Table of Contents

[CDS Overview 8](#_Toc148083733)

[Purpose of the Automated Testing Framework in Data Product 8](#_Toc148083734)

[Benefits of Automated Testing in Data Ingestion Product 9](#_Toc148083735)

[Test Requirement and Strategy for Ingestion Data Product 9](#_Toc148083736)

[Overall Testing Approach 9](#_Toc148083737)

[Testing Automation Framework Coverage Goals 9](#_Toc148083738)

[TAF Environment Setup 9](#_Toc148083739)

[TAF Design Principles 10](#_Toc148083740)

[TAF Architecture Overview 12](#_Toc148083741)

[TAF Components Summary 13](#_Toc148083742)

[TAF Components Details 15](#_Toc148083743)

[SourcePrimaryKeyVerifier 15](#_Toc148083744)

[Objective 15](#_Toc148083745)

[Business Value 15](#_Toc148083746)

[High Level Flow 15](#_Toc148083747)

[How to execute 15](#_Toc148083748)

[Resolution of Issues found 16](#_Toc148083749)

[Pipeline Input 16](#_Toc148083750)

[Logic 17](#_Toc148083751)

[Output 17](#_Toc148083752)

[Setup Required 17](#_Toc148083753)

[Technology Involved 17](#_Toc148083754)

[Artefact Details 17](#_Toc148083755)

[PL\_Taf\_SourcePrimaryKeyVerifier\_Main.json 17](#_Toc148083756)

[PL\_Taf\_SourcePrimaryKeyVerifier\_Source.json 17](#_Toc148083757)

[PL\_Taf\_SourcePrimaryKeyVerifier\_Lookup\_Oracle\_OnPrem.json 17](#_Toc148083758)

[PL\_Taf\_SourcePrimaryKeyVerifier\_Lookup\_Oracle\_OnPrem\_S.json 17](#_Toc148083759)

[PL\_Taf\_SourcePrimaryKeyVerifier\_Lookup\_SqlDbV1\_Cloud.json 17](#_Toc148083760)

[PL\_Taf\_SourcePrimaryKeyVerifier\_Lookup\_SqlDb\_Cloud.json 17](#_Toc148083761)

[PL\_Taf\_SourcePrimaryKeyVerifier\_Lookup\_SqlDb\_OnPrem.json 18](#_Toc148083762)

[taf.get\_sources\_for\_sourceTypes.StoredProcedure.sql 18](#_Toc148083763)

[taf.get\_source\_duplicate\_queries.StoredProcedure.sql 18](#_Toc148083764)

[Limitations 18](#_Toc148083765)

[Known Issues 18](#_Toc148083766)

[Required Maintenance 18](#_Toc148083767)

[Periodicity 18](#_Toc148083768)

[Applicability with Source Type 18](#_Toc148083769)

[Reusability of this verifier with other teams 18](#_Toc148083770)

[SourceConnectionVerifier 19](#_Toc148083771)

[Objective 19](#_Toc148083772)

[Business Value 19](#_Toc148083773)

[High Level Flow 19](#_Toc148083774)

[How to execute 20](#_Toc148083775)

[Pipeline Input 22](#_Toc148083776)

[Setup Required 22](#_Toc148083777)

[Logic 24](#_Toc148083778)

[Output 24](#_Toc148083779)

[Technology Involved 24](#_Toc148083780)

[Resolution of Issues found 24](#_Toc148083781)

[Artefact Details 24](#_Toc148083782)

[PL\_Taf\_SourceConnectionVerifier\_Main.json 24](#_Toc148083783)

[PL\_Taf\_SourceConnectionVerifier\_LinkedService.json 24](#_Toc148083784)

[PL\_Taf\_SourceConnectionVerifier\_SourceAndLinkedService.json 24](#_Toc148083785)

[taf.sp\_get\_linkedservices.StoredProcedure.sql 24](#_Toc148083786)

[taf.sp\_get\_sources\_ls.StoredProcedure.sql 25](#_Toc148083787)

[Limitations 25](#_Toc148083788)

[Known Issues 25](#_Toc148083789)

[Required Maintenance 25](#_Toc148083790)

[Periodicity 25](#_Toc148083791)

[Applicability with Source Type 25](#_Toc148083792)

[Reusability of this verifier with other teams 25](#_Toc148083793)

[MetadataComparatorVerifier 26](#_Toc148083794)

[Objective 26](#_Toc148083795)

[Business Value 26](#_Toc148083796)

[High Level Flow 26](#_Toc148083797)

[How to execute 27](#_Toc148083798)

[Pipeline Input 28](#_Toc148083799)

[Setup Required 28](#_Toc148083800)

[Logic 31](#_Toc148083801)

[Technology Involved 31](#_Toc148083802)

[Resolution of Issues found 31](#_Toc148083803)

[Artefact Details 31](#_Toc148083804)

[PL\_Taf\_MetadataComparatorVerifier\_Main.json 31](#_Toc148083805)

[PL\_Taf\_MetadataComparatorVerifier\_Copy\_Config.json 31](#_Toc148083806)

[Limitations 31](#_Toc148083807)

[Known Issues 32](#_Toc148083808)

[Required Maintenance 32](#_Toc148083809)

[Periodicity 32](#_Toc148083810)

[Applicability with Source Type 32](#_Toc148083811)

[Reusability of this verifier with other teams 32](#_Toc148083812)

[MetadataHistory 32](#_Toc148083813)

[Objective 32](#_Toc148083814)

[Business Value 32](#_Toc148083815)

[High Level Flow 32](#_Toc148083816)

[How to execute 33](#_Toc148083817)

[Pipeline Input 34](#_Toc148083818)

[Setup Required 34](#_Toc148083819)

[Logic 36](#_Toc148083820)

[Technology Involved 38](#_Toc148083821)

[Resolution of Issues found 38](#_Toc148083822)

[Artefact Details 38](#_Toc148083823)

[Limitations 39](#_Toc148083824)

[Known Issues 39](#_Toc148083825)

[Required Maintenance 39](#_Toc148083826)

[Periodicity 39](#_Toc148083827)

[Applicability with Source Type 39](#_Toc148083828)

[Reusability of this verifier with other teams 39](#_Toc148083829)

[Azure API Ingestor 39](#_Toc148083830)

[Objective 39](#_Toc148083831)

[Business Value 39](#_Toc148083832)

[High Level Flow 40](#_Toc148083833)

[How to execute 40](#_Toc148083834)

[Pipeline Input 41](#_Toc148083835)

[Setup Required 41](#_Toc148083836)

[Logic 43](#_Toc148083837)

[Technology Involved 43](#_Toc148083838)

[Resolution of Issues found 43](#_Toc148083839)

[Artefact Details 43](#_Toc148083840)

[PL\_Taf\_Generic\_Azure\_Api\_To\_SqlDb\_Main.json 43](#_Toc148083841)

[PL\_Taf\_Generic\_Azure\_Api\_To\_SqlDb\_Dataset.json 43](#_Toc148083842)

[taf.flatten\_json.StoredProcedure.sql 43](#_Toc148083843)

[taf.flatten\_single\_json.StoredProcedure.sql 43](#_Toc148083844)

[taf.master\_dataset\_api\_config.Table.sql 43](#_Toc148083845)

[taf.flatten\_json\_config.Table.sql 43](#_Toc148083846)

[Limitations 44](#_Toc148083847)

[Known Issues 44](#_Toc148083848)

[Required Maintenance 44](#_Toc148083849)

[Periodicity 44](#_Toc148083850)

[Applicability with Source Type 44](#_Toc148083851)

[Reusability of this verifier with other teams 44](#_Toc148083852)

[IngestionTestBasedVerifier 44](#_Toc148083853)

[Objective 44](#_Toc148083854)

[Business Value 44](#_Toc148083855)

[High Level Flow 45](#_Toc148083856)

[How to execute 45](#_Toc148083857)

[Pipeline Input 47](#_Toc148083858)

[Setup Required 47](#_Toc148083859)

[Logic 48](#_Toc148083860)

[Technology Involved 48](#_Toc148083861)

[Resolution of Issues found 48](#_Toc148083862)

[Artefact Details 48](#_Toc148083863)

[PL\_Taf\_IngestionVerifier\_TestRunBased\_Main.json 48](#_Toc148083864)

[PL\_Taf\_IngestionVerifier\_TestRunBased\_IngestAndVerifyDatasets.json 48](#_Toc148083865)

[PL\_Taf\_IngestionVerifier\_TestRunBased\_VerifyDatasets.json 48](#_Toc148083866)

[PL\_Taf\_IngestionVerifier\_TestRunBased\_VerifyDataset.json 48](#_Toc148083867)

[IngestionVerifier.py 48](#_Toc148083868)

[taf.ingestion\_verifier\_config.Table.sql 48](#_Toc148083869)

[Limitations 48](#_Toc148083870)

[Known Issues 48](#_Toc148083871)

[Required Maintenance 48](#_Toc148083872)

[Periodicity 49](#_Toc148083873)

[Applicability with Source Type 49](#_Toc148083874)

[Reusability of this verifier with other teams 49](#_Toc148083875)

[IngestionRealBasedVerifier 50](#_Toc148083876)

[Objective 50](#_Toc148083877)

[Business Value 50](#_Toc148083878)

[High Level Flow 51](#_Toc148083879)

[How to execute 51](#_Toc148083880)

[Pipeline Input 52](#_Toc148083881)

[Setup Required 52](#_Toc148083882)

[Logic 52](#_Toc148083883)

[Checks Covered 53](#_Toc148083884)

[Technology Involved 54](#_Toc148083885)

[Resolution of Issues found 54](#_Toc148083886)

[Artefact Details 54](#_Toc148083887)

[Limitations 55](#_Toc148083888)

[Known Issues 55](#_Toc148083889)

[Required Maintenance 55](#_Toc148083890)

[Periodicity 55](#_Toc148083891)

[Applicability with Source Type 55](#_Toc148083892)

[Reusability of this verifier with other teams 55](#_Toc148083893)

[TAFLogger 56](#_Toc148083894)

[Purpose 56](#_Toc148083895)

[High Level Diagram 56](#_Toc148083896)

[Artefact Details 56](#_Toc148083897)

[taf.taf\_log\_master.Table.sql 56](#_Toc148083898)

[taf.taf\_log\_detail.Table.sql 56](#_Toc148083899)

[taf.taf\_log\_check\_detail.Table.sql 56](#_Toc148083900)

[taf.taf\_log\_metric\_detail.Table.sql 56](#_Toc148083901)

[taf.sp\_insert\_taf\_log\_master.StoredProcedure.sql 56](#_Toc148083902)

[taf.sp\_insert\_taf\_log\_detail.StoredProcedure.sql 56](#_Toc148083903)

[taf.sp\_update\_taf\_log\_detail\_json.StoredProcedure.sql 56](#_Toc148083904)

[taf.sp\_update\_taf\_log\_detail.StoredProcedure.sql 56](#_Toc148083905)

[taf.sp\_update\_taf\_log\_master.StoredProcedure.sql 56](#_Toc148083906)

[TAFNotifier 57](#_Toc148083907)

[Purpose 57](#_Toc148083908)

[High Level Diagram 57](#_Toc148083909)

[Artefact Details 57](#_Toc148083910)

[taf.get\_verifier\_results\_in\_html.StoredProcedure.sql 57](#_Toc148083911)

[taf.spQueryToHtmlTable.StoredProcedure.sql 57](#_Toc148083912)

[insert\_taf\_config.sql 57](#_Toc148083913)

[Sample Email 57](#_Toc148083914)

[ETLVerifier 58](#_Toc148083915)

[ProfileEngine 58](#_Toc148083916)

[DataQualityEngine(PyDeequChecks, GreatExpectationChecks, SQLChecks, DuplicateChecks, TrendChecks) 58](#_Toc148083917)

[ReconEngine 58](#_Toc148083918)

[Data-Driven Testing 59](#_Toc148083919)

[Test Case Prioritization 59](#_Toc148083920)

[Mocking and Stubbing Data Sources 59](#_Toc148083921)

[Data Quality Testing 59](#_Toc148083922)

[Data Validation Rules 59](#_Toc148083923)

[Anomaly Detection 59](#_Toc148083924)

[Data Profiling 59](#_Toc148083925)

[Security Testing 59](#_Toc148083926)

[Data Security Measures 59](#_Toc148083927)

[Compliance and Reporting 59](#_Toc148083928)

[Continuous Improvement 59](#_Toc148083929)

[Collecting and Analyzing Test Metrics 60](#_Toc148083930)

[Conclusion 60](#_Toc148083931)

[Summary of Key Takeaways 60](#_Toc148083932)

[Future Enhancements and Roadmap 60](#_Toc148083933)

[Appendices 60](#_Toc148083934)

# CDS Overview

The Common Data Service manages the acquisition of data required for cross-commodity use-cases that wish to leverage the STRATOS data platform. The Common Data Service acquires data that is strategic in character and in acquisition patterns that adhere to STRATOS data governance standards.

