NYCPS TMS: Consolidated Prescriptive Data Governance, Management, Security & Compliance Strategy

I. Introduction: The Data Imperative

The NYCPS Transportation Management System (TMS) is fundamentally a data-driven system, handling vast quantities of highly sensitive information critical to student safety, operational efficiency, and regulatory compliance. This document establishes the **single, authoritative, and non-negotiable strategy** governing every aspect of data within the TMS project. It consolidates and details all requirements for data governance, management across the full lifecycle, security controls, privacy protection, compliance adherence, migration, archival, and destruction.

Given the involvement of Personally Identifiable Information (PII) pertaining to minors, Material Non-Public Information (MNPI), diverse data sources (cloud, on-premise, IoT, third-party), and the stringent legal landscape (FERPA, NY Education Law § 2-d, CIPA, HIPAA implications, NYC3/OTI/DIIT policies), **our approach embodies zero tolerance for non-compliance or security lapses.** Every process, technical control, and human action involving TMS data must strictly adhere to the principles and procedures mandated herein.

> **Failure to adhere meticulously to this strategy constitutes a critical project risk, potentially leading to regulatory penalties, legal action, loss of public trust, operational disruption, and project termination.**

II. Foundational Principles (Mandatory)

All data-related activities MUST adhere to these core principles:

1. **Compliance is Paramount:** Explicit adherence to FERPA, NY Ed Law 2-d, NYCPS A-820, NYC3/OTI/DIIT security policies, contractual data clauses, and all other applicable federal, state, and local laws is the absolute baseline.
2. **Security & Privacy by Design and Default:** Controls are embedded from the start; assume data is sensitive; least privilege is the default; security is everyone's responsibility.

3. **Data Minimization:** Only collect, process, and retain data absolutely essential for defined, legitimate TMS functions. Question the necessity of every data element.

4. **Purpose Limitation:** Use data solely for the specific purpose for which it was collected and authorized. Prohibit repurposing without formal review and approval.

5. **Controlled Access (Need-to-Know):** Implement strict Role-Based Access Control (RBAC) and attribute-based controls technically enforced at all layers. Access is granted only based on verified job function and legitimate need.

6. **Data Quality & Integrity:** Data must be accurate, complete, consistent, and timely. Implement validation and quality monitoring.

7. **Full Lifecycle Management:** Every data element has a defined, managed lifecycle from creation/ingestion to verifiable destruction.

8. **Accountability & Verifiable Auditability:** All access, modification, and deletion of sensitive data *must* be logged immutably and reviewed periodically. Actions must be traceable to specific users or systems.

9. **Transparency (Internal):** Data policies, classifications, standards, and procedures are clearly documented, accessible, and communicated to all relevant personnel via the TMS Data Governance Policy.

III. Formal Data Governance Framework

A robust governance structure ensures policies are defined, implemented, and enforced consistently.

A. Defined Data Governance Roles & Responsibilities

Implementation How-To:

1. Formally assign and document the following roles within the project organization chart and RACI matrix. These roles *must* be filled and actively engaged.
    - **Data Owner(s):** Senior NYCPS officials (Director level or above from OPT, potentially DIIT) designated as ultimately accountable for specific data domains (e.g., Student Transportation Records, Operational Route Data, Ridership Data, Vendor Performance Data).
      Responsibilities: Approve domain-specific policies (access, usage, quality, retention), resolve stewardship disputes,

provide final approval for high-risk data sharing or usage requests, champion data governance within their directorate.

- **Data Steward(s):** Designated NYCPS SMEs (e.g., Lead Router, OPT Operations Manager, Data Analyst Lead) responsible for the day-to-day management of specific data domains.
  Responsibilities: Define and document business terms/definitions (Data Dictionary), define data quality rules, define data access requirements based on business needs, review/approve access requests routed from technical teams, act as primary SME for data meaning/usage, monitor data quality metrics for their domain, identify data issues.

- **Data Custodian(s):** Technical teams (Vendor Dev/Ops/SRE, DIIT Infrastructure/DBAs where applicable) responsible for the secure technical implementation and operation of data systems. **Responsibilities:** Implement and operate databases, storage systems,

data pipelines; implement security controls (encryption, access controls based on IAM/RBAC policies defined via governance); manage backups/recovery; execute data archival/destruction procedures; monitor system health related to data stores. *Custodians do not define data policy or grant business access.*

- **NYCPS Chief Privacy Officer (CPO) / Designee:** Mandatory reviewer and approver for the TMS Data Governance Policy and any procedures impacting PII handling, student privacy rights, data sharing agreements, and incident response for privacy breaches. Ensures alignment with FERPA, NY Ed Law 2-d, etc.

