MLOps Lifecycle Documentation

Overview

DATA & ML PIPELINE + CI/CD + Monitoring + Retraining Loop

This documentation describes a **production-grade MLOps workflow** for flight delay prediction that implements **Continuous Integration (CI), Continuous Deployment (CD), monitoring, drift detection, automated retraining, and redeployment**. The complete lifecycle follows:

Initial Development → CI/CD Deployment → Production Monitoring → Drift Detection →

Retraining → Redeployment → Continuous Monitoring

Architecture Components

- **Data**: S3 (flight logs, historical delays), Weather APIs, Airport metadata
- **ML Framework**: **PyTorch / TensorFlow** (DelayNet model)
- **API**: FastAPI inference service
- **Orchestration**: AWS ECR/EKS (Kubernetes)
- **IaC**: **Terraform** (Infrastructure provisioning)
- **CI/CD**: Harness/GitHub Actions
- **Observability**: Prometheus + Grafana
- **Model Registry**: MLflow/Model Registry

Phase 1: DATA & ML PIPELINE (Initial Model Development)

1.1 Data Ingestion

- Load flight logs from S3 (historical + recent)

- Pull weather API data (temperature, wind, precipitation)

- Import airport metadata (runway capacity, gates)

- Read historical delays CSV (ground truth labels)

- Fetch holiday/event data (major disruptions)

1.2 Data Validation

- Schema validation (column types, required fields)

- Missing value checks (<5% threshold)

- Outlier detection (IQR method, domain rules)

- Range validation (flight numbers, distances)

- Duplicate row detection (flight_id + timestamp)

1.3 Feature Engineering

- Encode airline carrier (one-hot/target encoding)

- Extract hour-of-day, day-of-week features

- Normalize distance (min-max scaling)

- Weather severity score (composite index)

- Holiday/weekend binary flags

1.4 Model Training

- Train DelayNet using **PyTorch / TensorFlow** (binary classification)

- Adam optimizer (lr=0.001, weight_decay=1e-5)

- Early stopping (patience=10 epochs)

- Class imbalance handling (focal loss/SMOTE)

- GPU-accelerated training (batch_size=1024)

1.5 Model Evaluation

- Accuracy (>85% target)

- Recall for "delay" class (>80%)

- ROC-AUC (>0.88)

- Latency benchmark (<50ms p95)

- Robustness tests (adversarial, distribution shift)

1.6 Model Registration & Packaging

Model Registry:

- Save model_v1 (torchscript/ONNX/TensorFlow SavedModel)

- Store metrics JSON (F1, precision, recall)

- Store training params (hyperparams, git SHA)

- Store dataset version (S3 path + hash)

- Add lineage metadata (parents, tags)

Packaging:

- FastAPI inference service (/predict, /health)

- Dockerized (python:3.9-slim + model weights)

- Tag: delay-ml:model_v1

- Push to AWS ECR

Phase 2: CI/CD DEPLOYMENT PIPELINE

2.1 Infrastructure Provisioning (Terraform)

- Provision EKS cluster (terraform apply)

- Create ECR repositories (delay-ml)

- Setup IAM roles/policies for ML workloads

- Configure VPC, subnets, security groups

- Deploy Prometheus/Grafana monitoring stack

2.2 CI Stage (Build)

- Clone GitHub repo (main branch)

- Install dependencies (pip install -r requirements.txt)

- Build Docker image (delay-ml:model_v1)

- Push to AWS ECR (authenticated)

- Export image tag as pipeline artifact

2.3 CD Stage (Deploy)

- **Terraform**: Apply K8s infra changes (if needed)

- Apply Kubernetes manifests (Helm/Terraform)

- Update Deployment image field (model_v1 → model_v2)

- Rolling update (maxUnavailable=10%, maxSurge=20%)

- Wait for pods Ready (timeout=300s)

- Update Service/Ingress (traffic routing)

2.4 Verification Stage

- p95 latency check (<100ms)

- Error rate check (<1%)

- Prediction distribution (matches baseline ±10%)

- Pod health check (all Ready)

- Compare to baseline model (A/B metrics)

2.5 Promotion Decision

✅ Healthy: Auto-approve, mark "live", notify Slack

❌ Degraded: Auto-rollback, notify Slack, store metadata

Phase 3: PRODUCTION MONITORING

Real-time monitoring via Prometheus + Grafana:

MODEL METRICS:

- Data drift (KS-test, PSI > 0.1)

- Prediction drift (label shift detection)

- Latency p95/p99 (>2x degradation)

- Error rate spikes (>5% increase)

- Traffic anomalies (volume ±50%)

ALERTING:

- Drift → Trigger retraining pipeline

- SLA violations → PageOps escalation

- Infrastructure → Auto-scaling/healing

Phase 4: AUTOMATED RETRAINING (Drift-Triggered)

4.1 Data Refresh

- Last 7 days S3 flight data (fresh patterns)

- Updated weather feeds (new seasons)

- Airport changes (runway construction)

- Seasonal patterns (holidays, summer peaks)

- Real-time operational data (live disruptions)

4.2 Feature Re-computation

- Rebuild feature vectors (consistent pipeline)

- Update weather severity scoring

- Recalculate time-based features

- Re-normalize with new training statistics

- Handle new carriers/routes (online encoding)

4.3 Model Retraining

- Train DelayNet_v2 using **PyTorch / TensorFlow** (warm start from v1)

- Hyperparameter tuning (Optuna/Bayesian)

- Class re-balancing (new data distribution)

- GPU acceleration (same config)

- Save improved weights

4.4 Challenger Evaluation

Compare v2 vs current production v1:

- Accuracy improvement (>2% absolute)

- Recall improvement (>3%)

- Latency neutral (±10%)

- Robustness (better drift resilience)

- Statistical significance (bootstrap test)

Exit criteria: If v2 not superior → Stop pipeline

4.5 Registration & Packaging v2

- Register model_v2 (challenger → candidate)

- Store comparison metrics + drift reason

- Update FastAPI with v2 weights

- Build/tag: delay-ml:model_v2

- Push ECR → Trigger CI/CD

Phase 5: CONTINUOUS LOOP

model_v2 → **Terraform** (infra if needed) → CI/CD Pipeline → Production Deployment → Monitoring

The same verification gates apply to all new versions.

🔄 COMPLETE LOOP:

Monitor → [Drift] → Retrain → [Better] → Deploy → Verify → Monitor

↓ NO

Continue Monitoring

Key Benefits

MLOps Lifecycle Documentation

Capability	Implementation	Business Value
Continuous Training	Drift-triggered retraining	Adapts to changing flight patterns
Zero-Downtime Updates	Rolling K8s + **Terraform** IaC	99.99% uptime SLA
Automated Quality Gates	p95 latency + accuracy checks	Production reliability
Full Observability	Prometheus/Grafana dashboards	Instant issue detection
Model Lineage	MLflow registry + metadata	Audit trail + reproducibility
Infrastructure Automation	**Terraform** provisioning	Consistent, reproducible environments

---

This workflow ensures **production ML reliability** through automated quality gates, continuous monitoring, drift-adaptive retraining, and **Terraform-powered infrastructure consistency** while maintaining zero-downtime deployments.