NYCPS TMS: Prescriptive Data Engineering, Analytics & AI/ML Strategy

I. Introduction: Data as a Strategic Asset

This document establishes the comprehensive, best-in-class strategy for managing the entire data and AI/ML lifecycle within the NYCPS Transportation Management System (TMS) project. Recognizing that data is not just a byproduct but a core strategic asset for optimizing operations, ensuring safety, providing transparency, and driving continuous improvement, this strategy mandates rigorous, modern practices across Data Engineering (DE), Data Quality (DQ), Data Performance, Reporting, Visualization, Analytics, Data Science (DS), Machine Learning (ML), and Artificial Intelligence (AI).

Given the project's complexity, the sensitivity of PII/MNPI data, the diverse data ecosystem (streaming IoT, batch feeds, on-premise systems, 3rd party data, cloud services), and the absolute criticality of compliance and reliability, this strategy integrates DataOps, MLOps, and elements of AIOps principles within the overarching DevSecOps framework. It details the "what" and the "how" for building and operating a robust, secure, compliant, and high-performance data platform on AWS GovCloud using GitLab for CI/CD.

Adherence to this strategy, alongside the overarching Data Governance & Management Strategy, is mandatory to mitigate risks and ensure the responsible, ethical, and effective use of data.

II. Guiding Principles for Data & AI/ML

Core Data & AI/ML Principles (Mandatory Adherence):

- **Governance & Compliance First:** All data activities *must* align with the formal TMS Data Governance Policy, FERPA, NY Ed Law 2-d, NYCPS A-820, and all relevant security/privacy mandates.
- **DataOps Manifesto Alignment:** Embrace DataOps principles – automate everything possible, monitor quality/performance continuously, orchestrate workflows, foster collaboration between DE, Analytics, DS, and Ops.

- **MLOps Lifecycle Management:** Implement a structured MLOps approach for repeatable, reliable development, deployment, and monitoring of ML models.
- **Security & Privacy Embedded:** Apply security controls (encryption, access control, masking) consistently across data pipelines, storage, analytics environments, and ML workflows.
- **Data Quality as a Product:** Treat data quality not as an afterthought but as a critical feature, with defined metrics, automated testing, and clear ownership.
- **Automation & Self-Service (Governed):** Automate data pipelines, testing, and deployment. Enable governed self-service access to data and analytics tools for authorized users where appropriate.
- **Modularity & Reusability:** Build reusable data pipeline components, ETL/ELT logic (e.g., dbt models), feature engineering steps, and reporting templates.
- **Performance & Scalability by Design:** Architect data infrastructure and pipelines to handle current and future data volumes and processing needs efficiently within AWS GovCloud.
- **Ethical & Responsible AI:** Ensure fairness, transparency, explainability, and bias mitigation are considered and tested throughout the ML development lifecycle.
- **Continuous Improvement:** Regularly measure and analyze the performance, quality, and cost-effectiveness of data processes and models, feeding insights back into refinement cycles.

III. Data & Analytics Team Structure & Roles

We propose a hybrid model with a central Data Platform & Insights team providing core infrastructure, standards, and specialized expertise, potentially supplemented by analysts or data-savvy engineers embedded within domain-focused feature teams.

A. Central Data Platform & Insights Team

- **Data Engineering Lead:** Oversees DE strategy, architecture, pipeline development, DataOps practices.
- **Data Engineers (Multiple):** Design, build, test, deploy, and monitor data ingestion/transformation pipelines (streaming & batch), data lake/warehouse infrastructure. Implement DataOps CI/CD. Collaborate with DevOps on IaC for data resources.
- **Data Quality Analyst / Engineer:** Defines/implements DQ rules & monitoring, develops automated DQ tests, analyzes quality issues, works with Stewards/DE on remediation.
- **Data Performance Engineer (or DE focus):** Designs/executes performance tests for data pipelines and queries, analyzes bottlenecks, works with DE/DBA on optimization.
- **BI / Analytics Lead:** Oversees reporting/visualization strategy, defines standards, manages BI platform (QuickSight).