- **NYCPS Chief Information Security Officer (CISO) / Designee:** Mandatory reviewer and approver for data security architecture, technical security controls (encryption, IAM, network segmentation related to data), vendor security assessments, security incident response

plans related to data breaches. Ensures alignment with NYC3/OTI/DIIT standards.

- **Legal Counsel (NYCPS OLS):** Provides legal interpretation of regulations, reviews contractual data clauses, advises on data sharing agreements, reviews data breach notifications.
- **Data Governance Lead (Project Role - e.g., Lead BA or dedicated PM function):** Facilitates the overall data governance process for the project. Coordinates between Owners, Stewards, Custodians, CPO, CISO. Manages the Data Governance Policy document. Tracks data-related risks and issues. Organizes Data Steward meetings. Reports on data governance status.

2. Document detailed responsibilities and decision rights matrix in the TMS Data Governance Policy.

Responsibility: Project Leadership, NYCPS Directorate Leadership (to assign Owners/Stewards), PMO.

B. Formal TMS Data Governance Policy (The Controlling Document)

Implementation How-To:

1. Develop the dedicated **TMS Data Governance Policy** document as the *absolute source of truth* for all data handling within the project. This is a mandatory, non-negotiable deliverable requiring formal sign-off before production data processing begins.
2. Store centrally in Confluence, under strict version control, with access controlled but viewable by all project members.
3. **Mandatory Sections (Exhaustive Detail Required):**
    a. **Introduction:** Purpose, Scope (all TMS data, all lifecycle stages), Mandatory Compliance Statement, Link to Guiding Principles.
    b. **Governance Structure:** Detailed Roles & Responsibilities (as defined above), Data Domains mapped to Owners/Stewards, Meeting Cadences (e.g., Quarterly Data Steward Forum), Decision Rights Matrix, Escalation Paths for data-related issues.

c. **Data Classification Standard:** Reference and link to official NYCPS standard. Provide a **TMS Data Inventory & Classification Matrix** (as an appendix or linked page) explicitly listing key TMS data elements (database fields, API attributes, report fields) and their assigned classification (Highly Restricted, Confidential, Private, Public). This matrix must be maintained throughout the project.

d. **Data Quality Framework:** Overall approach, roles (Stewards define rules). Define standard Data Quality Dimensions (e.g., Accuracy, Completeness, Consistency, Timeliness, Validity). Detail the process for defining, implementing, and monitoring quality rules per domain. Define data quality issue reporting and resolution process.

e. **Data Security Policy:** Reference mandatory adherence to NYCPS security policies (NYC3/OTI/DIIT) and the project's specific security

architecture. Mandate specific controls based on data classification (e.g., encryption types/levels for Highly Restricted vs. Confidential, specific IAM controls).

f. **Data Access Control Policy:** Define the principle of least privilege. Detail the *formal process* for requesting, reviewing (by Data Steward/Owner), approving (by Data Owner for sensitive data), implementing (by Data Custodian), periodically recertifying (e.g., quarterly/annually by manager/Steward), and revoking data access for individuals and systems based on defined roles (RBAC). Maintain an access control matrix/log.

g. **Data Usage Policy:** Explicitly define *approved* uses for TMS data based on project objectives. Strictly *prohibit* unauthorized uses, including commercial exploitation, marketing, sale, or sharing beyond approved integrations/agreements.

Outline consequences for policy violation.

h. **Data Privacy Policy:** Explicitly detail procedures for ensuring compliance with FERPA (Parental access/amendment rights handled via DOE process), NY Ed Law 2-d (including Parent Bill of Rights integration, restrictions on disclosure, encryption requirements), CIPA/COPPA considerations for student-facing interfaces, and HIPAA considerations for health-related data. Define process for handling privacy inquiries or complaints.

i. **Data Sharing Policy (Internal & External):** Define strict criteria and approval workflow for any data sharing. Internal sharing between TMS components must follow least privilege and secure protocols. External sharing requires formal Data Sharing Agreement (DSA) reviewed by Legal/CPO/CISO/Data Owner, specifying purpose, exact data

elements, security controls, usage limits, and destruction requirements. Maintain register of all DSAs.

j. **Data Masking/Anonymization Standard:** Define approved techniques (e.g., hashing+salting for identifiers, randomization, generalization, k-anonymity principles) and processes for creating non-production datasets. Mandate verification of effectiveness. Prohibit re-identification attempts.

