NYC Yellow Taxi Data Platform

Architecture Overview

End-to-End Governed Data Engineering Solution

Author:

Mallikarjun Reddy

https://github.com/mallireddy0915/AWS-Data-Engineer

January 2026

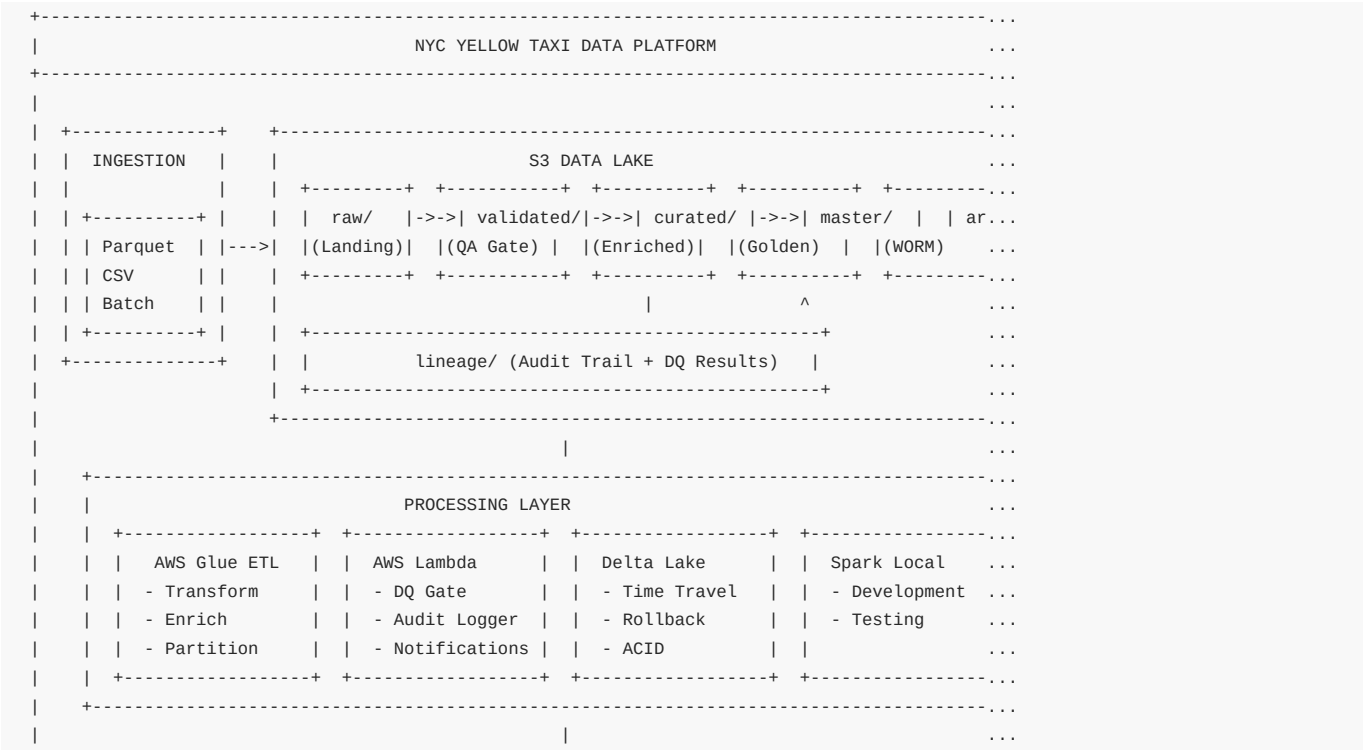
NYC Yellow Taxi Data Platform - Architecture Overview

