NYCPS Transportation Management System - AWS GovCloud Architecture (RFP R1804)

1. Introduction

This document outlines the technical architecture for the NYCPS Transportation Management System, designed to meet the functional requirements specified in the Solution Design and the stringent non-functional requirements (performance, scalability, security, availability, compliance) outlined in RFP R1804. The architecture prioritizes deployment within **AWS GovCloud (US)** regions, leveraging managed services where possible to enhance reliability, scalability, and security while adhering to necessary compliance standards (e.g., FedRAMP High, ITAR, CJIS, HIPAA as applicable within GovCloud).

2. Architectural Principles

- **GovCloud Native:** Utilize AWS services available within the GovCloud (US) regions.
- **Security by Design:** Implement security controls at every layer (network, compute, data, application). Assume a Zero Trust posture.
- **High Availability & Resilience:** Design for fault tolerance using multi-AZ deployments and managed services with built-in HA. Implement robust Disaster Recovery (DR).
- **Scalability & Elasticity:** Employ auto-scaling, serverless components, and elastic managed services to handle variable loads (daily peaks, seasonal changes) efficiently.
- **Modularity (Microservices):** Utilize a microservices architecture to promote independent development, deployment, scaling, and fault isolation of components.
- **Event-Driven:** Leverage asynchronous messaging and event streams for decoupling components and enabling real-time processing.

- **Infrastructure as Code (IaC):** Manage infrastructure provisioning and configuration through code for consistency, repeatability, and auditability.
- **Managed Services Preference:** Utilize AWS managed services (`RDS` , `Kinesis` , `DynamoDB` , `SQS` , `SNS` , `Lambda` , `Fargate` , etc.) to reduce operational overhead and leverage AWS expertise.

3. Overall Architecture Pattern

The proposed architecture is a **cloud-native, event-driven microservices architecture** deployed across multiple Availability Zones (AZs) within an AWS GovCloud (US) region.

- **Data Ingestion:** GPS devices communicate via MQTT/HTTPS to `AWS IoT Core` or directly to `API Gateway` endpoints. Data flows through streaming services (`Kinesis` / `MSK`).
- **Processing:** `Lambda` functions and containerized services (on `Fargate` / `ECS` / `EKS`) process events and data streams. The Routing Engine may use dedicated `EC2` instances for intensive computation.
- **Data Storage:** A mix of databases (`RDS PostgreSQL/PostGIS` , `DynamoDB` , `ElastiCache`) and object storage (`S3` , `Glacier`) is used based on data type and access patterns.
- **APIs & Communication:** RESTful APIs exposed via `API Gateway` serve front-end applications and internal service communication. `SNS` / `SQS` / `SES` handle asynchronous messaging and notifications.
- **Frontend:** Mobile applications (iOS/Android) and Web applications (React/Angular/Vue hosted on `S3` / `CloudFront`) interact with backend APIs.

4. Component-to-AWS Service Mapping

Functional Component	Primary AWS GovCloud Service(s)	Rationale & Notes
GPS Data Ingestion & Processing	`IoT Core` (optional, for MQTT), `API Gateway` + `Lambda` / `Fargate` , `Kinesis`	IoT Core provides scalable MQTT ingestion if devices support it. Kinesis/MSK handles high-throughput

Functional Component	Primary AWS GovCloud Service(s)	Rationale & Notes
	`Data Streams` or `MSK`, `S3` (raw data), `DynamoDB` (real-time location), `RDS PostgreSQL w/ PostGIS` (processed tracks, geofences), `Lambda` / `Fargate` (ETA calc, geofence logic), `CloudWatch` (device status)	streaming data. Lambda/Fargate enables scalable, event-driven processing. DynamoDB offers low-latency reads/writes for real-time location. RDS/PostGIS handles complex spatial queries & history.
Ridership Tracking Module	`API Gateway` + `Lambda` / `Fargate`, `DynamoDB` or `RDS PostgreSQL`, `S3` (for ID scan images if needed)	API endpoints handle scan/manual entry events. DynamoDB or RDS store ridership records, depending on query patterns. Lambda/Fargate process events, validate against rosters (from Student Mgmt).
Dynamic Routing Engine	`EC2` / `ECS` / `EKS` (potentially w/ GPU instances), `RDS PostgreSQL w/ PostGIS`, `ElastiCache` (Redis/Memcached), `AWS Location Service` (if available/suitable) or self-managed `ESRI ArcGIS` on `EC2` / `RDS`, `SQS` / `Step Functions` (async tasks), `Lambda` (rule triggers)	EC2/Containers needed for complex, potentially long-running routing algorithms. RDS/PostGIS stores road network, stops, routes, constraints. ElastiCache speeds up access to frequently used data (e.g., traffic, routes). AWS Location or ESRI for mapping/geocoding/traffic. SQS/Step Functions for orchestration.
Notification & Communication	`SNS` (Push, SMS), `SES` (Email), `Amazon Connect` (optional, for Robocalls/IVR), `Lambda` / `Fargate` (notification logic), `API Gateway` (for internal communication APIs)	SNS/SES provide scalable, managed notification delivery. Connect offers programmable voice capabilities. Lambda/Fargate implement business logic for triggering and formatting notifications based on events from other systems.
User Modules (Frontend)	`S3` + `CloudFront` (Web Apps), Native iOS/Android SDKs,	S3/CloudFront provide scalable, secure, low-latency hosting for web frontends.

