## ****Section 15: Dependency Matrix****

This matrix outlines the key **technical and procedural dependencies** required to execute the automation test framework successfully. It includes ownership, impact levels, readiness status, and mitigation plans for each dependency.

### 🔗 **Dependency Matrix**

| **Dependency ID** | **Dependency Description** | **Owner** | **Impact if Not Met** | **Readiness Status** | **Mitigation Strategy** |
| --- | --- | --- | --- | --- | --- |
| **D-01** | **Read Replica solution developed and unit tested** | Data Engineering | High – Cannot validate interim ingestion path | ✅ Completed | Validate again post-integration; proceed to execute Read Replica workflow first |
| **D-02** | **CDA ingestion solution developed and unit tested** | Data Engineering | High – Cannot begin parallel validation or switch ingestion source | 🟡 In Progress | Proceed with Read Replica testing; CDA to be validated as a secondary cycle |
| **D-03** | **Test Environment setup and accessible (clusters, notebooks, data)** | Infra / DevOps / Databricks Admin | High – Blocks all automation execution | ✅ Completed | Perform pre-checks before execution; run sample workflows as readiness test |
| **D-04** | **Source environment (STD001) is stable with no showstopper defects** | Guidewire Application Team | Medium – May cause inconsistent or incorrect test results | 🟠 Limited Stability | Maintain known defect list; exclude impacted tables from scope temporarily |
| **D-05** | **Test Data created and injected via Guidewire UI or backend** | Manual QA / UI Automation | Medium – Sample data needed for specific business cases | 🟡 In Progress | Coordinate with QA team to confirm availability before each cycle |
| **D-06** | **Staging layer for Oracle, Read Replica, and CDA available with 7-day snapshots** | Data Engineering | High – Essential for 3-way reconciliation and regression validation | ✅ Completed (Test only) | Refresh daily before test execution; confirm version control is functioning |
| **D-07** | **IAM + VPN access from Databricks to AWS S3** (for CDA Parquet reads) | Cloud Ops / Infra | High – Blocks CDA validation and execution | 🟡 In Progress | Validate VPN tunnel in lower environments; escalate delays immediately |
| **D-08** | **CI/CD deployment pipeline tested and working via Azure DevOps** | DevOps / Automation Lead | Medium – Delays notebook and script deployment | ✅ Completed | Manual fallback deployment enabled if required |

### ✅ **Legend:**

* ✅ Completed and confirmed
* 🟡 In Progress or Partial
* 🟠 Known risks / Limited stability

## ****Section 15: Detailed Dependencies Matrix****

| **Dependency ID** | **Dependency Description** | **Impact if Blocked** | **Primary Owner** | **Mitigation Strategy 1** | **Mitigation Strategy 2** |
| --- | --- | --- | --- | --- | --- |
| **D-01** | ✅ Read Replica solution fully developed and unit tested | High – blocks ingestion and delta validation for interim phase | Data Engineering / ETL Team | Use Oracle source staging for sample comparison to start schema validation | Validate against subset of tables already migrated to Bronze layer |
| **D-02** | ✅ CDA solution developed and unit tested (S3 availability, schema alignment) | High – blocks final ingestion validation and 3-way comparison | Data Engineering / Cloud Ops | Proceed with Read Replica-only test cycle and flag CDA as deferred | Simulate CDA structure in test environment using mock Parquet files |
| **D-03** | ✅ Test environment setup (Databricks workspace, cluster, VPN, S3 access) | High – delays full test cycle execution | Cloud Infra / QA | Use DEV environment to dry-run scripts and validate workflows | Split testing into isolated table groups once TEST is ready |
| **D-04** | ⚠️ Source environments (Read Replica / CDA / Oracle) must be stable and STD001-free | High – data inconsistency, unstable test results | Data Engineering / Application SMEs | Log data quality issues via JIRA and skip impacted tables | Align test execution with clean load window after source patching |
| **D-05** | ⚠️ Test data creation through Guidewire UI and manual teams | Medium – impacts field-level transformation and SCD testing | QA Manual Team / UI Automation | Use reusable test cases aligned with real policy and claims data | Use old Oracle + Gold snapshot as synthetic input to replicate transformation scenarios |

## ****Section 16: Automation Code Performance Check, Optimization Setup & Performance Plan****

This section outlines the approach to monitor, optimize, and report the performance of automation code and its execution during data validation. Although full-scale performance testing is out of scope, a **high-level performance benchmarking** and **runtime optimization** are integral to ensure timely, reliable automation.