Executive Summary

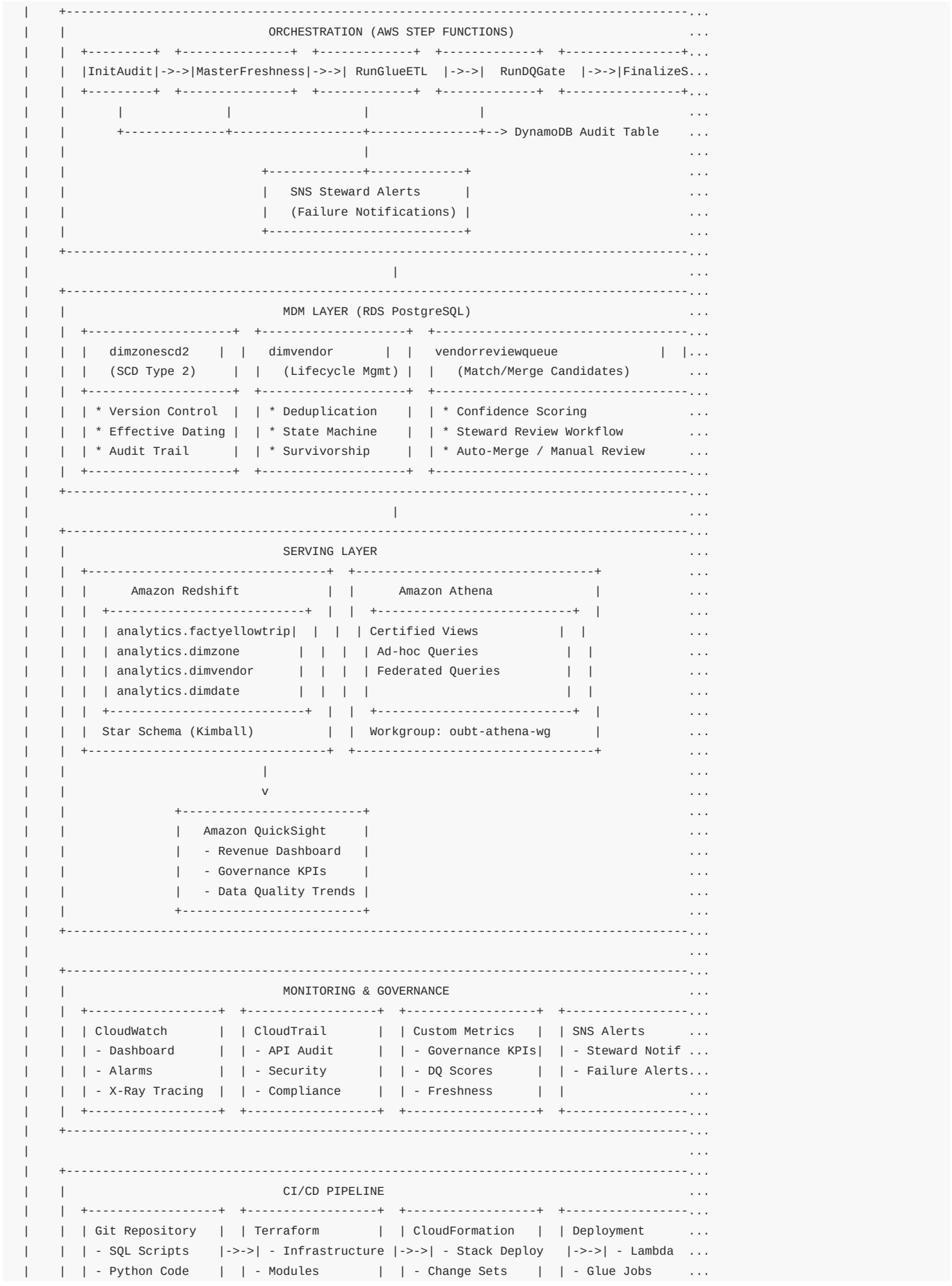
This document presents a comprehensive end-to-end Governed Data Platform built on AWS for NYC Yellow Taxi trip analytics. The platform implements enterprise-grade data engineering patterns including:

- Multi-zone data lake with governance controls
- Master Data Management (MDM) with deduplication and SCD Type 2
- Automated orchestration via AWS Step Functions
- Data quality gates with Great Expectations and AWS Glue DQ
- Dimensional analytics on Amazon Redshift and Athena
- Real-time monitoring with CloudWatch dashboards
- CI/CD deployment via CloudFormation

Architecture Diagram



NYC Yellow Taxi Data Platform - Architecture Overview



```
|    |  | - Config Files   |  | - Environments   |  | - Rollback       |  | - Step Functions...
|    |  +------------------+  +------------------+  +------------------+  +-----------------...
|    +-------------------------------------------------------------------------------------...
|                                                                                          ...
+------------------------------------------------------------------------------------------...
```

---

1. Ingestion Layer (Batch via S3)

Overview

Data arrives as batch files (Parquet, CSV) into the S3 landing zone.