Functional Component	Primary AWS GovCloud Service(s)	Rationale & Notes
	`Amplify` (optional, for mobile/web dev acceleration)	Native SDKs for mobile apps. Amplify can streamline frontend development and backend integration.
User Modules (Backend APIs)	`API Gateway`, `Lambda` / `Fargate` (ECS/EKS), `Cognito` or SAML integration via backend, `IAM`	API Gateway provides secure, scalable API endpoints. Lambda/Fargate host backend microservices logic. Cognito handles user authentication/pools if direct DOE integration isn't used. IAM secures service-to-service communication.
Student Management Backend	`API Gateway` + `Lambda` / `Fargate`, `RDS PostgreSQL` or `DynamoDB`, `Glue` (for ETL from NYCPS systems if needed), `SFTP` / `DataSync` (for file-based integration if needed)	Standard API-driven microservice pattern. Choice of DB depends on data model complexity and query patterns. Glue/SFTP/DataSync facilitate integration with potentially legacy NYCPS systems.
Reporting & Analytics	`S3` (Data Lake), `Glue` (ETL/Catalog), `Athena`, `Redshift`, `QuickSight`, `Kinesis Data Analytics` (optional, real-time KPIs), `Lambda` (scheduled report generation)	S3 provides a scalable data lake. Glue handles ETL and cataloging. Athena for ad-hoc queries on S3. Redshift for performant data warehousing. QuickSight for dashboards/visualizations. Kinesis Data Analytics for streaming analytics. Lambda for automated report generation.
Device Management & Inventory	`DynamoDB` or `RDS PostgreSQL`, `IoT Core` (Device Shadow/Registry), `Systems Manager` (potentially for config), API integration with MDM & Ticketing	DynamoDB/RDS stores inventory data. IoT Core can track device state/metadata if devices connect via MQTT. Systems Manager might assist with certain configurations. Requires integration points.

Functional Component	Primary AWS GovCloud Service(s)	Rationale & Notes
Ticketing System Integration	`API Gateway` + `Lambda` / `Fargate`	Acts as a secure facade/adapter layer to mediate API calls between AWS environment and the NYCPS/Vendor ticketing systems.

5. Networking & Security (AWS GovCloud)

- **VPC Structure:** Multiple VPCs may be used (e.g., Prod, Dev, Test). Each VPC spans multiple AZs. Use of public subnets restricted to necessary components (e.g., Load Balancers, NAT Gateways). Private subnets host core application/database resources. VPC Endpoints (Gateway and Interface) used for private access to AWS services (`S3`, `DynamoDB`, `Kinesis`, etc.).
- **Segmentation:** Security Groups (stateful) and Network ACLs (stateless) enforce strict ingress/egress rules based on the principle of least privilege between application tiers and components.
- **Connectivity:** `AWS Direct Connect` or `Site-to-Site VPN` established between AWS GovCloud VPC and NYCPS data centers for secure, private access to internal resources/databases.
- **Edge Security:** `CloudFront` for caching and DDoS protection for web applications. `AWS WAF` deployed with API Gateway and CloudFront to filter malicious requests (SQLi, XSS). `AWS Shield Advanced` for enhanced DDoS protection.
- **Encryption:**
    - *In Transit:* TLS 1.2+ enforced for all external and internal API Gateway endpoints, CloudFront distributions, Load Balancers, and direct service communications.
    - *At Rest:* Server-side encryption enabled for `S3` (SSE-S3 or SSE-KMS), `EBS` volumes, `RDS` instances/snapshots, `DynamoDB` tables, `SQS` queues, `SNS` topics using `AWS KMS` with customer-managed keys (CMKs) where appropriate for enhanced control and auditability.
- **Identity & Access Management (IAM):**
    - Strict use of IAM Roles for EC2 instances, Lambda functions, ECS/EKS tasks (via IAM Roles for Service Accounts - IRSA - in EKS). Avoid long-lived access keys.

