NYCPS Transportation Management System: Prescriptive SDLC & Setup

Introduction & Methodology

This document establishes the definitive Software Development Lifecycle (SDLC) to be rigorously followed for the NYCPS Transportation Management System project (RFP R1804). Recognizing the project's criticality, scale, security sensitivities, AWS GovCloud deployment context, and demanding 12-month delivery timeframe, this SDLC mandates an **Agile (Scrum-based) methodology deeply integrated with DevSecOps principles.**

Adherence to this SDLC is mandatory. Its core tenets are:

- **Iterative & Incremental Delivery:** We will deliver functional, tested software increments in defined Sprints (typically 2 weeks), aligned with the major project phases. Each Sprint delivers a potentially shippable product increment.
- **Stakeholder Collaboration:** Continuous, structured engagement with NYCPS (OPT, DIIT, Security, Schools, etc.) is embedded in the process

through defined meetings and review cycles.

- **Pervasive Automation:** We will automate testing (unit, integration, E2E, performance, security), infrastructure provisioning (IaC), builds, deployments, and monitoring to the maximum extent feasible.
- **Integrated Security ("Shift Left"):** Security is not an afterthought. Security requirements, threat modeling, secure coding practices, code analysis (SAST), vulnerability scanning (SCA, DAST), and security reviews are integral to every phase.
- **Transparency & Adaptability:** Project progress, risks, and decisions will be transparent via dashboards and regular reporting. The Agile framework allows for controlled adaptation based on feedback and evolving understanding.
- **Built-in Quality:** Quality assurance is embedded through TDD/BDD, stringent code reviews, comprehensive automated testing, clear Definitions of Ready (DoR) and Done (DoD), and dedicated QA resources.
- **Governance & Compliance:** Formal governance checkpoints and stakeholder sign-offs ensure alignment and compliance throughout the lifecycle.

Phase 1: Project Initiation, Planning & Requirements Definition

Goal: Formally initiate the project, establish governance, define the detailed scope and roadmap, and elicit, analyze, document, and validate requirements for the initial project phases.

1.1 Project Kick-off & Governance Setup

- **Activity:** Conduct formal Project Kick-off Meeting.
    - Define project vision, goals, and critical success factors.
    - Introduce key team members and stakeholders (Vendor & NYCPS).
    - Establish communication plan, meeting cadences, and reporting structure.
    - Review and confirm understanding of contractual obligations and RFP requirements.
- **Activity:** Establish Project Governance Structure.
    - Define roles and responsibilities for Steering Committee, Project Management Team, Technical Review Board, Security Review Board, Change Control Board.
    - Schedule recurring governance meetings (e.g., Bi-weekly Steering Committee, Weekly Project Status).
- **Activity:** Finalize High-Level Project Plan & Roadmap.
    - Review and refine the 12-month phased roadmap and timeline (from proposal).
    - Identify major dependencies and critical path items.
    - Establish initial risk register.

1.2 Detailed Requirements Elicitation & Analysis (Focus: Initial Phases)

- **Activity:** Conduct Requirements Workshops with NYCPS Stakeholders (OPT, DIIT, Security, Pilot Schools, etc.).
    - Utilize techniques like user interviews, process mapping, use case modeling, prototyping.
    - Focus on core GPS tracking, driver login/assignment, basic admin view, and initial device deployment requirements (aligned with Phases 0-2).
    - Elicit detailed functional requirements, non-functional requirements (performance, usability, accessibility targets), data requirements, integration points, and security/compliance constraints.
- **Activity:** Analyze and Document Requirements.
    - Translate elicited needs into clear, concise, testable requirements (e.g., User Stories with detailed Acceptance Criteria).
    - Document business process flows (current and future state).
    - Develop initial data dictionary and conceptual data models.
- **Activity:** Requirements Validation & Prioritization.
    - Review documented requirements with stakeholders for accuracy and completeness.

- Prioritize requirements for the initial sprints/phases using methods like MoSCoW (Must have, Should have, Could have, Won't have) or Value vs. Effort.

1.3 Product Backlog Creation & Initial Grooming

- **Activity:** Populate Project Management Tool (e.g., Jira) with defined Epics, Features, and User Stories.
- **Activity:** Conduct initial Backlog Grooming sessions with the development team and Product Owner (NYCPS designated) to refine stories, estimate effort (e.g., story points), and ensure Definition of Ready (DoR) is met for upcoming sprint(s).