- **BI Developers / Data Analysts (Multiple):** Develop certified dashboards/reports, support ad-hoc analysis requests, train business users on self-service tools, work with Data Stewards on requirements.
- **Data Science Lead:** Oversees ML/AI strategy, use case validation, ethical AI guidelines, MLOps practices.
- **Data Scientists (Multiple):** Explore data, develop/train ML models, perform advanced statistical analysis, evaluate model performance/bias.
- **Machine Learning Engineer (MLE - potentially overlapping DE):** Focuses on operationalizing ML models – building robust training/inference pipelines, deploying models as services, monitoring production models (MLOps/AIOps).
- **Database Administrator (DBA - potentially shared w/ Ops):** Manages performance tuning, security configuration, backup/recovery specifically for core databases (RDS, Redshift).

B. Embedded Roles (Potential)

- **Embedded Data Analyst:** Works within a specific feature team (e.g., Routing Optimization) performing domain-specific analysis and reporting, leveraging central platform/tools.
- **Data-Savvy Software Engineer:** Feature team developers trained in basic data engineering/quality practices to implement simple data pipelines or quality checks specific to their microservice.

C. Collaboration Model

Implementation How-To:

1. Central Data Platform team defines standards, builds core infrastructure (lake, warehouse, pipelines, ML platform), and provides tools/support via established processes (e.g., Jira request queue for new pipelines/reports).
2. Feature teams consume data via defined APIs or governed views in the data warehouse/lake.
3. Embedded analysts/engineers work closely with both their feature team and the central data team, adhering to central standards.
4. Regular sync meetings between central data team leads and feature team leads to discuss dependencies, requirements, and standards.
5. Utilize shared Slack/Teams channels for cross-team data-related communication.

Responsibility: Project Leadership, Data Engineering Lead, Analytics Lead, DS Lead.

IV. Prescriptive Data Engineering Strategy

Data Engineering provides the reliable foundation for all data-driven activities, ensuring data is available, accurate, secure, and performant across all environments.

A. Data Infrastructure Provisioning & Management (Non-Prod & Prod)

1. Non-Production Environments (Dev, QA, Staging, Perf, DS Sandboxes)

What:

Provide isolated, scalable, and cost-effective environments with production-like architecture but using masked/synthetic data for development, testing, and data science experimentation.

How (Implementation):

1. **IaC for Data Resources:** Use dedicated Terraform modules (from the central `infra-tf/modules/` repo) to

provision all data infrastructure (RDS, DynamoDB, Kinesis, SQS, S3, Glue Jobs, Redshift clusters/schemas, SageMaker instances/notebook environments) per environment.

2. **Environment Parameterization:** Use environment-specific `.tfvars` files (`dev.tfvars`, `qa.tfvars`, etc.) to control instance sizes, cluster counts, storage allocation, IAM roles, security groups, KMS keys, and *crucially, pointers to masked/synthetic data sources*.

3. **Data Isolation:** Ensure strict network isolation (separate VPCs or subnets/SGs) and IAM controls prevent non-prod environments from accessing production data stores.

4. **Masked/Synthetic Data Pipelines:** Implement automated ETL pipelines (Glue, Lambda) to populate non-prod databases/S3 buckets with refreshed, structurally valid, but anonymized/synthesized data according to the TDM strategy (see Testing Strategy) and Data Governance Policy. Schedule regular refreshes.

5. **Cost Controls:** Implement aggressive auto-scaling down/shutdown policies for non-prod resources during off-hours (via Lambda/EventBridge or instance scheduler). Use smaller instance types where feasible. Set strict AWS Budgets alerts for non-prod accounts/tags.

6. **DS Sandboxes:** Provision isolated sandbox environments (potentially using SageMaker Studio user profiles or dedicated accounts/VPCs) with access only to approved, anonymized datasets for experimentation.

Responsibility: DevOps Team (IaC Execution), Data Engineers (Data-specific IaC modules, Masking Pipelines), Cloud Ops/FinOps (Cost Controls), Security Team (IAM/Network Policies).

2. Production Environment

What:

Provision highly available, scalable, secure, performant, and meticulously monitored production data infrastructure.

How (Implementation):

1. **IaC Deployment:** All production data infrastructure provisioned and managed *exclusively* via the approved Terraform code base deployed through the rigorous GitLab CI/CD pipeline (requiring manual approvals). No manual console changes allowed.
2. **High Availability:** Deploy stateful resources (RDS, ElastiCache, MSK) across multiple (typically 3) Availability Zones using built-in HA features (Multi-AZ RDS, Redis cluster mode, MSK multi-broker replication). Ensure stateless processing layers (Lambda, Fargate) run across multiple AZs behind load balancers.
3. **Scalability:** Configure auto-scaling for compute (Fargate/ECS Services, Lambda Provisioned Concurrency if needed) and potentially databases (RDS Read Replicas,

DynamoDB On-Demand/Auto Scaling). Monitor capacity proactively (see SRE practices).