The objective of the Common Data Service is to get data to the authorised user faster and to lower the cost of data integration via economies of scale and promoting the usage of common design patterns. The data published to the STRATOS data platform is secure by design and the teams development and operational processes support T&S Data Governance objectives.

# Purpose of the Automated Testing Framework in Data Product

CDS Test Automation Framework aims to provide capabilities to ensure various functioning of CDS system is working properly and as expected in automated and proactive manner.

**Below are the objective of the framework**

* Make CDS more transparent, reliable and robust platform and proof to stakeholders that system is working as expected
* Provide pro-active checks to catch the system is working and protect the system from probable risks attached to various functioning of platform
* Create a base for easy troubleshooting/monitoring/testing/deployment
* Create a base for automated CI/CD pipeline, and path to prod to days and ensure increased confidence in deployment
* Create a base for proven SLAs for CDS ingestion and transformation
* Create a base for Data Mesh Principle Self Infra as Platform for CDS platform, Also provide ways to enable self serve to business users and remove the dependency on technical resources

**Following are the major functionalities of the framework**

* Ensure the CDS ingestion component(source to land) is working as expected
* Ensure all the source connection are working properly
* Ensure CDS Processing component(land to unharm) is working as expected
* Ensure a new change to CDS components has not broken existing capabilities
* Ensure the datasets in each zone is quality tested.
* Provide various dashboards to view various system metrics, test results, pipeline results, dataset profile metrics, log metrics
* Ensure the CDS Metadata is configured properly

# Benefits of Automated Testing in Data Ingestion Product

# Test Requirement and Strategy for Ingestion Data Product

# Overall Testing Approach

# Testing Automation Framework Coverage Goals

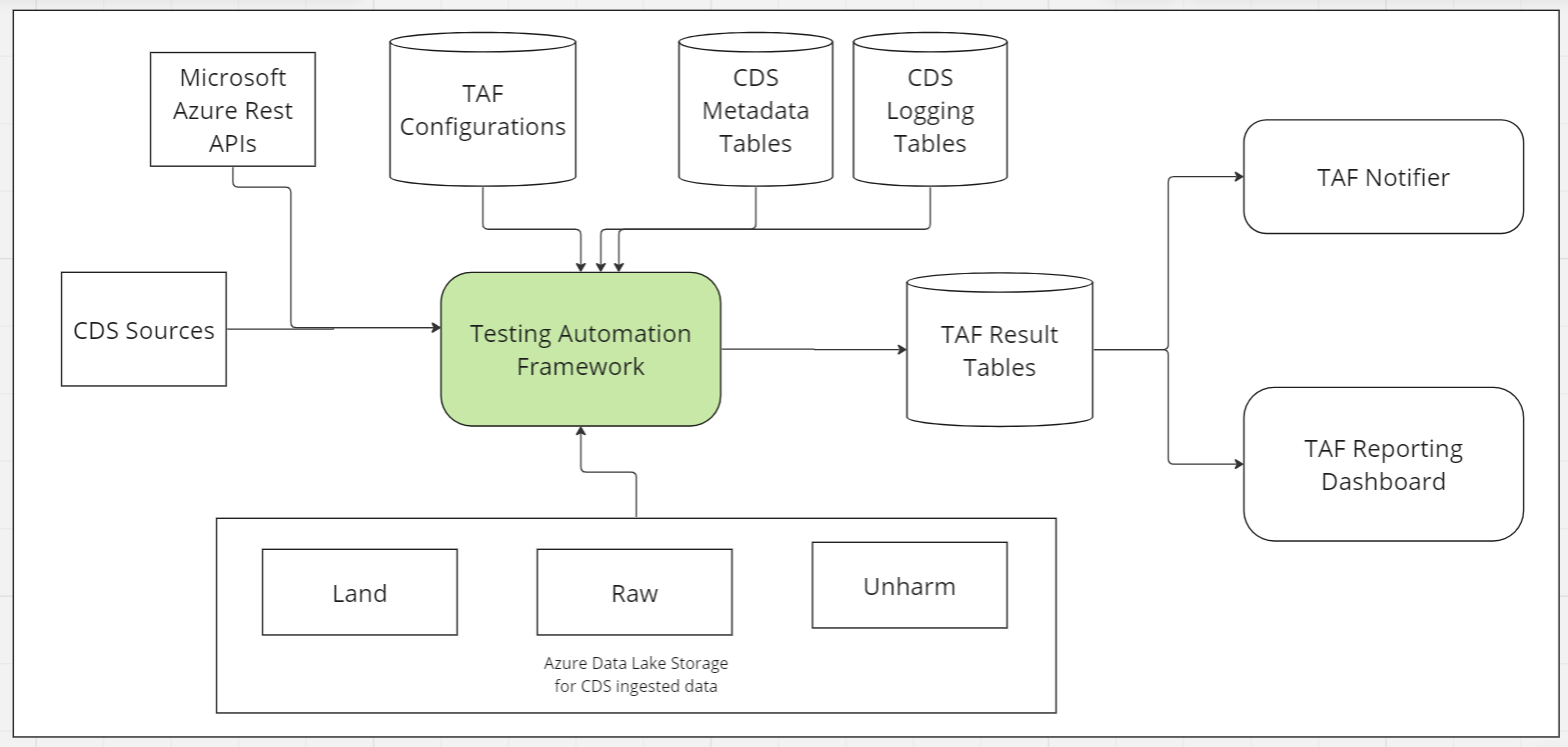
# TAF Environment Setup

# TAF Design Principles

|  |  |  |
| --- | --- | --- |
| **Seq** | **Category** | **Design Considerations** |
| 1 | Side Effects | There should be Zero impact to CDS real pipeline results(datasets in all the zones), watermark, Source systems while running test framework |
| 3 | Low Code Solution | Test framework should be less verbose, should have minimal config and minimal code change, should be a low code solution based on metadata, No per dataset effort or artifact |
| 4 | Less Code footprint for framework | Wherever possible it should hit the real pipeline artifacts, should be avoiding standing a separate code base mimicking the real work |
| 5 | Extensible | It should be extensible, new component/checks should be easily be able to be added without resulting verification of other component/checks |
| 6 | Cost | Framework should be cost effective, ex - Should minimize use of Databricks as it is costlier, unnecessary executions should be avoided |
| 7 | Instrumentation | Framework should be instrumented, so that troubleshooting the framework misfunctioning can be easier |
| 8 | Software Eng best practices | Framework should be following software eng best practices such as modularity, Loosely Coupled Architecture , clean code, naming standards depicting the responsibilities, design pattern, design principles like SOLID, DRY, KISS, YAGNI, Functional Programming |
| 9 | Reusability | Should reuse frameworks developed at Shell earlier, Should use widely used open source like PyDeequ and Great Expectations if applicable |
| 10 | Product thinking | Product Thinking – should be developed as a product, so that other team with similar requirements can use this framework |
| 11 | Data Mesh – Self Infra as platform and Self service to business | Implement Self Infra as platform data mesh principle and create base for self service to business users and CDS stakeholders(SOM/Consumers) for various functionalities |
| 12 | Base for Automated CI/CD | Should create as base for automated CI/CD, signing off a build when there is a change in framework |
| 13 | Base for SLA driven ingestion | Broken SLAs should be catched and reported |
| 14 | Orchestration Options | Should try to “close” the orchestration and sub-orchestrations so that a new check does not require change in orchestration  ADF, Azure Functions, Azure Dev Ops could be used as higher level orchestration option |
| 15 | Runtime Options | The actual check runtime could be Python, Pyspark, SQL, ADF depending on metrics in question |
| 16 | Test Run vs Real Run | Depending on scenario and check being performed need to decide whether it make sense to do a test run with test dataset or check should be based on Real Run results |
| 17 | Check Granularities | Different options of check granularities – source type, source, dataset |
| 18 | Check Trigger | Adhoc run, Schedule Run, Event Driven, Couple with ingestion and transformation pipeline(pre-ingestion, post ingestion, post processing etc), part of CI/CD |
| 19 | Env Applicability | Which env the check should run on, few checks will worth running on prod as well |
| 20 | Grouping Checks | There could be a need of grouping the checks and provide ways to run all the checks under one group, should also provide ways to run them individually |
| 21 | ADLS Gen1 and Gen2 | Gen2 Migration is due, test framework without much changes should support gen2 |
| 22 | Rerun of framework | What if framework is rerun, can we make the run idempotent and deterministic and pure function(no side effects) |

# TAF Architecture Overview

Testing Automation Framework



Testing Automation Framework uses below information , runs various checks and produce check results in the form of TAF result tables in SQL Database, which is being used by TAF notifier and in future will be used by Power BI to display various statistics.

1. CDS Sources
2. TAF Configurations
3. CDS Metadata Tables
4. CDS Logging tables such as load\_status
5. Azure Data Lake(LAND, RAW, Unharm)
6. Azure API for Triggers

# TAF Components Summary

|  |  |  |
| --- | --- | --- |
| SrNo | Component Name | Component Description |
| 1 | SourcePrimaryKeyVerifier | This verifier will validate if source table has duplicate records based on primary key configured in CDS metadata config for the source table |
| 2 | SourceConnectionVerifier | This verifier will validate if CDS is able to make connection to the source based on connection configuration for source in CDS metadata |
| 3 | IngestionVerifier | This verifier will verify whether ingestion(source to land) is working as expected, It has various checks which are described in detail section |
| 4 | MetadataComparatorVerifier | This verifier compares CDS metadata config from 3 CDS metadata DB env(DEV, UAT and PROD) and reports differences. |
| 5 | Metadata History | This component enables CDS to capture the changes happening to CDS metadata tables at database level(INSERT/UPDATE/DELETE), It also shows value difference between one version of record to another in case of UPDATE |
| 6 | Generic Azure API Ingestor | This components ingests Azure Rest API to SQL Database in configurable way, currently it is configured to ingest data for Azure Data Factory Triggers from Microsoft Azure Rest API for ADF Triggers |
| 7 | TAF Logging | Logging tables and SP to maintain the rules result log |
| 8 | TAFNotifier | This component takes pipeline run ID of TAF Run and fetches the results , convert to HTML and sends email by calling Logic App |

# TAF Components Details

## SourcePrimaryKeyVerifier

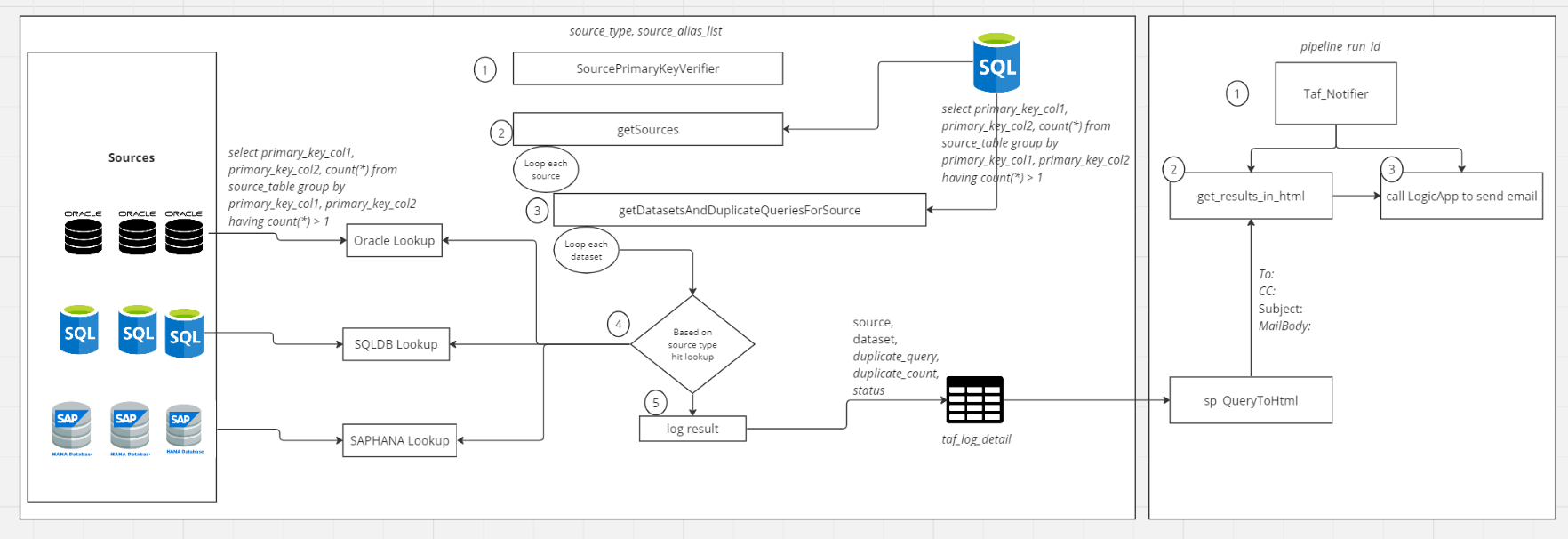
### Objective

This verifier checks whether source table has duplicates based on primary key configured for the dataset in CDS metadata config.

