# Data Processing Framework

## ****Overview****

##### The ****Data Processing Framework (DPF)**** is a modular, reusable, and agent-driven data orchestration system built using ****Python****, ****PySpark****, and ****Apache Airflow****, designed specifically for ****Cloudera Data Engineering (CDE)****. The framework enables execution of SQL queries, formatting of Excel reports, and automated email dispatch - suitable for enterprise-scale data processing pipelines.

## ****Objective****

##### To develop a ****scalable****, ****maintainable****, and ****standardized**** pipeline orchestration system that:

##### Supports generic execution of SQL transformations using parameterized inputs.

##### Automates report generation in excel format and report sharing via email.

## ****Technology Stack****

|  |  |
| --- | --- |
| ****Component**** | ****Version / Notes**** |
| Pytho | 3.10+ |
| PySpark | 3.5+ |
| Cloudera Data Engineering (CDE) | On Kubernetes |
| Apache Airflow | Deployed on Kubernetes |
| pandas | Used for Excel transformation |
| openpyxl, xlsxwriter | Excel file formatting |
| ruff | For linting and code quality enforcement |

## ****Framework Components****

### ****SQL Executor Job****

##### ****Purpose****

##### Execute parameterized SQL queries using Spark, log extensive execution metrics, and write results to data lake tables.

##### ****Features****

##### Accept SQL files with runtime parameters (supports Impala/Hive syntax).

##### Dynamically accept parameters from JSON configuration files.

##### Calculates necessary data partitions to be used for a particular run.

##### Execute queries in multi-stage pipelines using Spark SQL.

##### Log query metrics to dpf\_processing\_log table:

##### log\_id - Unique log identifier

##### dag\_name - Airflow DAG name

##### dag\_run\_id - DAG execution instance ID

##### job\_name - Job name within DAG

##### job\_run\_id - Job execution instance ID

##### stg\_name - Stage name identifier

##### run\_date - Job execution date

##### src\_query\_file\_path - SQL file path

##### tgt\_owner - Target table owner

##### tgt\_table - Target table name

##### tgt\_strategy - Load strategy (e.g., overwrite)

##### start\_time - Stage start timestamp

##### end\_time - Stage end timestamp

##### status - Execution status

##### row\_count - Number of rows loaded

##### loaded\_by - User or system loader

##### log\_entry\_time - Log record timestamp

##### stg\_description - Stage description text

##### tgt\_schema - Target schema name

##### query\_config - Query configuration details

##### exception\_message - Error or exception message

##### bdp\_date\_id – Partition date ID (yyyymmdd)

##### ****Arguments****

##### --run\_date: Optional; defaults to current date if not provided.

##### --src\_query\_file\_path: SQL query file location.

##### --query\_config: JSON variable with query parameters

##### --tgt\_strategy: Table write strategy (e.g. Insert Overwrite)

##### --tgt\_owner: Target database name.

##### --tgt\_table: Target table name.

##### ****Output****

##### Data written to ****data lake tables**** (with partition).

##### Temp view creation for intermediate data.

##### ****Best Practices****

##### Using consistent naming for job names, stages, and tables.

##### Avoiding excessive Spark stages; optimized query grouping.

### 2. ****Email Generator Job****

##### ****Purpose****

##### To generate anomaly reports from SQL queries, format them into clean Excel files, and email the reports with/without tabular content in the email body to target recipients. This job automates the generation and distribution of anomaly reports by combining two key operations: ****Excel formatting**** and ****email delivery****.

##### ****Features****

##### Reads data from final anomaly table created via sql executor job.

##### Convert SQL output into a well-formatted Excel sheet.

##### Supports custom email templates using Jinja2 for dynamic content rendering.

##### Format emails with:

##### Subject and message body as per provided template.

##### Tabular report embedded in the email body.

##### Single-sheet excel attachment with anomaly data.

##### Flexible recipient & content handling via external configuration.

##### Send the emails via SMTP configuration.

##### ****Arguments****

##### ****name:**** Job identifier.

##### ****description:**** Human-readable explanation of the job.

##### ****mail\_type:**** Type of email format.

##### ****mail\_template:**** Path to HTML/Jinja2 email body template.

##### ****mail\_body\_rule\_query\_file:**** SQL file to generate data for email body table.

##### ****mail\_attachment\_rule\_query\_file:**** SQL file to generate data for Excel attachment (can be same or different).

