds_raw",
  "columns": {
   "SecId":"string",
   "PerformanceId":"string",
   "Date":"string",
   "Deleted":"string",
   "Unit_BAS":"string",
   "Unit_USD":"string",
   "Unit_EUR":"string",
   "Unit_GBP":"string",
   "Unit_CHF":"string",
   "Unit_DKK":"string",
   "Unit_NOK":"string"
  },
  "partitions":[
   {
     "PartitionKey": "DataDate",
     "Type": "string"
   }
  ],
  "format": "TEXTFILE",
  "extra_sentences": "ROW FORMAT DELIMITED FIELDS TERMINATED BY ';'",
  "table_properties": "\"skip.header.line.count\"=\"1\"",
  "LfTags": [
   {
     "TagKey": "domain",
     "TagValues": [
       "funds"
     ]
   },
   {
     "TagKey": "taxonomy",
     "TagValues": [
       "public"
     ]
   }
  ]
 },
 {
  "name": "sells_rips",
  "db": "funds_staging",
  "columns": {
   "SecId":"string",
   "PerformanceId":"string",
   "Date":"date",
   "Deleted":"boolean",
   "Unit_BAS":"double",
   "Unit_USD":"double",
   "Unit_EUR":"double",
   "Unit_GBP":"double",
   "Unit_CHF":"double",
   "Unit_DKK":"double",
   "DataDate":"date"
  },
  "partitions":[
   {
     "PartitionKey": "DataDate"
   }
  ],
  "table_properties": "'table_type'='ICEBERG', 'format'='parquet', 'write_compression'='snappy'",
  "LfTags": [
   {
     "TagKey": "domain",
     "TagValues": [
       "funds"
     ]
   },
   {
     "TagKey": "taxonomy",
     "TagValues": [
       "public"
     ]
   }
  ]
 },
 {
  "name": "sells_rips_spark",
  "db": "funds_staging",
  "columns": {
   "SecId":"string",
   "PerformanceId":"string",
   "Date":"date",
   "Deleted":"boolean",
   "Unit_BAS":"double",
   "Unit_USD":"double",
   "Unit_EUR":"double",
   "Unit_GBP":"double",
   "Unit_CHF":"double",
   "Unit_DKK":"double",
   "Unit_NOK":"double",
   "Unit_SEK":"double",
   "Unit_JPY":"double",
   "LastUpdate":"timestamp",
   "Unit_SGD":"double",
   "ReturnType":"int",
   "Filled":"boolean",
   "Unit_TWD":"double",
   "Unit_HKD":"double",
   "Unit_MYR":"double",
   "Unit_CNY":"double",
   "Unit_ILS":"double",
   "Unit_INR":"double",
   "Unit_CAD":"double",
   "Unit_KWD":"double",
   "Unit_PLN":"double",
   "Unit_AUD":"double",
   "Unit_THB":"double",
   "Unit_KRW":"double",
   "Unit_NZD":"double",
   "DataDate": "date"
  },
  "partitions":[
   {
     "PartitionKey": "DataDate"
   }
  ],
  "table_properties": "'table_type'='ICEBERG', 'format'='parquet', 'write_compression'='snappy'",
  "LfTags": [
   {
     "TagKey": "domain",
     "TagValues": [
       "funds"
     ]
   },
   {
     "TagKey": "taxonomy",
     "TagValues": [
       "public"
     ]
   }
  ]
 },
 {
  "name": "sells_rips",
  "db": "funds_common",
  "columns": {
   "SecId":"string",
   "PerformanceId":"string",
   "Date":"date",
   "Deleted":"boolean",