k. **Data Lifecycle Management Policy:** Reference the mandatory 7-year retention period from RFP. Detail procedures for archival and destruction (linking to the formal procedure document - see Section VI).

l. **Auditing & Monitoring Policy:** Mandate logging requirements for data access/modification. Define frequency and responsibility for reviewing audit logs and data quality metrics.

m. **Compliance & Incident Reporting:** Define process for reporting potential data governance/privacy/security policy violations or data breaches internally and coordination with official DOE incident response procedures.

n. **Policy Exception Process:** Define a *highly restrictive* process for requesting exceptions, requiring strong justification, risk assessment, mitigation plan, and documented approval from Data Owner, CPO, CISO, and potentially Legal/Steering Committee. Exceptions must be time-bound and regularly reviewed.

o. **Training & Awareness:** Mandate initial and annual refresher training on this policy and relevant regulations (FERPA, 2-d) for all personnel with access to TMS data. Track training completion.

p. **Policy Enforcement & Review:** State consequences for non-compliance. Mandate annual review

and update cycle for the policy itself, approved by governance.

4. Obtain formal, documented sign-off on the policy and its appendices (especially Data Classification Matrix) from all designated approvers (Data Owners, CPO, CISO, Project Sponsor, Legal) via the established governance process.

Responsibility: Data Governance Lead/PM (Facilitation/Drafting), Data Owners/Stewards, CPO, CISO, Legal (Content/Approval), Steering Committee (Final Approval).

This policy is the cornerstone of data management; its development, approval, and enforcement are critical path items.

IV. Prescriptive Data Lifecycle Management Controls

We will implement specific, verifiable controls at each data lifecycle stage.

A. Stage: Data Identification & Classification

Detailed Implementation:

1. **Data Inventory Process:** During design of any new feature/service, developers/BAs *must* identify all data elements created, read, updated, or deleted (CRUD).
2. **Classification Mandate:** Each identified element *must* be assigned a classification (Highly Restricted, Confidential, Private, Public) according to the **TMS Data Governance Policy** and NYCPS standards. Use the **TMS Data Inventory & Classification Matrix** (living document in Confluence, linked from Policy) as the authoritative source. Updates require Data Steward review/approval.
3. **Tooling:** While tools like AWS Macie may *assist* in discovery on S3, the authoritative classification is the manually curated and approved matrix/dictionary. Macie alerts for potential misclassified PII found in unexpected locations will trigger investigation.
4. **Documentation:** Classification *must* be documented in the Data Dictionary, API Specifications (e.g., using custom OpenAPI extensions or descriptions), and database schema comments.

Responsibility: BAs/Developers (Initial Identification), Data Stewards (Classification Approval/Matrix Update), Security Team (Tool Config/Oversight).

Accurate, documented classification is mandatory for applying correct downstream controls.

B. Stage: Data Collection & Ingestion

Detailed Implementation:

1. **Secure Transport Protocols:** Mandate TLS 1.2+ with strong cipher suites for all API endpoints (API Gateway custom domains with ACM certs), IoT Core endpoints (MQTTS), SFTP servers (AWS Transfer Family), and database connections. Configure services to reject insecure protocols/ciphers.
2. **Rigorous Input Validation (Server-Side):**
    - API Gateway: Use Request Validation features based on OpenAPI schema.
    - Lambda/Application Code: Implement strict validation for *all* parameters, request bodies, headers using libraries (e.g., Pydantic for FastAPI, class-validator for NestJS, standard framework validation). Validate type, format (regex), length,

range, allowed values (enum). Use allow-lists. Sanitize inputs intended for downstream interpreters (SQL, OS, XML).

- Log validation failures clearly (without logging sensitive input data). Return specific 4xx error codes.

3. **Data Minimization Enforcement:** Code reviews *must* verify that only necessary data fields defined in approved API contracts/data models are being accepted and processed. Reject requests with extraneous data.

4. **Source Authentication/Authorization:**
    - External APIs: Use API Gateway API Keys/Usage Plans or Lambda Authorizers (validating JWTs or custom tokens).
    - IoT Devices: Use X.509 certificates managed via IoT Core registry and policies.
    - User Inputs (Apps): Rely on user authentication tokens (JWT validated by API Gateway/backend).

- SFTP: Use SSH key authentication managed by AWS Transfer Family.