##### template\_root\_dir****:**** Base directory for template files.

##### email ****from addr:**** Sender email address (SMTP authenticated).

##### ****subject:**** Subject line for the email.

##### ****to email:**** Primary recipients (semicolon-separated).

##### ****cc\_email:**** CC recipients (semicolon-separated).

##### ****department, sub unit, criticality:**** Metadata for internal classification.

##### ****Output****

##### A formatted Excel file with anomaly records.

##### An email sent to specified recipients with:

##### Embedded tabular data showing anomalies/metrics.

##### Attached Excel sheet of the anomaly data.

##### Custom message explaining the context of the anomaly.

##### ****Template Options****

##### ****Type 0:**** Notifying recipients when no anomalies are generated.

##### ****Type 1:**** Sending email with mail body anomalies only

##### ****Type 2:**** Sending email with attachment only.

##### ****Type 3:**** Sending email with mail body anomalies and attachment.

## ****In Scope****

* Supporting multi-sheet excel file.

##### User specific email generation.

##### JSON Schema Validation.

## ****Out of scope****

##### Sending multiple email attachments.

##### Supporting more than one tabular in email body.

## ****Coding Standards****

##### ****Linting****: Enforced via ruff, following modern Python code style conventions.

##### ****Modularization****: Each logical unit of work is developed as an independent, testable function.

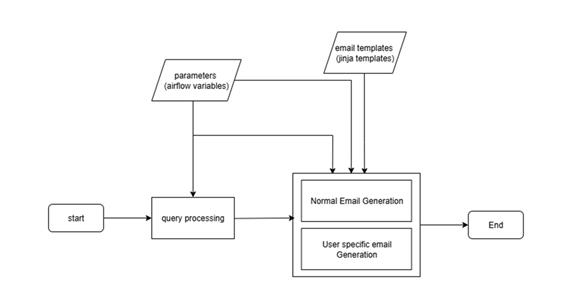
##### ****Documentation****: Every module and function include docstrings and inline comments.

##### ****Logging****: Follows structured logging standards for better observability.

##### ****Reusability****: Jobs can be reused in different DAGs or as standalone scripts.

##### 

##### ****Attached below is a diagrammatic representation:****



## ****Variable Parameter design****

##### Each airflow dag has one corresponding airflow variable.

##### Common environment/platform variables are managed separately.

##### ****Sample airflow variable****

{

"control\_variables": {

"control\_number": "cm\_aos\_control\_3",

"dag\_name": "dpf\_cm\_aos\_control\_3\_dag",

"run\_date": "2025-08-04",

"log\_db": "ds\_dashboard",

"log\_table": "dpf\_processing\_log"

},

"sql\_jobs": [

{

"cde\_job\_name":"dpf\_sql\_executor",

"name": "Preprocess\_Job",

"stages": [

{

"stg\_name": "Control\_3\_Preprocess\_Table",

"stg\_description": "",

"order\_of\_execution": 1,

"src\_query\_file\_path": "/app/mount/scripts/pyscripts/sql\_scripts/cm\_aos\_3\_1\_preprocess.sql",

"tgt\_owner": "ds\_dashboard",

"tgt\_table": "cm\_aos\_3\_1\_preprocess",

"tgt\_strategy": "Partition\_Overwrite",

"query\_config": {

"no\_of\_past\_days": 30,

"stand\_zone\_db": "ds\_dashboard",

"ds\_zone\_db": "ds\_dashboard"

}

}

]

