Contents

[**Streaming ETL Pipelines** 3](#_Toc207884851)

[JPMC REQUIRMENTS 6](#_Toc207884852)

[Build secrets rotation process for each of the extraction services 6](#_Toc207884853)

[**Example (AWS Secrets Manager)** 6](#_Toc207884854)

[Build extraction services /process to extract data from ~30 Data sources with 7](#_Toc207884855)

[over 750 Datasets 7](#_Toc207884856)

[• Extraction services to be built in 6 data patterns with a separate instance for each of the datasets 7](#_Toc207884857)

[**Example** 8](#_Toc207884858)

[Extraction services to be True CD & NFR Compliant 8](#_Toc207884859)

[For True CD Compliance 8](#_Toc207884860)

[For NFR Compliance 9](#_Toc207884861)

[Provision AWS Infrastructure using EAC 9](#_Toc207884862)

[Build Data Publish service for modernization 9](#_Toc207884863)

[Typical Features of a Modern Data Publish Service 9](#_Toc207884864)

[Example 10](#_Toc207884865)

[Build data model for 750 datasets and create documents for DRP approvals 10](#_Toc207884866)

["Create documents for DRP approvals" 10](#_Toc207884867)

[Example DRP Document Contents 11](#_Toc207884868)

[Build 1:1 DPL pipelines for 525 BAU (70% of 750) datasets to ingest data from Raw to Trusted and utilize shuttle service provision to Snowflake 11](#_Toc207884869)

[Build 1:1 DPL pipelines for 225 BAU (30% of 750) datasets to ingest data from Raw to Trusted and build transformation from Trusted to Refined and utilize shuttle service to provision data to Snowflake 11](#_Toc207884870)

[Summary Table 11](#_Toc207884871)

[Onboard seal to shuttle process so that same can be utilized to provision data to Snowflake 12](#_Toc207884872)

[Leverage Flakezaap(or similar JPMC tool)/CCB approved utility to perform Data set and pipeline registration, glue table creation 12](#_Toc207884873)

[What does this actually involve? 12](#_Toc207884874)

[Configure CI/CD pipeline and deploy codebase into higher environments leveraging existing processes 13](#_Toc207884875)

[Example Workflow 13](#_Toc207884876)

[Integrate dashboards for monitoring batch processes by configuring existing services like Dynatrace and CloudWatch 14](#_Toc207884877)

[Summary Table 14](#_Toc207884878)

[Schedule data pipeline execution based on available JPMC orchestration tool 14](#_Toc207884879)

[Matching datasets being extracted from each of the data sources as similar to legacy system. 15](#_Toc207884880)

[Data validation for the input dataset/files and target entities/tables in Production 15](#_Toc207884881)

[Functional Testing for data extraction. 16](#_Toc207884882)

[Perform SIT and UAT validation 17](#_Toc207884883)

[Performance testing is to be performed for key business critical API's/Batch jobs (limited to 20) identified during the requirements gathering phase. 18](#_Toc207884884)

[What Performance Testing Typically Involves for APIs 18](#_Toc207884885)

[Support UAT Execution, defect management and resolution for UAT Validation and sign off 18](#_Toc207884886)

[Summary Table 18](#_Toc207884887)

[Performance benchmarking will be done for BAU data pipeline 19](#_Toc207884888)

[Leverage Fermi tool for data validation between refined datasets and snowflake tables 19](#_Toc207884889)

[What This Involves 19](#_Toc207884890)

[Typical Steps: 20](#_Toc207884891)

[Why It’s Important 20](#_Toc207884892)

[Preparation of runbook and NFR documents 21](#_Toc207884893)

[Runbook 21](#_Toc207884894)

[NFR (Non-Functional Requirements) Document 21](#_Toc207884895)

[In summary: 22](#_Toc207884896)

[Manage all SIT, UAT and PROD Codebase Migration activities 22](#_Toc207884897)

**Streaming ETL Pipelines**

Kafka serves as the backbone for Extract, Transform, Load (ETL) operations, where data is continuously ingested, transformed, and loaded into data lakes or analytics systems.

System design for Streaming ETL  
-------------------------------  
Here’s a system design for Streaming ETL (Extract, Transform, Load)   
Pipelines using Apache Kafka for scalable, real-time data ingestion, transformation, and loading into analytics systems.  
  
1. High-Level Architecture  
---------------------------  
Code  
[Data Sources]  
 |  
 (DBs, APIs, Logs, IoT, Files)  
 |  
[Extraction Agents]  
 |  
(Kafka Producer or Kafka Connect Source)  
 |  
[Kafka Cluster]  
 |  
[Kafka Topics: raw-data, transformed-data, error-events]  
 |  
[Stream Processing]  
 |  
[Transformation Logic]  
 |  
(Kafka Producer or Kafka Connect Sink)  
 |  
[Kafka Topics: cleaned-data, enriched-data]  
 |  
[Loaders/Connectors]  
 |  
[Target Systems]  
 (Data Lake, Warehouse, Search, Dashboard)  
  
2. Components Breakdown  
------------------------  
  
A. Extraction  
  
Data Sources: Databases (CDC via Debezium), APIs, log files, IoT devices, cloud buckets, etc.  
Extraction Agents:  
Kafka Producers or Kafka Connect Source connectors ingest data into Kafka (raw-data topic).  
B. Kafka Cluster  
  
Central backbone for event transport.  
Topics:  
raw-data: Unprocessed ingested events.  
transformed-data: Cleaned, filtered, or enriched events.  
error-events: Events that failed transformation/validation.  
  
C. Stream Processing  
Tools: Kafka Streams, Apache Flink, Spark Streaming.  
Transformation Logic:  
Cleansing (remove duplicates, fix formats).  
Enrichment (add geo, user, or reference data).  
Filtering (drop irrelevant events).  
Validation (schema checks).  
  
D. Loading  
Kafka Connect Sink connectors or custom Kafka Consumers push processed data to target systems:  
Data Lakes: S3, HDFS, GCS.  
Data Warehouses: BigQuery, Snowflake, Redshift.  
Search/Analytics: Elasticsearch, Druid.  
Dashboards: Real-time visualization.  
  
E. Error Handling  
Invalid or failed events sent to error-events topic for audit, alerting, or reprocessing.  
  
3. Key Technologies  
--------------------  
Kafka: Event backbone.  
Kafka Connect: Source and sink connectors for integration.  
Stream Processing: Kafka Streams, Flink, Spark Streaming.  
Schema Registry: Validate and evolve event schemas.  
Target Systems: S3, BigQuery, Snowflake, Elasticsearch, etc.  
Monitoring: Prometheus, Grafana, ELK stack.  
  
4. Scalability & Reliability  
-------------------------------  
Topic partitioning for throughput.  
Replication for durability.  
Consumer groups for parallel processing and load balancing.  
Idempotent processing for safe event handling.  
Error handling: Dead-letter topics for failed events.  
  
5. Sample Streaming ETL Event Schema (JSON)  
-------------------------------------------  
JSON  
{  
 "event\_id": "evt-12345",  
 "source": "orders\_db",  
 "timestamp": "2025-09-01T15:06:49Z",  
 "payload": {  
 "order\_id": "ORD-98765",  
 "user\_id": "USR-12345",  
 "total": 150.00,  
 "status": "completed"  
 },  
 "metadata": {  
 "extracted\_by": "debezium",  
 "schema\_version": "v2"  
 }  
}  
  
6. Text-Based Diagram  
---------------------  
Code  
[DB/API/Files/IoT] → [Kafka Producer/Connect Source] → [Kafka Topic: raw-data]  
 ↓  
 [Stream Processor]  
 ↓  
 [Kafka Topic: transformed-data]  
 ↓  
 [Kafka Connect Sink/Loader]  
 ↓  
 [Data Lake/Warehouse/Search]  
  
7. Extensions  
--------------  
Real-time dashboards: For data monitoring and analytics.  
Backpressure handling: Monitor lag and scale consumers as needed.  
Schema evolution: Handle changes in source schemas gracefully.  
Reprocessing: Replay events for correction or migration.  
Security: Encrypt sensitive data, control access, audit logs.

# JPMC REQUIRMENTS

# Build secrets rotation process for each of the extraction services

**Example (AWS Secrets Manager)**

1. Extraction service uses database password stored in Secrets Manager.
2. AWS rotates the password according to schedule.
3. Service reads password from Secrets Manager at startup or before each connection.
4. Monitoring alerts if extraction fails due to authentication errors.

**Summary:**  
Building a secrets rotation process means automating the secure and regular update of sensitive credentials for every extraction service, ensuring security and minimizing manual intervention or risk.

# Build extraction services /process to extract data from ~30 Data sources with

# over 750 Datasets

# • Extraction services to be built in 6 data patterns with a separate instance for each of the datasets

* **6 Data Patterns:**
  1. SQL Table Extraction
  2. CSV File Extraction
  3. REST API Extraction
  4. Kafka Stream Extraction
  5. XML File Extraction
  6. NoSQL Document Extraction
* **Datasets:**
  1. 10 SQL tables
  2. 5 CSV files
  3. 20 API endpoints
  4. 5 Kafka topics
  5. 3 XML files
  6. 7 NoSQL collections  
     (Total: 50 datasets)

Add these > then we get total Sources

**Example**

Suppose you have:

* 5 databases (each with 100 tables = 500 datasets)
* 10 APIs (each with 10 endpoints = 100 datasets)
* 7 file shares (each with 20 files = 140 datasets)
* 8 cloud data buckets (each with 2 folders = 16 datasets)

**Total datasets:** 500 + 100 + 140 + 16 = 756 datasets  
**Total sources:** 30

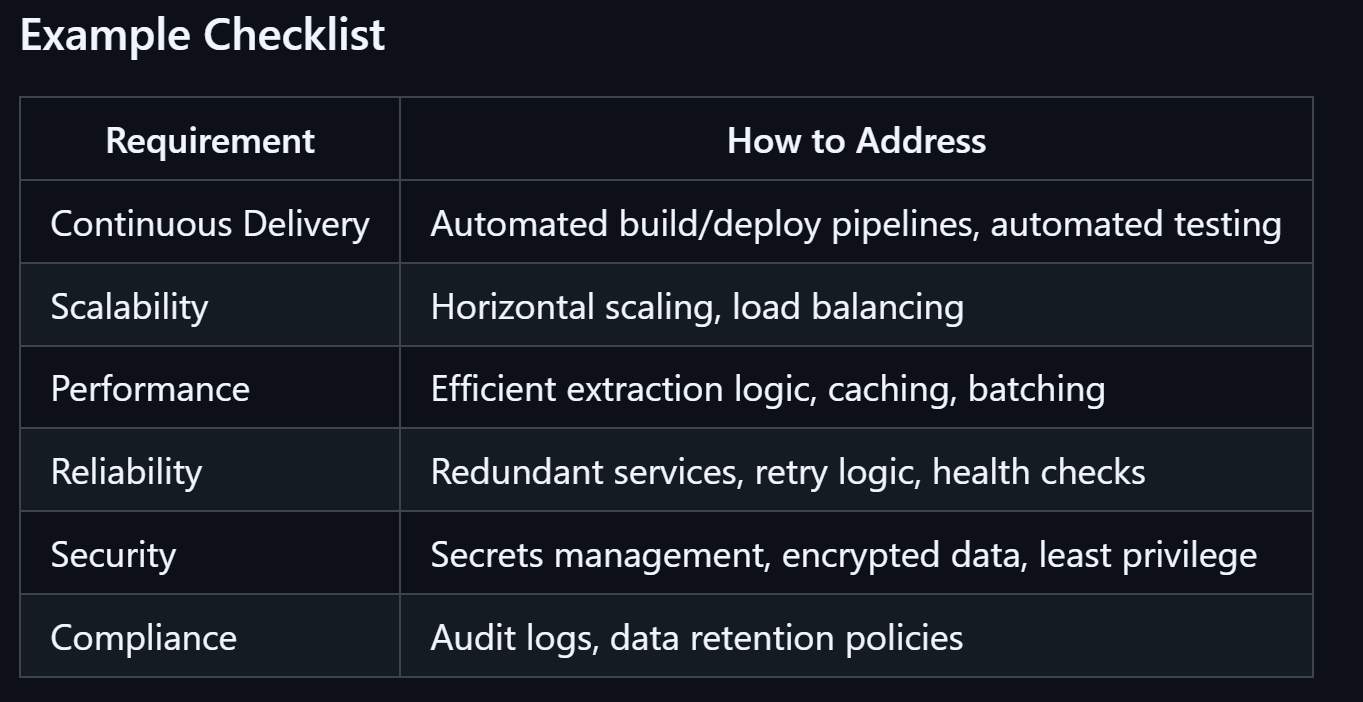
# Extraction services to be True CD & NFR Compliant

### For True CD Compliance

* CI/CD pipelines (e.g., Jenkins, GitHub Actions)
* Automated tests on every code change
* Automated deployment to all environments
* Monitoring and alerting on deployments

### For NFR Compliance

* Sizing and scaling strategies
* SLAs for latency, uptime, and error rates
* Security best practices (encryption, secrets rotation)
* Logging, auditing, and monitoring
* Documentation for operations and support



# Provision AWS Infrastructure using EAC

Terraform, CloudFormation needed – scripts to automatic create AWS resources and not through clicks

# Build Data Publish service for modernization

## Typical Features of a Modern Data Publish Service

* **Automated data delivery:** Pushes data to data lake, warehouse, or API endpoints.
* **Schema validation and transformation:** Ensures published data is clean and standardized.
* **Access controls:** Manages who can view/download/use published data.
* **Metadata tagging:** Publishes data with usage, provenance, and lineage info.
* **Notifications:** Alerts consumers when new/updated data is available.
* **Versioning:** Allows consumers to access previous versions if needed.
* **Monitoring:** Tracks publish job status, errors, and usage metrics.