5. **Reliable Ingestion Architecture:**
    - Use highly available services (API GW, IoT Core, Kinesis, MSK, SQS, Lambda, Fargate).
    - Configure Dead-Letter Queues (DLQs) for SQS queues and Lambda asynchronous invocations to capture processing failures. Set up CloudWatch Alarms on DLQ message counts.
    - Implement appropriate retry logic within processing Lambdas/services for transient failures talking to downstream systems.
    - Monitor Kinesis/MSK `IteratorAge` or Lag metrics to detect processing delays.

Responsibility: Developers (Code Implementation), DevOps (Service Config/IaC), Security (Review/Auth Config).

C. Stage: Data Processing & Transformation

Detailed Implementation:

1. **Secure Compute Environment:** All processing *must* occur in private VPC subnets using approved compute services (Lambda, Fargate/ECS). IAM Roles attached to compute resources *must* follow least privilege, granting only necessary permissions to specific resources (e.g., read from specific SQS queue, write to specific DynamoDB table).
2. **Data Validation Checks:** Re-validate critical data elements after transformations or enrichments, especially before writing to persistent stores or sharing externally. Use defined data quality rules.
3. **Robust Error Handling:** Implement comprehensive try/catch blocks and error handling logic. Log errors with correlation IDs and context (no PII). Failed processing steps should trigger alerts and route problematic data/messages to DLQs or error repositories for manual review, ensuring no data loss for critical streams.
4. **ETL/ELT Security & Quality (AWS Glue):**
    - Run Glue jobs using IAM roles with least-privilege access to

source/target data stores (S3, RDS).

- Integrate data quality checks using Glue Data Quality rules based on Data Steward definitions. Log quality violations.
- Parameterize Glue jobs and manage code in Git, deploying via CI/CD.
- Encrypt data processed and stored by Glue jobs (using Security Configurations).

5. **Data Masking/Anonymization Implementation (Mandatory for Non-Prod):**
    - Implement as part of the ETL pipeline (Glue job, Lambda script) that populates non-prod environments (QA, Staging).
    - Use approved techniques from Data Governance Policy consistently (e.g., SHA-256+Salt for identifiers, Faker for names/addresses, k-anonymization logic).
    - **Verification:** Include steps in the ETL/masking process and QA testing

to *verify* that PII/Highly Restricted data has been effectively removed/masked and cannot be easily reversed in non-prod environments. Document verification results.

6. **Audit Logging:** Log start/end/success/failure of major processing jobs (ETL, batch routing). Log significant data transformations if required for auditability.

Responsibility: Developers/Data Engineers (Code/ETL), DevOps (Compute/IAM), Data Stewards (Quality Rules), Security (Verification of Masking).

Failure to properly mask/anonymize data in non-production environments is a major compliance violation and security risk. Verification is mandatory.

D. Stage: Data Storage & Security (At Rest & In Transit)

Detailed Implementation:

1. **Encryption at Rest (Mandatory Configuration):**

- Configure *all* S3 buckets storing project data (logs, archives, intermediate files) with SSE-S3 or SSE-KMS (using dedicated, environment-specific CMKs for Highly Restricted/Confidential data) via Terraform/IaC. **Enforce encryption via bucket policies.**
- Enable encryption by default for *all* EBS volumes in the GovCloud region. Use specific KMS CMKs for volumes handling Highly Restricted data. Configure via Launch Templates/Terraform.
- Enable encryption at rest using KMS (dedicated CMKs preferred for sensitive DBs) for *all* RDS instances and automated/manual snapshots. Configure via Terraform.
- Enable encryption at rest using KMS (dedicated CMKs preferred) for *all* DynamoDB tables storing sensitive data. Configure via Terraform.

- Enable Server-Side Encryption (SSE) using KMS (AWS-managed `alias/aws/sqs` or dedicated CMKs) for *all* SQS queues handling sensitive data. Configure via Terraform.
- Enable SSE using KMS for *all* SNS topics transmitting sensitive data. Configure via Terraform.
- Implement strict KMS Key Policies granting key usage permissions only to necessary IAM roles/users (least privilege). Enable key rotation for CMKs.

2. **Encryption in Transit (Mandatory Configuration):**
   - Configure *all* Application Load Balancers (ALBs) and API Gateways with HTTPS listeners using ACM certificates and secure TLS policies (e.g., `ELBSecurityPolicy-TLS-1-2-Ext-2018-06` or newer). Redirect HTTP to HTTPS.
   - Ensure all application code and AWS SDK clients connect to internal

services (RDS, ElastiCache, other APIs) using TLS/SSL endpoints. Verify certificate validation is enabled.

- Configure AWS Transfer Family SFTP servers to use secure protocols only.