},

{

"cde\_job\_name":"dpf\_sql\_executor",

"name": "Anomaly\_Detect\_Job",

"stages": [

{

"stg\_name": "Detect\_Anomaly",

"stg\_description": "",

"order\_of\_execution": 1,

"src\_query\_file\_path": "/app/mount/scripts/pyscripts/sql\_scripts/cm\_aos\_3\_final\_report.sql",

"tgt\_owner": "ds\_dashboard",

"tgt\_table": "cm\_aos\_3\_1\_final\_report",

"tgt\_strategy": "Partition\_Overwrite",

"query\_config":{

"no\_of\_past\_days": "NA",

"preprocess\_table": "cm\_aos\_3\_1\_preprocess",

"anomaly\_table": "cm\_aos\_3\_1\_final\_report",

"stand\_zone\_db": "ds\_dashboard",

"ds\_zone\_db": "ds\_dashboard"

}

}

]

}

],

"normal\_email\_reports":[

{

"cde\_job\_name": "dpf\_email\_sender",

"name": "Report\_Mail",

"description": "Sending anomalies in the email body",

"mail\_type": "Type-1",

"mail\_type\_description": "Mail with tabulation in body only ",

"mail\_body\_rule\_query\_file": "/app/mount/scripts/pyscripts/sql\_scripts/cm\_control\_3\_rule\_query.sql",

"mail\_attachment\_sheets":[],

"query\_config": {

"ds\_zone\_db": "ds\_dashboard"

},

"template\_root\_dir": "/app/mount/scripts/pyscripts/email\_template/",

"mail\_template": "email\_template.html",

"email\_from\_addr": "controlmonitor\_cde\_dev\_alerts@adib.com",

"subject": "DC-AOS-03 Framework: Account closures without customer request",

"to\_email": "Sanjana.Krishnan@adib.com;Thaanish.Kurumadaki@adib.com;Karthikeyan.Nethaji@adib.com",

"cc\_email": "Maadhusri.Ulaganathan@adib.com;Smrithi.Sundar@adib.com",

"department": "AOS",

"sub\_unit": "AOS",

"criticality": "Low",

"mail\_description": "The below listed accounts do not have entry either in Ethix or BPM system for account closure requests. This defect has high business criticality and requires your immediate attention."

},

{

"cde\_job\_name": "dpf\_email\_sender",

"name": "Report\_Mail",

"description": "Send Anomalies as mail with excel attachment",

"mail\_type": "Type-2",

"mail\_type\_description": "Mail with excel attachment only",

"mail\_body\_rule\_query\_file": "",

"mail\_attachment\_sheets":[

{

"add\_account\_master":"/app/mount/scripts/pyscripts/sql\_scripts/add\_account\_master.sql"

},

{

"agency\_fee\_report":"/app/mount/scripts/pyscripts/sql\_scripts/agency\_fee\_report.sql"

},

{

"ambit\_ns\_users":"/app/mount/scripts/pyscripts/sql\_scripts/ambit\_ns\_users.sql"

}

],

"query\_config": {

"ds\_zone\_db": "ds\_dashboard"

},

"template\_root\_dir": "/app/mount/scripts/pyscripts/email\_template/",

"mail\_template": "email\_template.html",

"email\_from\_addr": "controlmonitor\_cde\_dev\_alerts@adib.com",

"subject": "DC-AOS-03 Framework: Account closures without customer request",

"to\_email": "Sanjana.Krishnan@adib.com;Thaanish.Kurumadaki@adib.com;Karthikeyan.Nethaji@adib.com",

"cc\_email": "Maadhusri.Ulaganathan@adib.com;Smrithi.Sundar@adib.com",

"department": "AOS",

"sub\_unit": "AOS",

"criticality": "Low",

"mail\_description": "The below listed accounts do not have entry either in Ethix or BPM system for account closure requests. This defect has high business criticality and requires your immediate attention."