   "Unit_BAS":"double",
   "Unit_USD":"double",
   "Unit_EUR":"double",
   "Unit_GBP":"double",
   "Unit_CHF":"double",
   "Unit_DKK":"double",
   "Unit_NOK":"double",
   "Unit_SEK":"double"
  },
  "partitions":[
   {
     "PartitionKey": "Date"
   }
  ],
  "table_properties": "'table_type'='ICEBERG', 'format'='parquet', 'write_compression'='snappy'",
  "LfTags": [
   {
     "TagKey": "domain",
     "TagValues": [
       "funds"
     ]
   },
   {
     "TagKey": "taxonomy",
     "TagValues": [
       "public"
     ]
   }
  ]
 },
 {
  "name": "sells_rips_spark",
  "db": "funds_common",
  "columns": {
   "SecId":"string",
   "PerformanceId":"string",
   "Date":"date",
   "Deleted":"boolean",
   "Unit_BAS":"double",
   "Unit_USD":"double",
   "Unit_EUR":"double",
   "DataDate": "date"
  },
  "partitions":[
   {
     "PartitionKey": "Date"
   }
  ],
  "table_properties": "'table_type'='ICEBERG', 'format'='parquet', 'write_compression'='snappy'",
  "LfTags": [
   {
     "TagKey": "domain",
     "TagValues": [
       "funds"
     ]
   },
   {
     "TagKey": "taxonomy",
     "TagValues": [
       "public"
     ]
   }
  ]
 }
]
```

data_quality folder

The **data_quality** folder contains different files that include the data quality rules written in Glue's Data Quality Language (Data Quality Definition Language (DQDL) reference - AWS Glue). Here is an example:

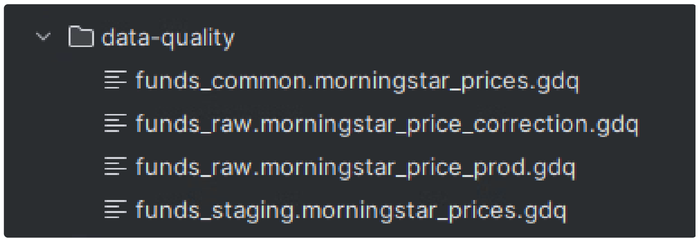
```
#Checks the primary key
IsPrimaryKey "performanceid" "date"
#Checks the column type
ColumnDataType "datadate" = "Date"
#Checks the format
CustomSql "SELECT COUNT(*) FROM primary WHERE CAST(dataDate AS STRING) REGEXP '^\d{4}-\d{2}-\d{2}$'" = 0
```

There is no separator between sentences. These sentences will be included in `data_catalog.json` each time there is a CI/CD action. Although it is not visible in the repository, it will be loaded into the AWS account.

For quality rules, a separate file will be created for each table and layer. The naming convention will be as follows: `funds_raw` for the raw layer, `funds_staging` for the staging layer, and `funds_common` for the common layer. After this, separated by a dot, the table name will follow, and then the file extension `.gdq` (Glue Data Quality).

Example of file name format:

{database name}.{table_name}.gdq → funds_common.morningstar_prices.gdq



```
data-quality
    funds_common.morningstar_prices.gdq
    funds_raw.morningstar_price_correction.gdq
    funds_raw.morningstar_price_prod.gdq
    funds_staging.morningstar_prices.gdq
```

The **test** folder contains unit test files along with their corresponding code. The developer can put the source data files used for the tests in the *files* subdirectory.