4. **Security:** Apply strictest security controls: least-privilege IAM roles, granular Security Groups/NACLs, encryption at rest (KMS CMKs) and in transit (TLS 1.2+), private subnets, VPC Endpoints, mandatory audit logging.
5. **Backup & DR:** Implement automated, encrypted backups (AWS Backup, RDS/DynamoDB native) with cross-region replication to the DR AWS GovCloud region. Regularly test restore procedures and full DR failover (as per DR Plan).

Responsibility: DevOps Team (IaC Execution), SRE/Ops Team (Monitoring, HA/DR Config), Security Team (Control Validation), DBA (DB Config).

B. Data Ingestion Pipelines (Streaming & Batch)

1. Streaming Ingestion (e.g., GPS, Real-time Ridership)

What:

Ingest high-volume, low-latency data reliably from devices and applications.

How (Implementation):

1. **Endpoint:** Use AWS IoT Core (for MQTT devices) with rules forwarding to Kinesis/MSK, OR use API Gateway (HTTP API for performance) invoking Lambda or directly integrating with Kinesis/MSK.
2. **Transport:** Use Kinesis Data Streams (easier management) or Managed Streaming for Kafka (MSK - for Kafka ecosystem compatibility/control) as the primary streaming backbone. Provision sufficient shards/brokers based on load testing. Enable server-side encryption.
3. **Processing:** Use Lambda functions triggered by Kinesis/MSK streams OR Fargate/ECS services consuming from streams for validation, transformation, enrichment, and routing to downstream stores (e.g., DynamoDB for real-time state, S3 data lake for raw/processed records, potentially other SQS queues).
4. **Error Handling:** Implement DLQs for Lambda triggers or robust error handling within consumer applications to route failed messages for investigation without blocking the stream. Monitor processing lag (`IteratorAge` for Kinesis).
5. **Schema Enforcement:** Use Glue Schema Registry (if using Kafka/Kinesis with Avro/Protobuf) or implement schema validation within ingestion Lambdas/services.

Responsibility: Data Engineers, Backend Developers, DevOps Team (Infra).

2. Batch Ingestion (e.g., Daily extracts from ATS/NPSIS, 3rd party files)

What:

Reliably ingest bulk data from files or database extracts on a scheduled basis.

How (Implementation):

1. **Transport:** Use AWS Transfer Family for secure SFTP endpoints where external systems push files. Use S3 Event Notifications to trigger processing when files arrive in designated landing zone buckets. For DB extracts, use AWS DMS (if feasible/approved for direct connection) or scheduled export jobs pushing files to S3.
2. **Orchestration:** Use AWS Step Functions or AWS Glue Workflows to orchestrate multi-step batch processing jobs (e.g., download -> validate -> transform -> load -> quality check -> archive).
3. **Processing:** Use AWS Glue ETL jobs (PySpark/Scala) for complex transformations and large volumes. Use Lambda functions for simpler tasks (file validation, triggering jobs).
4. **Validation & Quality:** Integrate data validation and Glue Data Quality checks within the batch workflow. Route failed files/records to an error location/queue.
5. **Scheduling:** Use EventBridge (CloudWatch Events) Scheduler or Glue Triggers to run batch jobs on the required cadence.

6. **Monitoring:** Monitor job success/failure, duration, and data quality results via CloudWatch Logs/Metrics and Glue Job metrics. Set alarms on failures or excessive runtimes.

Responsibility: Data Engineers, ETL Developers, DevOps Team (Infra/Scheduling).

C. Data Transformation & Modeling (Lakehouse Approach)

What:

Transform raw ingested data into cleaned, structured, and modeled datasets suitable for reporting, analytics, and ML, utilizing a Lakehouse architecture (combining Data Lake flexibility with Data Warehouse structure).

How (Implementation):

1. **Data Lake (S3):**
    - Store raw, immutable ingested data in a dedicated S3 "Raw Zone" bucket (partitioned by source/date).
    - Store cleaned, standardized, validated data (e.g., in Parquet format) in an S3 "Processed Zone" or "Staging Zone" bucket

(partitioned appropriately for query performance).

- Store highly curated, aggregated, or feature-engineered data for ML/Analytics in an S3 "Curated Zone" or "Gold Zone".
- Utilize AWS Glue Data Catalog to catalog schemas and partitions across all zones, enabling query engines like Athena and Redshift Spectrum to access S3 data.