Implementation Details

Component	Description
Source Files	NYC Yellow Taxi trip records (Parquet), Tax..
Landing Zone	s3://arjun-s3-776312084600/raw/
File Formats	Parquet (trips), CSV (reference data)
Ingestion Pattern	Batch upload via AWS CLI, boto3, or schedul..

Key Scripts

- data_ops.py - S3 upload/download operations

- s3_infra.py - Bucket creation and lifecycle configuration

---

2. Storage Layer - S3 Data Lake with Zones

Zone Architecture

```
+-------------------------------------------------------------------------+
|                          S3 DATA LAKE ZONES                             |
+--------------+--------------+--------------+--------------+--------------+
|    raw/      |  validated/  |   curated/   |   master/    |   archive/   |
|  (Landing)   |  (QA Gate)   |  (Enriched)  |  (Golden)    |   (WORM)     |
+--------------+--------------+--------------+--------------+--------------+
| * Immutable  | * Quality    | * Business   | * Golden     | * Compliance |
| * Original   |   checks     |   rules      |   records    |   retention  |
| * Source of  |   passed     |   applied    | * Strict     | * Immutable  |
|   truth      | * Schema     | * MDM        |   governance | * 7+ years   |
| * Open       |   validated  |   enriched   | * Steward    | * Legal hold |
|   ingestion  | * Auto-gated | * Analytics  |   approval   |   enabled    |
|              |              |   ready      |   required   |              |
+--------------+--------------+--------------+--------------+--------------+
```

```
                    |
          +---------------+---------------+
          |         lineage/              |
          |   (Audit Trail + DQ Results)  |
          |   * Data provenance tracking  |
          |   * Transformation history    |
          |   * Governance audit logs     |
          +-------------------------------+
```

Bucket Lifecycle Configuration

Zone	Retention	Transition	Access Control
raw/	90 days hot	-> Glacier after 90d	Open ingestion
validated/	30 days hot	Archive after 30d	Auto-gated
curated/	1 year	Standard-IA after 60d	Governed transforms
master/	Permanent	None	Strict access control
archive/	7+ years	Deep Archive	Legal hold, WORM

---

3. Processing Layer - Spark/Glue/Lambda with Delta Lake

AWS Glue ETL

```
# Key transformation logic (day7gluetaxicurated.py)
# 1. Quality gates (validated zone rules)
tripsq = trips.filter(col("tpeppickupdatetime").isNotNull()) \
               .filter(col("tpepdropoffdatetime").isNotNull()) \
               .filter(col("tpeppickupdatetime") <= col("tpepdropoffdatetime")) \
               .filter(col("tripdistance") >= 0) \
               .filter(col("totalamount") >= 0)
# 2. Add partitions
tripsq = tripsq.withColumn("year", year(col("tpeppickupdatetime"))) \
                 .withColumn("month", month(col("tpeppickupdatetime")))
# 3. Enrich with zone master data
tripsq = tripsq.join(puzones, "PULocationID", "left")
                 .join(dozones, "DOLocationID", "left")
```

AWS Lambda Functions

Function	Purpose
day11auditlogger	Records pipeline execution to DynamoDB
day11dqgate	Executes Glue Data Quality rulesets
day11masterfreshness	Validates MDM data freshness before ETL
day11notifysteward	Sends SNS alerts on failures

Delta Lake Features

- Time Travel: Query historical data versions

- ACID Transactions: Consistent reads during writes

- Rollback/Restore: day15deltarollback_restore.py

---

4. Transformation Layer - SQL Scripts with Version Control

SQL Script Organization

```
sql/
+-- 00admin/           # Admin DDL
+-- 10transforms/      # Core transformations
+-- 20views/           # Analytical views
+-- 30quality/         # DQ test queries
+-- 40tests/           # Unit tests for SQL
+-- 50scd/             # SCD Type 2 logic
+-- 51_versioning/      # Version control metadata
+-- athena/             # Athena-specific SQL
+-- redshift/           # Redshift star schema
```

Script Header Standard (Version Control)

```
-- ============================================================
-- Script:      day4mdmschema.sql
-- Author:      data_engineer
-- Created:     2026-01-15
-- Modified:    2026-01-20
-- Version:     1.2.0
-- Domain:      MDM
-- Description: Master data dimension tables with audit columns
-- ============================================================
```