## Example

Suppose you have:

* Legacy ETL system that generates daily sales data.
* Modernization efforts move data to AWS S3, Snowflake, or Databricks.
* **Data Publish service** automatically pushes cleaned sales data from staging to S3 bucket, registers it in a data catalog, and notifies BI teams.

# Build data model for 750 datasets and create documents for DRP approvals

### **"Create documents for DRP approvals"**

* **DRP (Data Readiness/Review/Remediation/Refinement Process) Documents:**  
  Prepare formal documentation for each dataset that describes its structure, content, source, quality, and compliance aspects, as required for DRP approvals.
* **Purpose:**  
  To submit these documents for review and sign-off by data governance, compliance, or quality teams before datasets are onboarded or made available for use.

## Example DRP Document Contents

* Dataset name and source
* Data model/schema (table/file structure, fields, types, keys)
* Data lineage (where data comes from, how it’s processed)
* Data quality assessment (accuracy, completeness, timeliness)
* Compliance and privacy checks
* Business owner and usage description
* Approval signatures or workflow status

# Build 1:1 DPL pipelines for 525 BAU (70% of 750) datasets to ingest data from Raw to Trusted and utilize shuttle service provision to Snowflake

# Build 1:1 DPL pipelines for 225 BAU (30% of 750) datasets to ingest data from Raw to Trusted and build transformation from Trusted to Refined and utilize shuttle service to provision data to Snowflake

## Summary Table

| **Term** | **Meaning** |
| --- | --- |
| 1:1 DPL pipelines | One pipeline per dataset for data ingestion |
| 525 BAU datasets | 525 datasets considered "Business As Usual" |
| Raw to Trusted | Data flow from raw zone to trusted/curated zone |
| Shuttle service to Snowflake | Automated transfer mechanism into Snowflake |

* **S3** is like a giant, secure, scalable file system for storing data.
* **Snowflake** is a powerful database platform for querying, analyzing, and transforming data at scale.

# Onboard seal to shuttle process so that same can be utilized to provision data to Snowflake

# Leverage Flakezaap(or similar JPMC tool)/CCB approved utility to perform Data set and pipeline registration, glue table creation

This refers to a specialized, internal tool at J.P. Morgan (JPMC) designed to manage data assets and integration with Snowflake, AWS Glue, etc.

* This makes the data queryable via Glue, Athena, or other analytics services.

## What does this actually involve?

1. **Use the approved utility (Flakezaap or similar):**
   * Access the tool and follow its workflow for onboarding data assets.
2. **Register datasets and pipelines:**
   * Provide all required metadata (name, description, source, owners, lineage, etc.).
   * Register both the dataset and the processing pipeline that moves/transforms the data.
3. **Create Glue tables:**
   * Use the tool’s integrated functionality (or automation) to create corresponding AWS Glue tables for each dataset.
   * This typically involves specifying the schema, location (S3 path), and other table properties.

# Configure CI/CD pipeline and deploy codebase into higher environments leveraging existing processes

## Example Workflow

1. **Developer pushes code to GitHub.**
2. **CI/CD pipeline runs:**
   * Build
   * Test
   * Package
   * Deploy to Dev
3. **Upon approval, pipeline deploys to QA, then Staging, then Production, using established deployment scripts and tools.**

# Integrate dashboards for monitoring batch processes by configuring existing services like Dynatrace and CloudWatch

## Summary Table

| **Step** | **Description** |
| --- | --- |
| Integrate dashboards | Build visual monitoring panels for batch jobs |
| Use Dynatrace | Cloud monitoring platform for dashboards/alerts |
| Use CloudWatch | AWS monitoring service for logs/metrics/alarms |

# Schedule data pipeline execution based on available JPMC orchestration tool

1. **Schedule Data Pipeline Execution**
   * Set up your data pipelines (ETL/ELT/jobs) to run automatically at specific times, intervals, or based on triggers.
   * Scheduling can be for daily, hourly, weekly jobs or triggered by events (file arrival, upstream job completion, etc.).
2. **Based on Available JPMC Orchestration Tool**
   * Use the orchestration tool(s) approved and available within JPMorgan Chase (JPMC).
   * Common orchestration tools in large enterprises include:
     1. **Control-M** (very common at JPMC)
     2. **Apache Airflow**
     3. **AWS Step Functions**
     4. **Talend**
     5. **Informatica**
     6. Or any other JPMC-internal scheduling/orchestration framework.
   * The tool is responsible for job scheduling, dependency management, retries, monitoring, and alerting.

# Matching datasets being extracted from each of the data sources as similar to legacy system.

* **As similar to legacy system:**  
  The goal is to ensure that the extracted datasets are structured, named, and populated in a way that closely resembles those from the legacy (older) system.

# Data validation for the input dataset/files and target entities/tables in Production

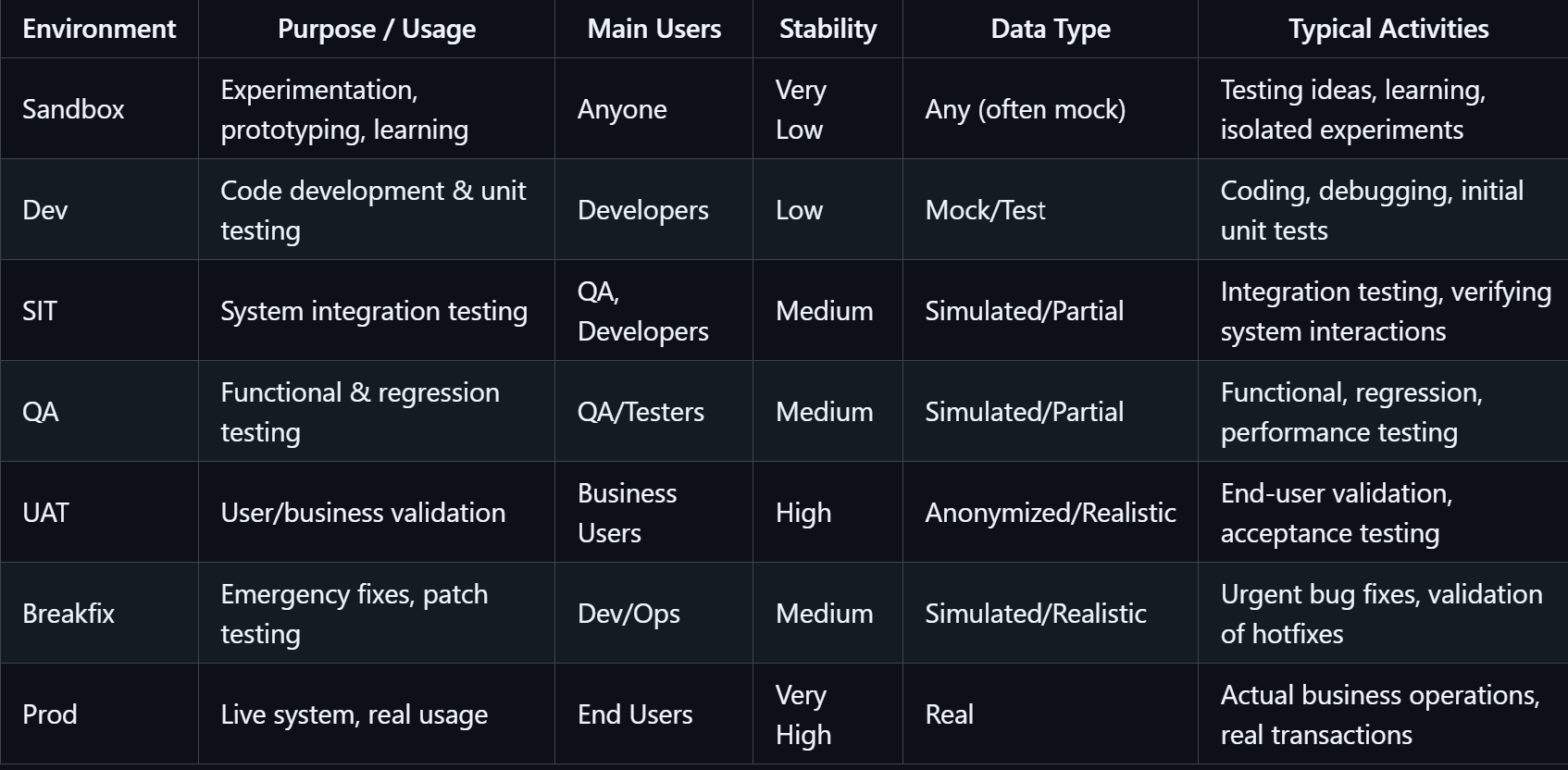
1. **Input Dataset/File Validation:**
   * **Schema Conformance:** Checking if the incoming data files conform to the expected schema (column names, data types, required fields, etc.).
   * **Value Range Checks:** Ensuring values fall within allowed ranges or categories.
   * **Null/Blank Validation:** Identifying missing or blank values in fields that should always be populated.
   * **Format Validation:** Confirming the format of dates, identifiers, codes, etc.
   * **Duplicate Detection:** Identifying duplicate records or keys.
2. **Target Entity/Table Validation:**
   * **Referential Integrity:** Ensuring relationships between tables (e.g., foreign keys) are maintained.
   * **Constraint Enforcement:** Making sure database constraints (unique, not null, check constraints) are not violated.
   * **Post-Load Validation:** Verifying that the data loaded matches source expectations (e.g., row counts, aggregates, checksums).
   * **Business Rules:** Applying business-specific rules (e.g., age must be >18, order amount >0).
3. **Process & Monitoring:**
   * **Automated Checks:** Use of scripts, validation frameworks, or ETL tools to automate validation.
   * **Alerting & Reporting:** Triggering alerts and logging issues when validation fails.
   * **Data Quality Metrics:** Tracking statistics like error rates, completeness, and timeliness.

# Functional Testing for data extraction.

**Functional Testing for data extraction** means verifying that the data extraction process works as intended and meets all specified business and technical requirements. It ensures that the system correctly extracts data from the source systems (databases, files, APIs, etc.) and delivers it in the expected format, structure, and completeness for downstream processes.

1. **Correctness of Extraction Logic**
   * Confirm that only the intended data is extracted (e.g., specific columns, rows, or entities).
   * Validate that filters, conditions, joins, and transformations are working as expected.
2. **Data Completeness**
   * Ensure all required data is extracted (no missing records).
   * Validate that data from all relevant sources is included.
3. **Data Accuracy**
   * Check that extracted data matches the source data exactly (no unintended changes or corruptions).
   * Verify that calculated fields or derived data are computed correctly.
4. **Format and Structure**
   * Confirm that the output data format (CSV, JSON, XML, etc.) meets specifications.
   * Validate schema: column names, data types, order, and structure.
5. **Boundary and Edge Cases**
   * Test extraction for empty data sets, very large data sets, or special characters.
6. **Error Handling**
   * Ensure that extraction fails gracefully and reports errors for unsupported scenarios (e.g., missing source, permission issues).
7. **Incremental and Full Loads**
   * Validate both full data extractions and incremental loads (e.g., only new or changed data since last extraction).

# Perform SIT and UAT validation



# Performance testing is to be performed for key business critical API's/Batch jobs (limited to 20) identified during the requirements gathering phase.

### What Performance Testing Typically Involves for APIs

* **Load Testing:** Simulating expected user traffic to see how the API behaves.
* **Stress Testing:** Pushing the API beyond normal limits to see how it handles overload.
* **Spike Testing:** Sending sudden increases in traffic to test stability.
* **Endurance Testing:** Running traffic over an extended period to ensure consistent performance.
* **Measurement:** Tracking metrics like response time, throughput, error rate, and resource usage.

# Support UAT Execution, defect management and resolution for UAT Validation and sign off

### **Summary Table**

| **Activity** | **What it Involves** |
| --- | --- |
| Support UAT Execution | Help with UAT setup, test execution, troubleshooting, and answering user queries |
| Defect Management and Resolution | Logging, tracking, fixing, retesting defects, and communicating status |
| UAT Validation and Sign Off | Confirming requirements are met, testing is complete, and formal approval from business is obtained |

# Performance benchmarking will be done for BAU data pipeline

# Leverage Fermi tool for data validation between refined datasets and snowflake tables

You should use the **Fermi tool** (a data validation or data comparison utility) to compare and verify the data between your **refined datasets** (typically, processed or cleaned data in files, data lakes, or intermediate storage) and the corresponding **Snowflake tables** (where the data is finally loaded in your data warehouse).

### What This Involves

* **Fermi Tool:**  
  A specialized software/tool designed to automate and simplify data validation tasks, such as comparing row counts, values, schemas, and identifying discrepancies between two data sources.
* **Refined Datasets:**  
  Data that has already been processed, cleansed, and transformed, and is ready for final loading or analysis.
* **Snowflake Tables:**  
  The destination tables in the Snowflake cloud data warehouse where refined data is stored for analytics and reporting.

### **Typical Steps:**

1. **Configure Fermi:**  
   Set up the tool to connect to both your refined dataset source (files, data lake, etc.) and Snowflake.
2. **Define Validation Rules:**  
   Specify what to compare (row counts, specific columns, aggregate values, schema consistency, etc.).
3. **Run Validation:**  
   Use Fermi to perform automated comparison of the datasets and Snowflake tables.
4. **Analyze Results:**  
   Review Fermi’s report to identify mismatches, missing records, or data quality issues.
5. **Resolve Issues:**  
   Investigate and fix any discrepancies found between source and target.

### **Why It’s Important**

* Ensures data loaded into Snowflake matches the refined source exactly.
* Detects and prevents data loss, corruption, or transformation errors.
* Supports compliance and audit requirements for data quality.
* Automates a typically manual and error-prone process.

**In summary:**  
You will use the Fermi tool to automatically and systematically validate that the refined data you’ve prepared matches what’s loaded into your Snowflake tables, helping guarantee data accuracy and integrity in your analytics environment.