### Business Value

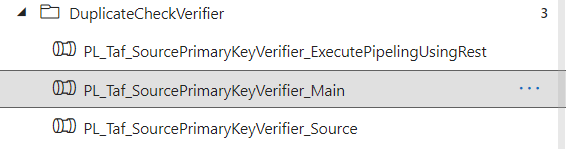
This verifier helps business analyst to check if primary keys are wrongly configured in CDS metadata, It will prevent duplicate data getting ingested to Data Lake. SCD Type1 and SCD Type 2 logic of CDS worked based on primary key defined hence this will prevent incorrect logic for SCD Type 1 and SCD Type 2.

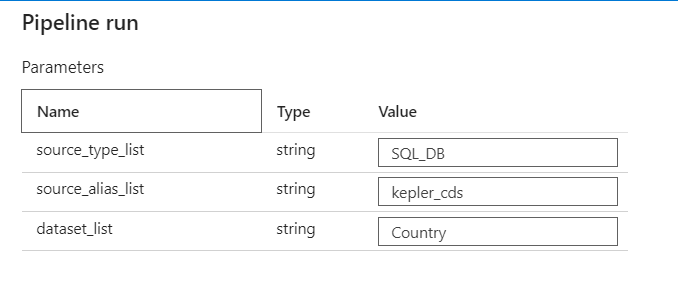
### High Level Flow



### How to execute

Under Testing Framework/DuplicateCheckVerifier





We can mention “ALL” in the parameters to run for all.

### Resolution of Issues found

How to resolve issue found by TAF for this verifier

We need to fix the primary key working with source business contact, we need to collect the new primary key, configure the new primary key in CDS metadata for the dataset and validate using the verifier, we can then run the full load again after modifying the primary key in all environments.

### Pipeline Input

Source\_type\_list = <*List of source types*>,*use ALL to run for all*

Source\_alias\_list=<*List of source alias*>, *use ALL to run for all*

Dataset\_list=<*List of datasets*>, *use ALL to run for all*

### Logic

1. Get all the sources from CDS metadata database applicable to the input calling the stored proc Get\_sources\_for\_sourceTypes
2. Loop through each source
3. For each source
   1. Call stored proc to get all the datasets under the source along with the duplicate query for source table based on primary key configured in CDS Metadata config\_primary\_keys
   2. Loop through each dataset
      1. Call applicable lookup for the source which will execute the duplicate query against source DB and write the results to taf\_log\_detail table if Duplicate is found along with duplicate query and duplicate count
4. Call TafNotifier passing the pipeline run id, where TafNotifier will query the result , convert the results records to html and call logic app to send email

### Output

Taf.Taf\_log\_detail will be populated per dataset with duplicate count found in Source table for this dataset

### Setup Required

No setup required

### Technology Involved

ADF, SQL Stored proc

### Artefact Details

|  |  |  |
| --- | --- | --- |
| SrNo | Artefact | Artefact Detail |
| 1 | [PL\_Taf\_SourcePrimaryKeyVerifier\_Main.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_SourcePrimaryKeyVerifier_Main.json) | Get sources and loop through them |
| 2 | [PL\_Taf\_SourcePrimaryKeyVerifier\_Source.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_SourcePrimaryKeyVerifier_Source.json) | Get datasets for a source and loop through them, and calls lookup pipeline |
| 3 | [PL\_Taf\_SourcePrimaryKeyVerifier\_Lookup\_Oracle\_OnPrem.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_SourcePrimaryKeyVerifier_Lookup_Oracle_OnPrem.json) | Lookup for Oracle Onprem |
| 4 | [PL\_Taf\_SourcePrimaryKeyVerifier\_Lookup\_Oracle\_OnPrem\_S.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_SourcePrimaryKeyVerifier_Lookup_Oracle_OnPrem_S.json) | Lookup for Oracle Onprem Service Name |
| 5 | [PL\_Taf\_SourcePrimaryKeyVerifier\_Lookup\_SqlDbV1\_Cloud.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_SourcePrimaryKeyVerifier_Lookup_SqlDbV1_Cloud.json) | Lookup for sqlDB Cloud |
| 6 | [PL\_Taf\_SourcePrimaryKeyVerifier\_Lookup\_SqlDb\_Cloud.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_SourcePrimaryKeyVerifier_Lookup_SqlDb_Cloud.json) | Lookup for sqlDB Cloud |
| 7 | [PL\_Taf\_SourcePrimaryKeyVerifier\_Lookup\_SqlDb\_OnPrem.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_SourcePrimaryKeyVerifier_Lookup_SqlDb_OnPrem.json) | Lookup for sqlDB Onprem |
| 8 | [taf.get\_sources\_for\_sourceTypes.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.get_sources_for_sourceTypes.StoredProcedure.sql) |  |
| 9 | [taf.get\_source\_duplicate\_queries.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.get_source_duplicate_queries.StoredProcedure.sql) |  |

### Limitations

Currently It supports only certain source types(SQL DB, Oracle)

It will only support SQL based databases, It will not support File or API based source types

### Known Issues

### Required Maintenance

We need to add support for more source types

### Periodicity

This can be executed for all supported source types once as first run

We need to decide as incr run, should it run for all the datasets

It can run daily or weekly

### Applicability with Source Type

|  |  |  |
| --- | --- | --- |
| Source Type | No of Sources | Supported |
| Oracle | 15 | Yes |
| SQL\_DB | 9 | Yes |
| Avro | 4 | Can not be supported |
| BW | 4 | Can be supported, need to develop |
| SQL\_DB\_V1 | 3 | Yes |
| API | 3 | Can not be supported |
| FILESYSTEM | 2 | Can not be supported |
| HANA\_DB | 2 | Can be supported, need to develop |
| bigquery | 1 | Can be supported, need to develop |
| TESLAAPI | 1 | Can not be supported |
| SFTP | 1 | Can not be supported |
| SharePoint | 1 | Can not be supported |

### Reusability of this verifier with other teams

If other teams are using same semantics as CDS then this is 100% usable.

If the other teams are not using same semantics as CDS team , then the feature is currently not 100% reusable due to CDS semantics(CDS table name, column names) used in the code, but it can be made 100% reusable with few modifications

Some part of the verifiers can be used as is

## SourceConnectionVerifier

### Objective

This verifier checks whether CDS Ingestion component (Ingestion ADF) can connect to source systems configured in CDS metadata.

### Business Value

This verifier will help CDS Support team to proactively check that the connection configured in metadata for source(e.g. database server, db name, username, pwd in Azure Key vault etc) are working fine. This will help in troubleshooting issues related to connection to the source systems from CDS Ingestion component.

### High Level Flow

To implement this verifier following Microsoft Azure APIs are used

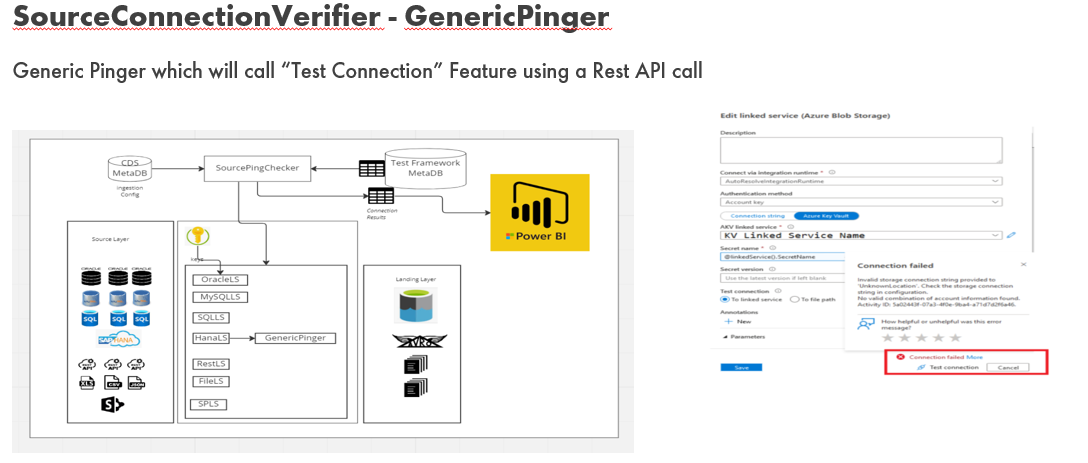
1. GET {endpoint}/linkedservices/{linkedServiceName}?api-version=2020-12-01

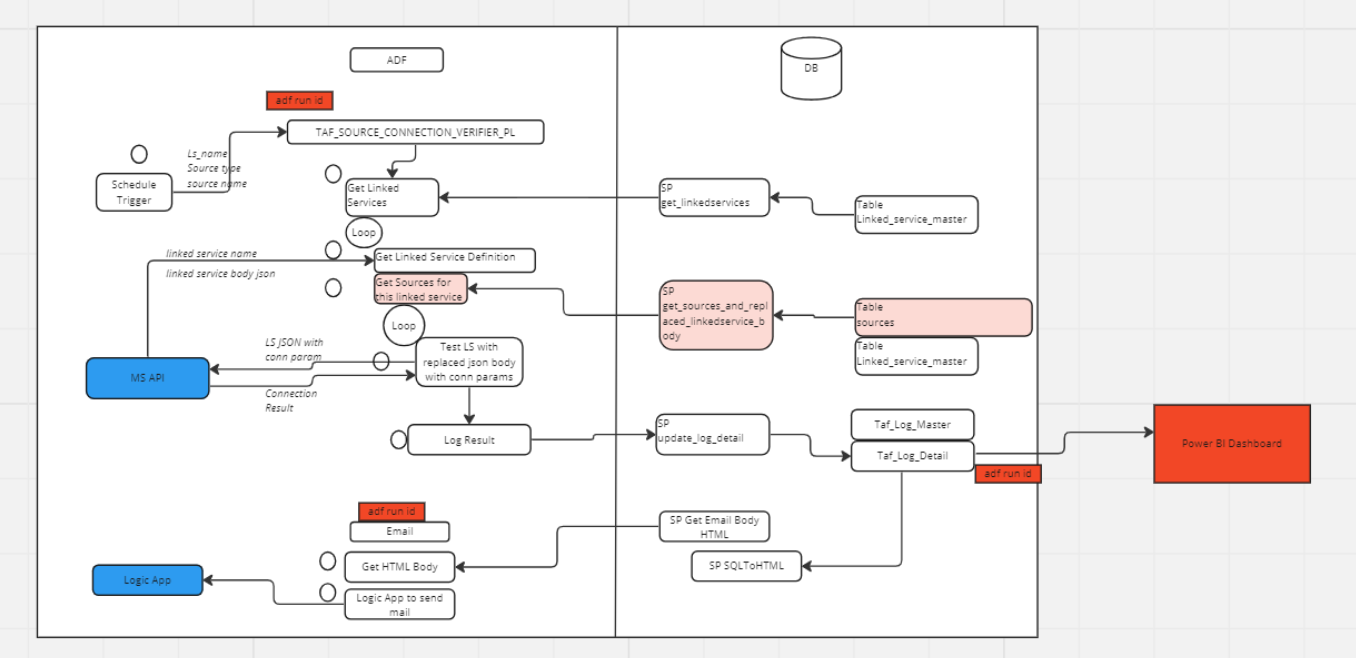
[Linked Service - Get Linked Service - REST API (Azure Synapse) | Microsoft Learn](https://learn.microsoft.com/en-us/rest/api/synapse/data-plane/linked-service/get-linked-service?tabs=HTTP)

This is to get Linked Service JSON body from Azure Data Factory Passing the linked service name

1. GET {endpoint}/testConnectivity?api-version=2018-06-01

This is an undocumented Microsoft API to check with filled value of parameters Linked service is able to connect to the source, It is same as clicking Test Linked Service Button in the ADF Linked service page