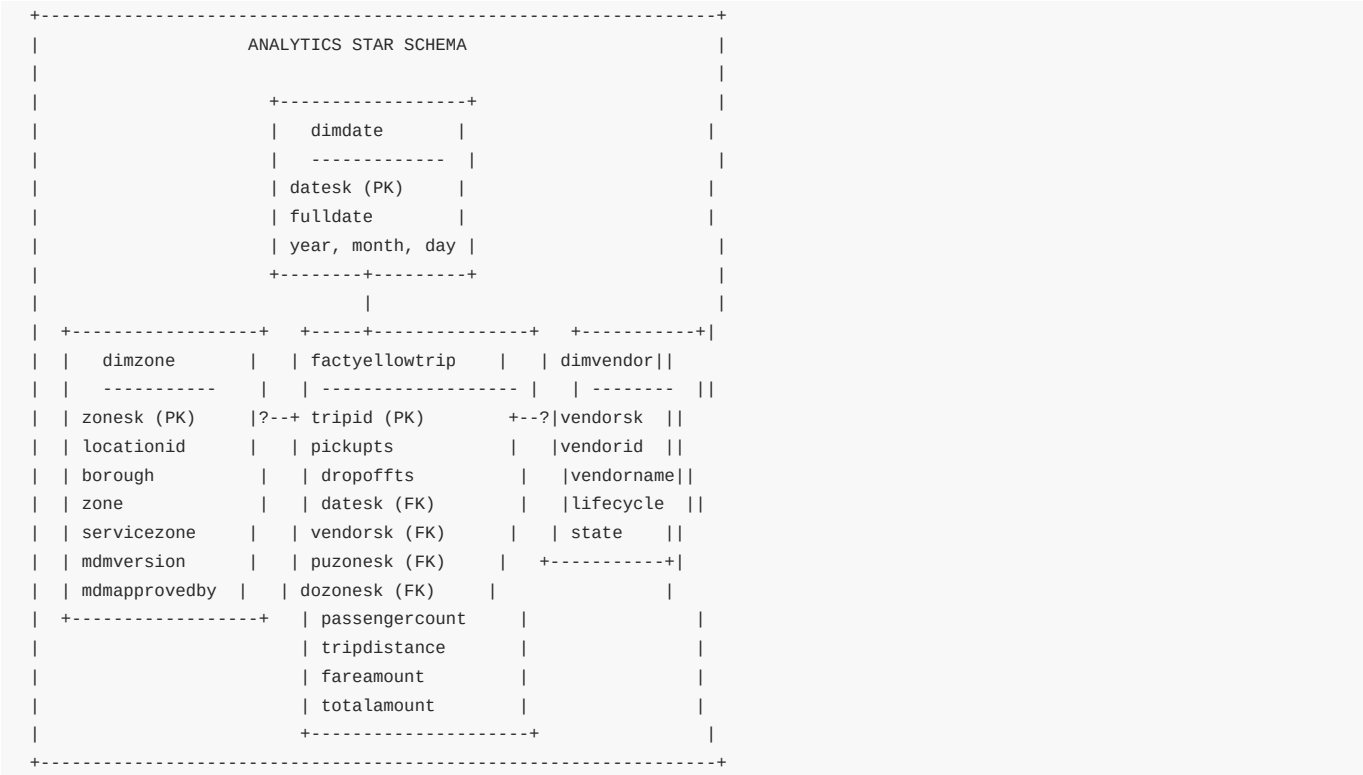
Key SQL Artifacts

Script	Purpose
day4mdmschema.sql	MDM dimension tables with versioning
510procapplyzonescd2.sql	SCD Type 2 stored procedure
day16starschema.sql	Redshift dimensional model
day17dashboardkpis_view.sql	Governance KPI views

---

5. Serving Layer - Redshift + Athena

Redshift Star Schema



Redshift Configuration

- Cluster: oobt-redshift (ra3.xlplus, single-node demo)