# Preparation of runbook and NFR documents

**“Preparation of runbook and NFR documents”** means:

### **Runbook**

A **runbook** is a detailed, step-by-step guide that describes how to operate, maintain, troubleshoot, or recover a system or application. It is typically used by operations teams, support staff, or anyone responsible for running production systems.

**Contents of a Runbook may include:**

* Startup and shutdown procedures
* Deployment steps
* Incident response and troubleshooting instructions
* Recovery actions (e.g., rollback, restore from backup)
* Monitoring and alert handling
* Key contacts and escalation paths
* Common issues and solutions

**Purpose:**  
To ensure consistent, reliable operations and fast response to incidents or outages.

### **NFR (Non-Functional Requirements) Document**

An **NFR document** defines the criteria that judge the operation of a system, rather than its behaviors—the “how” rather than the “what.” These requirements describe qualities and constraints such as:

**Examples of NFRs:**

* Performance (response time, throughput)
* Scalability
* Availability and reliability
* Security
* Maintainability
* Usability
* Compliance and regulatory requirements
* Disaster recovery

**Purpose:**  
To ensure the system meets critical business and technical quality standards beyond just functional capabilities.

### **In summary:**

Preparation of runbook and NFR documents means creating:

* A **runbook** that provides operational guidance for system support, troubleshooting, and recovery.
* An **NFR document** that specifies the system’s expected quality attributes and constraints, ensuring it meets standards for performance, security, reliability, etc.

# Manage all SIT, UAT and PROD Codebase Migration activities

You are responsible for planning, coordinating, executing, and tracking the migration (movement, deployment, or upgrade) of codebases across the following environments:

* **SIT (System Integration Testing):**  
  The environment where integrated system components are tested together.
* **UAT (User Acceptance Testing):**  
  The environment where users and business stakeholders validate the solution against requirements.
* **PROD (Production):**  
  The live environment where the application is used by end users.

### What This Typically Involves

1. **Planning Migration:**
   * Create migration plans and schedules for moving code to SIT, UAT, and PROD.
   * Identify dependencies, risks, and rollback strategies.
2. **Coordinating Releases:**
   * Align with development, QA, business, and operations teams.
   * Communicate timelines, prerequisites, and responsibilities.
3. **Executing Migration:**
   * Deploy codebase updates to each environment as per plan.
   * Ensure pre-deployment and post-deployment checks are performed (sanity, smoke tests).
   * Monitor for issues during deployment.
4. **Tracking and Reporting:**
   * Document migration activities, issues, and resolutions.
   * Track status and progress for each environment.
   * Report completion, success, or failures to stakeholders.
5. **Troubleshooting and Support:**
   * Resolve issues that arise during or after migration.
   * Implement fixes or rollbacks if necessary.
6. **Compliance and Documentation:**
   * Ensure migration activities adhere to organizational and regulatory standards.
   * Maintain clear documentation for audit and future reference.

**In summary:**  
You oversee and ensure the successful movement and deployment of codebases from development through testing (SIT, UAT) to production (PROD), minimizing risks and ensuring smooth transitions between environments.

# Warranty support for 4 weeks that includes handover to Daily/PROD support and fix mismatches or provide justifications

* **Warranty Support (4 weeks):**  
  After the solution (system, application, data pipeline, etc.) goes live in production, the delivery/project team will provide active support for a period of 4 weeks. During this time, they will:
  + Monitor the system.
  + Address any issues or defects that arise.
  + Ensure the solution operates as expected.
* **Handover to Daily/PROD Support:**  
  At the end of the 4-week warranty period, the responsibility for ongoing support will be formally transferred to the regular operations or production support team. This team will handle daily incidents and routine maintenance going forward.
* **Fix Mismatches or Provide Justifications:**  
  During the warranty period, if any data mismatches, functional issues, or unexpected behaviors are identified:
  + The delivery/project team will either fix the issue (if it’s a valid defect).
  + Or, if the mismatch is expected or justifiable (e.g., due to business rules or data anomalies), they will provide a documented explanation to stakeholders.

# Any performance bottleneck caused by platform or

# existing limitations in DPL or source

# connections will be resolved by JPMC

**"Any performance bottleneck caused by platform or existing limitations in DPL or source connections will be resolved by JPMC" means:**

* **Performance bottlenecks** refer to parts of the system (platform, DPL, or source connections) that slow down data processing or cause delays.
* **Platform** could mean the underlying infrastructure, cloud services, or data platform used for processing.
* **DPL** likely stands for Data Pipeline Layer—the component responsible for moving, transforming, and loading data.
* **Source connections** are the links to the original data sources (databases, files, APIs, etc.).

**JPMC** (JPMorgan Chase) will take responsibility for:

* Identifying and fixing any issues that slow down data flow or processing, whether those issues are due to the platform, the data pipeline, or the way data is fetched from source systems.
* Addressing limitations such as network latency, hardware constraints, software bugs, configuration problems, or inefficient connection setups.

# Source to Target mapping would be shared by JPMC before start of engagement

JPMC (JPMorgan Chase) will provide a document or specification **before the project/engagement begins** that clearly defines:

* **Source:** Where the data is coming from (e.g., source systems, databases, files, APIs).
* **Target:** Where the data is going to (e.g., target tables, data warehouse, Snowflake, etc.).
* **Mapping:** How each field, column, or data element in the source corresponds to a field, column, or data element in the target, including any transformations or business rules.

### Why is this important?

* Ensures **clarity** for the development and data migration teams.
* Serves as a **blueprint** for building ETL (Extract, Transform, Load) processes or migrations.
* Helps avoid misunderstandings and errors by specifying exactly how data should flow and be transformed.

**In summary:**  
Before work begins, JPMC will provide a detailed mapping document showing how data from the source systems should be transferred, transformed, and loaded into the target systems, ensuring all stakeholders are aligned.

# PI / PG classification would be done by JPMC before start of engagement

**"PI / PG classification would be done by JPMC before start of engagement" means:**

* **PI** stands for **Personally Identifiable Information** (data that can identify an individual, like name, SSN, email).
* **PG** typically stands for **Protected/Proprietary/General** data classification (depending on JPMC's internal terminology; often, PG refers to less sensitive or general business data).

**JPMC (JPMorgan Chase) will:**

* Review and categorize all relevant data **before the project begins**.
* Clearly identify which data is **PI** (requires higher protection and specific handling) and which is **PG** (may have different handling rules).
* Share this classification with project teams to ensure appropriate security, privacy, and compliance measures are followed during the engagement.

**In summary:**  
JPMC will determine and communicate the sensitivity and handling requirements of all data (PI/PG) so everyone knows which data must be protected and how, before any work starts.

# **"C3 / C4 architecture diagram will be shared by JPMC team before start of engagement"**

### What Are C3 and C4 Architecture Diagrams?

* **C4 Model:**  
  A widely used technique for visualizing software architecture at different levels of detail.
  + **Level 1 (Context):** System’s environment and interactions.
  + **Level 2 (Container):** Major containers (applications, databases, services).
  + **Level 3 (Component):** Components within each container.
  + **Level 4 (Code):** Internal code structure (classes, etc.).
* **C3 Model:**  
  Not as standard as C4, but sometimes refers to a simplified diagram focusing on three levels (Context, Container, Component), omitting code-level details.

### What Does This Mean for the Project?

* JPMC will provide diagrams showing:
  + How the system fits into its environment (context).
  + What major applications and databases are involved (containers).
  + Key components and their interactions.
* These diagrams will be shared **before the engagement starts**, so everyone has a clear understanding of the system’s structure, relationships, and boundaries.

**In summary:**  
JPMC will give you architecture diagrams (based on C3/C4 modeling) before work starts, ensuring all stakeholders understand the system’s design and integration points from the outset.

# Any Snowflake infrastructure set up and configuration would be done by JPMC

* **Setting up the Snowflake environment:**  
  This includes creating and configuring Snowflake accounts, warehouses, databases, schemas, roles, and user access as needed for the project.
* **Performing all configuration tasks:**  
  JPMC will handle setup of storage, compute resources, network integration, security settings, and any other platform-specific configurations.
* **Ownership:**  
  You and your team will NOT be expected to perform or own any core Snowflake infrastructure or configuration activities—these will be managed by JPMC.

# JPMC has base infrastructure with ECS clusters

# available, and only AWS resources required

# for the application will need to be provisioned

# using EAC

**"JPMC has base infrastructure with ECS clusters available, and only AWS resources required for the application will need to be provisioned using EAC" means:**

* **JPMC (JPMorgan Chase) already has foundational infrastructure in place, including ECS clusters.**
  + **ECS (Elastic Container Service):** AWS service for running containerized applications.
  + These clusters are already set up and available for use.
* **Only the additional AWS resources specific to your application need to be provisioned.**
  + Examples: databases (RDS, DynamoDB), object storage (S3), networking (VPC, subnets, security groups), load balancers, IAM roles, etc.
* **Provisioning should be done using EAC.**
  + **EAC (Enterprise Application Cloud or similar internal provisioning tool):** A JPMC or AWS tool/process for requesting and creating cloud resources in a controlled, compliant manner.

# AWS Infrastructure Architecture and Design will

# be provided by JPMC

* **Creating and supplying the overall architecture and design documents** for the AWS cloud infrastructure needed for your project or application.
* This includes decisions and documentation on:
  + Which AWS services and resources to use (e.g., EC2, S3, RDS, VPC, IAM, ECS, etc.).
  + How these resources will be organized, networked, and secured.
  + High-level and detailed diagrams, design principles, standards, and best practices to follow.
  + Requirements for scalability, availability, security, and compliance.

**You and your team do NOT need to design the AWS infrastructure yourselves;**  
You will be given all necessary architecture and design specifications by JPMC before you build or deploy anything on AWS.

# Any design change in Infrastructure Architecture

# at later point of time will lead to additional

# effort & timeline

# DPL, Snowflake, AWS CCB Data lake onboarding

# is completed before start of the engagement

Before the project or engagement officially begins, the following onboarding and setup activities will already be finished:

* **DPL (Data Pipeline Layer) onboarding:**  
  The necessary data pipelines (tools, processes, access, and configurations) required for moving and processing data are already set up and ready for use.
* **Snowflake onboarding:**  
  The Snowflake data platform (accounts, roles, access, schemas, databases, etc.) is already provisioned and configured for the project.
* **AWS CCB Data lake onboarding:**  
  The AWS Cloud-based Data Lake (often for storing, managing, and analyzing large volumes of data) is already set up, with required permissions, buckets, and resources in place.

# Re-planning of effort & timeline is required in case

# of environmental delays and delay in providing

# required information, infrastructure, and

# necessary access for the required servers, any

# deviation from considered inventory and

# complexity

# JPMC will provide priorities of data extraction

# and ingestion for various sources to Data Lake

# Complexity of data extraction logic is considered

# 50% Complex and 50% Medium for all data

# sources

# All the datasets from sources would be in CSV,

# TSV, JSON, AVRO,TXT and ingestion should

# be supported by DPL

# Same data extraction Logic can be implemented

# across data sources for the same data pattern

# Data publish service for modernization will not

# have any data transformation logic

# We assume the refined data entities are 30% of 750

# datasets i.e. 225 for BAU loads

* **30% of 750 = 225**, so **225 refined datasets** will be used for **BAU (Business-As-Usual) loads**.
  + **BAU loads** refer to regular, ongoing data processing or loading activities in daily operations, not special projects or one-time migrations.

# We assume the complexity of transformation on

# refined datasets is 70:25:5 as Simple: Medium:

# Complex

* Out of all the refined datasets to be transformed:
  + **70%** of them will require **simple transformations** (basic modifications, minimal logic, straightforward mapping).
  + **25%** will require **medium transformations** (moderate logic, some data enrichment, potentially involving multiple steps or sources).
  + **5%** will require **complex transformations** (advanced logic, multiple joins, aggregations, business rules, or complicated data restructuring).

# Sample data validation would be done using Fermi tool (up to 4000 records for each dataset)

# Transformation from Trusted to Refined will not be having any joins

**In summary:**  
The process of moving data from the Trusted layer to the Refined layer will only involve single-table operations, and will **not** include joining data from different tables or datasets.

# All the transformations required for Trusted and Refined are supported by DPL

# Snowflake would be AS IS replica of Refined and Trusted data lake layer

# SnowFlake v/s S3

### **Snowflake**

* **Type:** Cloud Data Warehouse (Database platform)
* **Purpose:**
  + Stores, processes, and analyzes structured and semi-structured data
  + Optimized for analytics, SQL queries, and business intelligence
* **Features:**
  + Supports SQL-based querying and analytics
  + Automatic scaling of compute and storage
  + Built-in security and compliance
  + Handles structured, semi-structured data (JSON, Parquet, Avro, etc.)
  + Data sharing, time travel, cloning, and advanced features
* **Use Case:**
  + Data warehousing
  + Analytics and reporting
  + ETL/ELT pipelines
  + Data science workloads

### **Amazon S3 (Simple Storage Service)**

* **Type:** Object Storage Service
* **Purpose:**
  + Stores any type of files (documents, images, videos, backups, datasets)
  + Not a database; does not support SQL querying directly
* **Features:**
  + Massive scalability for storing objects (files)
  + Highly durable and available
  + Fine-grained access control
  + Integrates with AWS services (Lambda, EC2, Athena, etc.)
  + Supports versioning, event notifications, lifecycle management
* **Use Case:**
  + Data lake storage
  + Backup and archival
  + Storing raw data, logs, media, unstructured datasets

### **Key Differences**

* **Snowflake:**
  + Purpose-built for analytics and data warehousing
  + Supports SQL queries on data
  + Manages compute and storage for fast analysis
* **S3:**
  + General-purpose storage for files and objects
  + No native querying or analytics—data must be loaded into a database or analytics service (like Athena, Redshift, or Snowflake itself) for querying

# There will be no additional transformations between Data Lake and Snowflake

# Critical and most used test scenarios will be covered as part of performance testing to ensure benchmark is achieved

# ~30 Data Source extraction to be considered in scope of functional testing.

* Around **30 different data sources** (such as databases, files, APIs, or other systems from which data is extracted) are included within the scope of **functional testing** for your project.