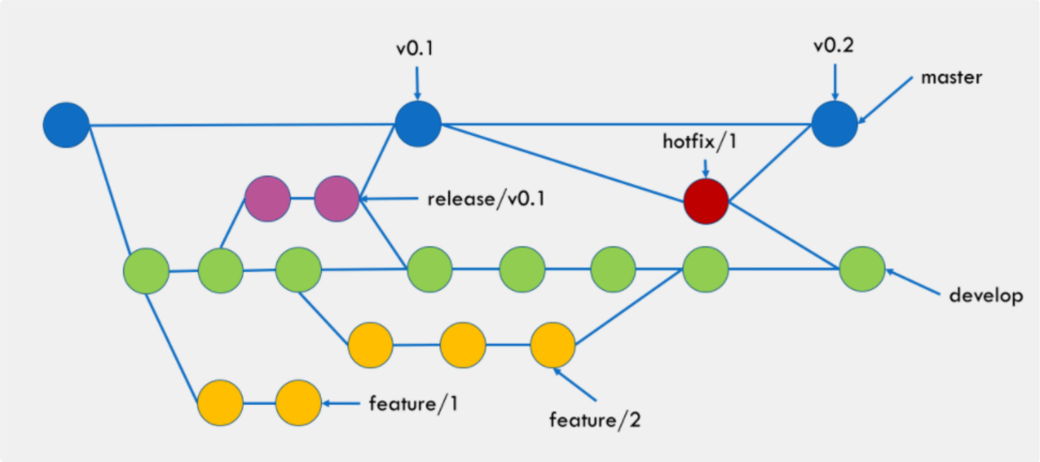
The **transformation** directory contains all the code required to transform data across the various layers of the data lake. A `transformation.json` file is a structured configuration used in data workflows used to define and document the rules and processes for transforming data between different layers in a data lake. It specifies transformation logic (including operations like filtering), renaming columns, and aggregating data. It also describes the output schema, data sources, destinations, and dependencies. The

src/dataflows/default directory contains the specific **python** code for transformations where the changes defined in `transformations.json` for each layer are executed. The utils directory should be used for helper functions or classes, and additional directories within `src/dataflows` for organizing other code as needed.

Structure of transformation.json

Branching Model

We will adopt the **Gitflow** branching model, which follows these rules:



Gitflow model

a. Main Branch (`main`):

- Contains stable and production-ready code.
- Only updated with versions that have passed thorough testing.

b. Development Branch (`develop`):

- Contains the latest version of the code under development.
- Serves as the base for new features and team collaboration.

c. Support Branches:

i. Feature Branches (`feature`):

- Created from `develop`.
- Used to develop new features.
- Merged back into `develop` once complete.

ii. Release Branches (`release`):

- Created from `develop` when preparing a new version for production.
- Used for final adjustments, bug fixes, and testing.
- Merged into both `main` and `develop`.
- Example: `release/1.0`.

iii. Hotfix Branches (`hotfix`):

- Created from `main` to address critical issues in production.

- Merged into both `main` and `develop`.
- Sometimes, although it is rare, hotfixes can be applied to release branches.

3. CI/CD

CI/CD (Continuous Integration/Continuous Deployment) is a software development practice that automates the process of integrating code changes, testing, and deploying applications.

Which kind of action trigger ci/cd?

These are the most common reasons:

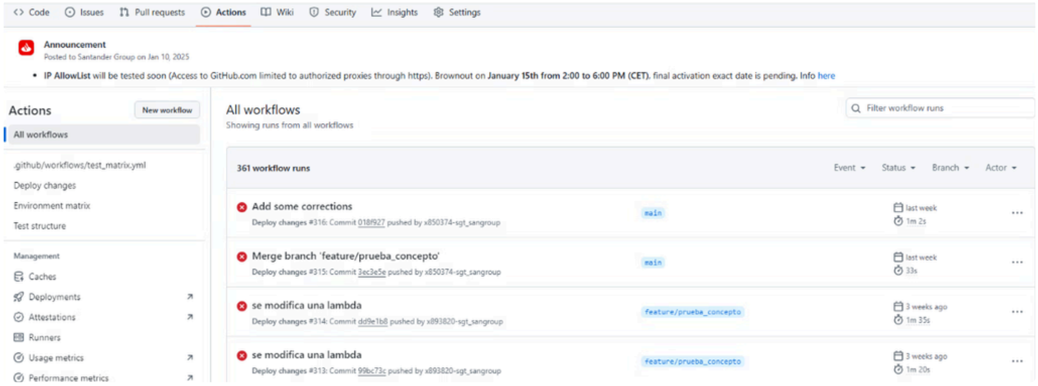
- **Code Changes:**
  - Pushing code to a repository.
  - Merging a pull request.
- **Manual Triggers:**
  - Workflow dispatch or manually starting a pipeline.