3. **Network Access Controls (Mandatory Configuration via IaC):**
    - Place all databases (RDS), caches (ElastiCache), and backend processing compute (Fargate/Lambda accessing VPC) in **private subnets**.
    - Implement granular Security Groups:
        - DB Security Group: Allows ingress *only* on the database port (e.g., 5432 for Postgres) *only* from the specific Security Group(s) of the application tier microservices that need access. Deny all

other ingress. Allow all egress (or restrict to specific needed endpoints if feasible).

- App Tier Security Group: Allows ingress *only* on the application port(s) *only* from the Security Group of the ALB or API Gateway (or other internal services that call it). Allows egress to necessary dependencies (DB SG, Cache SG, SQS/SNS VPC Endpoints, external APIs via NAT GW).

- Load Balancer Security Group: Allows ingress *only* on HTTPS

(443) from the internet (or specific source IPs if restricted). Allows egress only to the App Tier Security Group on the application port.

- Implement restrictive Network ACLs as a secondary defense layer (stateless filtering), although primary reliance is on Security Groups.
- Utilize VPC Endpoints (Gateway for S3/DynamoDB, Interface for KMS, Secrets Manager, SQS, SNS, ECR, etc.) to keep traffic to AWS services within the AWS network, accessed from private subnets.

4. **Backup Security:** Ensure all backups (RDS snapshots, DynamoDB backups, S3 versions) are encrypted using the same KMS keys as the source data. Restrict access to backups via IAM policies.

Responsibility: DevOps/Cloud Team (IaC Implementation), Security Team (Policy Definition/Review), DBA (DB Encryption Config).

Misconfigured Security Groups or lack of encryption are primary causes of data breaches. Rigorous implementation and auditing via IaC and AWS Config are essential.

E. Stage: Data Access & Use Control (Least Privilege Enforcement)

Detailed Implementation:

1. **Role-Based Access Control (RBAC) Implementation:**
    - Define application-specific roles (e.g., `TMS_OPT_ROUTER`, `TMS_PARENT`, `TMS_SCHOOL_ADMIN`, `TMS_L1_SUPPORT`) within the application or a central authorization service.
    - Map authenticated users (from SAML/Cognito/DB) to these roles.
    - Backend APIs and microservices *must* check the user's role (e.g., from JWT claims or session context)

before allowing access to specific functions/endpoints. Implement using decorators, middleware, or framework security features.

- Frontend UIs should conditionally render components/actions based on the user's role to provide a clear UX, but *server-side authorization is mandatory* as the definitive control.

2. **Attribute-Based Access Control (ABAC) / Scope Limitation:**
    - Implement fine-grained checks *in addition* to RBAC where necessary. E.g., A `TMS_SCHOOL_ADMIN` can only access student/route data associated with their specific `school_id`. A `TMS_PARENT` can only access data linked to their authorized `student_id`(s).
    - Authorization logic within backend services *must* incorporate these scope checks, typically by filtering database queries or checking resource attributes based on the

user's context (their school ID, their linked student IDs).

3. **IAM for System Access (Least Privilege):**
    - IAM Roles used by Lambda, ECS/Fargate, EC2 *must* have policies granting *only* the specific permissions needed (e.g., `dynamodb:GetItem` on `table/SpecificTable`, `s3:GetObject` on `bucket/SpecificBucket/prefix/*`, `kms:Decrypt` on a specific key). **Avoid `*:*` permissions.**
    - Use IAM Condition Keys where possible for further restriction (e.g., restrict access based on source IP, VPC endpoint, tags).
4. **Data Masking Implementation:** Implement server-side masking logic within APIs/services serving roles that need partial data visibility (e.g., L1 Support seeing `Student ID: ******1234`, `Address: REDACTED`). Masking rules defined by Data Stewards.
5. **Auditing Access (Mandatory):**

- Configure applications to log *every* successful and *failed* attempt to access or modify Highly Restricted or Confidential data. Log must include: Timestamp, User ID, Source IP, Role Used, Data Resource Accessed (e.g., Student ID, Route ID), Action Performed (Read/Update/Delete), Success/Failure Status.
- Enable CloudTrail data events for critical S3 buckets containing sensitive data.
- Enable RDS audit logging if detailed database access tracking is required by compliance.
- Forward relevant audit logs to a secure, central logging system (CloudWatch Logs or SIEM) with long-term retention and restricted access.
- Implement CloudWatch Alarms or SIEM rules to detect anomalous access patterns (e.g., excessive access by one user, access outside

business hours, attempts to access unauthorized data scope).