},

{

"cde\_job\_name": "dpf\_email\_sender",

"name": "Report\_Mail",

"description": "Send Anomalies as mail with excel attachment and tabular mail body",

"mail\_type": "Type-3",

"mail\_type\_description": "Mail with tabulation in body and excel attachment",

"mail\_body\_rule\_query\_file": "/app/mount/scripts/pyscripts/sql\_scripts/cm\_control\_3\_rule\_query.sql",

"mail\_attachment\_sheets":[

{

"add\_account\_master":"/app/mount/scripts/pyscripts/sql\_scripts/add\_account\_master.sql"

},

{

"agency\_fee\_report":"/app/mount/scripts/pyscripts/sql\_scripts/agency\_fee\_report.sql"

},

{

"ambit\_ns\_users":"/app/mount/scripts/pyscripts/sql\_scripts/ambit\_ns\_users.sql"

}

],

"query\_config": {

"ds\_zone\_db": "ds\_dashboard"

},

"template\_root\_dir": "/app/mount/scripts/pyscripts/email\_template/",

"mail\_template": "email\_template.html",

"email\_from\_addr": "controlmonitor\_cde\_dev\_alerts@adib.com",

"subject": "DC-AOS-03 Framework: Account closures without customer request",

"to\_email": "Sanjana.Krishnan@adib.com;Thaanish.Kurumadaki@adib.com;Karthikeyan.Nethaji@adib.com",

"cc\_email": "Maadhusri.Ulaganathan@adib.com;Smrithi.Sundar@adib.com",

"department": "AOS",

"sub\_unit": "AOS",

"criticality": "Low",

"mail\_description": "The below listed accounts do not have entry either in Ethix or BPM system for account closure requests. This defect has high business criticality and requires your immediate attention."

}

],

"user\_specific\_email\_reports":[

{

"cde\_job\_name": "dpf\_user\_specific\_email\_sender",

"name": "User\_Specific\_Report\_Mail",

"description": "Send Anomalies as tabular mail body",

"mail\_type": "Type-1",

"mail\_type\_description": "Mail with tabulation in body only",

"query\_file\_path": "/app/mount/scripts/pyscripts/sql\_scripts/",

"query\_file\_name": "cm\_internal\_control\_2\_report.sql",

"query\_config": {

"ds\_zone\_db": "ds\_dashboard"

},

"user\_grouping\_columns":{

"group\_by\_column": "user\_name",

"to\_email\_column": "line\_mgr\_email",

"cc\_email\_column": "staff\_email"

},

"template\_root\_dir": "/app/mount/scripts/pyscripts/email\_template/",

"mail\_template": "email\_template.html",

"email\_from\_addr": "controlmonitor\_cde\_dev\_alerts@adib.com",

"subject": "USER SPECIFIC EMAIL FRAMEWORK TEST",

"common\_to\_email": "Sanjana.Krishnan@adib.com;Thaanish.Kurumadaki@adib.com;Roopika.Nandagopal@adib.com",

"common\_cc\_email": "Maadhusri.Ulaganathan@adib.com;Smrithi.Sundar@adib.com",

"department": "Internal",

"sub\_unit": "Internal",

"criticality": "Low",

"mail\_description": "The below listed accounts requires your immediate attention."

},

{

"cde\_job\_name": "dpf\_user\_specific\_email\_sender",

"name": "User\_Specific\_Report\_Mail",

"description": "Send Anomalies as mail with excel attachment",

"mail\_type": "Type-2",

"mail\_type\_description": "Mail with excel attachment only",

"query\_file\_path": "/app/mount/scripts/pyscripts/sql\_scripts/",

"query\_file\_name": "cm\_internal\_control\_2\_report.sql",

"query\_config": {

"ds\_zone\_db": "ds\_dashboard"

},

"user\_grouping\_columns":{

"group\_by\_column": "user\_name",

"to\_email\_column": "line\_mgr\_email",

"cc\_email\_column": "staff\_email"

},

"template\_root\_dir": "/app/mount/scripts/pyscripts/email\_template/",

"mail\_template": "email\_template.html",

"email\_from\_addr": "controlmonitor\_cde\_dev\_alerts@adib.com",

"subject": "USER SPECIFIC EMAIL FRAMEWORK TEST",

"common\_to\_email": "Sanjana.Krishnan@adib.com;Thaanish.Kurumadaki@adib.com;Roopika.Nandagopal@adib.com",

"common\_cc\_email": "Maadhusri.Ulaganathan@adib.com;Smrithi.Sundar@adib.com",

"department": "Internal",

"sub\_unit": "Internal",

"criticality": "Low",

"mail\_description": "The below listed accounts requires your immediate attention."