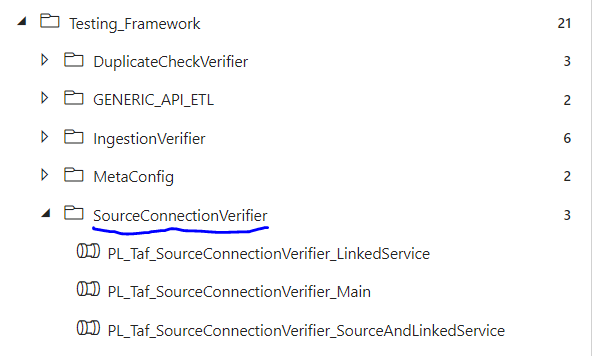




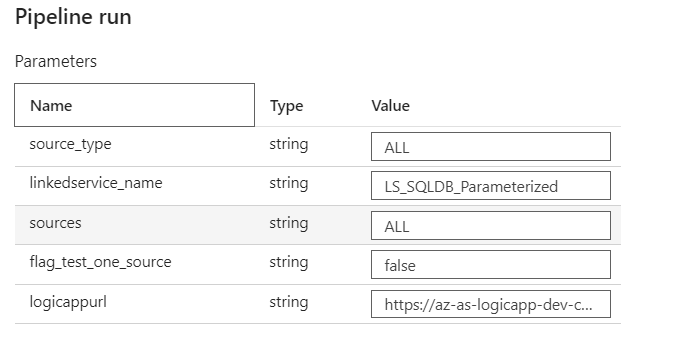
### How to execute

CDS Ingestion ADF ->

Testing Framework/SourceConnectionVerifier/PL\_Taf\_SourceConnectionVerifier\_Main



Following are the parameters



### Pipeline Input

Source\_type = Source Type e.g. SQL\_DB

Linkedservice\_name=Name of the linked service, *use ALL to run for all*

sources=<*List of source names*>, *use ALL to run for all*

flag\_test\_one\_source=false, keep it false, this is for developer testing, to test one source for a linked service

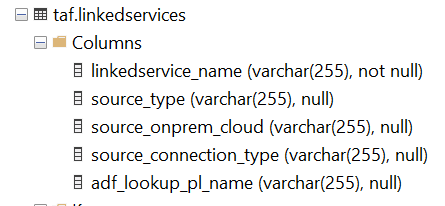
logicappurl=hardcoded, do not change(will be removed in future release)

### Setup Required

We have a couple of config tables which need to be configured before we run this verifier

Table Name: taf.linkedservices

Purpose: To keep master information about linked services used in CDS, it also keeps for which source type the linked service is being used



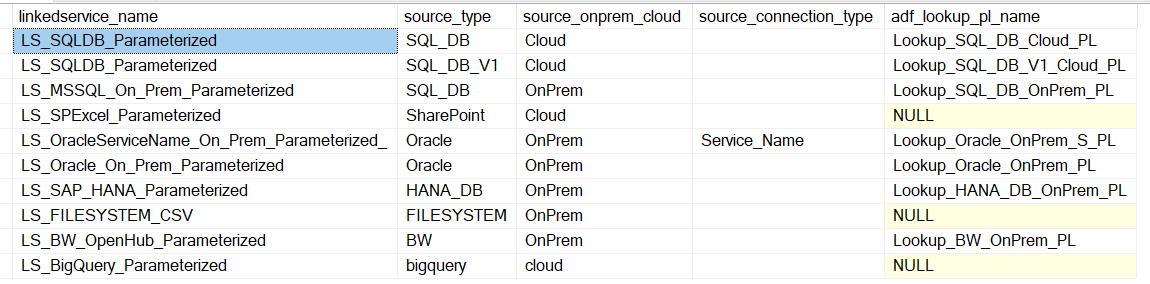
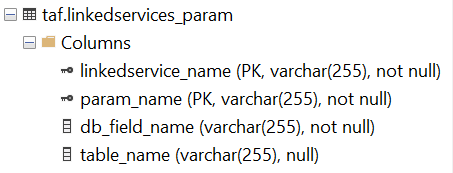
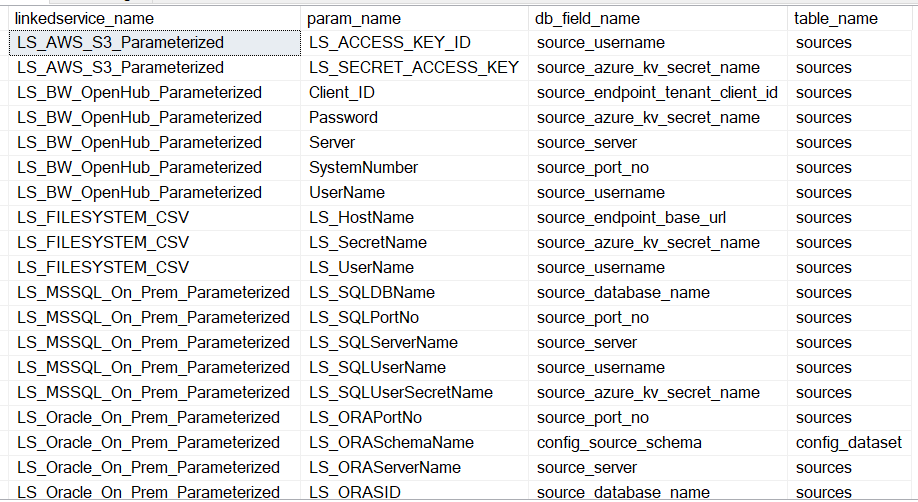


Table Name: taf.linkedservices\_param

Purpose: To keep all the parameters used in linked service and which config is used to keep the parameter in CDS metadata





### Logic

1. Get all the linked services which are applicable for the parameter
2. Loop through all the linked services
3. For each linked service
   1. Get the linked service json body hitting MS Azure API
   2. Get all the sources attached to this linked service from CDS metadata along with replaced JSON body in linked service with connection details
   3. Loop through all the sources returned
   4. For each source
      1. Hit Azure MS API to test the connectivity of the linked service with filled connection details
      2. Log the result with connection success or failed in taf log detail table
4. Call TafNotifier passing the pipeline run id, where TafNotifier will query the result , convert the results records to html and call logic app to send email

### Output

Taf.Taf\_log\_detail will be populated per source with connection status(success or failed) along with error encountered

### Technology Involved

ADF, SQL Stored proc

### Resolution of Issues found

How to resolve issue found by TAF for this verifier

If we are not able to make connection , below could be the possible issues

1. Connection configuration in cds metadata sources table might not be correct
2. AKV secret value might have been locked or expired
3. There could be issue with source system

### Artefact Details

|  |  |  |
| --- | --- | --- |
| SrNo | Artefact | Artefact Detail |
| 1 | [PL\_Taf\_SourceConnectionVerifier\_Main.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_SourceConnectionVerifier_Main.json) | Get linked services for applicable input |
| 2 | [PL\_Taf\_SourceConnectionVerifier\_LinkedService.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_SourceConnectionVerifier_LinkedService.json) | Loop through all linked services and get sources  Loop through all sources |
| 3 | [PL\_Taf\_SourceConnectionVerifier\_SourceAndLinkedService.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_SourceConnectionVerifier_SourceAndLinkedService.json) | Test connectivity |
| 4 | [taf.sp\_get\_linkedservices.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.sp_get_linkedservices.StoredProcedure.sql) | SP to get linked services |
| 5 | [taf.sp\_get\_sources\_ls.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.sp_get_sources_ls.StoredProcedure.sql) | Sp to get sources and their linked service filled with connection |

### Limitations

Currently It supports only certain source types

### Known Issues

### Required Maintenance

We need to add support for more source types

We need to configure the linked service taf config tables

### Periodicity

This can be executed daily

### Applicability with Source Type

|  |  |  |
| --- | --- | --- |
| Source Type | No of Sources | Supported |
| Oracle | 15 | Yes |
| SQL\_DB | 9 | Yes |
| Avro | 4 | Can be supported |
| BW | 4 | Yes |
| SQL\_DB\_V1 | 3 | Yes |
| API | 3 | Can be supported |
| FILESYSTEM | 2 | Can be supported |
| HANA\_DB | 2 | Yes |
| bigquery | 1 | Yes |
| TESLAAPI | 1 | Can be supported |
| SFTP | 1 | Can be supported |
| SharePoint | 1 | Can be supported |

### Reusability of this verifier with other teams

Other teams can use this with minimal changes

## MetadataComparatorVerifier

### Objective

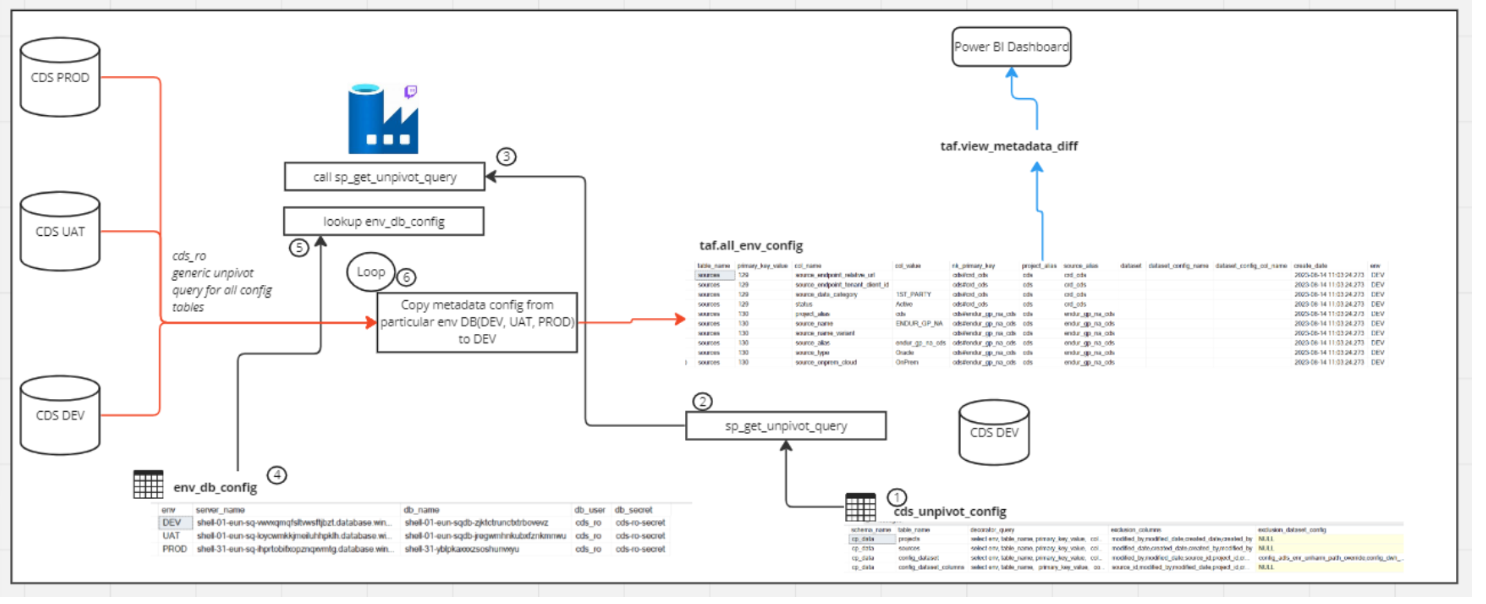
This verifier compares metadata configuration across 3 CDS Database environment i.e. DEV, UAT and PROD and reports the difference.

### Business Value

This verifier will help CDS team to check if there are any difference in metadata configuration , those difference might cause the datasets ingestion to run unexpectedly, hence it is important to know if there are difference of metadata exists, accordingly team can make the changes to keep the metadata configuration in sync.

There are configuration which meant to be different, tool has configurable way to exclude those configuration.