The most common trigger for this is a push action. It's worth emphasizing that the CI/CD process identifies the modified code and deploys only the affected parts. For example, if `data_catalog.json` is modified, only the CI/CD process related to the data catalog will be triggered.

Depending on your location, the trigger is executed in its corresponding environment. So if you are in `develop`, it will run the workflow in the `develop` repository; if you are in a feature, it will run in its respective repository; or if you are in `main`, it will run in the `main` repository.

It automatically upload and deploy catalog, transformations and orchestration processes.

If you want to check the workflow you will have to go to the **Actions** section in GitHub:



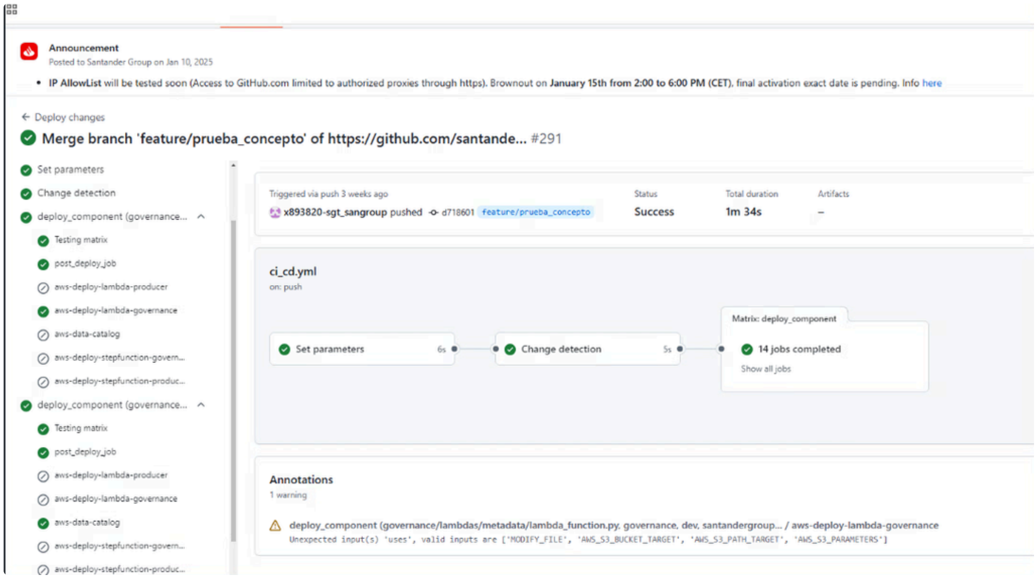
Announcement
Posted to Santander Group on Jan 10, 2025

- **IP AllowList** will be tested soon (Access to GitHub.com limited to authorized proxies through https). Brownout on **January 15th from 2:00 to 6:00 PM (CET).** final activation exact date is pending. Info here

All workflows
Showing runs from all workflows

361 workflow runs

Add some corrections Deploy changes #316: Commit 018f927 pushed by x850374-sgt_sangroup	`main`	last week 1m 2s
Merge branch 'feature/prueba_concepto' Deploy changes #315: Commit 3ec3e5e pushed by x850374-sgt_sangroup	`main`	last week 33s
se modifica una lambda Deploy changes #314: Commit dd9e1b8 pushed by x893820-sgt_sangroup	`feature/prueba_concepto`	3 weeks ago 1m 35s
se modifica una lambda Deploy changes #313: Commit 99bc73c pushed by x893820-sgt_sangroup	`feature/prueba_concepto`	3 weeks ago 1m 20s


Set parameters
Change detection
deploy_component (governance...
Testing matrix
post_deploy_job
aws-deploy-lambda-producer
aws-deploy-lambda-governance
aws-data-catalog
aws-deploy-stepfunction-govern...
aws-deploy-stepfunction-produc...
deploy_component (governance...