6. **Access Recertification:** Implement a mandatory quarterly or semi-annual process where managers or Data Stewards review and re-approve access permissions for users and system roles under their purview. Document certifications. Automate reporting of current permissions for review.

Responsibility: Developers (App RBAC/ABAC/Masking), Security Team (IAM Policies, Audit Config/Review), Data Stewards/Owners (Access Rules/Approval), Compliance Officer (Recertification Process).

F. Stage: Data Sharing & Integration Security

Detailed Implementation:

1. **Internal Microservice Security:** Enforce mTLS or use JWT-based service-to-service authentication for internal API calls within the cluster/VPC. Use IAM roles for AWS service interactions (SQS, SNS, S3, etc.).

2. **NYCPS System Integration Security:**
    - Use secure protocols (HTTPS APIs with robust auth, SFTP).

- Strictly validate data exchanged against defined schemas/formats.
- Grant least privilege access via API keys (rotated, stored in Secrets Manager) or specific IAM roles if using AWS native integration (e.g., Glue accessing S3).
- Log all data transfer events (success/failure, records transferred).

3. **Third-Party Sharing Controls:**
    - **Mandatory DSA:** No external sharing of PII/Confidential data without a fully executed Data Sharing Agreement reviewed by Legal/CPO/CISO/Data Owner.
    - **Technical Enforcement:** Implement sharing mechanism (secure API, SFTP) with strict authentication, authorization, and logging based on DSA terms. Transfer only specified data elements.
    - **Auditing:** Regularly audit third-party access logs and compliance

with DSA terms.

Responsibility: Developers/Data Engineers (Implementation), Architect (Design), Security Team (Review/Auth Config), Legal/CPO/Data Owner (DSAs).

G. Stage: Data Quality Management Implementation

Detailed Implementation:

1. **Rule Definition:** Data Stewards document specific Data Quality rules (using SMART criteria) in the Data Dictionary (Confluence) for CDEs within their domain.
2. **Rule Implementation:** Developers/Data Engineers implement these rules as:
    - Input validation constraints (API schemas, database constraints).
    - Checks within ETL/processing jobs (AWS Glue Data Quality rules, custom assertions in scripts).
    - Scheduled validation queries/scripts (Lambda running SQL against RDS, Athena against S3 data lake)

checking for consistency, completeness, validity across datasets.

3. **Monitoring & Reporting:**
    - Configure validation jobs/scripts to output quality metrics (e.g., % valid addresses, % complete records, count of duplicates) to CloudWatch Metrics or a dedicated quality database.
    - Create Data Quality Dashboards (QuickSight/Grafana) visualizing these metrics and trends over time, filterable by data domain/source.
    - Set CloudWatch Alarms on critical data quality metric thresholds (e.g., validation error rate > X%) to alert Data Stewards/Support.
4. **Issue Resolution Workflow:** Use Jira/ADO with a dedicated "Data Quality Issue" type. Automated checks or manual identification create tickets. Data Stewards triage and assign to relevant teams (e.g., source system team, ETL team, application team) for correction. Track resolution progress and verify fixes.

Responsibility: Data Stewards (Rules), Developers/Data Engineers (Implementation), QA (Testing Checks), Data Governance Lead (Monitoring Process).

H. Stage: Data Archival & Retention Enforcement

Detailed Implementation:

1. **Policy Implementation:** Configure S3 Lifecycle Policies via Terraform for all relevant buckets to automatically transition data based on age (e.g., -> Glacier -> Deep Archive) and expire data after 7 years (2557 days). **Mandatory review and testing of lifecycle policies in non-prod before applying to production.**
2. **Database Archival:** Implement and schedule thoroughly tested Lambda/Glue jobs to export historical data (> active window) from RDS/DynamoDB to S3 archive buckets (partitioned by date/type, in Parquet format) before database deletion (if applicable). Ensure export includes retention calculation metadata.
3. **Audit Log Archival:** Ensure CloudTrail and critical application audit logs are configured for delivery to

dedicated S3 buckets with lifecycle policies ensuring 7-year retention (potentially in Glacier/Deep Archive).

4. **Retrieval Testing:** Periodically (e.g., annually) test the retrieval process for data from Glacier/Deep Archive for key datasets to ensure feasibility and validate expected timelines/costs. Document test results.

Responsibility: DevOps/Cloud Team (S3 Policies), Data Engineers (DB Archival Scripts), Compliance Officer (Policy Validation).

Failure to implement correct lifecycle policies can lead to premature data loss or non-compliant retention. Rigorous testing is essential.

