## ****Section 17: Automation Components and Execution Flow****

### **17.1 Overview**

The automation framework is designed to validate Guidewire data ingestion using a **CI/CD-enabled**, **Databricks notebook-driven**, and **Delta Lake-backed** architecture. The framework supports schema validation, DDL comparisons, ingestion checks, and real-time reporting using dashboards.

It is implemented and deployed via **Azure DevOps**, executed within **Databricks Workflows**, and managed through version-controlled pipelines and parameterized notebooks.

### **17.2 Folder Structure in Azure DevOps Repo**

pgsql

CopyEdit

ODS\_SIT\_AUTOMATION\_POC/

│── notebooks/

│ │── UKS\_SIT\_DW\_E2E/

│ │ │── ODS\_BRONZE\_VALIDATION\_ESSENTIAL.ipynb → Core data validation logic

│ │ │── DLT\_REFRESH\_PIPELINE.ipynb → Refreshes DLT tables

│ │ │── DDL\_COMPARISION\_ORA\_ODS\_S3.ipynb → DDL structure comparison

│ │ │── ORA\_READREPLICA\_SCHEMA\_CHECK.ipynb → Schema diff checks

│ │ │── REFERENCE\_FILE/

│ │ │ │── exclusion\_list.csv → Tables to skip

│ │ │ │── ora\_schema\_file.csv → Reference Oracle schema

│ │ │ │── readreplica\_schema\_file.csv → Reference PostgreSQL schema

│ │── Release\_7\_RunNoteBook\_Deploy.ipynb → Notebook to trigger batch validation

│ │── Create\_Databricks\_Job.ipynb → Utility notebook to create/update jobs

│── azure-pipelines.yml → CI/CD pipeline file

│── EDM Scripts.sql → Metadata reference SQLs

│── IDW\_Load\_Test.sql → Load/ingestion test SQLs

│── README.md → Setup and usage guide

### **17.3 CI/CD and Deployment Process via Azure DevOps**

| **Stage** | **Action** | **Tool/Trigger** |
| --- | --- | --- |
| **Dev Commit** | Code and notebooks committed to DevOps Git | Developer |
| **CI Trigger** | azure-pipelines.yml builds and validates notebooks | Azure Pipelines |
| **Deploy to DEV** | Notebooks deployed to DEV Databricks workspace | Auto-trigger post-CI |
| **Approval Workflow** | Post-DEV testing, team lead approves push to TEST | Manual Approval |
| **Deploy to TEST** | Validated notebooks promoted to TEST workspace | Azure DevOps Release |
| **Live Promotion** | TEST workflow is copied to PROD via **Data Engineering’s deployment utility** | Data Engineering with config adjustment |

### **17.4 Workflow Execution and Orchestration**

#### **In TEST Environment**

* **Databricks Workflow** defines multiple tasks:
  + ODS\_BRONZE\_VALIDATION\_ESSENTIAL.ipynb: Validates bronze layer ingestion
  + DDL\_COMPARISION\_ORA\_ODS\_S3.ipynb: Compares Oracle vs ODS DDL structures
  + DLT\_REFRESH\_PIPELINE.ipynb: Refreshes Delta Live Tables with validation output
  + ORA\_READREPLICA\_SCHEMA\_CHECK.ipynb: Validates schema from Read Replica
* **Triggering**:
  + Manual via Release\_7\_RunNoteBook\_Deploy.ipynb
  + Scheduled via **Databricks Job Scheduler**
* **DLT Tables**:
  + Store validation results such as:
    - Table-wise pass/fail
    - Row count mismatches
    - Schema differences
    - SCD and duplicate issues
* **Dashboards**:
  + Built on top of DLT result views
  + Provide real-time views for QA, leads, and management
  + Visual KPIs: % of tables passed, schema drift count, daily data load size, error trend

### **17.5 Dashboard Features & Usability**

| **Feature** | **Description** |
| --- | --- |
| **Live Status View** | Auto-refresh views showing per-table validation results |
| **Drill-down Capability** | Clickable dashboards to view failed records, mismatched schema |
| **Validation Categories** | Filters for Schema/Count/Data/Referential/Performance |
| **Export Options** | CSV/Excel downloads for manual reporting or offline review |
| **Ease of Use** | No SQL knowledge required, designed for QA and business reviewers |

### **17.6 Production Workflow Transition**

* **Workflow Copy**:
  + Test workflow structure is **cloned to Production** via a controlled copy process provided by the **Data Engineering team**
  + Only config files, paths, and job parameters are adjusted to reflect PROD dataset references.
* **Validation Logic**:
  + Exactly mirrors what was tested in TEST
  + Only data paths and clusters differ

### **17.7 Test Results – Structure & Reporting**

| **Test Result Table Fields** | **Description** |
| --- | --- |
| source\_table | Table name being validated |
| test\_type | Schema / Count / Data / SCD / Duplicate |
| status | PASS / FAIL |
| error\_detail | Mismatch info (column, value, row ID) |
| execution\_time | Duration of validation |
| run\_id | Unique ID for traceability |
| run\_date | Timestamp of test execution |

### **17.8 Integration with Teams & Outlook**

* Work in progress: Email alerts for test failures or validation breaches
* Plan to integrate:
  + Daily summary of test runs to team DL (via Outlook)
  + Scheduled reports post major deployment

### **17.9 JIRA Integration Limitation**

| **Challenge** | **Details** |
| --- | --- |
| **Direct Integration Blocked** | Self-hosted JIRA cannot connect directly to cloud-based Databricks due to firewall restrictions |
| **Workaround** | Export test metrics from DLT table views into Excel or CSV |
| **Manual Update Process** | Dedicated QA/test lead updates corresponding test cases manually in JIRA using test result logs |

Would you like this paired with:

* A **workflow diagram**
* A **deployment sequence visual**
* Or included into the master test plan document?