Governance & Sign-offs:

- **Project Charter Review:** Steering Committee review and approval of the formal Project Charter outlining scope, goals, governance, and high-level plan.
- **Requirements Baseline Sign-off:** Formal sign-off by designated NYCPS Product Owner(s) and key stakeholders on the prioritized backlog and detailed requirements for the initial project phase(s).
- **Weekly Project Status Meetings:** Review progress, risks, issues, and upcoming activities.

Best Practices Emphasized:

- Agile Requirements Gathering (User Stories, Acceptance Criteria)
- Stakeholder Collaboration & Workshops

- Backlog Management & Prioritization
- Formal Governance Setup
- Risk Identification

Exit Criteria: Approved Project Charter; Prioritized Product Backlog for initial phases with signed-off User Stories meeting DoR; Established Governance Structure & Communication Plan.

Phase 2: Architecture & Design

Goal: Translate validated requirements into detailed technical designs for the system architecture, components, interfaces, data models, and user interfaces, ensuring security, scalability, and compliance are designed in.

2.1 System Architecture Definition

- **Activity:** Refine AWS GovCloud Architecture Document.
  - Finalize selection of specific AWS services, instance types, storage configurations, database engines based on detailed NFRs and cost analysis.
  - Detail network architecture: VPC/subnet layout, routing, Security Group/NACL rules, VPN/Direct Connect configuration, VPC Endpoints strategy.

        - Document HA/DR strategy: Multi-AZ deployment patterns, database replication, backup/restore procedures, cross-region DR setup (if applicable).
- **Activity:** Define Microservice Boundaries & Interactions.
    - Apply DDD principles to confirm service boundaries.
    - Define communication patterns (Synchronous REST via API Gateway vs. Asynchronous Events via Kinesis/MSK/SQS/SNS).
    - Specify API contracts using OpenAPI Specification (Swagger) for all inter-service communication. Version control API specifications.

2.2 Component & Interface Design

- **Activity:** Design Individual Microservices.
    - Create detailed design documents for each microservice, outlining internal logic, data access patterns, error handling, and dependencies.
    - Define technology stack for each service (e.g., language, framework, libraries).
- **Activity:** Design Data Models.
    - Create detailed logical and physical data models for relational databases (RDS PostgreSQL/PostGIS).
    - Design data structures and access patterns for NoSQL databases (DynamoDB).
    - Define schemas for data streams (Kinesis/MSK).

        - Plan data migration approach for existing NYCPS data (if applicable).
- **Activity:** Design User Interfaces (UI) & User Experience (UX).
    - Develop detailed wireframes and high-fidelity mockups based on stakeholder feedback.
    - Create interactive prototypes for key user flows.
    - Conduct usability testing sessions with representative users (drivers, parents, admins).
    - Finalize UI style guide (leveraging NYCPS style guide) and component library.
    - Ensure designs meet WCAG 2.0 AA requirements.
- **Activity:** Design Integrations.
    - Define detailed specifications for APIs or file formats for integrating with internal NYCPS systems (ATS, NPSIS, Ticketing, etc.) and external services (Traffic data, Maps).

2.3 Security Design & Infrastructure as Code Planning

- **Activity:** Conduct Detailed Threat Modeling.
    - Perform STRIDE/DREAD analysis on critical components (authentication, PII handling, routing engine, public-facing APIs).
    - Identify potential attack vectors and define specific mitigation controls in the design.

- **Activity:** Design Security Controls.
    - Detail authentication flows (SAML/OIDC integration, Cognito setup).
    - Design fine-grained authorization logic (RBAC implementation).
    - Specify encryption methods and key management strategy (KMS usage).
    - Define logging/auditing requirements for security events.
    - Plan WAF rulesets and Security Group configurations.
- **Activity:** Plan Infrastructure as Code (IaC).
    - Structure IaC repositories (e.g., Terraform modules, CloudFormation stacks).
    - Define standards for IaC development, review, and testing.
    - Plan parameterization for environment-specific configurations.

Governance & Sign-offs:

- **Architecture Review:** Presentation and review of the detailed AWS GovCloud architecture by the Technical Review Board (including DIIT/Security representatives). Formal sign-off required.
- **Security Design Review:** Detailed review of security architecture, threat models, and control designs by the Security Review Board/NYC3 representatives. Formal sign-off required.

- **UI/UX Design Review:** Review and approval of mockups and prototypes by NYCPS Product Owner and key user representatives.
- **API Contract Review:** Review and approval of internal and external API specifications.
- **Component Design Reviews:** Peer and technical lead reviews of detailed designs for key microservices/components.