- Distribution: AUTO for fact, ALL for dimensions

- Sort Keys: pickup_ts on fact for time-series queries

- IAM Role: S3 COPY access for data loading

Athena Configuration

- Workgroup: oubt-athena-wg

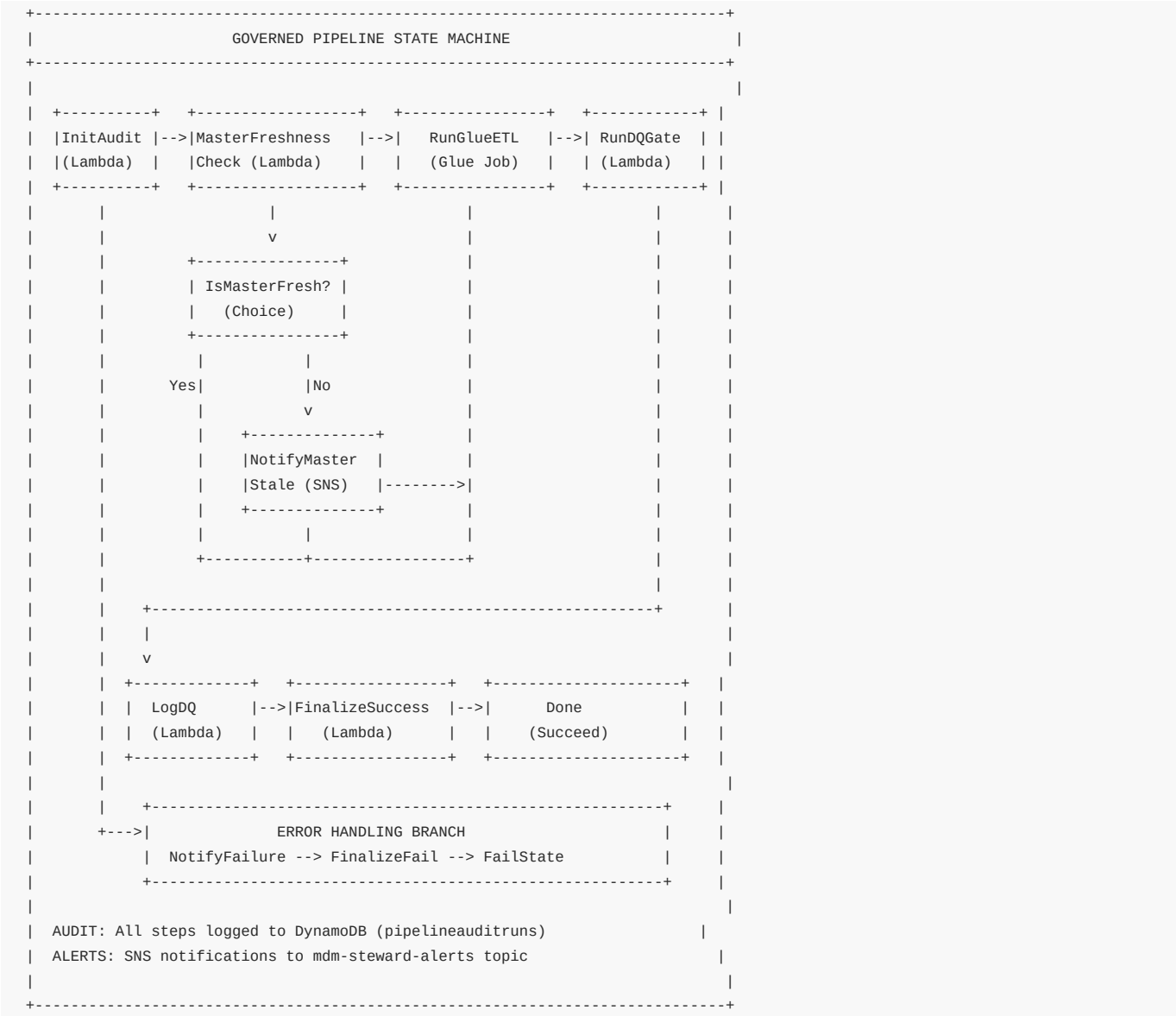
- Catalog: AWS Glue Data Catalog

- Use Cases: Ad-hoc queries, federated queries, certified views

---

6. Orchestration - AWS Step Functions

Pipeline State Machine



Step Functions Features

- Retry Logic: 3 retries with exponential backoff

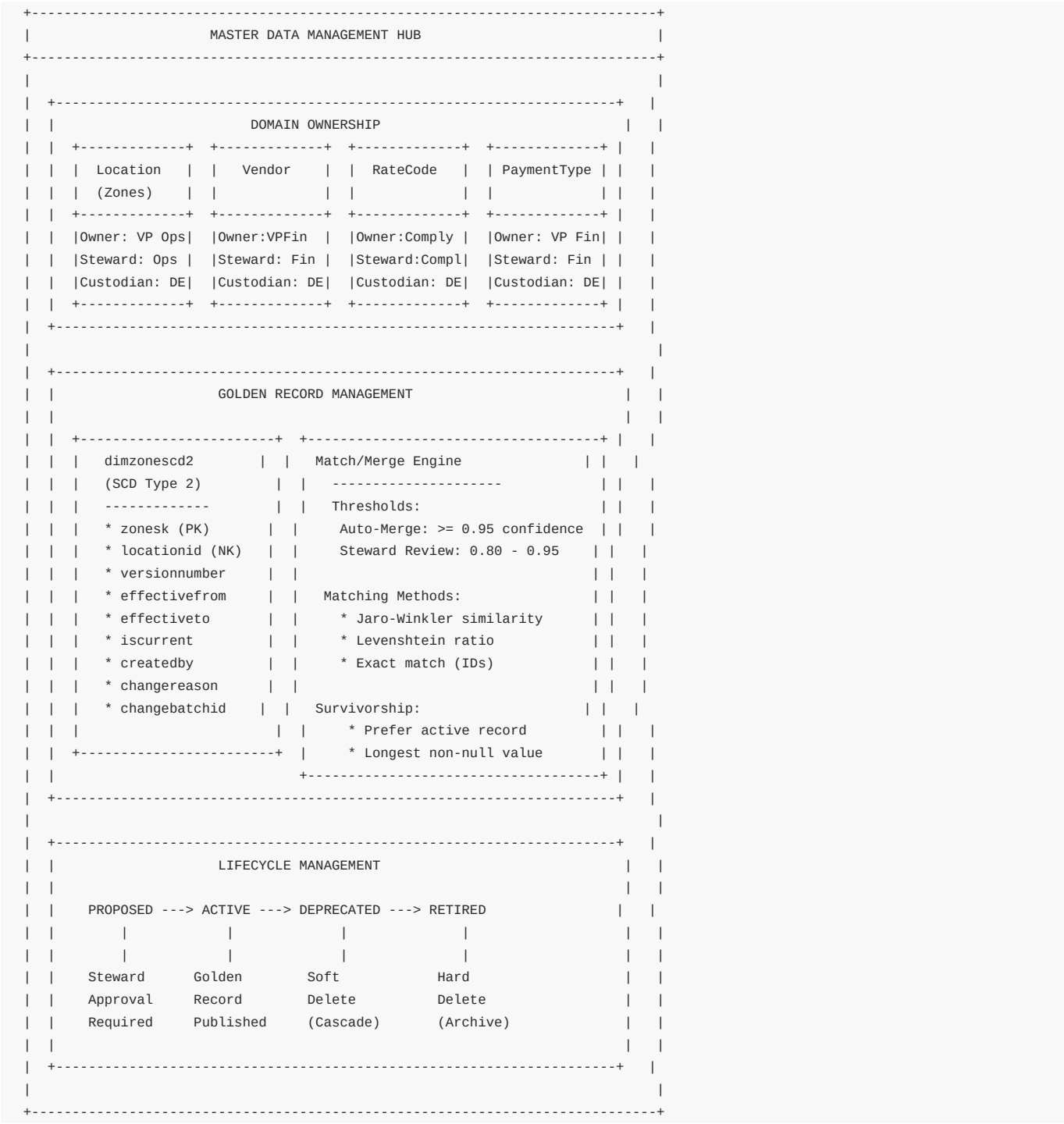
- Error Handling: Catch all errors, notify stewards

- Audit Trail: Every step logged to DynamoDB

- State Persistence: Execution history for debugging

7. Master Data Management (MDM) Layer

MDM Architecture



SCD Type 2 Implementation

```
-- Stored procedure: applyzonescd2fromstage
-- Key logic:
```

```
IF vchanged THEN
  -- Expire current row
  UPDATE mdm.dimzonescd2
  SET effectiveto = vnow, iscurrent = FALSE
  WHERE zonesk = vcurrent.zonesk;
  -- Insert new version
  INSERT INTO mdm.dimzonescd2(
    locationid, borough, zone, servicezone,
    versionnumber, effectivefrom, iscurrent,
    createdby, changereason, changebatchid
  ) VALUES (...);
  -- Audit trail
  INSERT INTO mdm.masterversionaudit(...);
END IF;
```