}

]

}

## ****Folder Structure Design****

##### resources/automated\_controls

##### {pipeline1\_name}

##### {job1\_name}

##### {target\_table1\_name}.sql

##### {target\_table2\_name}.sql

##### {job2\_name}

##### {target\_table1\_name}.sql

##### {target\_table2\_name}.sql

##### {pipeilne2\_name}

##### ......

##### {pipelineN\_name}

## ****Standard Naming Convention****

##### ****Airflow dag****

##### dpf\_cm\_{control\_department}\_control\_{control\_number}\_dag.py

##### ****Job name****

cm\_{control\_department}\_{control\_number}\_final\_report.py

cm\_{control\_department}\_{control\_number}\_preprocess.py

cm\_{control\_department}\_{control\_number}\_master.py

### ****Table names****

**Final Report tables:** cm\_{control\_department}\_{control\_number}\_final\_report

**Preprocess tables:** cm\_{control\_department}\_{control\_number}\_preprocess

**Master tables:** cm\_{control\_department}\_{control\_number}\_master

**Static tables**: ext\_static\_{table\_name}

### ****Excel report filename****

Normal emails : cm\_{control\_department}\_control\_{control\_number}\_{date}.xlsx

User specific emails: cm\_{control\_department}\_control\_{control\_number}\_{user\_specific\_value}\_{date}.xlsx

## ****Pseudocode****

### 1. SQL Executor - dpf\_common\_sql\_executor.py

#### Import Required Modules:

* Import required Python modules (e.g., sys, datetime, logging, json, etc.)
* Add custom library path for importing project-specific modules

#### Define Utility Classes:

##### Class: ExceptionHandler

* + extract\_error\_message(): Clean and return concise message from exception object

##### Class: DataFrameLogger

* + Initialize logging dictionary with metadata keys
  + get\_log\_schema(): Return predefined Spark schema for logging
  + add\_log(key, value): Update log entry
  + get\_log(): Return current log dictionary
  + post\_logs(db, table, status): Write log to Hive table

##### Class: SQLExecutor

* + get\_query(path, params, extra): Load SQL from file and substitute parameters
  + execute\_query(query): Run Spark SQL and return DataFrame
  + insert\_overwrite(df, db, table, partition\_id): Overwrite or drop Hive partition
  + get\_schema(df): Return schema as JSON
  + process\_df(df, stage\_config, partition\_id): Depending on strategy:
    - Create temp view or
    - Insert data into target table
    - Return row count and schema

#### Define Utility Functions:

* dict\_to\_namespace(obj) – Convert nested dictionary to SimpleNamespace
* add\_prefix(value) – Add "P\_" prefix to a number (used for partitioning)
* get\_partitions(run\_date, totalNoOfDays) – Return current and historical partitions based on run date
* parse\_spark\_configuration(spark) – Extract config variables from SparkContext
* get\_partition\_id(run\_date) – Return partition ID string for a given date
* add\_stage\_config\_to\_logs(logger, stage\_json) – Log stage config into logger

#### Main Function: process()

**Input:** sql\_executor, table\_logger, spark\_conf, control\_config, job\_config

**Steps:**

1. Add job-level metadata to logs
2. Get run\_date and calculate partition ID
3. Loop through all stages in job\_config (sorted by execution order):
   * Add stage config to logs
   * Get additional query parameters (based on no\_of\_past\_days)
   * Prepare SQL query by reading from file and substituting variables
   * Log start time, post "STARTED" status
   * Execute SQL query and return DataFrame
   * Process DataFrame (insert or create temp view)
   * Log row count, schema, end time
   * Post log as "SUCCESS"
4. If exception occurs:
   * Log exception message and post as "FAILED"