Let me know and I can generate those next.

##### You said:

produce Elaborate detail about the orchestration, workflow schedule, test result view in tables , test results contains details about the validation results and how the validation being performed and then dashboard funcnality and usabity , easiness

###### ChatGPT said:

Certainly! Here's an **Elaborate and Detailed Section** describing the **Orchestration**, **Workflow Scheduling**, **Test Result Structure**, **Validation Process**, and the **Dashboard Functionality & Usability** for your **Guidewire Data Migration Automation Framework**.

## ****Section 18: Orchestration, Workflow Scheduling, Test Results & Dashboard Usability****

### **18.1 Orchestration of Automation Workflows**

The automation framework is orchestrated using **Databricks Workflows**, which serve as the scheduling and execution engine for the data validation notebooks. Each workflow is a sequence of tasks representing one or more validation steps for a particular source system (Oracle, Read Replica, or CDA).

#### 🔁 **Workflow Structure in Test Environment:**

Each workflow typically includes the following tasks in sequence:

| **Task Name** | **Notebook** | **Purpose** |
| --- | --- | --- |
| DDL Comparison | DDL\_COMPARISION\_ORA\_ODS\_S3.ipynb | Compares Oracle vs S3/Read Replica schema metadata |
| Schema Validation | ORA\_READREPLICA\_SCHEMA\_CHECK.ipynb | Validates that source schema matches ODS structure |
| Data Validation | ODS\_BRONZE\_VALIDATION\_ESSENTIAL.ipynb | Performs count, field-level, SCD, duplicate, referential checks |
| DLT Pipeline Refresh | DLT\_REFRESH\_PIPELINE.ipynb | Stores validation output into Delta Live Tables |
| Notification (Optional) | Email trigger (future feature) | Alerts teams on success/failure |

### **18.2 Workflow Scheduling Strategy**

| **Environment** | **Trigger Type** | **Frequency** | **Purpose** |
| --- | --- | --- | --- |
| **Dev** | Manual via Release Notebook | On code update | CI/CD validation |
| **Test** | Scheduled via Databricks Jobs UI | Daily or post-ingestion | Full validation of delta loads |
| **Pre-Prod** | Manual or Scheduled | Before go-live cutover | Sanity check on production-like data |
| **Prod** | Scheduled (optional post-deployment) | One-time per release | Confirm final load and outbound generation |

**Scheduling Tools:**

* **Databricks Jobs UI** with parameterized runs
* Can be integrated with Azure DevOps or triggered via REST API for CI/CD extension

### **18.3 Test Result Table (DLT Table) Structure**

Validation outcomes are stored in **Delta Live Tables (DLT)** for traceability, analysis, and reporting. Each validation type (Schema, Count, Data, Duplicate, SCD, Referential) contributes to the result tables.

| **Field Name** | **Description** |
| --- | --- |
| run\_id | Unique identifier per validation execution |
| source\_table | Name of the table being validated |
| validation\_type | Type of test (e.g., SCHEMA, COUNT, DATA, DUPLICATE) |
| status | PASS / FAIL |
| expected\_value | Expected result (e.g., row count, column name) |
| actual\_value | Actual result returned from query |
| error\_message | Detail of mismatch or failure |
| timestamp | Validation run date-time |
| execution\_time | Duration of validation logic in seconds |

### **18.4 How Validation is Performed**

Each notebook handles validation using a defined, modular process:

1. **Parameter Input**: Table name, source, validation type.
2. **Metadata Extraction**: Uses schema snapshots or live JDBC connections.
3. **Validation Logic**: Based on Spark SQL / PySpark logic:
   * Count comparisons
   * Field-level joins
   * Hash sum matching
   * Referential and null check joins
   * SCD logic via effective\_date or hash diff detection
4. **Error Logging**: Failures logged with full context in Delta tables.
5. **Result Storage**: All results appended to DLT table with run metadata.

### **18.5 Dashboard Functionality & Usability**

The dashboards are built directly on top of **DLT Views**, exposed inside Databricks. They are designed for **QA engineers, test leads, project managers**, and **business users** to easily interpret results without SQL or notebook access.

#### 📊 **Dashboard Features**

| **Feature** | **Functionality** |
| --- | --- |
| **Real-Time Status View** | Shows current pass/fail count by validation type |
| **Filter by Validation Type** | Schema / Count / Data / Duplicate / Referential Integrity |
| **Drill-Down per Table** | View failure reason and compare expected vs actual |
| **Execution Metrics** | Runtime per notebook, validation type, per table |
| **Date Filter** | Run-date filter for daily or specific period validation |
| **Status Summary KPIs** | % passed, failed, skipped, and total validations |
| **Downloadable Reports** | CSV export for JIRA manual update or audit logs |

#### 🎯 **Usability Highlights**

* **No technical knowledge required**: QA and business users can use intuitive filters and visuals.
* **Interactive**: Clickable elements for deep dives into validation mismatches.
* **Connected to live Delta tables**: Always current with no lag.
* **Auto-refresh (optional)**: Scheduled refresh supports near-real-time reporting.

### ✅ Summary Benefits

| **Benefit** | **Value** |
| --- | --- |
| **Modular Workflow** | Easy to extend for new ingestion sources or validations |
| **Clear Orchestration** | Every job is tracked via Workflows and Databricks Jobs |
| **Transparent Reporting** | QA and stakeholders see exactly what passed/failed and why |
| **Scalable Architecture** | Supports large volumes and many tables |
| **Audit-Ready** | Delta Live Tables store full logs, re-runnable anytime |