Best Practices Emphasized:

- Cloud-Native Design (Leveraging Managed Services)
- Microservices Architecture & API-First Design
- Security by Design & Threat Modeling
- Infrastructure as Code Planning
- UX Design & Usability Testing
- HA/DR Planning

Exit Criteria: **Signed-off Architecture Document; Signed-off Security Design Document; Approved API Specifications; Approved UI/UX Mockups/Prototypes; Detailed Designs for initial microservices; Approved IaC plan.**

Phase 3: Development (Implementation)

Goal: Construct the software components, infrastructure, and tests iteratively within Sprints, adhering to designs and secure coding practices.

3.1 Sprint Execution

- **Activity:** Sprint Planning.
    - Team selects User Stories from the prioritized backlog based on capacity and sprint goals.
    - Tasks are broken down from stories.
    - Team commits to the sprint backlog.
- **Activity:** Daily Stand-ups.
    - Brief daily meetings to synchronize, discuss progress, and identify impediments.
- **Activity:** Feature Development.
    - Developers implement features based on user stories and technical designs.
    - Code is written following secure coding standards (OWASP Top 10, NYCPS standards).
    - Input validation, output encoding, proper error handling are implemented.
    - Authentication/Authorization checks are integrated.
    - Code is commented appropriately.
- **Activity:** Unit Testing (TDD).

        - Developers write and execute unit tests for their code, aiming for high code coverage (e.g., >80-90% branch coverage).
        - Tests are automated and run as part of the local build and CI process.
- **Activity:** Infrastructure as Code (IaC) Development.
    - DevOps/Cloud Engineers develop/update IaC scripts (Terraform/CloudFormation) to provision necessary AWS resources for the features being built.
    - IaC code is version controlled alongside application code.
- **Activity:** Version Control & Code Integration.
    - Developers commit code frequently to feature branches in Git.
    - Pull Requests (PRs) are created for merging feature branches into main development branch (e.g., `develop`).
    - PRs include links to user stories/tasks, description of changes, and evidence of passing tests.
- **Activity:** Code Reviews.
    - Mandatory peer review for all PRs before merging.
    - Reviewers check for correctness, security vulnerabilities, performance issues, adherence to coding standards, and maintainability.
    - Automated SAST scans are integrated into the PR workflow as a status check.

- **Activity:** Continuous Integration (CI).
    - CI server (CodeBuild/Jenkins) automatically triggers builds upon code commits/merges.
    - Build process includes compiling code, running unit tests, performing SAST scans, checking dependencies (SCA), and packaging artifacts (e.g., Docker images).
    - Build failures immediately notify the development team.
- **Activity:** Sprint Review.
    - Team demonstrates completed user stories (potentially shippable increment) to the Product Owner and stakeholders.
    - Feedback is gathered on the delivered functionality.
- **Activity:** Sprint Retrospective.
    - Team reflects on the sprint process – what went well, what didn't, and identifies actionable improvements for the next sprint.

Governance & Sign-offs:

- **Sprint Review Acceptance:** Product Owner formally accepts or rejects completed user stories based on acceptance criteria.
- **Code Review Approvals:** Documented approval within the Pull Request system by designated reviewers.
- **CI Pipeline Status:** Builds must pass all defined quality gates (tests, scans) before artifacts are considered eligible for deployment to test environments.

Best Practices Emphasized:

- Scrum Framework (Sprints, Ceremonies)
- Secure Coding Standards
- Test-Driven Development (TDD)
- Version Control (Git) & Branching Strategies
- Code Reviews & SAST
- Dependency Scanning (SCA)
- Continuous Integration (CI)

Exit Criteria (per Sprint): **Potentially shippable product increment demonstrated and accepted by Product Owner; All code reviewed and merged; All automated tests passing in CI pipeline.**

Phase 4: Integrated Testing & Quality Assurance

Goal: Verify the integrated system components meet functional, non-functional, security, and compliance requirements through rigorous testing in dedicated environments.

4.1 Test Environment Setup & Management

- **Activity:** Provision Test Environments (QA, Staging/UAT, Performance) using IaC. Ensure environments mirror production configuration as closely as possible.
- **Activity:** Implement Test Data Management Strategy. Populate test environments with anonymized or synthetic data. Establish processes for refreshing test data.