Match/Merge Rules (vendormatchrules.yaml)

```
thresholds:
  automerge: 0.95      # Automatic merge
  stewardreview: 0.80  # Human review required
fields:
  - name: vendorname
    weight: 0.85
    method: "stringsimilarity"
  - name: vendorid
    weight: 0.15
    method: "exact"
survivorship:
  automergestrategy: "preferactiverecord"
  fieldrules:
    vendorname: "longestnon_null"
```

---

8. Monitoring Layer - CloudWatch Dashboards

Observability Dashboard

```
+-------------------------------------------------------------------------+
|                   OUBT OBSERVABILITY DASHBOARD                          |
+-------------------------------------------------------------------------+
|  Domain: NYC_Taxi    Owner: Operations Team    Certified: Yes           |
+-----------------------------------+-------------------------------------+
|   STEP FUNCTIONS EXECUTIONS       |   GOVERNANCE KPIs (Custom Metrics)  |
|   --------------------------      |   -------------------------------   |
|   +-------------------------+     |   +-----------------------------+   |
|   | ??????????????????????? |     |   |                             |   |
|   | ??????????????????????? |     |   | Pipeline Success Rate: 98.5% |   |
|   | ??????????????????????? |     |   | Master Freshness: 2.3 hours  |   |
|   | ??????????????????????? |     |   | Orphan Rate PU: 0.02%        |   |
|   +-------------------------+     |   | Orphan Rate DO: 0.01%        |   |
|   # Started  # Succeeded          |   +-----------------------------+   |
|   # Failed   # TimedOut           |                                     |
+-----------------------------------+-------------------------------------+
|   DATA QUALITY SCORECARD                                                |
|   ----------------------                                                |
|   +-------------------------------------------------------------------+ |
```

```
|   | Dimension    | Metric                  | Threshold     | Status    |  |
|   +--------------+-------------------------+---------------+-----------+  |
|   | Completeness | Non-null timestamps     | >= 99.9%       | ? PASS   |  |
|   | Validity     | Distances >= 0           | 100%          | ? PASS   |  |
|   | Consistency  | Known zones (no orphan) | <= 0.1% orphan | ? PASS   |  |
|   | Timeliness   | Curated within 24h      | < 24 hours    | ? PASS   |  |
|   +-------------------------------------------------------------------+  |
+-------------------------------------------------------------------------------+
```

CloudWatch Alarms

Alarm	Condition	Action
Pipeline Failure	ExecutionsFailed > 0	SNS -> Steward email
DQ Gate Failure	DQ score < threshold	Block pipeline, notify
Master Stale	Freshness > 24h	Alert for review
Lambda Errors	Error count > 5/5min	PagerDuty integration

---

9. CI/CD Pipeline - CloudFormation

Deployment Architecture

```
+-------------------------------------------------------------------------------+
|                        CI/CD DEPLOYMENT PIPELINE                              |
+-------------------------------------------------------------------------------+
|                                                                               |
|  +----------+   +----------+   +----------+   +----------+   +----------+ |
|  |   Git    |-->| Terraform|-->|CloudForm |-->|  Deploy  |-->| Validate | |
|  |  Push    |   |  Plan    |   |  Stack   |   | Resources|   |  & Test  | |
|  +----------+   +----------+   +----------+   +----------+   +----------+ |
|       |              |              |              |              |        |
|       v              v              v              v              v        |
|  +----------+   +----------+   +----------+   +----------+   +----------+ |
|  | SQL      |   | Modules: |   | Change   |   | Lambda   |   | DQ Tests | |
|  | Scripts  |   | * S3     |   | Sets     |   | Glue     |   | SQL      | |
|  | Python   |   | * RDS    |   | Review   |   | Step Fn  |   | Tests    | |
|  | YAML     |   | * Lambda |   | Rollback |   | Redshift |   |          | |
|  +----------+   +----------+   +----------+   +----------+   +----------+ |
|                                                                               |
+-------------------------------------------------------------------------------+
```

Terraform Modules

```
infra/terraform/
+-- envs/
|   +-- dev.tf
|   +-- prod.tf
|   +-- variables.tfvars
+-- modules/
    +-- lambda/
    |   +-- main.tf
```

```
    |   +-- variables.tf
    +-- rds/
    |   +-- main.tf
    |   +-- outputs.tf
    |   +-- variables.tf
    +-- s3/
        +-- main.tf
        +-- outputs.tf
        +-- variables.tf
```

Infrastructure Components (Terraform)