# Only Onboarding of applications on JPMC observability tools is factored in efforts DPL Ingestion process is supporting file format across 6 different patterns

# JPMC to provide the attribute level transformation rules, description, PG Classification and mapping document for all the datasets

# JPMC to provide intake form required for data modelling

# All existing DPL packages are benchmarked for optimal performance to meet SLA defined and no further optimization is required

* The current **DPL (Data Pipeline Layer) packages** (software modules or components used in your data platform) are **fully capable** of performing every transformation needed for the **Semantic Pipelines** you intend to build.
* **No additional development, customization, or new packages are required** for transformation logic. The existing tools/modules already have the necessary features, functions, and capabilities.
* This ensures that the transformation requirements for Semantic Pipelines (such as data cleansing, enrichment, mapping, aggregation, etc.) are already addressed by the current DPL packages.

# Any pre-processing rules to enable raw files ingestion to be handled by JPMC Team

# Any changes to DPL code repository /new feature build to be owned by JPMC Team

# JPMC's Flakezapp(or similar JPMC tool) dataset registration utility supports all the file format which is in scope of this engagemen

# DRP process will be done for a minimum of group of 25 datasets/iteration

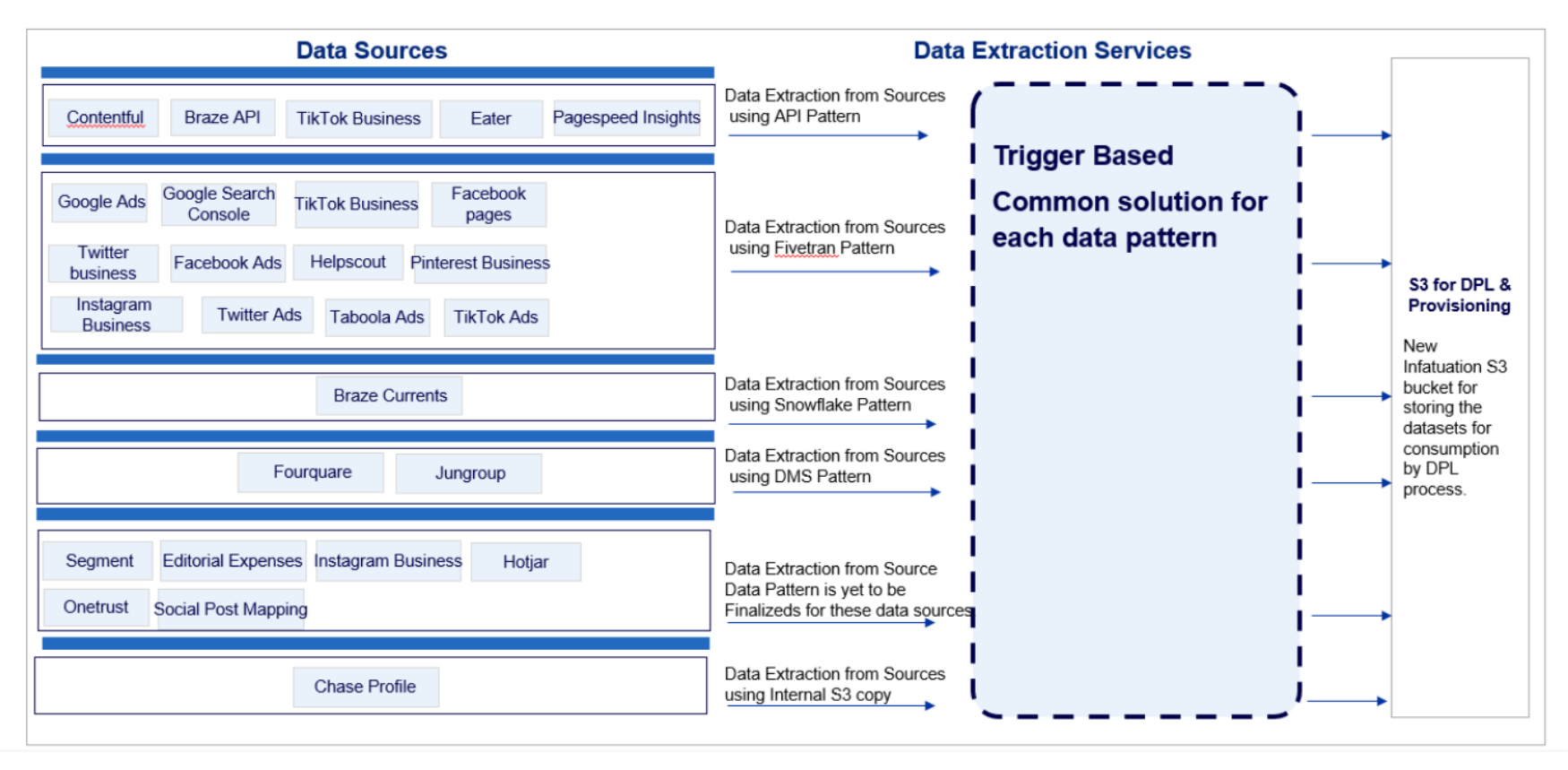
**"DRP process will be done for a minimum of group of 25 datasets/iteration" means:**

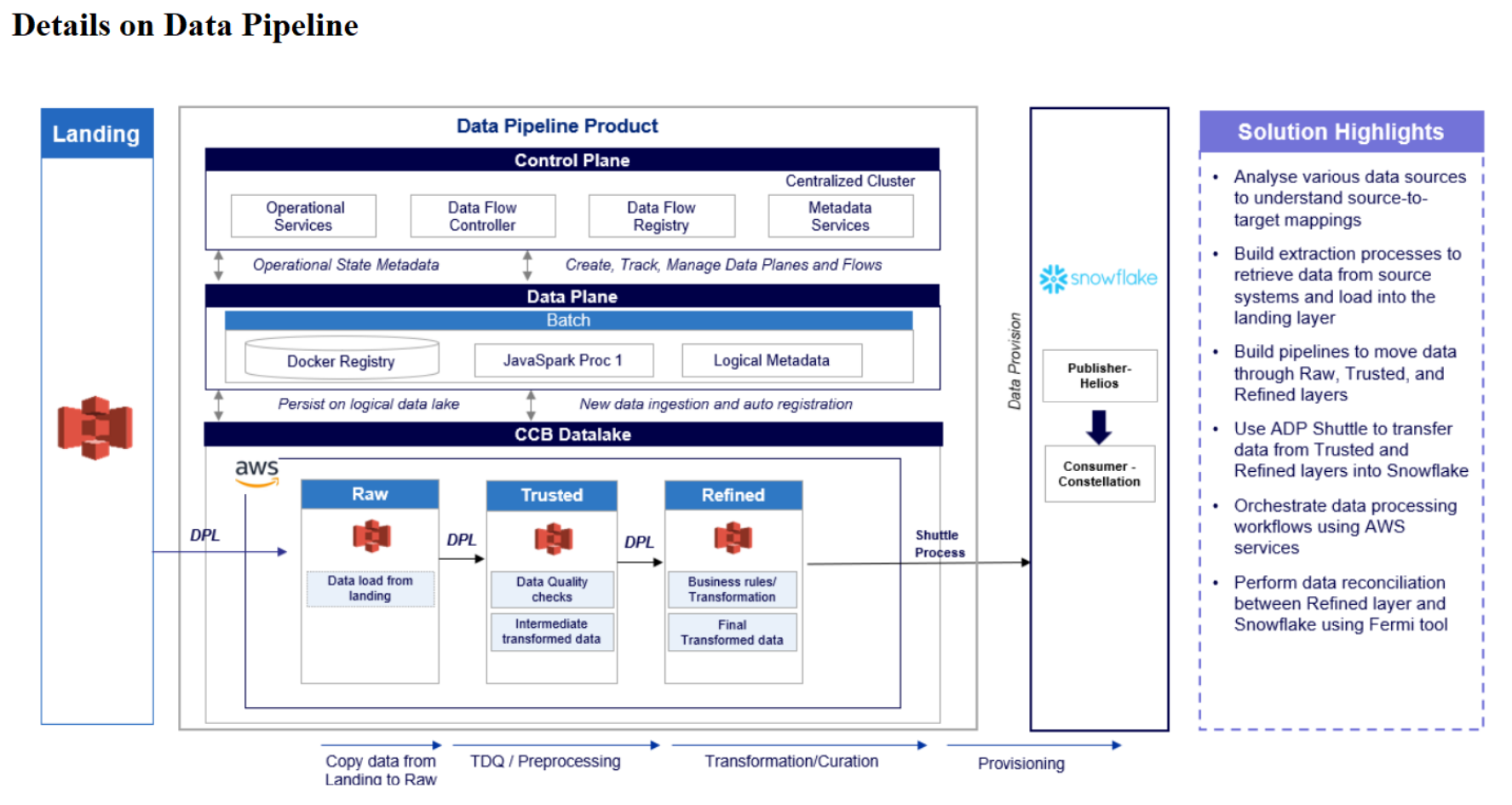
* The **DRP process** (likely referring to Data Readiness/Recovery/Replication/Review Process or similar) will be performed in **iterations** (rounds or cycles).
* In each iteration, a **minimum of 25 datasets** will be processed together as a group.
* The process will not be triggered for fewer than 25 datasets at a time.

JPMC recommended inhouse Datalens tool will be leveraged for data modelling for simple and medium datasets(which constitutes 70% of total datasets)

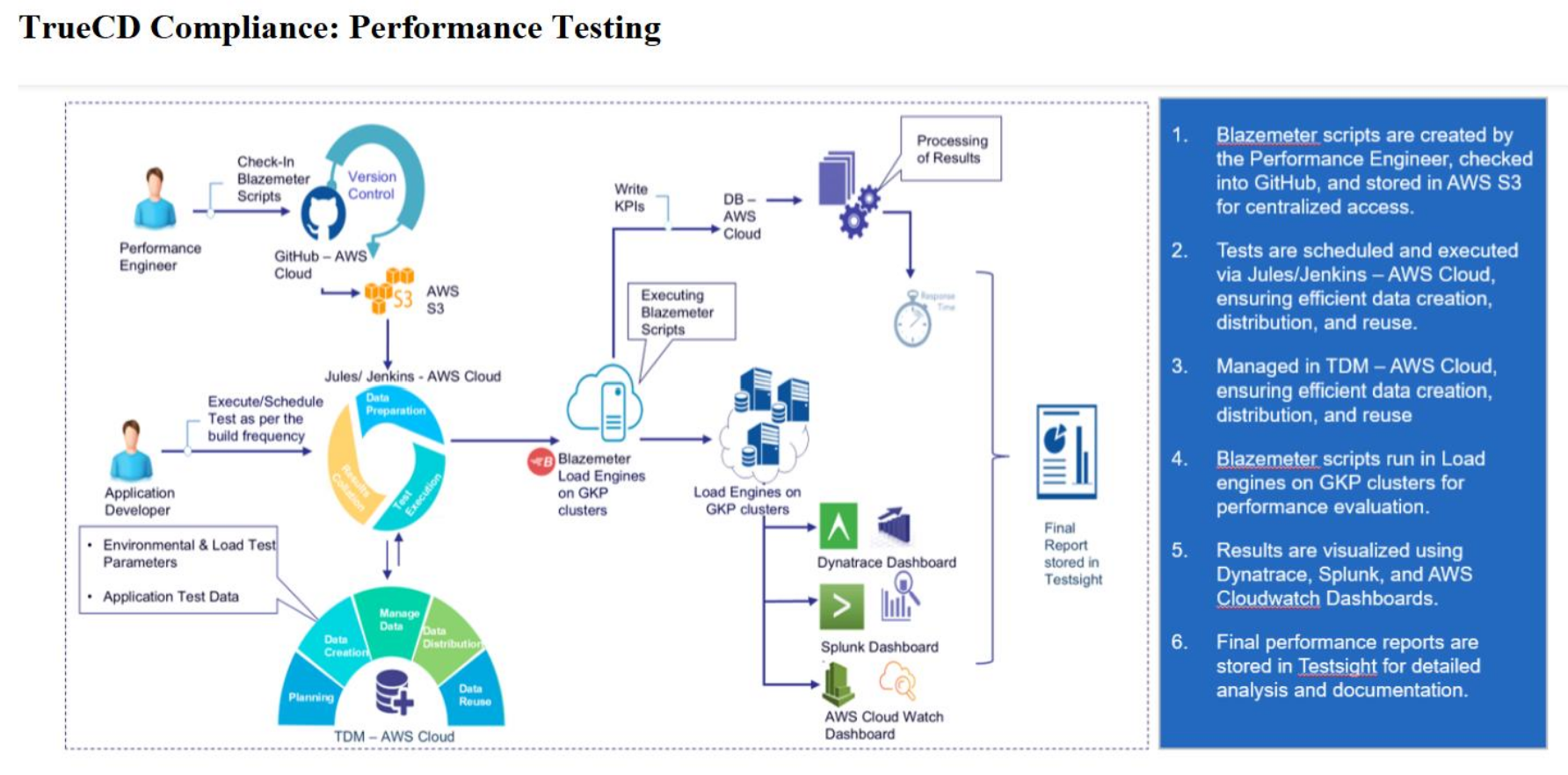
# For first 120 datasets, to be deployed to production in 2025, data mapping between source to target will be 1:1