4.2 Test Execution Cycles (Iterative within & between Sprints)

- **Activity:** Integration Testing. Verify interactions and data flow between microservices and components in the QA environment. Automate where possible.
- **Activity:** API Testing. Test service APIs directly for functional correctness, error handling, and security (authentication/authorization checks). Automate using tools like Postman/RestAssured.
- **Activity:** End-to-End (E2E) Testing. Execute test scripts simulating user workflows across UI and backend services in the QA/Staging environment. Automate critical paths using Selenium/Cypress/Playwright.
- **Activity:** Security Testing.
    - Run DAST scans against applications deployed in test environments.
    - Perform manual security testing/validation based on threat models.
    - Coordinate third-party penetration testing against the Staging environment before major releases.

- **Activity:** Performance & Load Testing. Execute load test scripts against the dedicated Performance environment to validate response times, throughput, and resource usage under expected and peak loads. Identify and report bottlenecks.
- **Activity:** Accessibility Testing. Conduct automated scans (Axe/WAVE) and manual reviews (keyboard navigation, screen reader testing) against WCAG 2.0 AA criteria in the Staging environment.
- **Activity:** User Acceptance Testing (UAT). Facilitate UAT sessions with designated NYCPS stakeholders and end-user representatives in the Staging/UAT environment. Users execute predefined test cases and provide feedback/sign-off.
- **Activity:** Regression Testing. Execute automated regression test suites frequently (e.g., nightly, per deployment) in QA/Staging environments to catch unintended side effects of changes.

4.3 Defect Management

- **Activity:** Log all identified defects in the tracking system (Jira) with clear steps to reproduce, severity, priority, environment found, and assigned owner.
- **Activity:** Triage defects regularly (e.g., daily bug scrub meeting). Prioritize fixes based on severity and impact.
- **Activity:** Verify defect fixes in the relevant test environment before closing issues.

Governance & Sign-offs:

- **Test Plan Review:** Approval of the Master Test Plan and phase-specific test plans by QA Lead and relevant stakeholders.

- **UAT Sign-off:** Formal sign-off from NYCPS Product Owner(s) and UAT participants indicating acceptance of functionality for a given release/phase.
- **Security Test Report Review:** Review of penetration test reports and remediation plans by the Security Review Board/NYC3. Sign-off on residual risk required.
- **Performance Test Report Review:** Review and approval of performance test results against NFRs by Technical Lead/Architect.
- **Release Readiness Review:** Go/No-Go decision meeting before production deployment, reviewing test results, outstanding defects, and rollback plan. Participants include Project Management, Dev Lead, QA Lead, Ops Lead, Security Lead, Product Owner.

Best Practices Emphasized:

- Comprehensive Test Strategy (Multi-Layered)
- Test Automation at Scale
- Dedicated Test Environments (IaC Managed)
- Rigorous Security Testing (DAST, Pen Testing)
- Performance Engineering & Load Testing
- Formal User Acceptance Testing (UAT)
- Accessibility Validation
- Structured Defect Management

Exit Criteria: **Successful completion of all planned test cycles for the release candidate; UAT sign-off received; Critical/High severity defects resolved or**

have accepted risk mitigation plan; Performance and security NFRs met; Release Readiness Review approved.

Phase 5: Deployment & Release

Goal: Deploy the validated system increment to the production AWS GovCloud environment safely, reliably, and with minimal disruption, according to the phased rollout plan.

5.1 Pre-Deployment Preparation

- **Activity:** Finalize Production Infrastructure (IaC). Ensure production IaC templates are up-to-date, tested, and peer-reviewed.
- **Activity:** Pre-Deployment Checklist Review. Verify all prerequisites are met (e.g., backups complete, monitoring configured, rollback plan ready, stakeholder approvals obtained).
- **Activity:** Schedule Release Window. Coordinate deployment timing (often during off-peak hours) with OPT and relevant stakeholders. Communicate planned downtime/impact if applicable.
- **Activity:** Production Readiness Review. Final Go/No-Go decision based on testing results, risk assessment, and operational readiness.

5.2 Automated Deployment Execution

- **Activity:** Execute CI/CD Pipeline for Production. Trigger the automated pipeline to deploy application artifacts and apply IaC changes to the production environment.
- **Activity:** Utilize Deployment Strategy (Blue/Green or Canary).
    - **Blue/Green:** Provision a parallel production environment ("Green"), deploy the new version, test it, then switch traffic from the old ("Blue") environment. Allows instant rollback by switching traffic back.
    - **Canary:** Gradually roll out the new version to a small subset of users/buses, monitor closely, then incrementally increase traffic if stable. Allows limiting the blast radius of potential issues.
- **Activity:** Database Migrations/Updates. Apply necessary database schema changes or data migrations using controlled, tested scripts, often performed just before traffic shifting in Blue/Green or during initial Canary phase.