#### Initialize & Run Job

* Create SparkSession with Hive
* Configure logger and Spark settings
* Parse job and control config
* Instantiate SQLExecutor, DataFrameLogger
* Call process()
* Error handling : On error → log as FAILED

### 2. Email Generator - dpf\_email\_sender.py

#### Import Required Modules

* Import: os, sys, json, datetime, logging, smtplib, pandas, pyspark, jinja2, etc.
* Append custom paths
* Import custom module: control\_monitor\_framework

##### Define Utility Functions

* render\_template(): Load and render HTML email template using Jinja2
* send\_email(): Build and send email (HTML body + optional attachments)
* get\_query(): Load SQL file, replace placeholders with actual params
* dict\_to\_namespace(): Convert dicts to SimpleNamespace recursively
* generate\_no\_anomalies\_email(): Send notification when no anomalies are found

##### Define Main Logic: process()

**Input:** spark, logger, control\_config, job\_config, t\_1

* Create /tmp/control\_report/ directory if not exists
* Based on job\_config.mail\_type, choose one of three branches:

**Type-1 (Only Mail Body):**

* + Run SQL query for email body
  + If no anomalies → send "No anomalies" email
  + Else → convert Spark DataFrame to Pandas, send email with table in body

**Type-2 (Only Attachment):**

* + Run multiple SQL queries (from config) to create sheets in Excel
  + Save all sheets to a single Excel file
  + If all sheets are empty → send "No anomalies" email
  + Else → send email with Excel attachment

**Type-3 (Mail Body + Attachment):**

* + Run SQL for mail body and multiple queries for Excel sheets
  + If all data is empty → send "No anomalies" email
  + Else → send email with table in body and Excel as attachment

#### Spark Session & Job Initialization

* Create SparkSession with Hive support
* Setup logger
* Load Spark config variables: control\_config, job\_config
* Convert config dictionaries to SimpleNamespace
* Calculate t\_1 as day before control\_config.run\_date

#### Execute Main Function

* Call process(...)
* On success → log success
* On failure → log exception, raise error

### 3. User specific Email Generator - dpf\_user\_specific\_email\_sender.py

#### Import Modules

* Import standard libraries: os, sys, json, datetime, logging, smtplib, etc.
* Import data/templating libraries: pandas, pyspark, jinja2, etc.
* Add custom paths and import control\_monitor\_framework

#### Define Utility Functions

* render\_template(): Load and render HTML template using Jinja2
* send\_email(): Send email with HTML body and optional attachments
* get\_query(): Load SQL query and substitute variables
* dict\_to\_namespace(): Convert dict to SimpleNamespace
* get\_user\_specific\_df\_details(): Filter DataFrame for user, extract email info
* generate\_no\_anomalies\_email(): Send "no anomalies" email

#### Define Main Process – process()

* Create /tmp/control\_report/ directory
* Log job details
* Load and render SQL query
* Execute query via Spark → get DataFrame
* Count rows in DataFrame
* Prepare email subject

**If no anomalies:**

* + Send "no anomalies" email

**If anomalies found:**

* + Convert Spark DataFrame to Pandas
  + Group data by user
  + For each user:
    - Filter user-specific rows
    - Extract to and cc email addresses

**If Type-1:**

* + Send email with table in body only

**If Type-2:**

* + Save user data as Excel file
  + Send email with Excel attachment only

#### Spark & Logger Setup

* Initialize Spark session with Hive
* Set Hive partition mode
* Configure logging
* Parse Spark config: control\_config, job\_config
* Convert configs to SimpleNamespace
* Compute t\_1 as run\_date - 1 day

#### Execute Main

* Call process(...)
* On success: log success
* On failure: log error and raise exception