Resource	Configuration
S3 Buckets	Versioning, lifecycle, encryption
RDS PostgreSQL	MDM hub, private subnet
Redshift	ra3.xlplus, S3 COPY role
Lambda Functions	Python 3.11, VPC access
Step Functions	Pipeline orchestration
IAM Roles	Least privilege policies

---

10. Live Demo Guide (20 Minutes)

Demo 1: Master Data Operations (5 min)

```
# 1. Show current MDM state
psql -c "SELECT * FROM mdm.dimzone WHERE locationid = 1;"
# 2. Stage an update
INSERT INTO mdm.stgzoneupdates (locationid, borough, zone, servicezone, changereason)
VALUES (1, 'Manhattan', 'Newark Airport Updated', 'EWR', 'Demo update');
# 3. Apply SCD2 procedure
CALL mdm.applyzonescd2fromstage('demouser', 'Live demo', 'Manual');
# 4. Show version history
SELECT zonesk, locationid, zone, versionnumber, iscurrent, effectivefrom
FROM mdm.dimzonescd2 WHERE locationid = 1 ORDER BY version_number;
```

Demo 2: Data Quality Monitoring (4 min)

```
# 1. Show Great Expectations suite
cat governance/greatexpectations/yellowtripssuite.json
# 2. Run DQ evaluation
python scripts/day8rundqevaluation.py
# 3. Show results in CloudWatch
aws cloudwatch get-metric-data --metric-name "DQRulesPassed" ...
# 4. Show quality scorecard
cat docs/quality_scorecard.md
```

Demo 3: Analytics Dashboard (3 min)

```
-- Connect to Redshift
-- 1. Show star schema
SELECT  FROM analytics.factyellowtrip LIMIT 10;
SELECT  FROM analytics.dimzone LIMIT 10;
-- 2. Run analytics query
SELECT
  dz.borough,
  COUNT(*) as trips,
  SUM(f.totalamount) as revenue
FROM analytics.factyellowtrip f
JOIN analytics.dimzone dz ON f.puzonesk = dz.zonesk
GROUP BY dz.borough
ORDER BY revenue DESC;
```

Demo 4: Batch ETL with Step Functions (4 min)

```
# 1. Start pipeline execution
python scripts/day11startexecutionmanual.py
# 2. Monitor in AWS Console (Step Functions)
# Show state machine graph, execution progress
# 3. Check DynamoDB audit table
aws dynamodb scan --table-name pipelineaudit_runs
# 4. Show SNS notification (if failure)
```

Demo 5: CI/CD Deployment (4 min)

```
# 1. Show Terraform plan
cd infra/terraform
terraform plan
# 2. Show module structure
ls -la modules/
# 3. Deploy change (Lambda update)
terraform apply -auto-approve
# 4. Verify deployment
aws lambda get-function --function-name day11auditlogger
```

Summary Table

Layer	Technology	Key Features
Ingestion	S3, boto3	Batch upload, Parquet/CSV
Storage	S3 Data Lake	5 zones (raw -> master), lif..
Processing	Glue, Lambda, Spark	ETL, DQ gates, Delta Lake
Transformation	SQL (versioned)	SCD2, quality tests
Serving	Redshift, Athena	Star schema, certified views
Orchestration	Step Functions	Governed pipeline, retry/catch
MDM	PostgreSQL	Dedup, match/merge, SCD2
Monitoring	CloudWatch	Dashboards, alarms, X-Ray
CI/CD	Terraform, CloudFormation	IaC, modular deployment

Appendix: Key File References

Category	File Path
Architecture Doc	docs/day20_architecture.md
MDM Architecture	docs/day19mdmarchitecture.md
Lake Zone Design	docs/day6datalakearchitecture.svg
Governance Charter	docs/governance_charter.md
Step Functions	stepfunctions/day11_pipeline.asl.json
Glue ETL	gluejobs/day7gluetaxicurated.py
MDM Schema	sql/day4mdmschema.sql
SCD2 Procedure	sql/50scd/510procapplyzone_scd2.sql
Redshift Schema	sql/redshift/day16starschema.sql
Match Rules	governance/mdm/vendormatchrules.yaml
DQ Suite	governance/greatexpectations/yellowtrips_su..
Dashboard	dashboards/day18/oobtobservabilitydashboard..
Terraform	infra/terraform/modules/

Document Version: 1.0.0

Last Updated: 2026-01-21

Author: Data Engineering Team