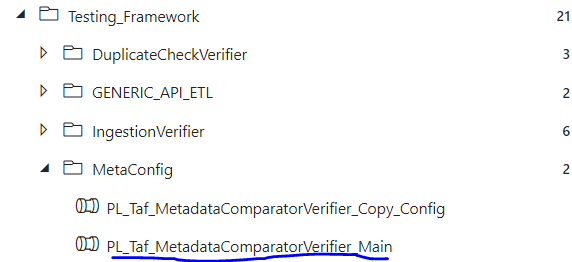
### High Level Flow



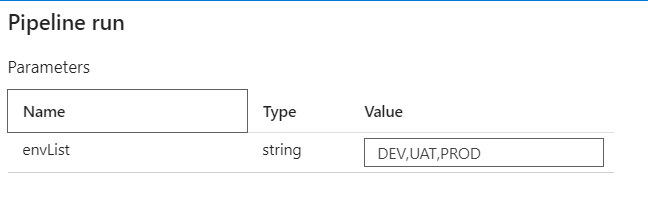
### How to execute

CDS Ingestion ADF ->

Testing Framework/MetaConfig/PL\_Taf\_MetadataComparatorVerifier\_Main



Following are the parameters



### Pipeline Input

envList=hardcoded, do not change, will be removed in future release

### Setup Required

We have a couple of config tables which need to be configured before we run this verifier

Table Name: taf.env\_db\_config

Purpose: To keep CDS metadata DB connection details for each env

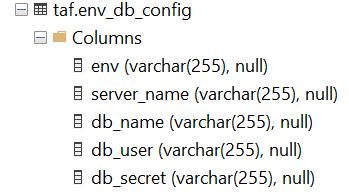
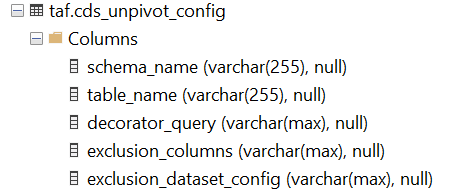


Table Name: taf.cds\_unpivot\_config

Purpose: To keep details of unpivoting cds tables



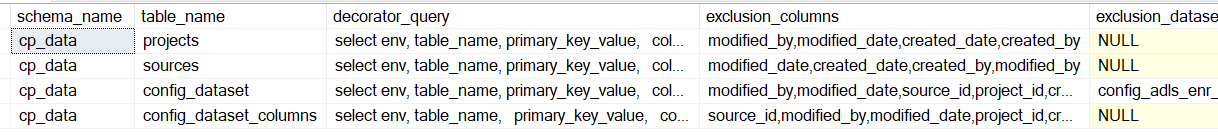
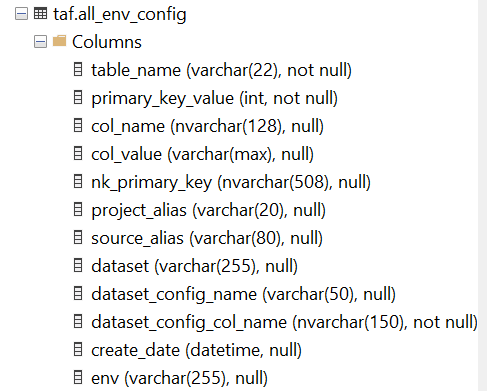


Table Name: taf.all\_env\_config

Purpose: To copy and keep metadata configuration from various environment so that comparison can be performed



### Logic

1. Get the database connection details for all CDS env DB from env\_db\_config
2. Generate CDS unpivot queries
3. Loop through each env
4. For each env
   1. Copy metadata config from particular env to DEV env using the CDS unpivot query

Following query can be used thereafter to see the configs with difference

select \* from taf.view\_metadata\_diff  
where (dev\_col\_value <> uat\_col\_value or  
dev\_col\_value <> prod\_col\_value or  
uat\_col\_value <> prod\_col\_value  
)

### Technology Involved

ADF, SQL Stored proc

### Resolution of Issues found

How to resolve issue found by TAF for this verifier

If the difference is not meant to be existing then lower environment config should be changed to match prod.

### Artefact Details

|  |  |  |
| --- | --- | --- |
| SrNo | Artefact | Artefact Detail |
| 1 | [PL\_Taf\_MetadataComparatorVerifier\_Main.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_MetadataComparatorVerifier_Main.json) | Get all db config  Loop through them |
| 2 | [PL\_Taf\_MetadataComparatorVerifier\_Copy\_Config.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_MetadataComparatorVerifier_Copy_Config.json) | Copy config from env db to dev |

### Limitations

### Known Issues

### Required Maintenance

### Periodicity

This can be executed daily

### Applicability with Source Type

### Reusability of this verifier with other teams

This verifier can be used by other teams without any changes.

## MetadataHistory

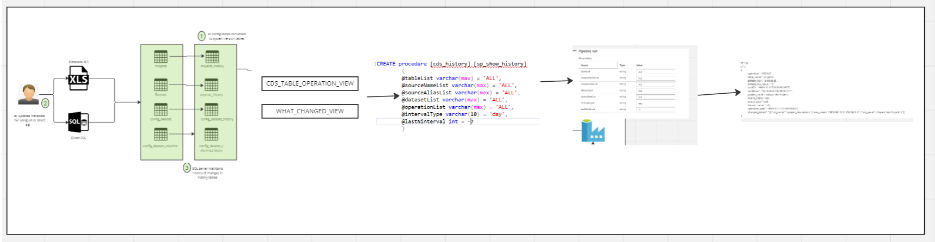
### Objective

This component will enable CDS to maintain history of changes at SQL Table level. It uses temporal table (aka. System versioned ) feature of SQL Server.

### Business Value

This verifier will help CDS team to triage any issues related to changes occurred in metadata. If there are datasets which are not working as expected

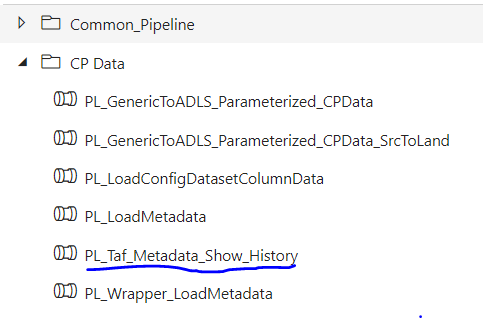
### High Level Flow



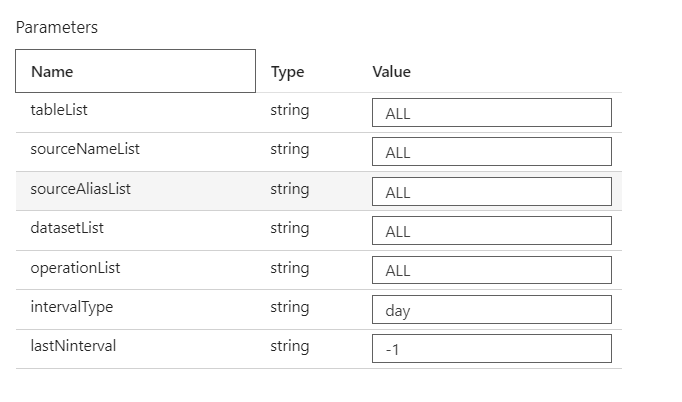
### How to execute

CDS Ingestion ADF ->

CP Data/PL\_Taf\_Metadata\_Show\_History



Following are the parameters



### Pipeline Input

tableList=*list of tables, ALL for all tables*

sourceNameList=*list of SourceNames, ALL for all Sourcenames*

sourceAliasList=*list of SourceAliases, ALL for all SourceAliases*

datasetList=*list of Datasets, ALL for all Datasets*

operationList=*list of operation, ALL for all Operations[UPDATE, DELETE, INSERT]*

intervalType=*day/minute*

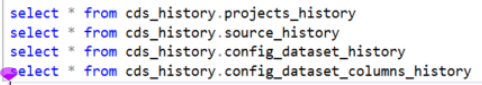
lastNinterval=*A number to show last N interval(day/minute)*

### Setup Required

1. Add ValidTo and ValidFrom columns in all the config tables, This is required by SQL Server to convert a table to system versioned (temporal table)
2. Convert all the config tables system versioned tables  
   Below SQL script has the code for the same

<https://github.com/sede-x/CDS_SQL/blob/main/CDS_HISTORY/temporal_ddl.sql>

1. History tables for each config tables will be created by SQL Server when we convert a table to system versioned table



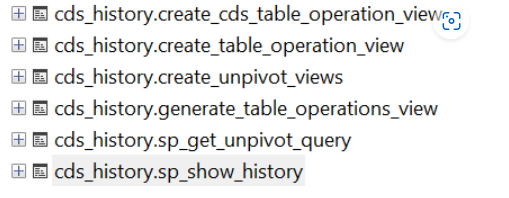
With above setup whenever there is a DML operation on main table, there will be changes in history table maintained by SQL Server

|  |  |
| --- | --- |
| **Operation on Main table** | **What will happen in History table** |
| INSERT | Nothing will happen in History table |
| UPDATE | Record updated in main table will be inserted will old value(s) in History table |
| DELETE | Record deleted in main table will be inserted in History table |

### Logic

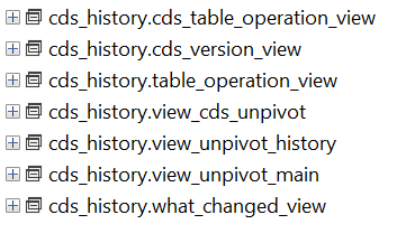
**Generate meaningful report tables out of main table and history table**

1. Below are the list of stored procs which will create views



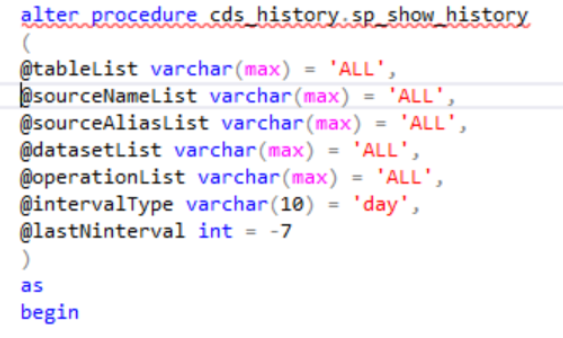
Stored proc will identify all the DML operations performed on config table and populate couple of result tables, For update operation it will also identify which columns changed with old and new values

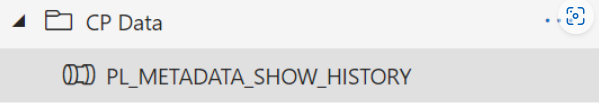
Below are the views

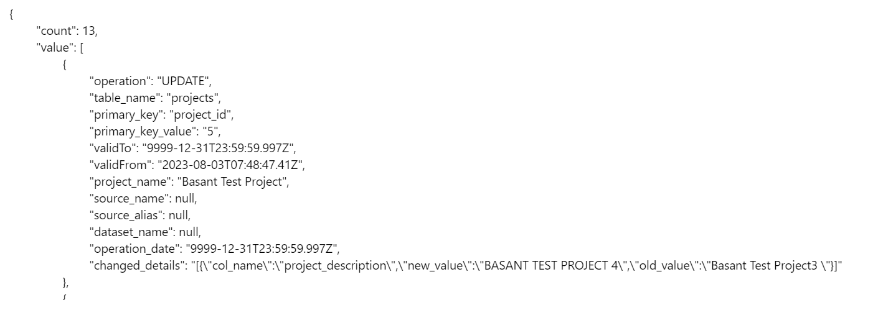


**Stored proc and ADF to query the report tables**

* cds\_history.sp\_show\_history to show the historical changes based on various parameters
* This stored proc queries cds\_table\_operation\_view and what\_changed\_view views







### Technology Involved

ADF, SQL Stored proc, Views

### Resolution of Issues found

### Artefact Details

GIT Link  
<https://github.com/sede-x/CDS_SQL/tree/main/CDS_HISTORY>

|  |  |  |  |
| --- | --- | --- | --- |
| **SrNo** | **Name** | **Type** | **Details** |
| 1 | sp\_get\_unpivot\_query | SP | This SP will give UNPIVOT query based on columns present in table and primary key of the table |
| 2 | create\_unpivot\_views | SP | This SP will call sp\_get\_unpivot\_query and create unpivot history and unpivot main views |
| 3 | generate\_table\_operations\_view | SP | This will generate tables operation SQL |
| 4 | create\_table\_operation\_view | SP | This will call generate\_table\_operations\_view and create table operation view |
| 5 | create\_cds\_table\_operation\_view | SP | This will decorate table\_operation\_view with application(in this case CDS) specific columns |
| 6 | sp\_show\_history | SP | This will join cds\_table\_operations\_view and what\_changed\_view |
| 7 | view\_unpivot\_history | VIEW | This is view for unpivoting history table, it featches only data which has got updated |
| 8 | view\_unpivot\_main | VIEW | This is view for unpivoting main table |
| 9 | view\_cds\_unpivot | VIEW | THis is union of history and main pivot |
| 10 | cds\_version\_view | VIEW | This will assign version to each record |
| 11 | what\_changed\_view | VIEW | THis will comparte old and new values of modified columns and show the result |
| 12 | table\_operation\_view | VIEW | THis is tabke operation view which shows INSERT, DELETE, UPDATE happened on a table |
| 13 | cds\_table\_operation\_view | VIEW | This is decorated view to have CDS specific columns |

### Limitations

### Known Issues

### Required Maintenance

### Periodicity

This can be executed whenever there is a change in metadata

### Applicability with Source Type

### Reusability of this verifier with other teams

This verifier can be used by other teams without any changes.

## Azure API Ingestor

### Objective

This component is a framework to ingest Azure Microsoft Rest API data to a SQL table.