Testing matrix
post_deploy_job
aws-deploy-lambda-producer
aws-deploy-lambda-governance
aws-data-catalog
aws-deploy-stepfunction-govern...
aws-deploy-stepfunction-produc...
Set parameters 6s
Change detection 5s
Matrix: deploy_component
14 jobs completed
Show all jobs

Announcement

Posted to Santander Group on Jan 10, 2025

- **IP AllowList** will be tested soon (Access to GitHub.com limited to authorized proxies through https). Brownout on **January 15th from 2:00 to 6:00 PM (CET)**, final activation exact date is pending. Info here

← Deploy changes

Merge branch 'feature/prueba_concepto' of https://github.com/santande... #291

Triggered via push 3 weeks ago	Status	Total duration	Artifacts
x893820-sgt_sangroup pushed -o- d718601 `feature/prueba_concepto`	Success	1m 34s	–

ci_cd.yml

on: push

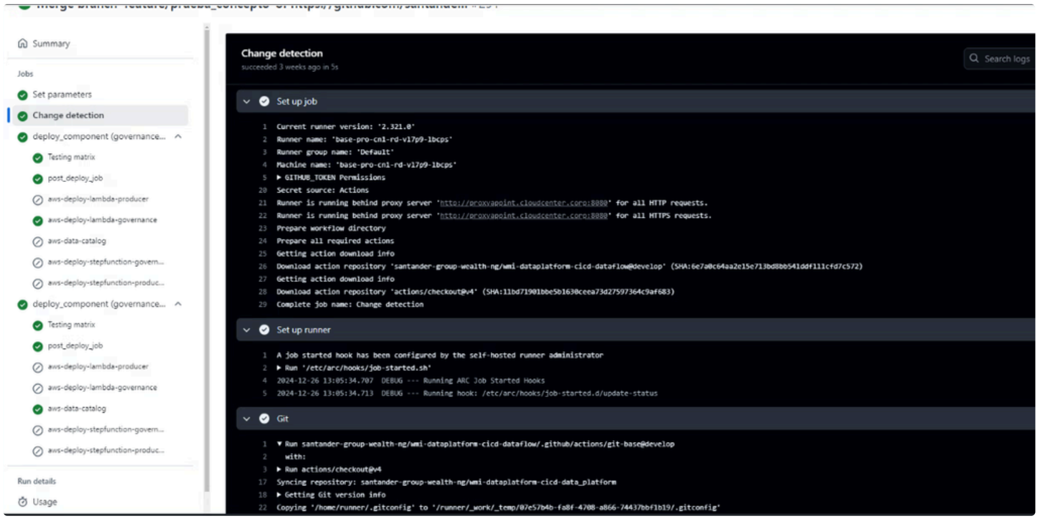
Annotations

1 warning

⚠ deploy_component (governance/lambdas/metadata/lambda_function.py, governance, dev, santandergroup... / aws-deploy-lambda-governance

```
Unexpected input(s) 'uses', valid inputs are ['MODIFY_FILE', 'AWS_S3_BUCKET_TARGET', 'AWS_S3_PATH_TARGET', 'AWS_S3_PARAMETERS']
```

You can click on each job in the workflow to view the corresponding log:


Summary
Jobs
Set parameters
Change detection
deploy_component (governance...
Testing matrix
post_deploy_job
aws-deploy-lambda-producer
aws-deploy-lambda-governance
aws-data-catalog
aws-deploy-stepfunction-govern...
aws-deploy-stepfunction-produc...
deploy_component (governance...
Testing matrix
post_deploy_job
aws-deploy-lambda-producer
aws-deploy-lambda-governance
aws-data-catalog
aws-deploy-stepfunction-govern...
aws-deploy-stepfunction-produc...
Run details
Usage
Search logs

Change detection

succeeded 3 weeks ago in 5s

Set up job

```
Current runner version: '2.321.0'
Runner name: 'base-pro-cn1-rd-v17p9-1bcps'
Runner group name: 'Default'
Machine name: 'base-pro-cn1-rd-v17p9-1bcps'
▶ GITHUB_TOKEN Permissions
Secret source: Actions
Runner is running behind proxy server 'http://proxyapoint.cloudcenter.corp:8080' for all HTTP requests.