    - Fine-grained IAM Policies adhering to least privilege.
    - Multi-Factor Authentication (MFA) enforced for all human access to the AWS Management Console and API, especially for privileged accounts.
    - Federated identity management via SAML 2.0 integration with NYCPS's Identity Provider (e.g., ADFS, Azure AD) for console/API access if feasible, otherwise stringent controls on IAM users.
- **Secrets Management:** `AWS Secrets Manager` used to store and rotate database credentials, API keys, and other secrets securely. Applications retrieve secrets at runtime via IAM roles.
- **Monitoring & Logging:**
    - `CloudTrail` enabled in all regions, logging to a central, secured S3 bucket with log file validation enabled.
    - `CloudWatch` Logs collected from all services (Lambda, Fargate, EC2, RDS, etc.). CloudWatch Alarms configured for critical metrics (CPU, memory, latency, error rates, queue depths).
    - `AWS Config` used to track resource configurations and compliance.
    - `Security Hub` aggregated findings from GuardDuty, Inspector, Macie (if used), Config, and partner integrations.
    - `GuardDuty` enabled for intelligent threat detection.
    - `Inspector` used for vulnerability assessments on EC2 instances (if used).

6. Data Management

- **Storage:** Tiered storage using `S3` Standard, S3 Intelligent-Tiering, `S3 Glacier`, and S3 Glacier Deep Archive to meet access needs and 7-year retention requirements cost-effectively.
- **Databases:** Use appropriate databases per component needs (`RDS` for relational/spatial, `DynamoDB` for key-value/high-throughput, `ElastiCache` for caching). Regular backups and point-in-time recovery configured.
- **Data Privacy:** Implement controls aligned with Solution Design, including encryption, RBAC, audit logging, and data minimization principles.
- **Data Ownership & Export:** Mechanisms (e.g., APIs, data exports to S3) provided for NYCPS to access and extract their data on demand, reaffirming DOE data ownership.

7. Deployment & Operations (DevSecOps)

- **Infrastructure as Code (IaC):** `AWS CloudFormation`, `AWS CDK`, or `Terraform` used to define and provision all infrastructure resources. Templates stored in version control (`AWS CodeCommit` or GitHub/GitLab).
- **CI/CD Pipeline:** `AWS CodePipeline` orchestrating `CodeCommit` (source), `CodeBuild` (build/test, including SAST/DAST scans), and `CodeDeploy` or ECS/EKS deployment strategies (blue/green, canary) for automated, secure deployments across environments (Dev, Test, Staging, Prod).
- **Monitoring:** Leverage `CloudWatch` dashboards, alarms, and logs. Integrate with centralized logging/monitoring solutions if used by NYCPS. Potentially use APM tools (e.g., Datadog, Dynatrace, configured appropriately for GovCloud) for deeper application insights.

8. High Availability & Disaster Recovery (DR)

- **High Availability (HA):**
  - Deploy critical components (API Gateway, Lambda, Fargate/ECS/EKS clusters, RDS, ElastiCache, DynamoDB) across multiple AZs (typically 3) within the primary GovCloud region.
  - Use Elastic Load Balancing (ALB/NLB) to distribute traffic across AZs.
  - Configure RDS Multi-AZ deployments for automatic failover.
  - Leverage DynamoDB global tables (if cross-region active-active is needed and available) or rely on inherent multi-AZ replication.
- **Disaster Recovery (DR):**
  - Establish a DR strategy (e.g., Pilot Light, Warm Standby) in a second AWS GovCloud (US) region.
  - Regularly back up data (using `AWS Backup`, RDS snapshots, DynamoDB backups) and replicate backups/snapshots to the DR region.
  - Use IaC to provision infrastructure in the DR region quickly.
  - Utilize `Route 53` health checks and DNS failover mechanisms.
  - Regularly test the DR plan (at least annually).

    - Design to meet RFP RPO/RTO targets (RPO=0 for GPS, RPO<=1hr for Routing/Notifications; RTO=0 for GPS, RTO<=15min for Routing/Notifications). Achieving RTO=0/RPO=0 for GPS likely requires an active-active or hot standby approach across AZs/regions for critical ingestion/processing components.

9. Compliance

- The architecture leverages services within AWS GovCloud (US), designed to host sensitive data and regulated workloads, meeting standards like FedRAMP High, ITAR, CJIS, DoD SRG IL4/5.
- Specific configurations (encryption, logging, IAM, network controls) align with security requirements from NYCPS, NYC3, OTI, DIIT, FERPA, CIPA, HIPAA (as applicable within GovCloud).
- Regular audits and security testing (as required by RFP) will validate ongoing compliance.