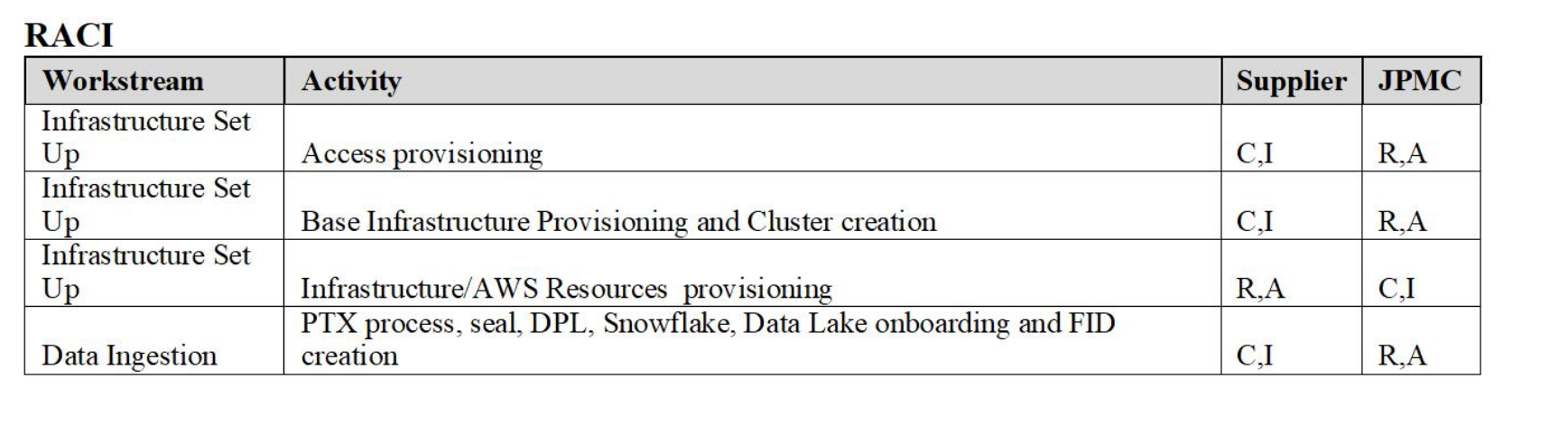
# Any deviations which are not meeting the above assumptions and dependencies and impacts the milestones listed in below section will undergo change management process

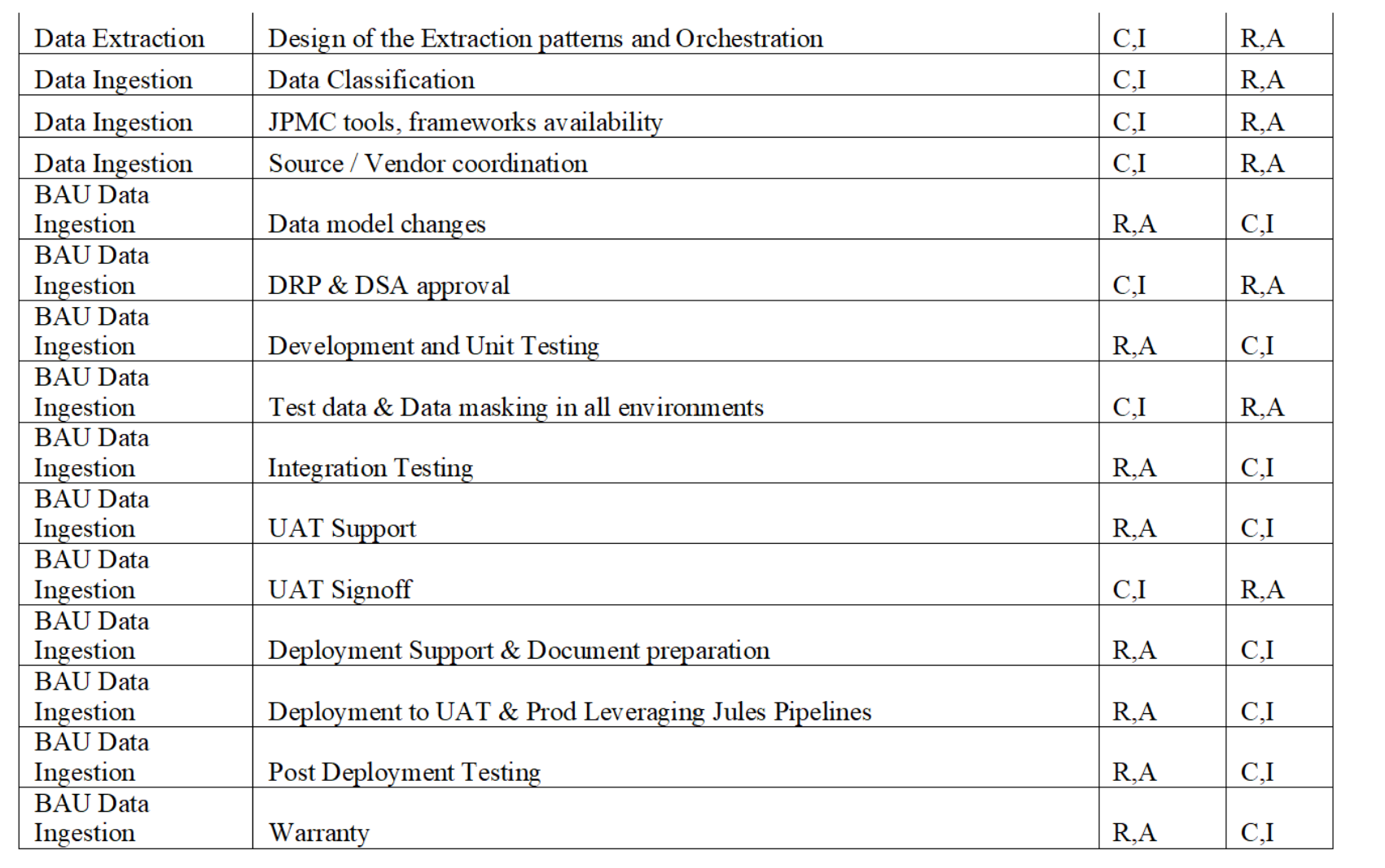


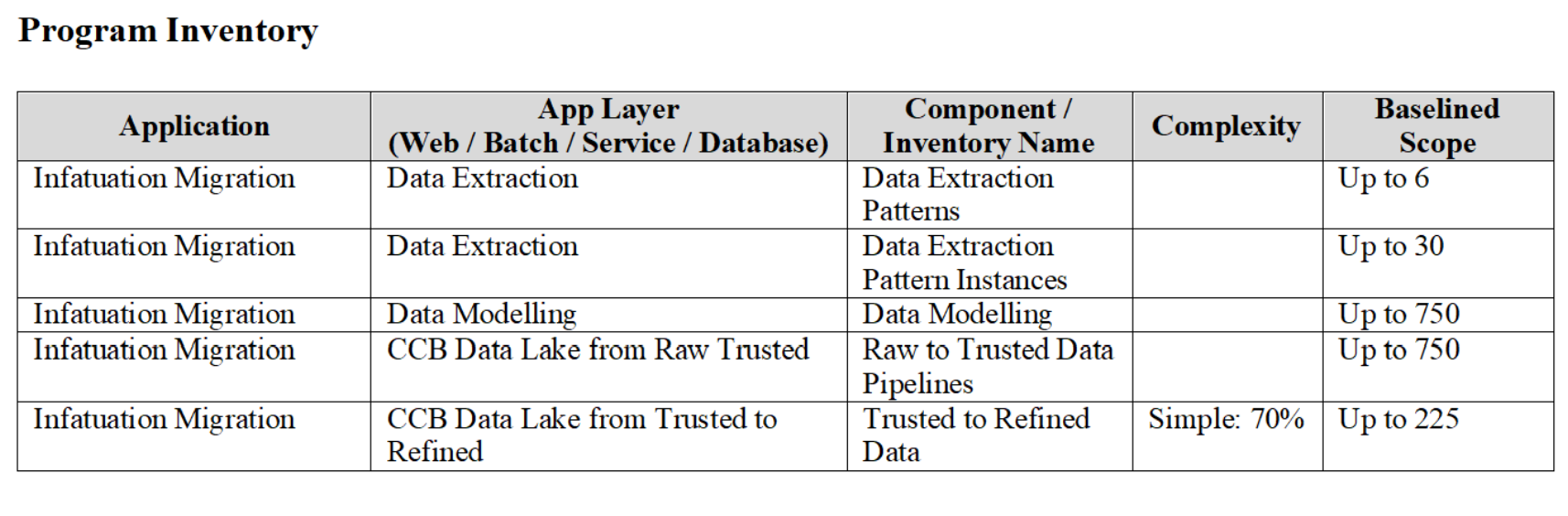


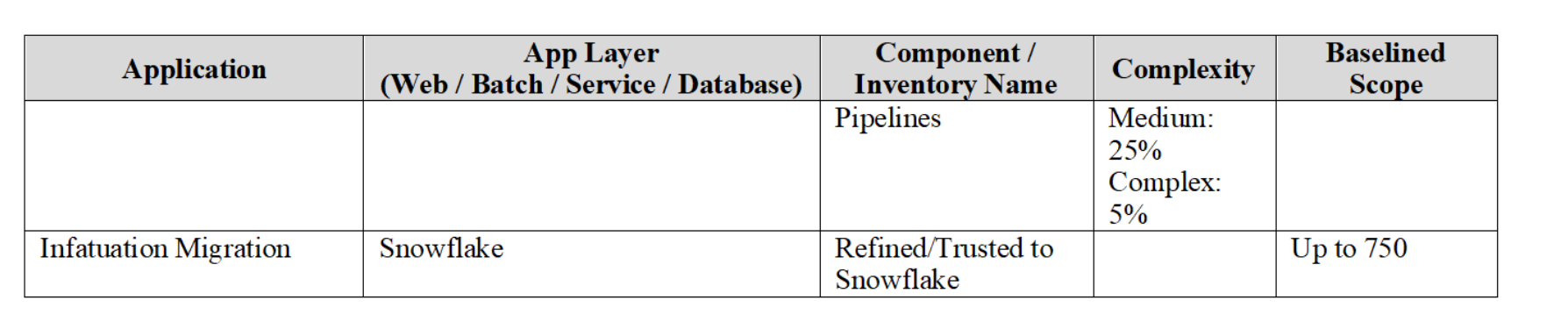












# Milestone 1

# Validate the inventory to ensure DPL compatibility and document the incompatibilities as DPL feature gap Perform Analysis of the mapping document with transformations details provided by JPMC for 150 data pipelines Build data model and obtain DRP approval for up to 50 datasets

# Milestone 2

# Perform analysis of the current data extraction process from data sources and understand the extraction logic, volumes, format, frequency. Understand the Data patterns required for each of the data source as per design provided and determine the instances needed for each of the services designed as per the vendor dataset counts. Build and test one dataset with minimal fields (<20) with one Datasource for the POC provided for any one pattern. Build data model and obtain DRP approval for up to 80 datasets Build data pipeline from Raw until Refined and Snowflake for BAU load of up to 50 datasets

### **1. Perform analysis of the current data extraction process from data sources and understand the extraction logic, volumes, format, frequency.**

* **Analyze how data is currently being extracted** from each data source (like databases, files, APIs, etc.).
* **Understand:**
  + **Extraction logic:** The rules, steps, or code used to extract data (e.g., SQL queries, ETL scripts).
  + **Volumes:** How much data is being extracted (size, record counts, etc.).
  + **Format:** The structure of extracted data (CSV, JSON, XML, etc.).
  + **Frequency:** How often data is extracted (real-time, hourly, daily, weekly, etc.).

### **2. Understand the Data patterns required for each of the data sources as per design provided**

* Review the **design documentation** to learn what data structures, schemas, or patterns are expected from each data source.
* Identify if the source data matches the required patterns, formats, and structures.

### **3. Determine the instances needed for each of the services designed as per the vendor dataset counts**

* For each service or application involved in processing the data, **decide how many instances (deployments, server nodes, or containers) are needed**.
* This should be based on the **number of datasets from each vendor** and the expected processing load.
* Basically, estimate the required infrastructure/resources for handling the data based on vendor dataset counts.

# Build and test one dataset with minimal fields (<20) with one Datasource for the POC provided for any one pattern. Build data model and obtain DRP approval for up to 80 datasets Build data pipeline from Raw until Refined and Snowflake for BAU load of up to 50 datasets

### **1. Build and test one dataset with minimal fields (<20) with one Datasource for the POC provided for any one pattern**

* You need to **select one dataset** that contains **less than 20 fields/columns**.
* Use **one data source** (such as a database, file, or API) for this.
* This is for a **Proof of Concept (POC)**, to demonstrate functionality for any one of the supported data patterns (perhaps CSV, JSON, etc.).
* You’ll need to **build** the process (ingestion, transformation, etc.) and **test** it.

### **2. Build data model and obtain DRP approval for up to 80 datasets**

* For **up to 80 datasets**, you’re required to **design and build the data model** (structure, schema, metadata).
* After building, **submit the data models for approval** to the **DRP** (likely Data Review Panel or equivalent governance group).
* This ensures the models meet standards and are formally accepted.

### **3. Build data pipeline from Raw until Refined and Snowflake for BAU load of up to 50 datasets**

* For **up to 50 datasets**, you should **build the complete data pipeline**:
  + **From Raw:** Starting from the initial, unprocessed data.
  + **To Refined:** Transform and process the data into a cleaned/structured state.
  + **To Snowflake:** Load the processed data into the **Snowflake** data warehouse.
* This pipeline should handle **BAU (Business As Usual) loads**, meaning it must support regular, ongoing data loads.