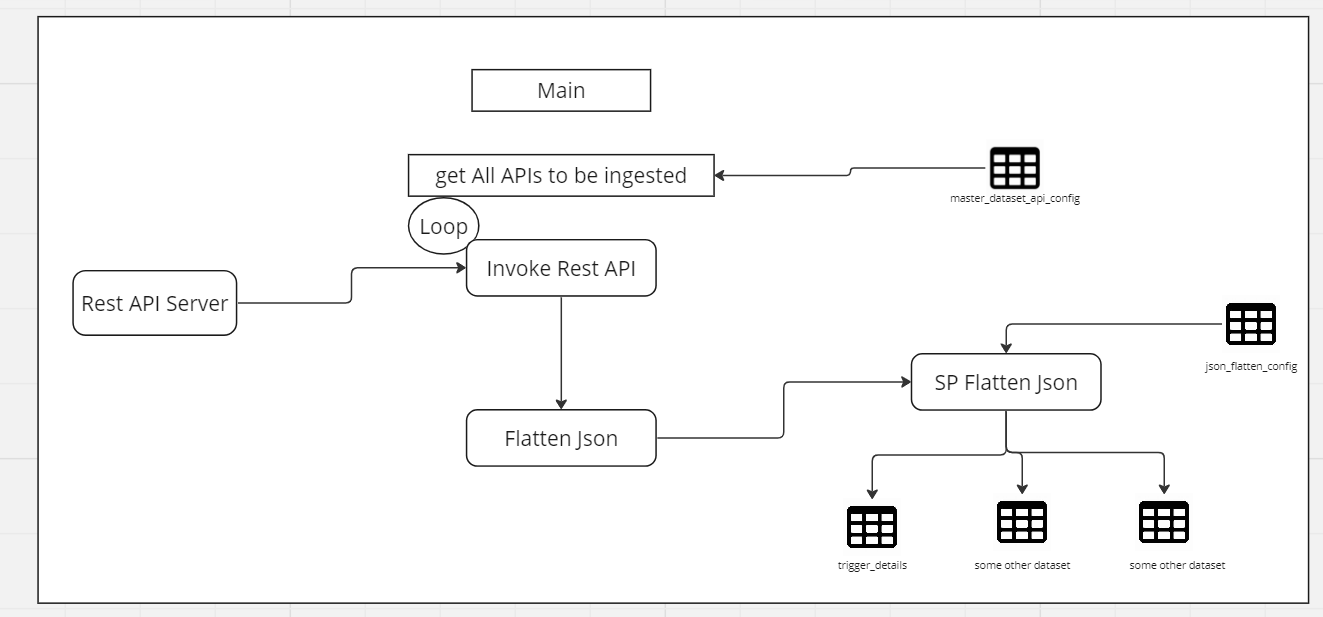
For instance this component can be used to ingest Azure Data Factory Trigger details to a SQL Table which captures trigger details

### Business Value

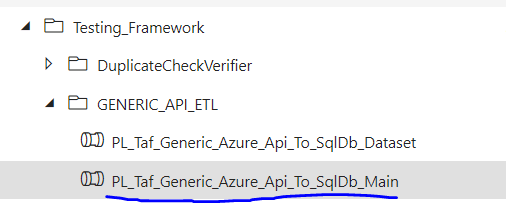
This component is a no-code solution to ingest Rest API(Microsoft Azure) and parse(flatten) the out of rest API and store the same in a SQL Table. Following are the business benefits

1. Many times we need to ingest data from Microsoft Azure API to SQL tables so that these information can be consumed for various purpose
2. Currently we have configured Trigger API to ingest data about Azure Data Factory Triggers which is being consumed for various purpose
3. This framework can be configured for various other MS Rest APIs such as Pipeline, LinkedService, Azure Data Factory pipeline out etc without any code change
4. This will reduce the development and testing effort drastically

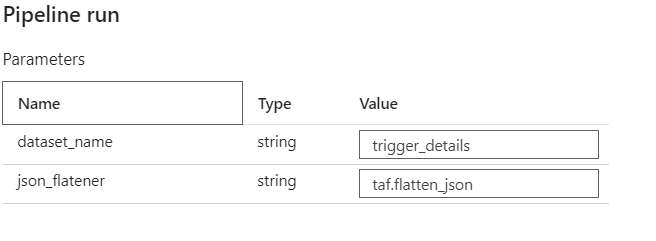
### High Level Flow



### How to execute



Following are the parameters



### Pipeline Input

Dataset\_name=Name of the dataset to be ingested

Json\_flatenner=hardcoded, will be removed in later release

### Setup Required

We have a couple of config tables which need to be configured before we run this verifier

Table Name: taf.master\_dataset\_api\_config

Purpose: To keep master config for APIs to be ingested

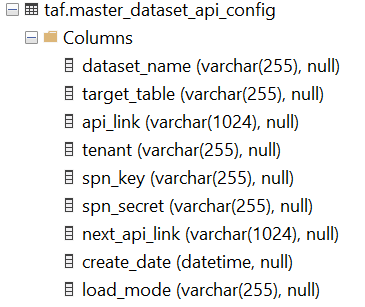
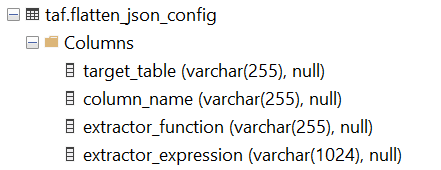
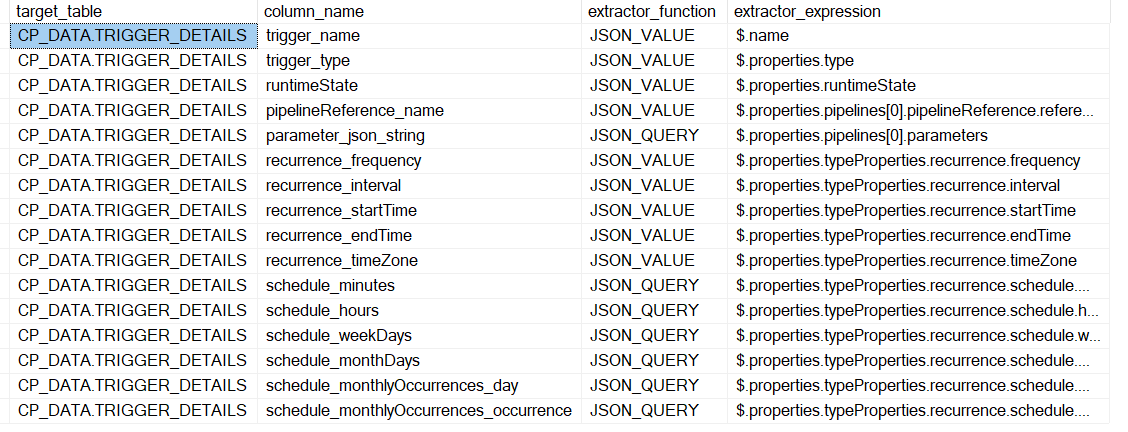


Table Name: taf.flatten\_json\_config

Purpose: To keep details of flattening the JSON returned by the API





### Logic

1. Get all the APIs to be ingested from master\_dataet\_api\_config
2. Loop through all the API
3. For each API
   1. Call Rest API and get the response
   2. Call SP Flatten Json to flatten the json and store in respective table for that dataset

### Technology Involved

ADF, SQL Stored proc

### Resolution of Issues found

### Artefact Details

|  |  |  |
| --- | --- | --- |
| SrNo | Artefact | Artefact Detail |
| 1 | [PL\_Taf\_Generic\_Azure\_Api\_To\_SqlDb\_Main.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_Generic_Azure_Api_To_SqlDb_Main.json) | Orchestrator, get all APIs and loop |
| 2 | [PL\_Taf\_Generic\_Azure\_Api\_To\_SqlDb\_Dataset.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_Generic_Azure_Api_To_SqlDb_Dataset.json) | Hit Rest API  Call SP to flatten and store in table |
| 3 | [taf.flatten\_json.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.flatten_json.StoredProcedure.sql) | Flatten Json SP |
| 4 | [taf.flatten\_single\_json.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.flatten_single_json.StoredProcedure.sql) | Flatten Single Json SP |
| 5 | [taf.master\_dataset\_api\_config.Table.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/Table/taf.master_dataset_api_config.Table.sql) | Table to store API details |
| 6 | [taf.flatten\_json\_config.Table.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/Table/taf.flatten_json_config.Table.sql) | Table to store extractor details |

### Limitations

### Known Issues

### Required Maintenance

### Periodicity

This can be executed daily

### Applicability with Source Type

### Reusability of this verifier with other teams

This verifier can be used by other teams without any changes.

## IngestionTestBasedVerifier

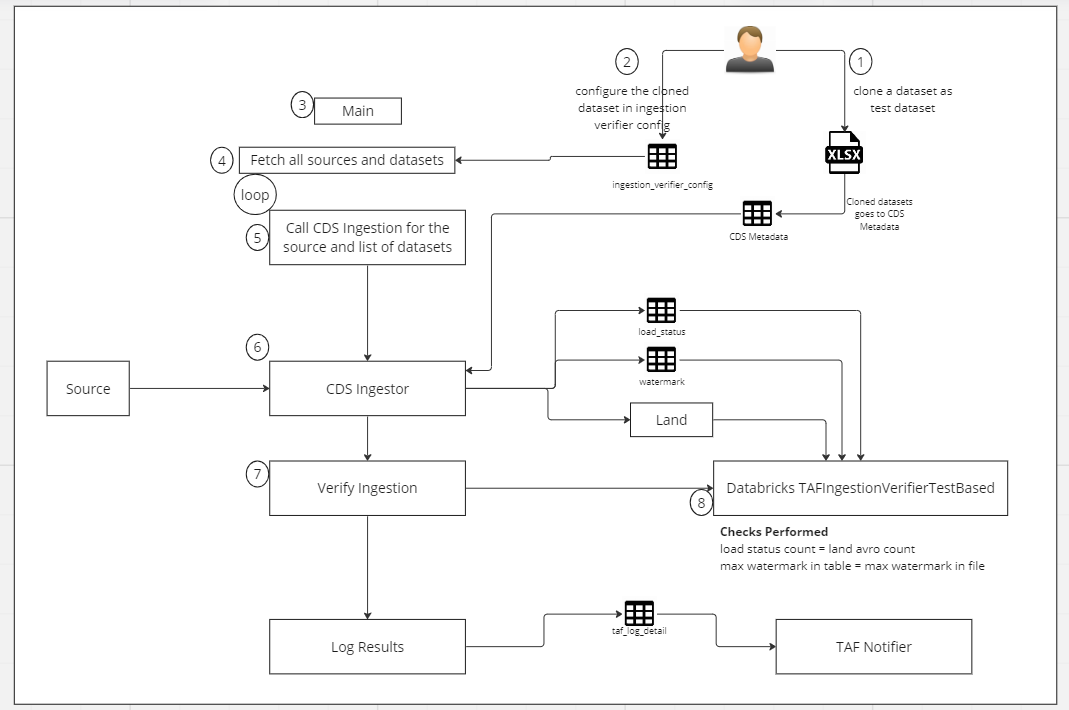
### Objective

This verifier will run the CDS Ingestion for cloned test datasets and verify whether it has worked as expected, and this way it validates that the ingestion component of CDS is working fine

### Business Value

* This verifier will help CDS team to validate that the CDS Ingestion Component is working as expected, if there are recent changes to the ingestor then this verifier can be executed to know that the ingestor is not broken.
* This verifier can be used to perform regression testing for Ingestion component of CDS.
* Please note that CDS team have ingestors with respect to the source type. Currently IngestionTestBasedVerifier is supporting subset of these ingestors.