2. **Data Warehouse (Redshift):**
    - Provision an AWS Redshift cluster (Serverless for variable workloads or Provisioned RA3 nodes for predictable high performance) for storing modeled, aggregated data optimized for BI and analytics queries.
    - Design dimensional models (Star/Snowflake schemas) within Redshift for key business domains (Ridership Performance, Route Efficiency, Student Transportation Needs).
    - Load data into Redshift from the S3 Processed/Curated Zone using `COPY` commands, Glue ETL jobs, or Redshift

Spectrum external tables (querying S3 directly).

3. **Transformation Logic (ETL/ELT):**
    - Implement transformations primarily using **AWS Glue** (PySpark/Scala for complex logic) or potentially **dbt (Data Build Tool)** running on EC2/ECS/Glue interacting with Redshift/Athena/S3 for SQL-based transformations and modeling.
    - Code/Scripts version controlled in GitLab.
    - Integrate data quality checks directly into transformation jobs (Glue DQ, dbt tests).
    - Orchestrate transformation jobs using Step Functions or Glue Workflows.

Responsibility: Data Engineers, Data Architects, ETL Developers, potentially Analytics Engineers (using dbt).

D. DataOps Pipelines & Practices

DataOps applies DevOps principles to data pipelines, focusing on automation, testing, collaboration, and monitoring to improve speed and reliability.

Implementation How-To:

1. **CI/CD for Data Pipelines:**
    - Store all data pipeline code (Glue scripts, Python Lambda functions, SQL transformation scripts, dbt models, Terraform for data infra) in GitLab.
    - Implement dedicated GitLab CI/CD pipelines for data jobs:
        - Lint code (SQLFluff, Pylint).
        - Unit test transformation logic (pytest for Python, dbt test for dbt models).
        - Integration test pipeline components against test environments (e.g., run Glue job against test S3 data/Redshift schema).
        - Package code (e.g., zip Lambda, package Glue scripts).
        - Deploy infrastructure changes (Terraform apply

for Glue jobs, Lambda functions, Step Functions).

- Deploy updated pipeline logic/code.

- Use environment branches or parameterization for deploying pipelines to Dev/QA/Prod environments.

2. **Automated Data Quality Testing:** Integrate automated DQ checks (Glue DQ, Great Expectations, dbt tests) directly into pipeline execution. Fail pipelines if critical quality thresholds are breached.

3. **Automated Schema Management:** Use tools like Glue Schema Registry or manage database schema changes via migration tools (Flyway/Liquibase triggered by CI/CD) integrated with pipeline deployments.

4. **Pipeline Monitoring & Alerting:** Implement detailed monitoring for pipeline execution (Glue/Step Functions metrics, CloudWatch logs). Alert on failures, excessive runtimes, or data quality issues detected within the pipeline.

5. **Data Lineage Tracking:** Utilize Glue Data Catalog and potentially third-party tools to automatically track data lineage from source through transformations to consumption layers (Warehouse/Reports).

6. **Collaboration:** Use shared code repositories (GitLab), documented processes (Confluence), and integrated tracking

(Jira) for collaboration between DE, Analytics, DS, and Ops.

Responsibility: Data Engineers, DevOps Team, QA Team (DQ aspects), Analytics/DS Teams (Consumers/Collaborators).

V. Prescriptive Data Quality & Performance Testing

Ensuring the quality and performance of data itself, and the pipelines processing it, is critical for trust and usability.

A. Data Quality Framework & Testing

Implementation How-To:

1. **Rule Definition:** Data Stewards, working with BAs and Analysts, define specific, measurable Data Quality rules for CDEs in the TMS Data Governance Policy / Data Dictionary (e.g., `student_id` must exist in ATS/NPSIS master, `gps_timestamp` must be within +/- 5 mins of system

time, `route_eta` must not be null for active routes, `vehicle_capacity_used` <= `vehicle_capacity_total`).

2. **DQ Check Implementation:**
    - **Ingestion:** Implement basic format/type validation at ingestion points (APIs, Lambdas).
    - **Transformation (ETL/ELT):** Embed complex DQ rule checks directly into Glue jobs (using Glue Data Quality) or dbt models (using `dbt test`). Checks can include uniqueness, referential integrity, accepted values, null checks, range checks, regex patterns, custom SQL assertions.
    - **At Rest:** Develop scheduled Lambda functions or use tools like Great Expectations (potentially run via Glue/EMR) to run quality checks directly against Data Lake (S3/Athena) or Warehouse (Redshift) tables periodically.
3. **Automation & Gating:** Integrate DQ check execution into DataOps CI/CD pipelines. **Quality Gate:** Fail pipeline builds/deployments if critical DQ checks fail (e.g., > X% nulls in required field, referential integrity broken).