# Milestone 3

# Build data model and obtain DRP approval for up to 80 datasets Build data pipeline from Raw until Refined and Snowflake for BAU load of up to 60 datasets UAT deployment for up to 40 datasets

UAT deployment for up to 40 datasets

Build Extraction Service for 1st data patterns and complete the testing for all data sets mapped to that pattern. Perform Analysis of the mapping document with transformations details provided by JPMC for 75 data pipelines

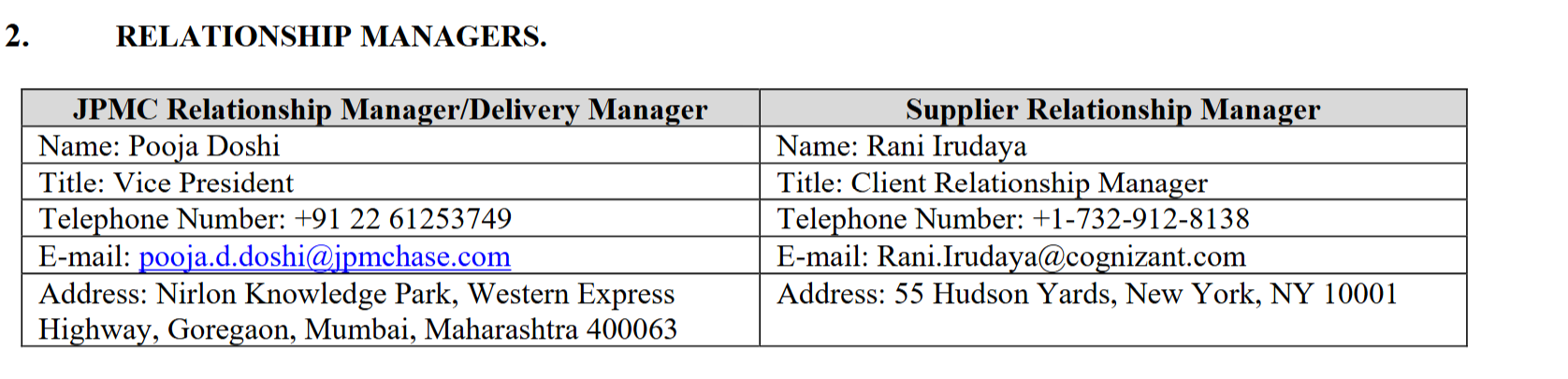
### **1. Build data model and obtain DRP approval for up to 80 datasets**

* For up to **80 datasets**, you need to **design and build data models** (define schema, structure, relationships, metadata, etc.).
* After building these models, you must **submit them for DRP approval** (likely Data Review Panel or an equivalent governance/review committee) to ensure they meet standards and requirements.

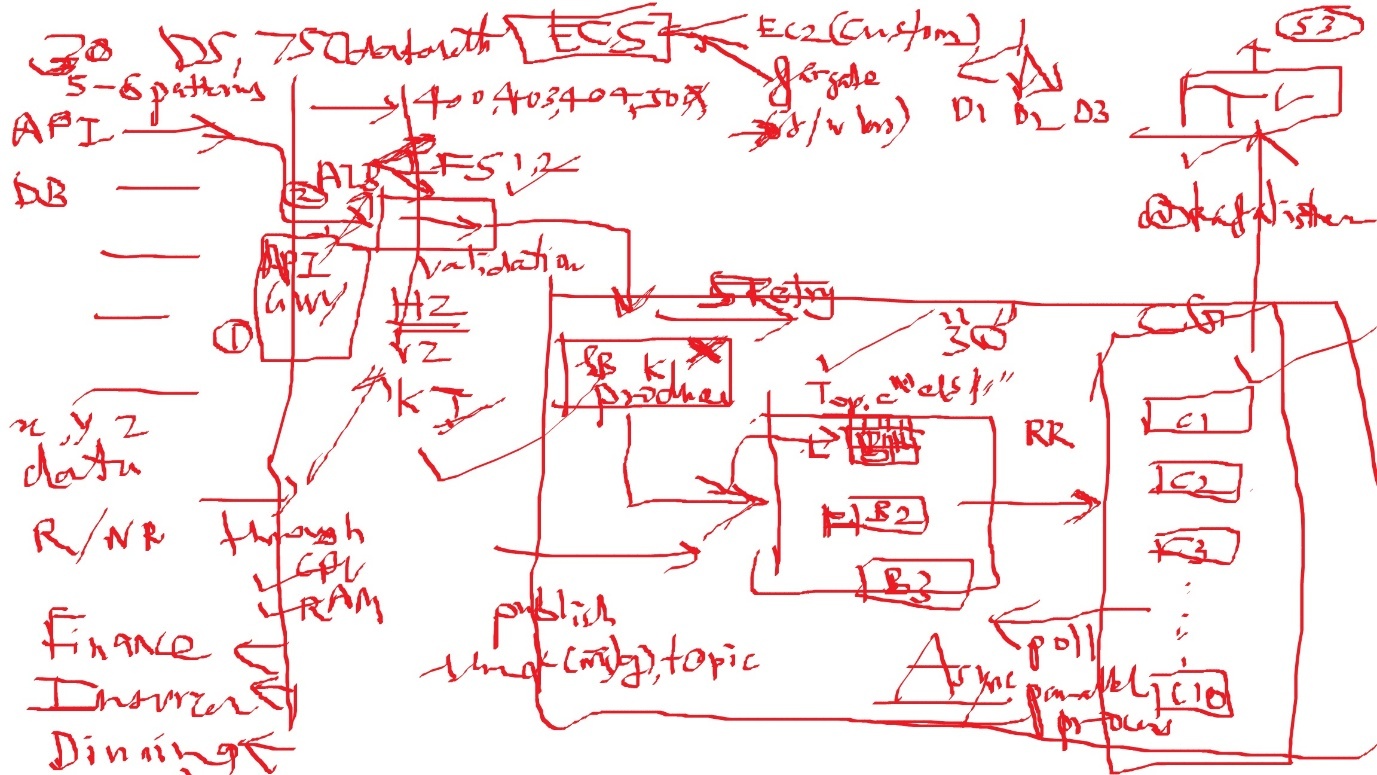
### **2. Build data pipeline from Raw until Refined and Snowflake for BAU load of up to 60 datasets**

* For up to **60 datasets**, you need to **build the complete data pipeline**:
  + **From Raw:** Start with the initial, unprocessed data.
  + **To Refined:** Transform the data into a cleaned, structured, and usable state.
  + **To Snowflake:** Load the refined data into the **Snowflake** data warehouse.
* The pipeline should support **BAU (Business As Usual) loads**, meaning it must handle regular, operational data processing for these datasets.

# Milestone 4 Build data model and obtain DRP approval for up to 120 datasets Build data pipeline from Raw until Refined and Snowflake for BAU load for up to120 datasets UAT deployment for up to 80 datasets Build Extraction Service for 2nd data patterns and complete the testing for all data sets mapped to that pattern.



# System 1 Architecture Streaming ETL Pipelines



Kafka serves as the backbone for Extract, Transform, Load (ETL) operations, where data is continuously ingested, transformed, and loaded into data lakes or analytics systems.

System design for Streaming ETL  
-------------------------------  
Here’s a system design for Streaming ETL (Extract, Transform, Load)   
Pipelines using Apache Kafka for scalable, real-time data ingestion, transformation, and loading into analytics systems.  
  
1. High-Level Architecture  
---------------------------  
Code  
[Data Sources]  
 |  
 (DBs, APIs, Logs, IoT, Files)  
 |  
[Extraction Agents]  
 |  
(Kafka Producer or Kafka Connect Source)  
 |  
[Kafka Cluster]  
 |  
[Kafka Topics: raw-data, transformed-data, error-events]  
 |  
[Stream Processing]  
 |  
[Transformation Logic]  
 |  
(Kafka Producer or Kafka Connect Sink)  
 |  
[Kafka Topics: cleaned-data, enriched-data]  
 |  
[Loaders/Connectors]  
 |  
[Target Systems]  
 (Data Lake, Warehouse, Search, Dashboard)  
  
2. Components Breakdown  
------------------------  
  
A. Extraction  
  
Data Sources: Databases (CDC via Debezium), APIs, log files, IoT devices, cloud buckets, etc.  
Extraction Agents:  
Kafka Producers or Kafka Connect Source connectors ingest data into Kafka (raw-data topic).  
B. Kafka Cluster  
  
Central backbone for event transport.  
Topics:  
raw-data: Unprocessed ingested events.  
transformed-data: Cleaned, filtered, or enriched events.  
error-events: Events that failed transformation/validation.  
  
C. Stream Processing  
Tools: Kafka Streams, Apache Flink, Spark Streaming.  
Transformation Logic:  
Cleansing (remove duplicates, fix formats).  
Enrichment (add geo, user, or reference data).  
Filtering (drop irrelevant events).  
Validation (schema checks).  
  
D. Loading  
Kafka Connect Sink connectors or custom Kafka Consumers push processed data to target systems:  
Data Lakes: S3, HDFS, GCS.  
Data Warehouses: BigQuery, Snowflake, Redshift.  
Search/Analytics: Elasticsearch, Druid.  
Dashboards: Real-time visualization.  
  
E. Error Handling  
Invalid or failed events sent to error-events topic for audit, alerting, or reprocessing.  
  
3. Key Technologies  
--------------------  
Kafka: Event backbone.  
Kafka Connect: Source and sink connectors for integration.  
Stream Processing: Kafka Streams, Flink, Spark Streaming.  
Schema Registry: Validate and evolve event schemas.  
Target Systems: S3, BigQuery, Snowflake, Elasticsearch, etc.  
Monitoring: Prometheus, Grafana, ELK stack.  
  
4. Scalability & Reliability  
-------------------------------  
Topic partitioning for throughput.  
Replication for durability.  
Consumer groups for parallel processing and load balancing.  
Idempotent processing for safe event handling.  
Error handling: Dead-letter topics for failed events.  
  
5. Sample Streaming ETL Event Schema (JSON)  
-------------------------------------------  
JSON  
{  
 "event\_id": "evt-12345",  
 "source": "orders\_db",  
 "timestamp": "2025-09-01T15:06:49Z",  
 "payload": {  
 "order\_id": "ORD-98765",  
 "user\_id": "USR-12345",  
 "total": 150.00,  
 "status": "completed"  
 },  
 "metadata": {  
 "extracted\_by": "debezium",  
 "schema\_version": "v2"  
 }  
}  
  
6. Text-Based Diagram  
---------------------  
Code  
[DB/API/Files/IoT] → [Kafka Producer/Connect Source] → [Kafka Topic: raw-data]  
 ↓  
 [Stream Processor]  
 ↓  
 [Kafka Topic: transformed-data]  
 ↓  
 [Kafka Connect Sink/Loader]  
 ↓  
 [Data Lake/Warehouse/Search]  
  
7. Extensions  
--------------  
Real-time dashboards: For data monitoring and analytics.  
Backpressure handling: Monitor lag and scale consumers as needed.  
Schema evolution: Handle changes in source schemas gracefully.  
Reprocessing: Replay events for correction or migration.  
Security: Encrypt sensitive data, control access, audit logs.

# System 2 Architecture ETL Design

Designing an ETL (Extract, Transform, Load) system on AWS involves leveraging various AWS services to efficiently extract data from sources, transform it as needed, and load it into a target data store. Below is a high-level system design, including recommended AWS services and architecture components.

## **AWS ETL System Design**

### **1. Data Sources**

* **Examples:** Databases (RDS, DynamoDB, on-prem DB), files (S3), APIs, third-party sources.

### **2. Extraction Layer**

* **AWS Glue:** Serverless data integration service to discover, catalog, and extract data.
* **AWS Lambda:** For event-driven extraction (e.g., ingesting API data or reacting to file uploads).
* **Amazon Kinesis / SQS:** For streaming data or message-based extraction.

### **3. Transformation Layer**

* **AWS Glue Jobs:** Python, Spark, or Scala scripts to clean, enrich, and transform data.
* **AWS EMR:** For large-scale distributed processing with Hadoop, Spark, etc.
* **AWS Lambda:** Lightweight transformations, filtering, or enrichment.

### **4. Loading Layer**

* **Amazon S3:** Landing zone or data lake for raw/processed files.
* **Amazon Redshift:** Data warehouse for analytics-ready data.
* **Amazon RDS/Aurora, DynamoDB:** For transactional or operational data stores.

### **5. Orchestration & Workflow**

* **AWS Glue Workflows:** Manage and schedule ETL pipelines.
* **AWS Step Functions:** Visual workflows for complex or multi-step ETL processes.
* **Amazon Managed Workflows for Apache Airflow (MWAA):** For advanced scheduling and pipeline management.

### **6. Monitoring & Logging**

* **Amazon CloudWatch:** Monitor jobs, trigger alarms, log events.
* **AWS CloudTrail:** Track API calls and changes in resources.

### **7. Security & Access Control**

* **IAM Roles & Policies:** Fine-grained access control for services and users.
* **AWS KMS:** Encryption for data at rest and in transit.
* **VPC & Security Groups:** Network isolation and control.

# System 2 Architecture Using AWS.

**[Data Sources]**

**│**

**▼**

**[Extraction Layer: Glue/Lambda/Kinesis]**

**│**

**▼**

**[Transformation Layer: Glue/EMR/Lambda]**

**│**

**▼**

**[Loading Layer: S3/Redshift/RDS/DynamoDB]**

**│**

**▼**

**[Orchestration: Glue Workflow/Step Functions/Airflow]**

**│**

**▼**

**[Monitoring: CloudWatch]**

## **Example Use Case: Batch ETL from S3 to Redshift**

1. **Data arrives in S3 bucket (raw files).**
2. **AWS Glue Crawler** catalogs the data.
3. **AWS Glue Job** extracts, transforms, and loads data into Amazon Redshift.
4. **Glue Workflow / Step Function** schedules and manages the pipeline.
5. **CloudWatch** monitors job status and sends alerts on failure.
6. **IAM/KMS** ensures security throughout.