5.3 Post-Deployment Validation & Monitoring

- **Activity:** Automated Smoke Tests. Run predefined automated tests against the production environment immediately after deployment to verify critical paths.
- **Activity:** Monitoring Dashboard Review. Closely monitor application performance (latency, error rates), infrastructure metrics (CPU, memory), and logs for anomalies immediately following deployment.
- **Activity:** Manual Verification. Perform targeted manual checks of key functionality by QA or designated personnel.

- **Activity:** Rollback Trigger (if needed). If critical issues are detected via smoke tests or monitoring, initiate the predefined rollback procedure immediately.

5.4 Release Communication

- **Activity:** Notify stakeholders (NYCPS, users via appropriate channels) upon successful completion of the deployment.
- **Activity:** Publish release notes detailing new features, changes, and bug fixes.

Governance & Sign-offs:

- **Production Readiness Review Sign-off:** Formal approval from all designated stakeholders (Dev, QA, Ops, Sec, Product Owner) before deployment begins.
- **Change Control Board (CCB) Approval:** Formal approval recorded for the production deployment according to NYCPS change management policies.
- **Post-Deployment Verification Sign-off:** Confirmation from QA/Ops that smoke tests passed and system is stable post-deployment.

Best Practices Emphasized:

- Continuous Deployment (CD) Pipeline
- Infrastructure as Code (IaC) for Production
- Blue/Green or Canary Deployment Strategies

- Automated Rollbacks
- Configuration Management
- Post-Deployment Smoke Testing & Monitoring
- Formal Release Management & Communication

Exit Criteria: **Software increment successfully deployed to production; Post-deployment validation checks passed; System stable and performing as expected under load; Release communicated to stakeholders.**

Phase 6: Operations, Maintenance & Optimization

Goal: Ensure the long-term health, performance, security, and cost-effectiveness of the production system while providing ongoing support and facilitating continuous improvement.

6.1 System Monitoring & Alerting

- **Activity:** Continuous 24/7 Monitoring. Utilize CloudWatch, APM tools, and custom health checks to monitor infrastructure, application performance, security events, and key business metrics.
- **Activity:** Alert Management. Respond promptly to automated alerts based on predefined runbooks and escalation paths (aligned with SLAs).

- **Activity:** Dashboard Maintenance. Regularly review and update monitoring dashboards (CloudWatch, QuickSight, Grafana) to ensure relevance and clarity.

6.2 Incident & Problem Management

- **Activity:** Incident Response. Follow defined procedures to quickly diagnose, mitigate, and resolve production incidents. Coordinate response efforts across teams (Dev, Ops, Sec, Vendor Support).
- **Activity:** Root Cause Analysis (RCA). Conduct thorough RCAs for significant incidents to identify underlying causes and prevent recurrence.
- **Activity:** Problem Management. Track recurring incidents and implement long-term fixes or architectural improvements.

6.3 Maintenance & Security Management

- **Activity:** Patch Management. Regularly schedule, test, and deploy OS, runtime, and dependency patches (prioritizing security patches) using automated tools (e.g., Systems Manager Patch Manager) and the CI/CD pipeline.
- **Activity:** Security Log Review & Audit. Periodically review security logs (CloudTrail, WAF, GuardDuty), IAM configurations, and compliance reports (Config, Security Hub).
- **Activity:** Vulnerability Management. Regularly run vulnerability scans (Inspector) and address findings promptly. Coordinate periodic penetration tests.

6.4 Backup, Recovery & DR

- **Activity:** Backup Verification. Regularly verify the integrity and restorability of database and S3 backups.
- **Activity:** DR Testing. Conduct scheduled DR drills (at least annually) involving failover to the secondary region and validation of RPO/RTO.

6.5 Performance & Cost Optimization

- **Activity:** Performance Monitoring & Tuning. Analyze long-term performance trends. Optimize database queries, caching, resource allocation, and code based on findings.
- **Activity:** Cost Analysis & Optimization. Use AWS Cost Explorer, Trusted Advisor, and other tools to identify cost-saving opportunities (e.g., right-sizing, Reserved Instances/Savings Plans, storage tiering). Implement approved optimizations.

6.6 Continuous Improvement & Feedback

- **Activity:** Gather User Feedback. Collect feedback through support channels, surveys, and direct engagement.
- **Activity:** Operational Data Analysis. Analyze logs, metrics, and incident data to identify areas for improvement in the system or processes.
- **Activity:** Input to Backlog. Feed improvement ideas, bug fixes, and technical debt reduction tasks back into the product backlog for prioritization in future sprints.