Runner is running behind proxy server 'http://proxyapoint.cloudcenter.corp:8080' for all HTTPS requests.
Prepare workflow directory
Prepare all required actions
Getting action download info
Download action repository 'santander-group-wealth-ng/wmi-dataplatform-cicd-dataflow@develop' (SHA:6e7a0c64aa2e15e713bd8bb541ddf111cfd7c572)
Getting action download info
Download action repository 'actions/checkout@v4' (SHA:11bd71901bbe5b1630ceea73d27597364c9af683)
Complete job name: Change detection
```

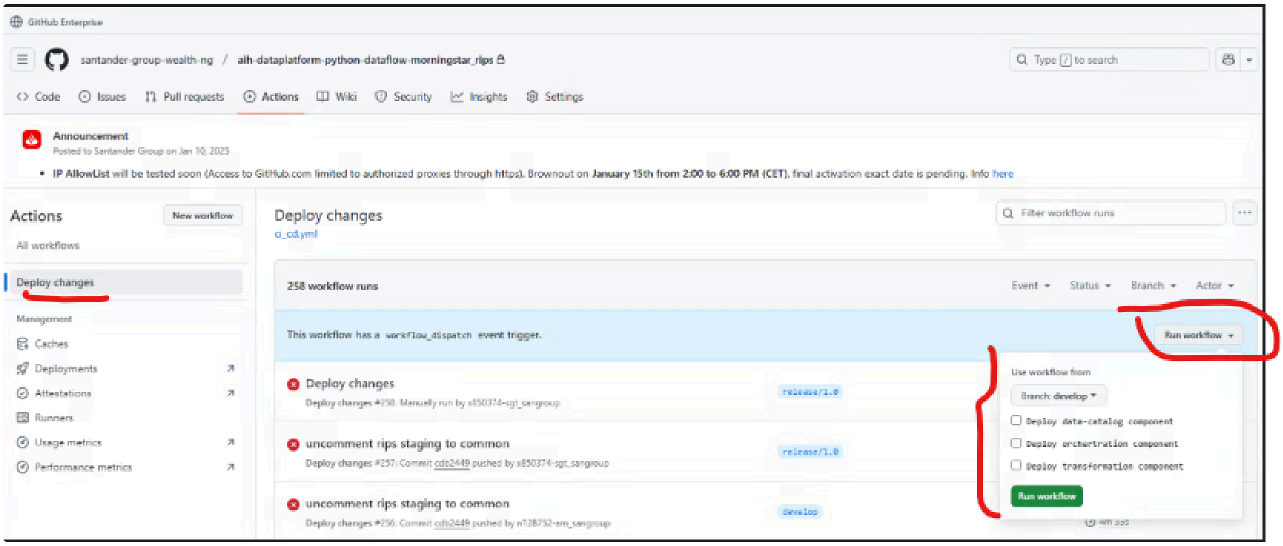
Set up runner

```
A job started hook has been configured by the self-hosted runner administrator
▶ Run '/etc/arc/hooks/job-started.sh'
2024-12-26 13:05:34.707  DEBUG --- Running ARC Job Started Hooks
2024-12-26 13:05:34.713  DEBUG --- Running hook: /etc/arc/hooks/job-started.d/update-status
```

Git

```
▼ Run santander-group-wealth-ng/wmi-dataplatform-cicd-dataflow/.github/actions/git-base@develop
  with:
▶ Run actions/checkout@v4
Syncing repository: santander-group-wealth-ng/wmi-dataplatform-cicd-data_platform
▶ Getting Git version info
Copying '/home/runner/.gitconfig' to '/runner/_work/_temp/07e57b4b-fa8f-4708-a866-74437bbf1b19/.gitconfig'
```

It is also possible to trigger a workflow without performing any action on the repository, from **"deploy changes"**, with the option to choose what you want to launch and from which branch:



Announcement
Posted to Santander Group on Jan 10, 2025

- IP AllowList will be tested soon (Access to GitHub.com limited to authorized proxies through https). Brownout on January 15th from 2:00 to 6:00 PM (CET). final activation exact date is pending. Info here

Deploy changes
ci_cd.yml

258 workflow runs

This workflow has a `workflow_dispatch` event trigger.