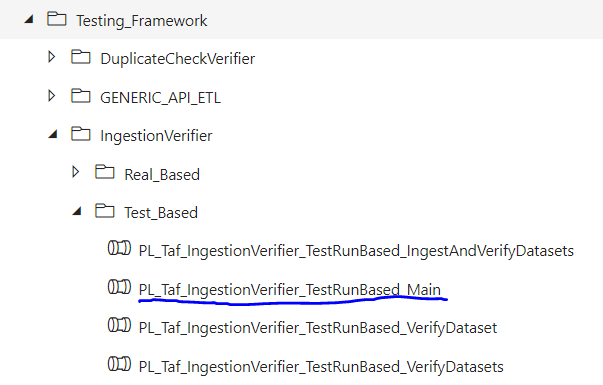
### High Level Flow



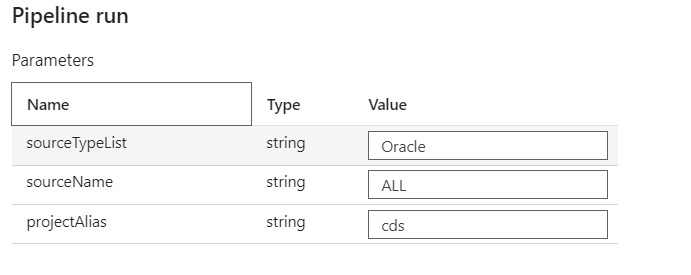
### How to execute

CDS Ingestion ADF ->

Testing Framework/IngestionVerifier/Test\_Based/ PL\_Taf\_IngestionVerifier\_TestRunBased\_Main



Following are the parameters



### Pipeline Input

sourceTypeList=list of sourceTypes

SourceName=Name of source

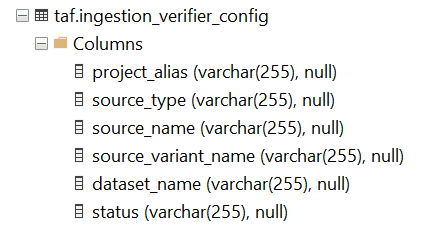
projectAlias= Project Alias

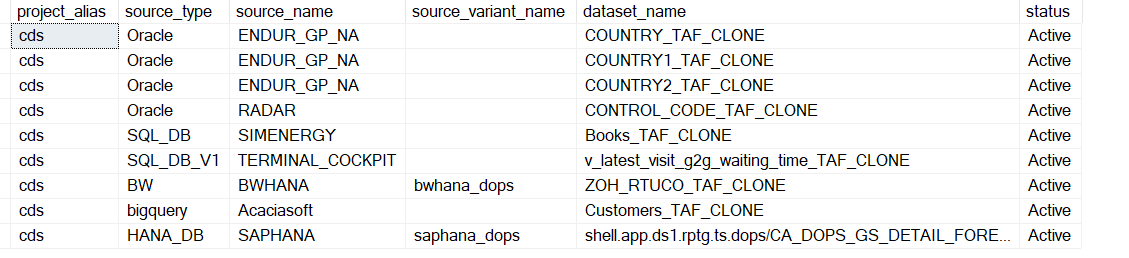
### Setup Required

1. Tester need to decide on which dataset to be cloned to make it as test data
2. Tester need to clone the dataset in metadata xls, change the parameters accordingly and run the pl load metadata to load the same in CDS metadata database
3. Tester need to configure these datasets in taf.ingestion\_verifier\_config

Table Name: taf.ingestion\_verifier\_config

Purpose: To keep cloned dataset config





### Logic

1. Get the source and datasets from config
2. Loop through it
   1. Call CDS Ingestion
3. Verify Ingestion
   1. Load status count should be same as land avro file count
   2. Watermark value in table should be same as max value of that column in land avro file
4. Call TAF Verifier to send mail if the test failed

### Technology Involved

ADF, SQL Stored proc

### Resolution of Issues found

How to resolve issue found by TAF for this verifier

If the ingestion is not working then we need debug the reason and solve it

### Artefact Details

|  |  |  |
| --- | --- | --- |
| SrNo | Artefact | Artefact Detail |
| 1 | [PL\_Taf\_IngestionVerifier\_TestRunBased\_Main.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_IngestionVerifier_TestRunBased_Main.json) | Get source and datasets from config |
| 2 | [PL\_Taf\_IngestionVerifier\_TestRunBased\_IngestAndVerifyDatasets.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_IngestionVerifier_TestRunBased_IngestAndVerifyDatasets.json) | Ingest and Verify |
| 3 | [PL\_Taf\_IngestionVerifier\_TestRunBased\_VerifyDatasets.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_IngestionVerifier_TestRunBased_VerifyDatasets.json) | Verify Datasets |
| 4 | [PL\_Taf\_IngestionVerifier\_TestRunBased\_VerifyDataset.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_IngestionVerifier_TestRunBased_VerifyDataset.json) | Verify Dataset |
| 5 | [IngestionVerifier.py](https://github.com/sede-x/CDS_ADB/blob/develop/Shared/CP_Data/Test_framework/IngestionVerifier.py) | ADB code to verify dataset |
| 6 | [taf.ingestion\_verifier\_config.Table.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/Table/taf.ingestion_verifier_config.Table.sql) | SQL for table ingestion\_verifier\_config |

### Limitations

Currently supports few source types

Need to add additional test/checks

### Known Issues

### Required Maintenance

Need to maintain cloned dataset config

Need to make entry for cloned dataset in metadata CDS table

### Periodicity

This can be executed when there is a change in the code of ingestion framework

### Applicability with Source Type

|  |  |  |
| --- | --- | --- |
| Source Type | No of Sources | Supported |
| Oracle | 15 | Yes |
| SQL\_DB | 9 | Yes |
| Avro | 4 | Can be supported |
| BW | 4 | Yes |
| SQL\_DB\_V1 | 3 | Yes |
| API | 3 | Can be supported |
| FILESYSTEM | 2 | Can be supported |
| HANA\_DB | 2 | Yes |
| bigquery | 1 | Yes |
| TESLAAPI | 1 | Can be supported |
| SFTP | 1 | Can be supported |
| SharePoint | 1 | Can be supported |

### Reusability of this verifier with other teams

This verifier can be used by other teams with minimal changes.

## IngestionRealBasedVerifier

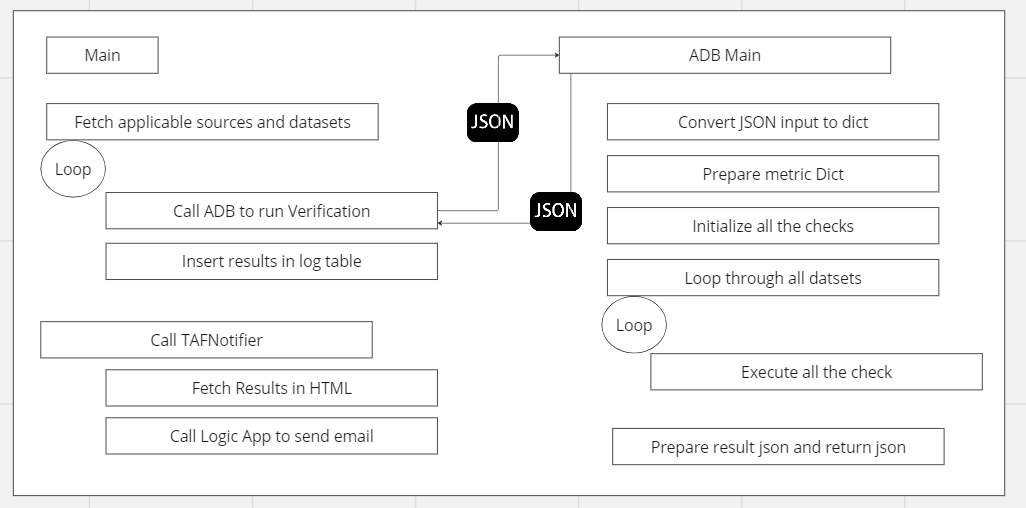
### Objective

This verifier will run various checks post ingestion of the datasets.

### Business Value

This verifier will help to check if ingestion for all the datasets are working as expected in automated manner

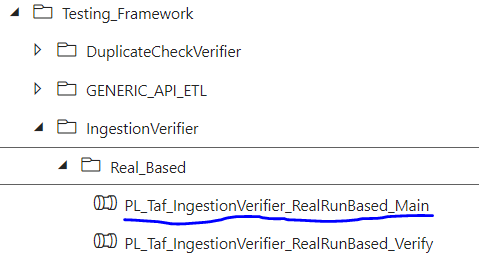
### High Level Flow



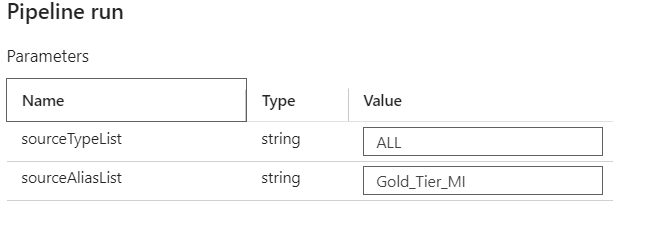
### How to execute

CDS Ingestion ADF ->

Testing Framework/IngestionVerifier/Real\_Based/ PL\_Taf\_IngestionVerifier\_RealRunBased\_Main



Following are the parameters



### Pipeline Input

sourceTypeList=list of sourceTypes

sourceAliasList =List of source aliases

### Setup Required

No setup required

### Logic

* ADF
  + Get the sources for input param
  + Loop through the sources
  + For each source
    - Get the datasets
    - Call ADB IngestionVerfierRealBased
  + Get the results from JSON and call SP to insert the results in log tables
    - Table: taf:tag\_log\_detail – This will keep dataset/verifier level log
    - Table: taf:tag\_log\_check\_detail – This will keep dataset/verifier/check level log
    - Table: taf:tag\_log\_metric\_detail – This will keep dataset/verifier/metric level log
* IngestionVerfierRealBased
  + Get all the json input from ADF
  + Convert all the json to dict
  + MetricCollector – to collect the metrics in the form of dict(map of key and value) which will be used in running the checks,
  + InitializeChecks – List of check functions to be executed
  + RunChecks – This will run all the checks and store the results in a metric
  + Return Result Json to ADF

### Checks Covered

|  |  |  |
| --- | --- | --- |
| SN | Check Name | Check Detail |
| 1 | check\_land\_file\_load\_status\_count\_comparison | Latest run of dataset from load\_status, target\_count should be matching with count of latest avro file |
| 2 | check\_watermark\_value | For load type=”Inc” , max inc column of avro land file should be matching with watermark for that latest in watermark table, please note in some cases when inc query is present this might not be possible to check |
| 3 | check\_trend\_compare\_wrt\_avg\_last\_N\_run | Trend Check - Last N runs avg and current run count should have threshold deviation |
| 4 | check\_land\_file\_access\_from\_adb | ADB is able to read the avro file |
| 5 | check\_one\_trigger\_exist\_for\_the\_dataset | Dataset should have trigger present |
| 6 | check\_one\_trigger\_exist\_for\_the\_dataset | Dataset should not have more than one trigger present |
| 7 | check\_if\_full\_load\_count\_is\_more\_than\_previous | For full load, count of current run should always be greater or equal than prev count |
| 8 | check\_columns\_removed\_comparing\_to\_previous\_landing\_files | Schema diff between current avro file vs prev avro landing file, there should not be any column removed |
| 9 | check\_columns\_added\_comparing\_to\_previous\_landing\_files | Schema diff between current avro file vs prev avro landing file, if there is column added then it should propagate to unharm and raw layer as well |
| 10 | check\_datatype\_changed\_comparing\_to\_previous\_landing\_files | Schema diff between current avro file vs prev avro landing file, is there a datatype mismatch |
| 11 | check\_if\_duplicate\_record\_in\_landing | Duplicate check on Landing if primary keys are present |
| 12 | TBD | Check based on Check Run date vs Latest Non Zero load vs Latest Trigger Run vs Latest Success Trigger Run vs Trigger Frequency – This to ensure triggers are running and loading some data as per frequency |
| 13 | TBD | Source vs Unharm Reconciliation |
| 14 | TBD | Max of source vs watermark(this will tell us how stale we are) |
| 15 | TBD | Is there any col in source table which is not present in landing |
|  |  |  |

### Technology Involved

ADF, SQL Stored proc

### Resolution of Issues found

How to resolve issue found by TAF for this verifier

If the ingestion is not working then we need debug the reason and solve it

### Artefact Details

|  |  |  |
| --- | --- | --- |
| SrNo | Artefact | Artefact Detail |
| 1 | [PL\_Taf\_IngestionVerifier\_RealRunBased\_Main.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_IngestionVerifier_RealRunBased_Main.json) | Main Orchestration, get all the datasets for source, loop through source |
| 2 | [PL\_Taf\_IngestionVerifier\_RealRunBased\_Verify.json](https://github.com/sede-x/CPData_Ingestion/blob/develop/pipeline/PL_Taf_IngestionVerifier_RealRunBased_Verify.json) | Call metadata SP to load metadata for datasets  Call ADB to verify all the datasets |
| 3 | [IngestionVerifierRealBasedMain.py](https://github.com/sede-x/CDS_ADB/blob/develop/Shared/CP_Data/Test_framework/IngestionVerifierRealBasedMain.py) | ADF will call this main ADB |
| 4 | [InputConverter.py](https://github.com/sede-x/CDS_ADB/blob/develop/Shared/CP_Data/Test_framework/InputConverter.py) | This converts json inputs to dict |
| 5 | [MetricCollector.py](https://github.com/sede-x/CDS_ADB/blob/develop/Shared/CP_Data/Test_framework/MetricCollector.py) | This set metricDict from all the input dict, on metric dict all the rules will be applied |
| 6 | [OutputConverter.py](https://github.com/sede-x/CDS_ADB/blob/develop/Shared/CP_Data/Test_framework/OutputConverter.py) | Convert the output to json |
| 7 | [CheckFactory.py](https://github.com/sede-x/CDS_ADB/blob/develop/Shared/CP_Data/Test_framework/CheckFactory.py) | This will have all the check functions |
| 8 | [CheckExecuter.py](https://github.com/sede-x/CDS_ADB/blob/develop/Shared/CP_Data/Test_framework/CheckExecuter.py) | This is orchestration of check execution |