> Governance & Sign-offs:

- **Operational Reviews:** Regular meetings (e.g., monthly) with OPT/DIIT stakeholders to review system performance, availability, security posture, incidents, and costs against SLAs.
- **Change Control Board (CCB) Approval:** Required for significant maintenance activities or configuration changes in production.
- **Security Audit Reviews:** Formal review of internal and external security audit findings and remediation plans.
- **DR Test Report Sign-off:** Approval of DR test results and any required remediation actions.

Best Practices Emphasized:

- Proactive Monitoring & Alerting
- Mature Incident & Problem Management
- Automated Patch Management
- Regular Backup & DR Testing
- Continuous Performance & Cost Optimization
- Security Operations (SecOps)
- Feedback Loops for Continuous Improvement

Exit Criteria: **This phase is ongoing throughout the system's operational life.**

Phase 7: System Retirement / Decommissioning (End of Life)

Goal: Securely and completely retire the system and its infrastructure at the end of the contract or when superseded by a replacement system, ensuring data retention/destruction requirements are met.

7.1 Planning & Communication

- **Activity:** Develop Decommissioning Plan. Outline steps, responsibilities, timeline, data archival/destruction procedures, and communication strategy. Align with Disengagement Plan (RFP requirement).
- **Activity:** Notify Stakeholders. Inform users and dependent system owners of the planned retirement date and migration plan (if applicable).

7.2 Data Archival & Destruction

- **Activity:** Final Data Backup/Export. Perform final backups of all required data. Provide data exports to NYCPS in a usable format if required for migration or long-term archival outside the system.
- **Activity:** Data Destruction. Securely delete data from databases, S3 buckets (excluding long-term archives), and other storage according to NYCPS policy and compliance requirements (e.g., cryptographic erasure, physical destruction if applicable). Document destruction process.

7.3 Infrastructure Teardown

- **Activity:** Decommission Applications. Stop application services, remove them from load balancers, and disable auto-scaling groups.
- **Activity:** Terminate Compute Resources. Terminate EC2 instances, Fargate/ECS/EKS services, Lambda functions.
- **Activity:** Delete Databases & Storage. Delete RDS instances, DynamoDB tables, ElastiCache clusters, EBS volumes (after ensuring data is backed up/destroyed appropriately). Empty and delete S3 buckets (except archives).
- **Activity:** Tear Down Networking. Remove Load Balancers, NAT Gateways, VPC Endpoints, VPN/Direct Connect connections (coordinate with network team), Security Groups, NACLs, Route Tables, Subnets, and finally the VPC itself.
- **Activity:** Clean Up IAM & Monitoring. Remove specific IAM roles/policies, CloudWatch alarms/dashboards, and other configurations related to the decommissioned system.
- **Activity:** Update IaC Repository. Mark infrastructure code as decommissioned or remove it.

7.4 Final Verification & Reporting

- **Activity:** Verify Decommissioning. Confirm all resources have been terminated and data appropriately handled.
- **Activity:** Final Report. Document the decommissioning process, data destruction confirmation, and final status.

Governance & Sign-offs:

- **Decommissioning Plan Approval:** Sign-off from NYCPS stakeholders (OPT, DIIT, Security, Legal/Compliance) on the plan before execution.
- **Data Destruction Certification:** Formal certification provided to NYCPS confirming data destruction according to policy.
- **Final Decommissioning Report Approval:** Sign-off confirming successful completion of the decommissioning process.

Best Practices Emphasized:

- Formal Decommissioning Planning
- Secure Data Handling (Archival/Destruction)
- Systematic Infrastructure Teardown (Using IaC where possible)
- Clear Communication
- Final Verification & Documentation

Exit Criteria: **All system components removed from production; Data handled according to retention/destruction policy; Final decommissioning report approved.**

SDLC Setup: Detailed Implementation Steps

Implementing the SDLC described above requires careful configuration of tools, establishment of process discipline, and fostering a collaborative culture. Below are detailed steps to set up the core components.