I. Stage: Secure Data Destruction & Disposal Verification

Detailed Implementation:

1. **Primary Method (Automation):** Rely on tested and verified S3 Lifecycle Policy expiration actions for automated, permanent deletion after 7 years.

2. **Verification Logging:** Configure S3 Server Access Logging and potentially CloudTrail data events (for

critical buckets) to capture deletion events initiated by lifecycle policies or direct API calls.

3. **Destruction Audit:** Conduct annual internal audits:
    - Review S3 inventory reports/metrics to confirm objects older than 7 years are no longer present (consider object versions/delete markers).
    - Sample CloudTrail/S3 access logs to verify deletion actions.
    - Query active databases to confirm records successfully exported/archived previously are now deleted (if applicable).
4. **Manual Deletion Control:** Prohibit direct deletion capabilities in production except via highly restricted, audited break-glass procedures requiring multiple approvals (including Compliance/Legal) and specific justification tied to legal/privacy obligations.
5. **Formal Certification:** Generate and securely store annual **Certificates of Data Destruction** (as per template/process defined in Risk Plan Section VI), signed by Compliance Officer/Data Custodian Lead, based on automated logs and audit verification.

Responsibility: DevOps/Cloud Team (Lifecycle Policy), Security/Compliance Officer (Audit, Certification), Internal Audit.

Maintaining verifiable proof of destruction via logs and formal certification is mandatory for audit and compliance.

VI. Prescriptive Data Migration Strategy (Legacy Systems)

Migrating data from legacy systems like MapInfo/FoxPro and Edulog extracts requires extreme care due to potential data quality issues, format differences, and the need to avoid disrupting cutover. This process will be managed as a distinct sub-project with dedicated testing and validation.

A. Planning, Analysis & Design

Implementation How-To:

1. **Scope Freeze & Data Identification:** Absolutely finalize the *minimum essential* legacy data

required for Day 1 operation of TMS (e.g., currently active routes/stops/student assignments, critical historical data needed for specific functions identified by OPT SMEs). **Resist migrating non-essential historical data unless explicitly required and justified. Document scope in Confluence.**

2. **Deep Source Analysis:** Perform detailed profiling of legacy source data (MapInfo/FoxPro DBFs, Edulog files/DB extracts). Identify data types, formats, constraints, relationships, NULL values, inconsistencies, known quality issues. Document thoroughly.**

3. **Target Schema Mapping & Transformation Logic:** Create detailed source-to-target mapping documents (spreadsheet/Confluence) for every required field. Define *explicit* transformation rules (e.g., data type conversion `VARCHAR`->`INTEGER`, code translation `LegacyCode`->`NewCode`, address standardization/validation using GIS, default value handling, data cleansing logic for known issues). Obtain Data Steward sign-off on mapping and rules.**

4. **Tool Selection & Proof of Concept (POC):**
    - **Select primary tool (AWS Glue strongly preferred for ETL, Data Quality integration, scalability; Python/Lambda for simpler tasks).**

- **Conduct small-scale POCs with representative sample data to validate tool capabilities, transformation logic feasibility, and performance characteristics *before* full development.**

5. **Environment Setup:** Provision dedicated resources (S3 buckets for extracts, Glue crawlers/jobs/connections, Lambda functions, IAM roles with least privilege) in non-prod AWS environments specifically for migration development and testing. Ensure secure connectivity to legacy data sources or secure mechanisms for receiving extracts.

6. **Detailed Migration Test Plan:** Create a comprehensive test plan covering:
    - **Pre-migration source data validation (counts, basic checks).**
    - **Migration script execution validation (logs, error handling).**
    - **Post-migration target data validation (record counts source vs. target, schema validation, referential integrity checks, spot checks on critical fields, validation**

of transformation logic via sampling, data quality rule checks on migrated data).

- Performance testing results.
- Rollback test results.

Responsibility: Data Architect, Data Engineers, ETL Developers, OPT SMEs (Source knowledge/Validation), DBA, QA Lead (Test Plan).

Data migration is high-risk. Inaccurate scope definition, poor source analysis, incorrect transformations, or inadequate testing can lead to critical go-live failures.

B. Migration Development & Testing Cycles

Implementation How-To:

1. Develop migration scripts/jobs (Glue/Python) iteratively, following project coding standards (version control in GitLab, code reviews).
2. Unit test individual transformation functions/logic components within the scripts.
3. Execute test migrations against *anonymized* or synthetic legacy data extracts in the DEV environment frequently. Validate results using

automated checks where possible (e.g., SQL queries comparing source/target counts, Python scripts checking transformations).