### Limitations

Currently supports few source types

Need to add additional test/checks

### Known Issues

### Required Maintenance

Need to maintain cloned dataset config

Need to make entry for cloned dataset in metadata CDS table

### Periodicity

This can be executed when there is a change in the code of ingestion framework

### Applicability with Source Type

All source types will be supported

### Reusability of this verifier with other teams

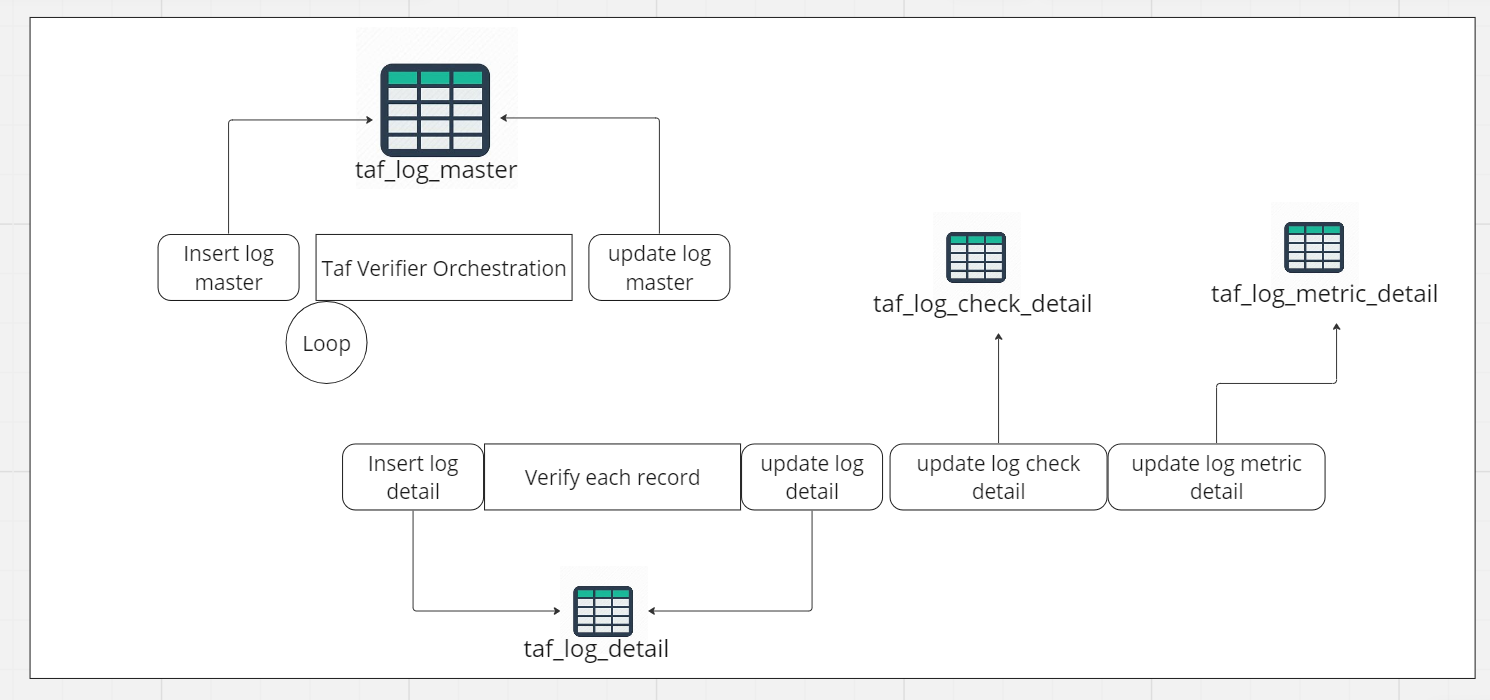
This verifier can be used by other teams with some changes.

## TAFLogger

### Purpose

Taf logger have log tables and procedures to insert and update log tables

### High Level Diagram



### Artefact Details

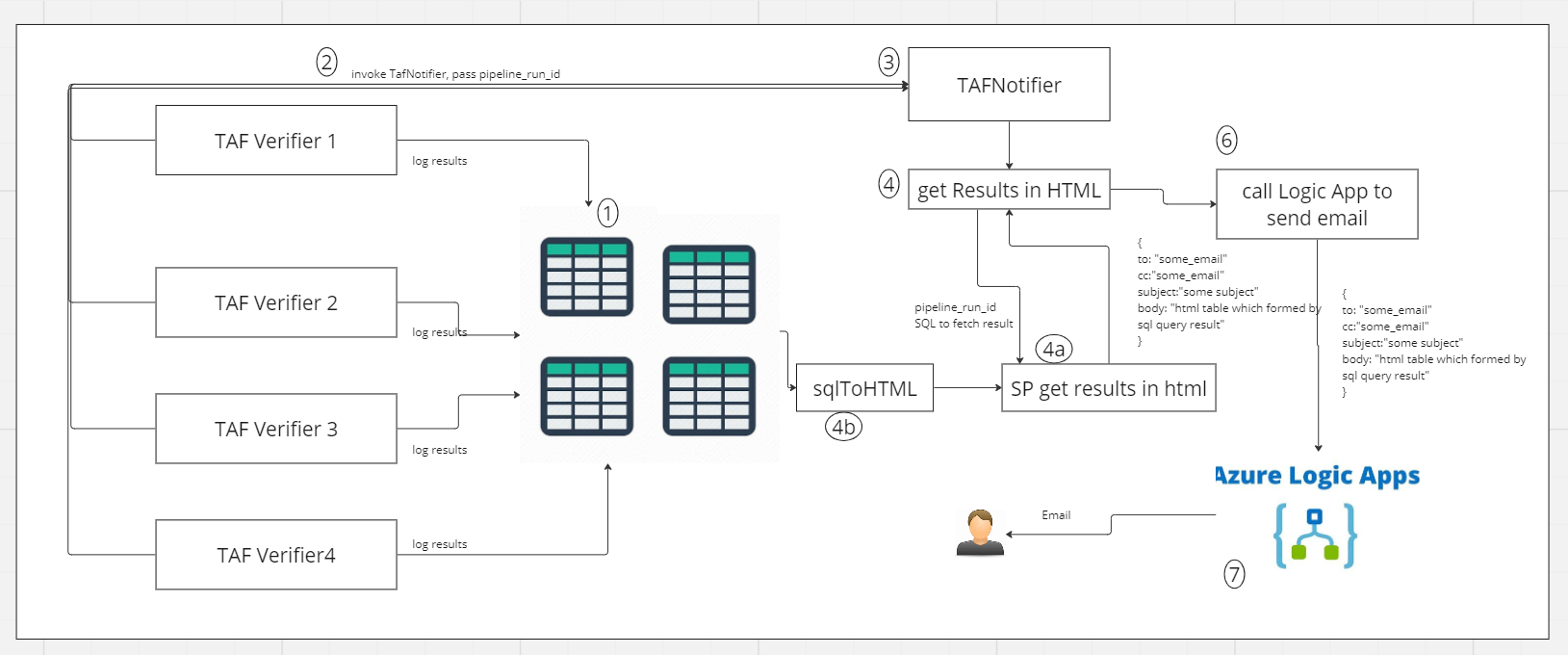
|  |  |  |
| --- | --- | --- |
| SrNo | Artefact | Details |
| 1 | [taf.taf\_log\_master.Table.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/Table/taf.taf_log_master.Table.sql) | Log master |
| 2 | [taf.taf\_log\_detail.Table.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/Table/taf.taf_log_detail.Table.sql) | Log detail |
| 3 | [taf.taf\_log\_check\_detail.Table.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/Table/taf.taf_log_check_detail.Table.sql) | Log check detail |
| 4 | [taf.taf\_log\_metric\_detail.Table.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/Table/taf.taf_log_metric_detail.Table.sql) | Log metric detail |
| 5 | [taf.sp\_insert\_taf\_log\_master.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.sp_insert_taf_log_master.StoredProcedure.sql) | Insert log master |
| 6 | [taf.sp\_insert\_taf\_log\_detail.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.sp_insert_taf_log_detail.StoredProcedure.sql) | Insert log detail |
| 7 | [taf.sp\_update\_taf\_log\_detail\_json.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.sp_update_taf_log_detail_json.StoredProcedure.sql) | Insert log detail with check and metric |
| 8 | [taf.sp\_update\_taf\_log\_detail.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.sp_update_taf_log_detail.StoredProcedure.sql) | Update log detail |
| 9 | [taf.sp\_update\_taf\_log\_master.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.sp_update_taf_log_master.StoredProcedure.sql) | Update log master |

## TAFNotifier

### Purpose

This component sends email post TAF run.

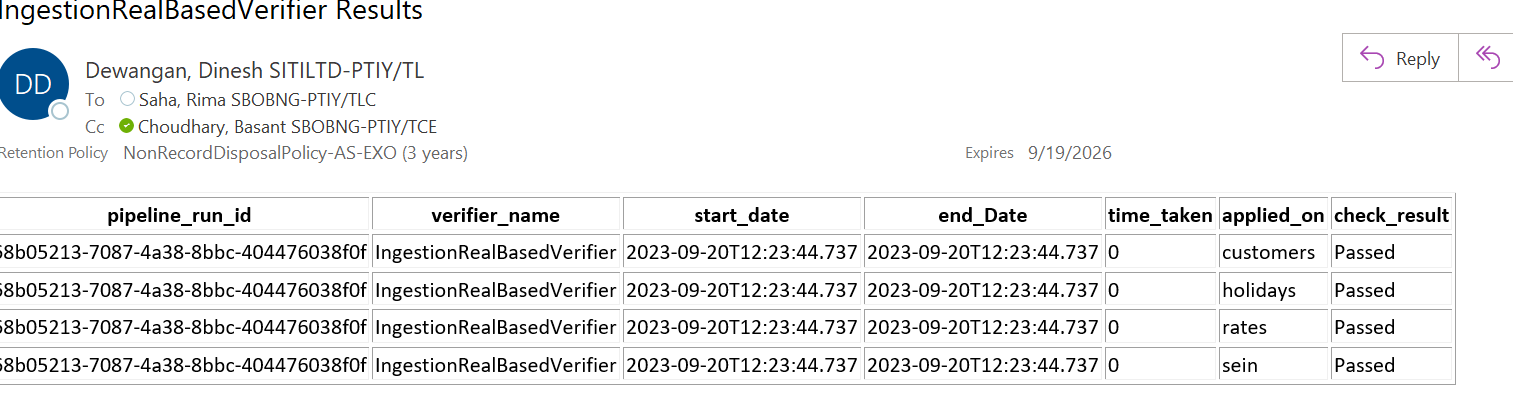
### High Level Diagram



### Artefact Details

|  |  |  |
| --- | --- | --- |
| Srno | Artefact | Details |
| 1 | [taf.get\_verifier\_results\_in\_html.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.get_verifier_results_in_html.StoredProcedure.sql) | SP to get result in HTML |
| 2 | [taf.spQueryToHtmlTable.StoredProcedure.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/SP/taf.spQueryToHtmlTable.StoredProcedure.sql) | SQL to HTML |
| 3 | [la\_cds\_send\_email - Microsoft Azure](https://portal.azure.com/#@ShellCorp.onmicrosoft.com/resource/subscriptions/f8ce0582-0af8-4bed-83e9-d83690b0f72d/resourceGroups/AZ-AS-RGP-EX-N-DEV-COMMON_DATA_SERVICES/providers/Microsoft.Logic/workflows/la_cds_send_email/logicApp) | Logic App |
| 4 | [insert\_taf\_config.sql](https://github.com/sede-x/CDS_SQL/blob/main/TAF/DML/insert_taf_config.sql) | For email IDs |
|  |  |  |
|  |  |  |
|  |  |  |

### Sample Email



## ETLVerifier

To be developed

## ProfileEngine

To be developed

## DataQualityEngine(PyDeequChecks, GreatExpectationChecks, SQLChecks, DuplicateChecks, TrendChecks)

To be developed

## ReconEngine

To be developed

# Data-Driven Testing

# Test Case Prioritization

# Mocking and Stubbing Data Sources

# Data Quality Testing

# Data Validation Rules

# Anomaly Detection

# Data Profiling

# Security Testing

# Data Security Measures

# Compliance and Reporting

# Continuous Improvement

# Collecting and Analyzing Test Metrics

# Conclusion

# Summary of Key Takeaways

# Future Enhancements and Roadmap

# Appendices

Glossary of Terms