1. **Establish Foundational AWS GovCloud Environment:**
    1. Using the AWS GovCloud Management Console or IaC, create separate AWS accounts (or leverage Organizational Units within AWS Organizations) for DEV, TEST (QA/UAT/PERF), and PROD environments to ensure strict isolation.
    2. Configure AWS Organizations for centralized governance and billing across accounts. Apply Service Control Policies (SCPs) to enforce security guardrails (e.g., restricting regions, preventing disabling of security services).
    3. Set up AWS Budgets and Cost Anomaly Detection for cost control.
    4. Implement foundational IAM:
        1. Create specific IAM groups for roles (Developers, Testers, Ops, Security, Admins).
        2. Define granular IAM policies adhering to least privilege, attaching them to groups.
        3. Create IAM roles for AWS services (EC2, Lambda, ECS/EKS Tasks) with

minimal necessary permissions.

4. Enforce MFA for all IAM users accessing the console/API.
5. Configure SAML 2.0 federation with NYCPS IdP for human access, mapping IdP groups to IAM roles.

5. Deploy baseline network infrastructure using IaC (CloudFormation/Terraform):
    1. Define VPCs (e.g., 10.x.0.0/16 range) spanning 3 AZs for each environment.
    2. Create public subnets (for LBs, NAT GWs) and private subnets (for application/data tiers).
    3. Configure Route Tables, Internet Gateways, and NAT Gateways.
    4. Implement baseline Security Groups (e.g., allow SSH/RDP from bastion only, allow HTTP/S from LB) and restrictive Network ACLs.
    5. Provision VPC Endpoints (Interface & Gateway) for private access to services like S3, DynamoDB, KMS, Secrets Manager, etc.

6. Configure AWS Direct Connect or Site-to-Site VPN tunnels for secure connectivity to NYCPS on-premises networks, establishing BGP routing.

6. Configure core security and logging services:
    1. Enable CloudTrail in all regions within each account, configured to deliver logs to a central, encrypted S3 bucket in a dedicated Logging/Audit account (using cross-account roles). Enable Log File Validation.
    2. Enable AWS Config and deploy conformance packs (e.g., for FedRAMP compliance) to continuously monitor resource configurations.
    3. Enable GuardDuty, Security Hub, and Inspector in all relevant regions. Configure Security Hub to aggregate findings and potentially send notifications (SNS -> Lambda/Email).
    4. Configure baseline CloudWatch Logs agents/collectors for instances/containers. Set up log groups with appropriate retention policies.

5. Configure AWS WAF with baseline managed rulesets (e.g., OWASP Top 10) and associate with public-facing ALBs/CloudFront.
6. Create Customer Managed Keys (CMKs) in KMS for encrypting sensitive data (EBS, S3, RDS, etc.). Define key rotation policies.

2. **Configure Development & Collaboration Toolchain:**
    1. Set up Git repositories (e.g., in AWS CodeCommit or GitHub/GitLab):
        1. Create repositories per microservice or component group.
        2. Configure branch protection rules for `main`/`master` and `develop` branches (require PRs, require status checks, require approvals).
        3. Define and document branching strategy (e.g., feature branches created from `develop`, merged back via PR; releases tagged from `main`).
        4. Define PR template standards (link to Jira issue, summary, testing performed).
    2. Configure Project Management Tool (e.g., Jira):

1. Set up project boards (Scrum board per team).
2. Define workflow states (e.g., To Do, In Progress, Code Review, Ready for QA, In QA, Ready for UAT, In UAT, Done).
3. Configure issue types (Epic, Feature, User Story, Task, Bug).
4. Integrate Jira with Git repository to link commits/PRs to issues.
5. Set up dashboards for sprint burndown, velocity, defect tracking.

3. Configure CI/CD Platform (e.g., AWS CodePipeline):
    1. Create pipelines per microservice/application.
    2. Configure Source stage (linking to CodeCommit/GitHub).
    3. Configure Build stage (using CodeBuild): Define buildspec.yml for steps like dependency install, compile, unit tests, SAST scan (e.g., SonarQube integration), SCA scan (e.g., Snyk/OWASP DC), Docker build & push to ECR, packaging deployment

artifacts. Define quality gates (e.g., fail build if tests fail or high/critical vulnerabilities found).

4. Configure Deploy stages for Dev, Test, Staging environments (using CodeDeploy, ECS/EKS blue/green deployment actions, Lambda updates, CloudFormation/Terraform apply actions).

5. Implement manual approval gates between environments (e.g., before deploying to Staging or Prod).

4. Configure Collaboration Tools: Set up dedicated Slack/Teams channels for project communication, automated notifications from CI/CD and monitoring. Set up Confluence/SharePoint space for documentation.

5. Configure Artifact Repositories: Set up ECR for Docker images. Configure CodeArtifact or Nexus/Artifactory if needed for language-specific packages (npm, Maven, PyPI).