### **16.1 Performance Objectives**

* Ensure that validation scripts run efficiently across large datasets (especially during Bronze-layer delta validation).
* Benchmark **execution time** for major test types (schema validation, count checks, data accuracy).
* Compare ADF ingestion runtime vs. Databricks ingestion runtime for equivalent loads.
* Ensure **pipeline resource optimization** through cluster sizing, memory management, and execution scheduling.

### **16.2 Performance Monitoring Strategy**

| **Performance Aspect** | **Monitoring Method** | **Tool/Metric** |
| --- | --- | --- |
| Validation script runtime | Notebook cell execution time | %time magic command, Spark logs |
| Cluster resource usage | Auto-scaling + memory consumption | Databricks Spark UI |
| Ingestion timing (ADF vs. Databricks) | Capture start-end timestamps | Workflow logs, executionTime column |
| Delta table write time | Transaction time and file stats | Spark logs, file metadata |
| Parallel workflow execution efficiency | Compare single vs. multi-table notebook runs | Time per test cycle |
| Bottlenecks or skewed joins | Physical plan monitoring | explain(), Spark UI DAG |

### **16.3 Optimization Setup**

| **Optimization Area** | **Best Practices Implemented** |
| --- | --- |
| **Data Partitioning** | Read CDA/Replica tables partitioned by date, lineOfBusiness, or centerName for efficient reads. |
| **Broadcast Joins** | Use Spark hints for smaller reference tables to avoid shuffle joins. |
| **Caching Intermediate DataFrames** | Use .cache() for reference datasets used across validations. |
| **Cluster Configuration** | Adjust node size & autoscaling in Test and Pre-Prod environments. |
| **Column Pruning** | Explicitly select only required columns for each test case. |
| **Coalesce & Repartition** | Used during writes to reduce file fragmentation and memory overhead. |
| **Workflow Parallelization** | Distribute validation across ingestion types and table groups in parallel Databricks Workflows. |

### **16.4 Performance Benchmarking Plan**

| **Metric** | **Measured Between** | **Goal** |
| --- | --- | --- |
| Validation Runtime | Script start vs. end | ≤ 10 minutes per table for large datasets |
| ADF vs Databricks Ingestion | Start/End timestamps in pipeline logs | ≤ 10–15% variance for same load size |
| File Write Efficiency | Delta table transaction times | ≤ 2 mins per target |
| Spark Task Distribution | Spark UI (shuffle vs. cached) | Balanced task execution, avoid skew |
| Cluster Utilization | CPU/Memory usage | Peak usage < 80%, autoscaling enabled |

### **16.5 Performance Validation Deliverables**

| **Deliverable** | **Description** |
| --- | --- |
| Runtime Log Snapshot | Per-table execution duration, ingestion timing |
| Performance Summary View | Databricks Dashboard showing avg/max/min run times |
| Cluster Usage Snapshot | Spark UI logs captured as HTML exports for review |
| Ingestion Comparison Report | Time comparison of ADF vs Databricks per layer |
| Optimization Notes | Suggestions and applied strategies per table group |

### **16.6 Improvement Loop**

Performance issues or bottlenecks discovered during validation cycles will be logged and tagged as **optimization tasks**, with the following process:

1. Identify: Runtime exceeds threshold or Spark DAG indicates skew.
2. Analyze: Review notebook logic, cluster logs, and data volume.
3. Fix: Implement partitioning, caching, or join strategy adjustments.
4. Validate: Re-run for comparison and impact measurement.
5. Document: Record changes and effect in the performance tracker.