4. Conduct full test migrations in the dedicated QA environment using the Migration Test Plan. QA team formally executes validation procedures and logs defects in Jira.
5. Iteratively fix bugs found in scripts or transformation logic. Re-run tests until QA validation passes according to predefined quality criteria.
6. Execute performance tests using anonymized, production-volume data to finalize runtime estimates and identify scaling needs for the production cutover. Optimize scripts/jobs as needed.

Responsibility: Data Engineers, ETL Developers, QA Team, DBA (Performance Tuning).

Quality Gate: QA sign-off on migration scripts and validation results in QA environment, confirming scripts run reliably and migrated data meets accuracy/completeness/quality targets defined in test plan. Performance test results meet cutover window constraints.

C. Production Cutover Execution (Meticulously Planned)

Implementation How-To:

1. **Finalize Cutover Strategy:** Based on testing and dependencies, finalize the approach (Big Bang preferred for simplicity if feasible within downtime window, otherwise Phased with clear data sync strategy). Obtain formal approval from Steering Committee/Project Sponsors.
2. **Create Hyper-Detailed Cutover Runbook:** Document every step, including:
    - Precise timings and durations for each step.
    - Responsible individuals/teams for each action.
    - Pre-cutover activities (communication freeze, legacy system quiesce, final legacy backup, environment checks).
    - Migration script execution commands/procedures.
    - Mandatory validation checkpoints (specific queries/counts to run at key stages).

- **Go/No-Go decision points based on validation results.**
- **Detailed Rollback Plan (see below).**
- **Post-migration steps (enable TMS access, final validation, communications).**
- **War room setup and communication plan *during* cutover.**

3. **Conduct Full Dress Rehearsal:** Execute the *entire* Cutover Runbook (including rollback steps where safe) in the Staging/UAT environment using a recent, full (anonymized) data set. Time all steps accurately. Identify and resolve any issues in the runbook or scripts. Obtain formal UAT sign-off on the rehearsal outcome.
4. **Schedule Production Cutover Window:** Negotiate and communicate a clear downtime window with all stakeholders (including OPT operations, schools if impacted).
5. **Execute Production Cutover:** Follow the rehearsed Runbook *precisely*. Maintain constant communication in the war room. Perform all validation checks at defined points. Make Go/No-Go decisions based on validation results and

timelines. Escalate issues immediately per defined incident process.

6. **Post-Cutover Hypercare:** Provide heightened monitoring and support immediately following go-live to quickly address any migration-related issues.

Responsibility: Cutover Manager (PM/Ops Lead), Data Engineers, DBAs, SRE/Ops, QA, OPT SMEs.

> ***Production data migration cutover requires executive awareness and a highly controlled execution window.***

D. Migration Rollback Plan (Mandatory)

Implementation How-To:

1. Develop a detailed, step-by-step Rollback Plan as an integral part of the Cutover Runbook.
2. Plan *must* include:
    - Clear trigger criteria for initiating rollback (e.g., validation check failure threshold, exceeding time window for critical step).

- **Decision authority for invoking rollback (e.g., Cutover Manager with PM/Sponsor concurrence).**
- **Steps to safely stop/undo migration activities in progress.**
- **Procedures to restore legacy systems from pre-cutover backups.**
- **Steps to revert any supporting configuration changes (DNS, firewalls, etc.).**
- **Communication plan for rollback status.**

3. **Test critical elements of the rollback plan during the dress rehearsal (e.g., restoring legacy DB backup to a test instance).**

Responsibility: Data Architect/Engineer, DBA, Ops/SRE, Cutover Manager.

An untested or inadequate rollback plan significantly increases cutover risk.

IX. Conclusion: Commitment to Data Integrity, Security & Compliance

This Consolidated Data Governance, Management, Security, and Compliance Strategy establishes the non-negotiable framework for all data handling within the NYCPS TMS project. Through the meticulous implementation and enforcement of these detailed policies, procedures, technical controls (leveraging AWS GovCloud capabilities), automated checks, rigorous testing, continuous monitoring, and clear governance structures, we will ensure the absolute confidentiality, integrity, availability, quality, and regulatory compliance of sensitive student and operational data. There is zero tolerance for deviation. This comprehensive, "water-tight" approach is fundamental to mitigating risks, meeting legal and ethical obligations, maintaining public trust, and delivering a secure, reliable, and successful Transportation Management System that serves the NYCPS community effectively and responsibly.