3. **Define & Document Standard Operating Procedures (SOPs):**
   1. **Coding Standards:** Document language-specific style guides and secure coding guidelines (referencing OWASP, NYCPS standards).

2. **Code Review Checklist:** Create a checklist for reviewers covering key areas (logic, security, performance, readability, test coverage, etc.).
3. **Testing Procedures:** Document how to write different types of tests (unit, integration, E2E), how to run them, where results are stored, and criteria for passing.
4. **Deployment Runbook:** Create detailed runbooks for deploying each major component, including pre-deployment checks, execution steps, monitoring points, validation steps, and rollback procedures.
5. **Incident Management Process:** Define severity levels, escalation paths, communication procedures, on-call rotation, RCA template, and post-mortem process.
6. **Change Management Process:** Document how changes (code, infrastructure, configuration) are requested, reviewed, approved, scheduled, and tracked, aligning with NYCPS CCB requirements.
7. **Onboarding Guide:** Create documentation for new team members covering project overview, architecture, tool setup, and key processes.
8. Store all SOPs in a central, version-controlled location (e.g., Confluence, SharePoint, Git repo).

4. **Integrate Security Scanning Tools:**

1. **SAST:** Integrate SonarQube/Checkmarx/similar into the CodeBuild stage of the CI pipeline. Configure quality gates to fail the build based on severity thresholds (e.g., fail on Critical/High vulnerabilities). Configure connection to scan results dashboard.
2. **SCA:** Integrate Snyk/OWASP Dependency-Check/similar into the CodeBuild stage. Configure to scan package manifests (package.json, pom.xml, requirements.txt). Configure quality gates based on vulnerability severity and license compliance.
3. **DAST:** Configure OWASP ZAP/Burp Suite/similar to run scans against deployed applications in Test/Staging environments, potentially triggered by the CD pipeline after a deployment. Configure reporting of findings.
4. **Container Scanning:** Configure ECR image scanning or integrate tools like Trivy/Clair into the CI pipeline after Docker image build to scan for OS and library vulnerabilities within containers.
5. Ensure scan results are reviewed regularly by the security team and findings are tracked as defects/tasks.

5. **Configure Monitoring, Logging, and Alerting:**
   1. Deploy CloudWatch Agent to EC2 instances and configure containerized applications (e.g., via Fluentd/Fluent Bit sidecar/daemonset) to forward

application logs and system metrics to CloudWatch Logs.

2. Define structured logging format for applications.
3. Create CloudWatch Log Groups with appropriate retention policies (e.g., shorter retention for debug logs, longer for audit logs).
4. Set up CloudWatch Metric Filters to extract key metrics from logs (e.g., error counts, specific event occurrences).
5. Create CloudWatch Alarms based on key metrics (CPU, memory, disk, latency, error rates, queue depth, custom application metrics) with defined thresholds and actions (e.g., trigger auto-scaling, send SNS notification).
6. Configure SNS topics for different alert severities/teams. Subscribe Ops/Dev team members (via email/SMS) or integrate with incident management tools (PagerDuty/Opsgenie).
7. Build initial CloudWatch Dashboards visualizing key health and performance metrics for each environment/service.
8. If using APM tools (Datadog, etc.), deploy agents and configure dashboards/alerts within that platform.

6. **Provision Initial Environments & Seed Data:**
   1. Run IaC scripts (CloudFormation/Terraform) to provision the baseline infrastructure for the DEV and

TEST (QA) environments based on the architecture design.

2. Deploy initial versions of core services (e.g., databases, basic APIs) via the CI/CD pipeline to DEV/TEST.

3. Develop and run scripts to populate TEST environment databases with anonymized or synthetic seed data necessary for initial development and testing.

7. **Conduct Team Onboarding & Kick-off Sprints:**
    1. Run dedicated onboarding sessions covering project goals, architecture, toolchain setup (local dev environments, Git access, Jira access, AWS access), coding standards, and SDLC processes (code reviews, testing, deployment).
    2. Ensure all team members have necessary access and permissions.
    3. Conduct the first Sprint Planning meeting, pulling initial setup tasks and potentially simple user stories into Sprint 1.
    4. Begin executing the defined Agile ceremonies (Daily Stand-ups, etc.).
8. **Establish Governance Rhythms:**
    1. Schedule and conduct initial meetings for the Steering Committee, Technical Review Board, and Security Review Board.

2. Establish regular cadence for backlog grooming, sprint reviews involving stakeholders, and process retrospectives.
3. Set up project status reporting mechanisms (e.g., automated dashboards from Jira, weekly status reports).