## **Best Practices**

* Use serverless options (Glue, Lambda) for scalability and cost-effectiveness.
* Store raw and processed data in S3 for audit and rollback.
* Partition data in S3 and Redshift for performance.
* Use Glue Data Catalog for metadata management.
* Schedule jobs during off-peak hours for cost savings.

**AWS Glue** is a fully managed, serverless data integration service provided by Amazon Web Services (AWS). It helps you easily discover, prepare, move, and integrate data from multiple sources for analytics, machine learning, and application development.

### Key Features of AWS Glue:

* **Serverless:** No need to manage servers or infrastructure.
* **ETL (Extract, Transform, Load):** Automates the process of extracting data from various sources, transforming it, and loading it into your data stores.
* **Data Catalog:** Automatically discovers and catalogs metadata about your data across AWS services, making it searchable and queryable.
* **Job Scheduling:** Allows you to schedule and orchestrate ETL jobs.
* **Support for Multiple Data Stores:** Works with Amazon S3, Amazon Redshift, Amazon RDS, and other databases.
* **Scalable:** Handles large volumes of data and scales automatically.
* **Flexible Programming:** Supports Python and Scala for custom ETL scripts using Apache Spark.

### Typical Use Cases:

* Building and running ETL pipelines.
* Data preparation for analytics and machine learning.
* Data migration between different storage and database systems.
* Creating and maintaining a central data catalog.

### Example Workflow:

1. **Crawl and catalog data sources** (e.g., S3, RDS).
2. **Create ETL jobs** to transform and process the data.
3. **Load the transformed data** into a data warehouse or analytics platform.

### Benefits:

* Reduces the time and effort required to build complex data pipelines.
* Integrates natively with other AWS services.
* Pay only for the resources you consume.

**Learn more:** [AWS Glue Documentation](https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html)

[AWS Glue concepts - AWS Glue](https://docs.aws.amazon.com/glue/latest/dg/components-key-concepts.html)

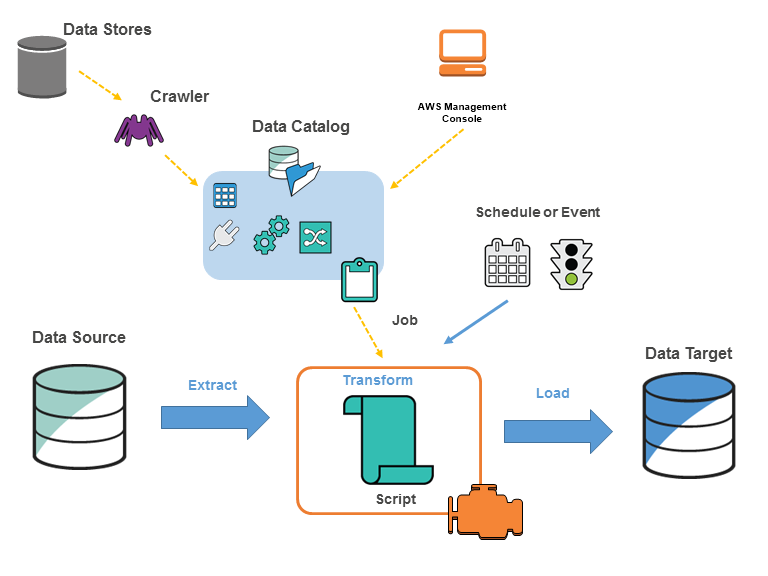
AWS Glue is a fully managed ETL (extract, transform, load) service that allows you to easily move data between different data sources and targets. The key components are:

* **Data Catalog**: A metadata store containing table definitions, job definitions, and other control information for your ETL workflows.
* **Crawlers**: Programs that connect to data sources, infer data schemas, and create metadata table definitions in the Data Catalog.
* **ETL Jobs**: The business logic to extract data from sources, transform it using Apache Spark scripts, and load it into targets.
* **Triggers**: Mechanisms to initiate job runs based on schedules or events.

The typical workflow involves:

1. Define data sources and targets in the Data Catalog.
2. Use Crawlers to populate the Data Catalog with table metadata from data sources.
3. Define ETL jobs with transformation scripts to move and process data.
4. Run jobs on-demand or based on triggers.
5. Monitor job performance using dashboards.

The following diagram shows the architecture of an AWS Glue environment.



You define jobs in AWS Glue to accomplish the work that's required to extract, transform, and load (ETL) data from a data source to a data target. You typically perform the following actions:

* For data store sources, you define a crawler to populate your AWS Glue Data Catalog with metadata table definitions. You point your crawler at a data store, and the crawler creates table definitions in the Data Catalog. For streaming sources, you manually define Data Catalog tables and specify data stream properties.

In addition to table definitions, the AWS Glue Data Catalog contains other metadata that is required to define ETL jobs. You use this metadata when you define a job to transform your data.

* AWS Glue can generate a script to transform your data. Or, you can provide the script in the AWS Glue console or API.
* You can run your job on demand, or you can set it up to start when a specified trigger occurs. The trigger can be a time-based schedule or an event.

When your job runs, a script extracts data from your data source, transforms the data, and loads it to your data target. The script runs in an Apache Spark environment in AWS Glue.

###### Important

Tables and databases in AWS Glue are objects in the AWS Glue Data Catalog. They contain metadata; they don't contain data from a data store.

|  |
| --- |
| **Text-based data, such as CSVs, must be encoded in UTF-8 for AWS Glue to process it successfully. For more information, see**[**UTF-8**](https://en.wikipedia.org/wiki/UTF-8)**in Wikipedia.** |

## AWS Glue terminology

AWS Glue relies on the interaction of several components to create and manage your extract, transform, and load (ETL) workflow.

### AWS Glue Data Catalog

The persistent metadata store in AWS Glue. It contains table definitions, job definitions, and other control information to manage your AWS Glue environment. Each AWS account has one AWS Glue Data Catalog per region.

### Classifier

Determines the schema of your data. AWS Glue provides classifiers for common file types, such as CSV, JSON, AVRO, XML, and others. It also provides classifiers for common relational database management systems using a JDBC connection. You can write your own classifier by using a grok pattern or by specifying a row tag in an XML document.

### Connection

A Data Catalog object that contains the properties that are required to connect to a particular data store.

### Crawler

A program that connects to a data store (source or target), progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in the AWS Glue Data Catalog.

### Database

A set of associated Data Catalog table definitions organized into a logical group.

### Data store, data source, data target

A data store is a repository for persistently storing your data. Examples include Amazon S3 buckets and relational databases. A data source is a data store that is used as input to a process or transform. A data target is a data store that a process or transform writes to.

### Development endpoint

An environment that you can use to develop and test your AWS Glue ETL scripts.

### Dynamic Frame

A distributed table that supports nested data such as structures and arrays. Each record is self-describing, designed for schema flexibility with semi-structured data. Each record contains both data and the schema that describes that data. You can use both dynamic frames and Apache Spark DataFrames in your ETL scripts, and convert between them. Dynamic frames provide a set of advanced transformations for data cleaning and ETL.

### Job

The business logic that is required to perform ETL work. It is composed of a transformation script, data sources, and data targets. Job runs are initiated by triggers that can be scheduled or triggered by events.

### Job performance dashboard

AWS Glue provides a comprehensive run dashboard for your ETL jobs. The dashboard displays information about job runs from a specific time frame.

### Notebook interface

An enhanced notebook experience with one-click setup for easy job authoring and data exploration. The notebook and connections are configured automatically for you. You can use the notebook interface based on Jupyter Notebook to interactively develop, debug, and deploy scripts and workflows using AWS Glue serverless Apache Spark ETL infrastructure. You can also perform ad-hoc queries, data analysis, and visualization (for example, tables and graphs) in the notebook environment.

### Script

Code that extracts data from sources, transforms it, and loads it into targets. AWS Glue generates PySpark or Scala scripts.

### Table

The metadata definition that represents your data. Whether your data is in an Amazon Simple Storage Service (Amazon S3) file, an Amazon Relational Database Service (Amazon RDS) table, or another set of data, a table defines the schema of your data. A table in the AWS Glue Data Catalog consists of the names of columns, data type definitions, partition information, and other metadata about a base dataset. The schema of your data is represented in your AWS Glue table definition. The actual data remains in its original data store, whether it be in a file or a relational database table. AWS Glue catalogs your files and relational database tables in the AWS Glue Data Catalog. They are used as sources and targets when you create an ETL job.

### Transform

The code logic that is used to manipulate your data into a different format.

### Trigger

Initiates an ETL job. Triggers can be defined based on a scheduled time or an event.

### Visual job editor

The visual job editor is a graphical interface that makes it easy to create, run, and monitor extract, transform, and load (ETL) jobs in AWS Glue. You can visually compose data transformation workflows, seamlessly run them on AWS Glue's Apache Spark-based serverless ETL engine, and inspect the schema and data results in each step of the job.

### Worker

With AWS Glue, you only pay for the time your ETL job takes to run. There are no resources to manage, no upfront costs, and you are not charged for startup or shutdown time. You are charged an hourly rate based on the number of **Data Processing Units** (or DPUs) used to run your ETL job. A single Data Processing Unit (DPU) is also referred to as a worker. AWS Glue comes with multiple worker types to help you select the configuration that meets your job latency and cost requirements. Workers come in Standard, G.1X, G.2X, G.4X, G.8X, G.12X, G.16X, G.025X, and memory-optimized R.1X, R.2X, R.4X, R.8X configurations.

# Note About EC2 /ECS / Fargate :

### **EC2, ECS, and Fargate: Key Differences**

* **Amazon EC2:** You control the virtual machines, install software, and manage scaling. You can use EC2 for custom ETL workloads, but you are responsible for scaling and maintenance.
* **Amazon ECS (Elastic Container Service):** Orchestrates containers (like Docker) across EC2 instances or Fargate.
* **AWS Fargate:** Serverless compute engine that runs containers without you managing EC2 servers.

### **Handling High-Volume Data for ETL**

#### **EC2 Instances**

* **Yes, EC2 can handle high-volume ETL** if you provision adequate resources (CPU, memory, storage, network).
* **You must manage scaling:** For very high volume, you’ll need to scale out manually (Auto Scaling Groups, Load Balancers).

#### **ECS with EC2 Launch Type**

* **Yes, ECS with EC2 can handle high-volume ETL** by distributing the workload across multiple EC2 instances in a cluster.
* **Scalability:** You can increase the number of EC2 instances as needed.

#### **ECS with Fargate Launch Type**

* **Yes, ECS with Fargate can handle high-volume ETL**, but there are some considerations:
  + **Fargate is serverless:** You don’t manage servers, just specify CPU/memory per task.
  + **Scaling:** Fargate scales automatically with the number of tasks you launch.
  + **Data Volume Limits:** Fargate tasks have limits on CPU/memory (up to 16 vCPU and 120 GB RAM per task as of 2024), and storage is ephemeral (up to 200 GB per task). For very large ETL jobs, you may need to split your workload into smaller chunks or use persistent storage (like S3, EFS).

#### **Best Practices for High-Volume ETL**

* Use S3 as your data landing/processing zone.
* Break large ETL jobs into smaller, parallelizable chunks.
* Use ECS/Fargate for stateless, scalable processing.
* Use managed services like **AWS Glue** or **EMR** for very large-scale distributed ETL.
* Monitor and scale based on throughput and resource usage.

## **Summary Table**

| **Option** | **Handles High Volume?** | **Scaling** | **Who Manages Servers?** | **Persistent Storage** |
| --- | --- | --- | --- | --- |
| EC2 | Yes | Manual | You | Yes (EBS, EFS) |
| ECS (EC2) | Yes | Cluster | You | Yes (EBS, EFS) |
| ECS (Fargate) | Yes\* | Automatic | AWS | Limited (EFS) |
| AWS Glue/EMR | Yes (Best) | Automatic | AWS | Yes (S3, HDFS) |

\*Fargate has task size limits. For truly massive ETL, Glue/EMR is preferred.

**Recommendation:**  
For very high-volume, distributed ETL, AWS Glue or EMR is usually better. Use ECS/Fargate for scalable, container-based processing—just be aware of task resource limits and design your ETL to parallelize where possible.

# What is Amazon Athena?

[**PDF**](https://docs.aws.amazon.com/pdfs/athena/latest/ug/athena-ug.pdf#what-is)

[**RSS**](https://docs.aws.amazon.com/athena/latest/ug/amazon-athena-release-notes.rss)

Focus mode

Amazon Athena is an interactive query service that makes it easy to analyze data directly in Amazon Simple Storage Service (Amazon S3) using standard [SQL](https://docs.aws.amazon.com/athena/latest/ug/ddl-sql-reference.html). With a few actions in the AWS Management Console, you can point Athena at your data stored in Amazon S3 and begin using standard SQL to run ad-hoc queries and get results in seconds.

For more information, see [Get started](https://docs.aws.amazon.com/athena/latest/ug/getting-started.html).

Amazon Athena also makes it easy to interactively run data analytics using Apache Spark without having to plan for, configure, or manage resources. When you run Apache Spark applications on Athena, you submit Spark code for processing and receive the results directly. Use the simplified notebook experience in Amazon Athena console to develop Apache Spark applications using Python or [Use Athena notebook APIs](https://docs.aws.amazon.com/athena/latest/ug/notebooks-spark-api-list.html).

For more information, see [Get started with Apache Spark on Amazon Athena](https://docs.aws.amazon.com/athena/latest/ug/notebooks-spark-getting-started.html).

Athena SQL and Apache Spark on Amazon Athena are serverless, so there is no infrastructure to set up or manage, and you pay only for the queries you run. Athena scales automatically—running queries in parallel—so results are fast, even with large datasets and complex queries.