Deploy changes
Deploy changes #258: Manually run by x850374-sgt_sangroup

uncomment rips staging to common
Deploy changes #257: Commit cdb2449 pushed by x850374-sgt_sangroup

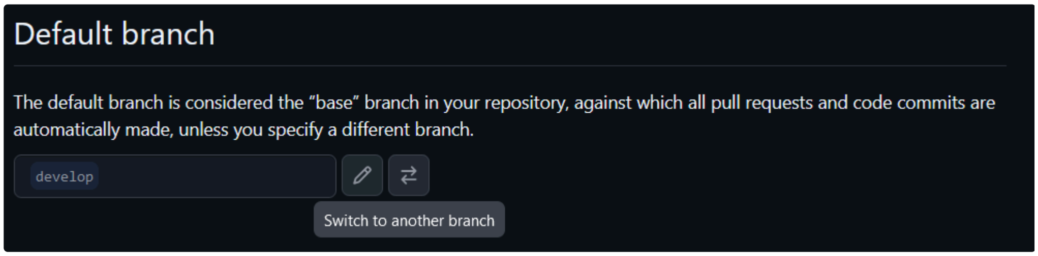
uncomment rips staging to common
Deploy changes #256: Commit cdb2449 pushed by n128752-em_sangroup

Use workflow from
Branch: develop

☐ `Deploy data-catalog component`
☐ `Deploy orchertration component`
☐ `Deploy transformation component`

Run workflow

In order to “Run workflow” button appear, the workflow must to have been runned in the default branch of the repository. By default the main branch is the default branch, so, if a commit hasn’t been done in the main branch (main), the button will not be visible. You should go to “Settings” → “General” → “Default branch” → “Switch to another branch” and select “develop” for make “Run workflow” button visible.



Default branch

The default branch is considered the “base” branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.

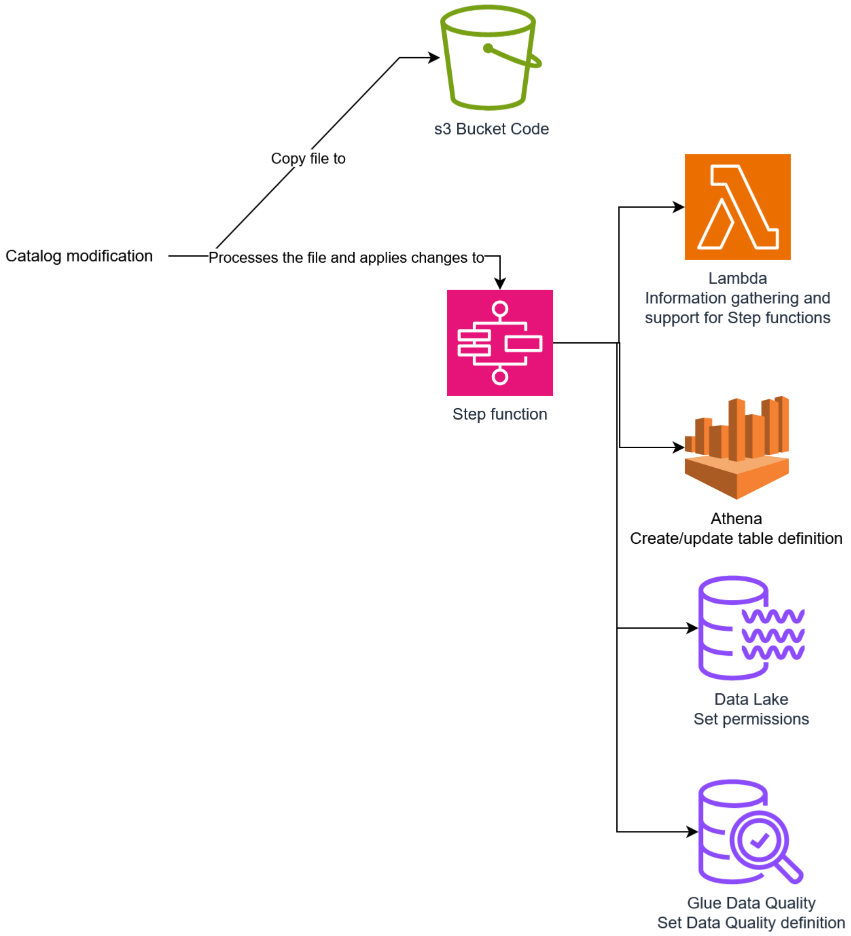
develop

Default branch configuration menu

It is important to note that the CI/CD pipeline includes a step for **Fortify**, a tool used to analyze source code for vulnerabilities and security issues. This process scans the dataflow code using Fortify to detect potential security problems. Additionally, the pipeline includes a process for **Sonar**, a tool that assesses code quality and identifies issues such as bugs, vulnerabilities, and code smells. It is important to highlight that in the **dev** environment, passing Fortify and Sonar is not mandatory, but it is required in the **pre** and **pro** environments.