4. **Monitoring & Reporting:** Send DQ check results (pass/fail counts, specific failing records) to CloudWatch Metrics/Logs. Create Data Quality Dashboards (QuickSight/Grafana) visualizing quality scores/trends per data domain/source. Alert Data Stewards/Support on critical DQ failures.

5. **Issue Management:** Log DQ failures as defects in Jira/ADO, assign to Data Stewards for triage and relevant DE/Source System team for remediation. Track resolution.

Tools: AWS Glue Data Quality, Great Expectations, dbt test, Custom SQL/Python Scripts, CloudWatch, Jira/ADO, QuickSight/Grafana.

Responsibility: Data Stewards (Rules), Data Engineers (Implementation), QA Team (Verification/Reporting), Data Governance Lead (Process Owner).

Demonstrable data quality processes and metrics are essential for building trust in reports and analytics derived from TMS data.

B. Data Performance Testing

Implementation How-To:

1. **Identify Critical Paths:** Determine key data pipelines (e.g., GPS stream processing -> real-time location update) and critical analytical queries/reports with performance NFRs (throughput, latency).

2. **Define Performance NFRs:** Set specific, measurable targets (e.g., GPS end-to-end latency P99 < 10 seconds, Daily reporting ETL must complete within 1 hour, Dashboard load time < 5 seconds).
3. **Environment Setup:** Use the dedicated, production-scaled PERF environment. Ensure realistic, anonymized data volumes representative of peak load.
4. **Test Script Development:**
    - **Pipeline Load:** Use tools (e.g., Kinesis Data Generator, custom Python scripts, JMeter feeding API Gateway) to generate realistic high-volume streaming or batch input data.
    - **Query/Report Load:** Use tools (e.g., Apache Bench, JMeter, custom scripts using DB drivers/QuickSight API) to simulate concurrent user queries against Redshift/Athena/QuickSight dashboards.
5. **Execution & Monitoring:** Run performance test scripts against the PERF environment while closely monitoring relevant system metrics (CloudWatch, Redshift/DB metrics, Glue job metrics, Lambda metrics).
6. **Analysis:** Analyze results against NFRs. Identify bottlenecks (e.g., insufficient Kinesis shards, under-provisioned Redshift capacity, inefficient SQL queries,

Lambda timeouts, slow ETL transformations). Use CloudWatch, X-Ray, DB performance insights tools.

7. **Optimization & Retesting:** Work with DE/DBA/Dev teams to implement optimizations (tune queries, adjust partitioning/distribution keys, scale resources, refactor code). Rerun tests to verify improvement.

8. **Gating:** Include performance test execution and result validation as a mandatory step/quality gate before major production releases impacting critical data paths.

Tools: k6, JMeter, Gatling, Kinesis Data Generator, Custom Scripts, CloudWatch, Redshift Performance Dashboards, X-Ray, Athena Query Stats.

Responsibility: Performance Engineer/QA, Data Engineers, DBAs, SRE/Ops.

Quality Gate (Pre-Release): Documented performance test results must meet defined NFRs for critical data pipelines and queries before production deployment.

VIII. Conclusion: Enabling Data-Driven Excellence

This prescriptive Data, Analytics, and AI/ML Strategy provides the comprehensive framework required to manage and leverage data as a strategic asset for the NYCPS TMS project effectively, securely, and responsibly. By mandating best-in-class practices in Data Engineering (DataOps), Data Quality, Performance Testing, Reporting/Visualization, Data Science, and MLOps/AIOps, integrated with stringent governance and ethical AI considerations, we establish a foundation for trust, reliability, and continuous improvement.

The detailed implementation steps, focus on automation (especially via AWS SageMaker and GitLab CI/CD), clear role definitions, and embedded quality/security gates ensure that data pipelines are robust, analytics are insightful, ML models are reliable and fair, and all activities comply with the critical legal and policy requirements governing NYCPS data. Adherence to this strategy is fundamental to unlocking the full potential of data to optimize transportation, enhance safety, ensure compliance, and ultimately deliver superior value to the NYCPS community.