# About Snow Flake

**Snowflake** is a cloud-based data platform known for its scalable and high-performance data warehousing, data lake, data engineering, data science, and data application capabilities.

### **Key Features of Snowflake**

* **Cloud-Native:** Runs on AWS, Azure, and Google Cloud.
* **Data Warehousing:** Stores and analyzes large volumes of structured and semi-structured data (like JSON, Parquet).
* **Separation of Storage and Compute:** Allows you to scale compute resources independently from data storage.
* **Multi-Cluster Architecture:** Multiple compute clusters can access the same data simultaneously, supporting concurrency without performance bottlenecks.
* **Zero Management:** No infrastructure to manage; scaling, maintenance, and tuning are handled by Snowflake.
* **Secure Data Sharing:** Enables organizations to share data easily and securely across accounts and clouds.
* **Supports SQL:** Uses familiar SQL syntax for querying, transforming, and managing data.
* **Automatic Scaling:** Automatically adds or removes compute resources as needed.

### **Typical Use Cases**

* Enterprise data warehousing
* Analytics and business intelligence
* Data lake and data engineering
* Machine learning and AI data pipelines
* Secure data sharing between organizations

### **How Snowflake Differs from Traditional Data Warehouses**

* **Fully Managed:** No hardware or software to install or manage.
* **Elastic:** Instantly scales up or down based on workload.
* **Pay-per-Use:** You only pay for what you use (storage and compute).
* **Cross-Cloud:** Works seamlessly across major cloud providers.

### **Snowflake Architecture Overview**

[Data Sources] → [Snowflake Storage] ← [Compute Clusters (Virtual Warehouses)]

↑

[SQL Interface/API]

**In summary:**  
Snowflake provides a simple, scalable, and powerful solution for storing, processing, and analyzing big data in the cloud, making it popular for modern analytics and data engineering needs.

concise comparison of **Snowflake** vs **Amazon Athena**—both popular cloud data platforms, but with key differences in architecture, use cases, and capabilities:

# Snow Flake v/s Amazon Athena

## **Snowflake**

**Type:** Cloud Data Warehouse Platform  
**Architecture:** Proprietary, multi-cluster shared data architecture  
**Storage:** Manages its own storage, but can access external data lakes  
**Compute:** Compute and storage are separated; scalable virtual warehouses  
**Data Formats:** Structured & semi-structured (CSV, JSON, Parquet, Avro, ORC, etc.)  
**Query Language:** Standard SQL  
**Performance:** High performance, scalable, supports concurrent workloads  
**Management:** Fully managed, zero infrastructure management  
**Features:** Time travel, cloning, secure data sharing, automatic scaling, cross-cloud  
**Pricing:** Pay-per-use for storage & compute

**Typical Use Cases:**

* Centralized analytics, enterprise data warehousing
* Data sharing across organizations
* Large-scale BI workloads
* Advanced security & compliance

## **Amazon Athena**

**Type:** Serverless Query Service for Data Lakes  
**Architecture:** Query engine for data stored in Amazon S3  
**Storage:** Uses data in S3 directly; no data warehouse layer  
**Compute:** Serverless; you pay per query (per TB scanned)  
**Data Formats:** Structured & semi-structured (CSV, JSON, Parquet, Avro, ORC, etc.)  
**Query Language:** Standard SQL (Presto/Trino engine)  
**Performance:** Depends on data partitioning and format; scales automatically  
**Management:** Fully managed, no infrastructure; just point to S3  
**Features:** Integrates with Glue Data Catalog, supports federated queries, easy setup  
**Pricing:** Pay-per-query (per TB scanned)

**Typical Use Cases:**

* Ad-hoc analytics on S3 data
* Data lake exploration
* Log analysis
* Quick, low-cost analysis without building a warehouse

## **Summary Table**

| **Feature** | **Snowflake** | **Amazon Athena** |
| --- | --- | --- |
| Data Location | Internal + external (staged) | Amazon S3 |
| Compute | Dedicated, scalable clusters | Serverless, pay-per-query |
| Use Case | Data warehouse, enterprise BI | Data lake analytics, ad-hoc SQL |
| Performance | High, supports concurrency | Depends on S3 file structure |
| Management | Fully managed | Fully managed (serverless) |
| Pricing | Storage + compute | Per TB scanned |
| Security/Sharing | Advanced (cross-account/cloud) | IAM-based, basic sharing |
| Data Catalog | Built-in | AWS Glue Data Catalog |

## **Which to Choose?**

* **Choose Snowflake** if you need:
  + Enterprise-grade data warehousing
  + High concurrency and performance
  + Data sharing, time travel, advanced features
* **Choose Athena** if you need:
  + Quick, cost-effective ad-hoc queries
  + Analytics directly on S3 data lake
  + No infrastructure or warehouse setup

**In summary:**  
**Snowflake** is best for large-scale, persistent, multi-user analytics and warehousing.  
**Athena** is best for serverless, ad-hoc querying of S3 data lakes.

# System 3 Architecture

# High-level system design for using **Apache Kafka** and **Apache Spark**

High-level system design for using **Apache Kafka** and **Apache Spark** together for ETL (Extract, Transform, Load) operations:

## **System Design: Kafka + Spark for ETL**

### **1. Data Ingestion (Extract)**

* **Producers:** Various applications, databases, or services send raw data to **Kafka topics**.
* **Kafka Cluster:** Acts as a distributed, high-throughput message broker, decoupling producers and consumers.

### **2. Real-time Data Processing (Transform)**

* **Apache Spark Streaming:** Consumes data from Kafka topics.
  + **Spark Structured Streaming** or **Spark Streaming** APIs can be used.
  + Spark workers process data in micro-batches or real-time.
  + Transformation logic is applied (cleaning, enrichment, aggregation, etc.).

### **3. Data Output (Load)**

* **Sink Targets:** Processed data is written to target systems:
  + **Data Lake:** Amazon S3, HDFS, Azure Data Lake, etc.
  + **Data Warehouse:** Amazon Redshift, Snowflake, Google BigQuery, etc.
  + **NoSQL/Relational DB:** MongoDB, Cassandra, MySQL, etc.
  + **Another Kafka Topic:** For further downstream consumers.

### **4. Orchestration & Monitoring**

* **Workflow Orchestration:** Apache Airflow, AWS Step Functions, or custom scripts to manage pipeline scheduling and error recovery.
* **Monitoring:** Tools like Spark UI, Kafka Manager, Prometheus/Grafana for health and performance.

## **Architecture Diagram**

Code

[Data Sources]

│

▼

[Kafka Producers] ---> [Kafka Cluster] ---> [Spark Streaming/Structured Streaming]

│

▼

[ETL Transform Logic]

│

▼

[Output Sinks: S3, Redshift, DB, etc.]

│

▼

[Athena/QuickSight/BI]

### **Typical ETL & Analytics Flow on AWS**

Code

[Data Sources] → [AWS Glue/EMR/Lambda/Kinesis] → [Transform + Load] → [S3/Redshift/Elasticsearch] → [Athena/QuickSight/BI]

## **How it Works (Step-by-Step):**

1. **Producers** send data (events, logs, transactions) to **Kafka topics**.
2. **Spark Streaming Application** subscribes to Kafka topics.
3. Spark processes the data in real-time or micro-batches, applying ETL transformations.
4. Processed data is written ("loaded") into one or more target systems.
5. Orchestration and monitoring tools ensure the pipeline runs smoothly, handles errors, and scales as needed.

## **Benefits of Using Kafka + Spark Together**

* **Scalable & Fault-tolerant:** Both Kafka and Spark are distributed systems.
* **Real-time ETL:** Enables low-latency data processing.
* **Decoupling:** Kafka separates data producers from consumers.
* **Flexible Output:** Spark can write to multiple sinks.

## **Sample Spark Streaming Code (Pseudo-code)**

Python

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("KafkaSparkETL").getOrCreate()

# Read from Kafka

df = spark \

.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers", "broker1:9092,broker2:9092") \

.option("subscribe", "source-topic") \

.load()

# Transformation logic (example)

transformed\_df = df.selectExpr("CAST(value AS STRING)").transform(custom\_etl\_function)

# Write to sink (e.g., S3)

transformed\_df.writeStream \

.format("parquet") \

.option("path", "s3://your-bucket/processed/") \

.option("checkpointLocation", "/tmp/checkpoints") \

.start()

## **Best Practices**

* Partition Kafka topics for parallelism.
* Use Spark Structured Streaming for better reliability.
* Tune Spark resources based on data volume.
* Use checkpoints for Spark jobs for fault tolerance.
* Monitor Kafka and Spark clusters for bottlenecks.

# AWS Services that can be used for Analytics

main **AWS tools/services** commonly used for analytics and ETL (Extract, Transform, Load) operations:

## **1. AWS Glue**

* **Purpose:** Serverless ETL service for data preparation, transformation, and loading.
* **Features:** Data catalog, job scheduling, integration with S3, Redshift, RDS.

## **2. Amazon EMR (Elastic MapReduce)**

* **Purpose:** Big data processing using frameworks like Apache Spark, Hadoop, Hive.
* **Features:** Scalable clusters, integration with S3, supports custom analytics and machine learning workloads.

## **3. Amazon Redshift**

* **Purpose:** Cloud data warehouse for fast analytics at scale.
* **Features:** Columnar storage, MPP architecture, integrates with BI tools, supports complex analytics queries.

## **4. Amazon Athena**

* **Purpose:** Serverless, interactive query service for data in S3 using standard SQL.
* **Features:** No infrastructure setup, pay-per-query, integrates with Glue Data Catalog.

## **5. Amazon Kinesis**

* **Purpose:** Real-time analytics and processing of streaming data.
* **Features:** Kinesis Data Streams, Kinesis Data Firehose, Kinesis Data Analytics (SQL over streams).

## **6. AWS Lambda**

* **Purpose:** Serverless compute for lightweight ETL and event-driven analytics.
* **Features:** Triggered by events (S3, DynamoDB, Kinesis), suitable for simple transformations.

## **7. Amazon QuickSight**

* **Purpose:** Business intelligence (BI) and data visualization.
* **Features:** Dashboards, ML insights, integrates with Redshift, Athena, S3.

## **8. AWS Data Pipeline**

* **Purpose:** Orchestration of data movement and transformation between AWS services.
* **Features:** Scheduling, dependency management, batch processing.

## **9. Amazon OpenSearch Service (formerly Elasticsearch Service)**

* **Purpose:** Search and analytics on log and text data.
* **Features:** Full-text search, real-time analytics, integrates with Kinesis and Lambda.

### **Typical ETL & Analytics Flow on AWS**

Code

[Data Sources] → [AWS Glue/EMR/Lambda/Kinesis] → [Transform + Load] → [S3/Redshift/Elasticsearch] → [Athena/QuickSight/BI]

**Summary:**

* Use **Glue** for serverless ETL, **EMR** for big data frameworks, **Redshift** for warehousing analytics, **Athena** for querying S3 data lakes, **Kinesis** for real-time analytics, and **QuickSight** for visualization.
* These can be combined as needed for your end-to-end analytics pipeline.

[AWS Glue: essential tips for enhancing ETL development and operations | by George Matheou | Data Engineer Things](https://blog.dataengineerthings.org/aws-glue-essential-tips-for-enhancing-etl-development-and-operations-39662b9ebdba)

[AWS Glue ETL Workflow for Data Analytics | by Allen Ma | Medium](https://pony-run.medium.com/aws-glue-etl-workflow-for-data-analytics-b6f3c3e41eec)

[AWS Glue components - AWS Glue](https://docs.aws.amazon.com/glue/latest/dg/components-overview.html#data-catalog-intro)

[AWS Glue Documentation](https://docs.aws.amazon.com/glue/)

[Spark Streaming - Spark 4.0.1 Documentation](https://spark.apache.org/docs/latest/streaming-programming-guide.html)

[Apache Spark™ - Unified Engine for large-scale data analytics](https://spark.apache.org/)

[AWS Glue vs Spark on Kubernetes — Which One is Right for You? | by Aleksandar Brkljac | Medium](https://abrkljac.medium.com/aws-glue-vs-spark-on-kubernetes-which-one-is-right-for-you-5eca53d1045b)

[Getting started with AWS Glue for ETL Operations | by Vaibhav Srivastava | Medium](https://vabnix.medium.com/getting-started-with-aws-glue-for-etl-operations-5d74bec0470d)

[Understanding AWS Glue for ETL. In the big data world, the biggest… | by Joan Ngugi | Medium](https://ngugijoan.medium.com/understanding-aws-glue-for-etl-51db6561fe45)

[AWS Glue for ETL (extract, transform and load) processes | by Alejandro Cora González | Medium](https://alek-cora-glez.medium.com/aws-glue-for-etl-extract-transform-and-load-processes-25d3ef794c85)

[Supported data sources for crawling - AWS Glue](https://docs.aws.amazon.com/glue/latest/dg/crawler-data-stores.html)

[AWS Glue components - AWS Glue](https://docs.aws.amazon.com/glue/latest/dg/components-overview.html)

[When should I use Athena? - Amazon Athena](https://docs.aws.amazon.com/athena/latest/ug/when-should-i-use-ate.html)

[What is AWS Lake Formation? - AWS Lake Formation](https://docs.aws.amazon.com/lake-formation/latest/dg/what-is-lake-formation.html)

[What Is AWS CloudTrail? - AWS CloudTrail](https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html)

[Amazon Redshift Spectrum - Amazon Redshift](https://docs.aws.amazon.com/redshift/latest/dg/c-using-spectrum.html)

[Use resource-based policies for Amazon EMR access to AWS Glue Data Catalog - Amazon EMR](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-iam-roles-glue.html)

(Using Java)

[Spark Streaming - Spark 4.0.1 Documentation](https://spark.apache.org/docs/latest/streaming-programming-guide.html#a-quick-example)