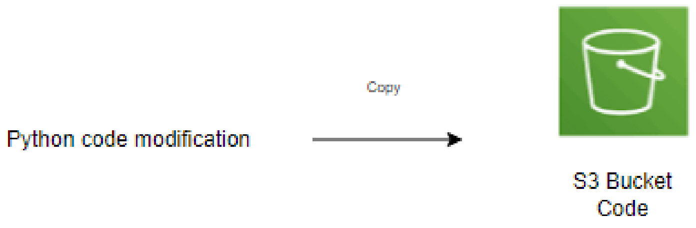
There are three steps that the CI/CD process is prepared to deploy:

Catalog

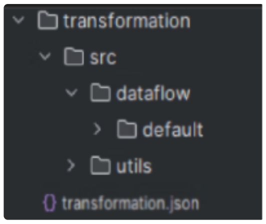

s3 Bucket Code
Copy file to
Catalog modification
Processes the file and applies changes to
Lambda
Information gathering and
support for Step functions
Step function
Athena
Create/update table definition
Data Lake
Set permissions
Glue Data Quality
Set Data Quality definition

Each modification to the data_catalog.json file triggers two actions. The first one copies the code to its corresponding bucket, and the second calls the step function that applies the changes to the database and Glue Data Catalog (updating metadata, tables and quality rules)

Python transformation code

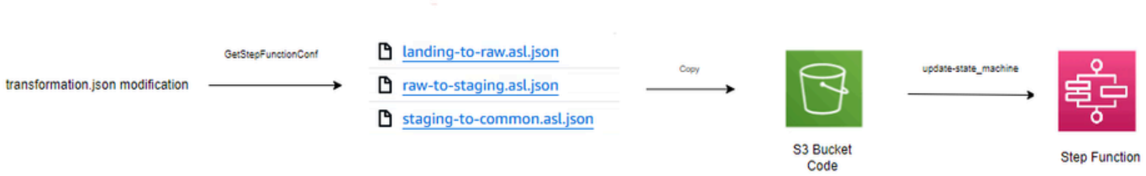


Any change in the Python code within the transformation directory requires uploading all the transformation code, compressed, to the S3 bucket.



```
transformation
  src
    dataflow
      default
    utils
  transformation.json
```

Transformation.json



This process automates the update of **AWS Step Functions** by generating new `.asl.json` definitions based on a `transformations.json` file, storing them in an S3 bucket, and then using them to update the corresponding Step Functions. Each `.asl.json` file describes the workflow logic for a specific layer.