RAKEZ Case Study: Deploying and Monitoring a Lead Scoring Model

Role: Machine Learning Engineer
Duration: 5-7 days
Status: ■ Complete

Executive Summary

This document presents a comprehensive solution for deploying, monitoring, and maintaining RAKEZ's lead scoring model in a production environment. The solution addresses all requirements including deployment strategy, online testing, monitoring, automation, and retraining capabilities.

Key Deliverables:
- Production-ready deployment architecture
- Online testing strategy (A/B testing and shadow deployment)
- Comprehensive monitoring and alerting system
- Automated retraining pipeline
- Real-time monitoring dashboard
- Complete documentation and presentation

---

1. Deployment Strategy

Architecture Overview

The deployment architecture leverages **Databricks** for data processing and **MLflow** for model management, with a **FastAPI** REST API for real-time serving.

Key Components

1.1 Model Registry (MLflow)

Implementation: `03_api/fastapi_app.py`, `02_notebooks/model_inference_databricks.py`

- **Stages**: Production, Staging, Archived
- **Versioning**: Automatic version tracking with metadata
- **Auditability**: Complete experiment tracking and model lineage
- **Rollback**: One-click reversion to previous versions

```
# Model loading from registry
model_uri = f"models:/{MODEL_NAME}/{STAGE}"
model = mlflow.sklearn.load_model(model_uri)
```

1.2 REST API (FastAPI)

Implementation: `03_api/fastapi_app.py`

- **Endpoint**: `/score-lead` for real-time scoring
- **Features**:

- Input validation (Pydantic models)
- Shadow model support
- Request logging
- Error handling
- Health checks

1.3 Batch Inference (Databricks)

Implementation: `02_notebooks/model_inference_databricks.py`

- Scheduled daily batch jobs
- Processes new leads from Delta Lake
- Updates CRM via Delta tables
- Handles feature engineering consistently

Deployment Methods

Canary Deployment
- Phase 1: 10% traffic (1 day)
- Phase 2: 50% traffic (2 days)
- Phase 3: 100% traffic
- Automatic rollback on failure

Shadow Deployment
- Parallel evaluation without affecting production
- Silent comparison of predictions
- Real-world performance data collection

Frameworks & Tools

- ■ **MLflow**: Model versioning and registry
- ■ **FastAPI**: REST API framework
- ■ **Databricks**: Data processing platform
- ■ **Delta Lake**: Data storage and versioning

---

2. Online Testing Approach

Strategy Overview

We implement a **two-phase testing approach**: Shadow deployment followed by A/B testing.

Phase 1: Shadow Deployment

Duration: 1-2 weeks

Implementation: `03_api/fastapi_app.py` (shadow model support)

- New model runs in parallel with production
- All traffic routed to both models
- Predictions compared silently
- Zero risk to business operations

Metrics Tracked:
- Prediction distribution differences
- Model performance (AUC, Precision, Recall)
- Business KPIs (conversion rate, revenue)

Phase 2: A/B Testing

Traffic Split: 90% production, 10% new model

Success Criteria:
- **Technical**: +2% AUC improvement OR
- **Business**: +5% conversion rate OR +5% revenue per lead
- **Operational**: No increase in error rate, latency maintained

Decision Process:
1. Monitor for statistical significance
2. Compare business metrics
3. Gather sales team feedback
4. Gradual rollout if successful

Safety Measures

- **Automatic Rollback**: Triggered if error rate > 2% or latency degrades > 50%
- **Real-time Monitoring**: Continuous tracking during testing
- **Approval Gates**: Manual review before full promotion
- **Gradual Rollout**: 10% → 50% → 100% traffic

---

3. Monitoring Plan

3.1 Data Drift Detection

Implementation: `02_notebooks/drift_detection.py`

Metrics:

PSI (Population Stability Index)
- **Calculation**: Per-feature and overall
- **Thresholds**:

- PSI < 0.1: No significant change
- PSI 0.1-0.25: Moderate change (Warning)
- PSI > 0.25: Significant change (Critical)

KL Divergence
- Measures distribution shift
- Threshold: KL > 0.1 indicates drift

Statistical Tests
- **Kolmogorov-Smirnov**: Continuous features
- **Chi-square**: Categorical features
- P-value < 0.05 indicates significant drift

Monitoring Schedule:

- **Real-time**: Feature statistics
- **Hourly**: Distribution checks
- **Daily**: PSI calculation
- **Weekly**: Comprehensive drift report

3.2 Prediction Drift

Implementation: `02_notebooks/monitoring_metrics.py`

- Model performance degradation detection
- AUC, Precision, Recall tracking over time
- Calibration metrics (Brier score)

3.3 Latency & Throughput

Metrics:
- **Latency**: P50, P95, P99 percentiles
- Target: P95 < 200ms
- Warning: P95 > 500ms
- **Throughput**: Requests per second
- Target: > 10 req/s
- **Error Rate**: HTTP errors
- Target: < 1%
- Critical: > 1%

3.4 Business Performance

Metrics:
- **Conversion Rate**: Leads → Customers
- **Revenue per Lead**: Average revenue
- **Score Distribution**: Lead score buckets
- **Conversion by Score**: Performance by score category

3.5 Alerting System

Implementation: `01_architecture/monitoring_architecture.md`

Alert Levels:
- **Level 1 (Info)**: Logged to dashboard

- **Level 2 (Warning)**: Slack notification (#ml-alerts)
- **Level 3 (Critical)**: Email + Slack + PagerDuty

Alert Triggers:
- PSI > 0.5: Critical drift
- Latency P95 > 500ms: Critical
- Error rate > 1%: Critical
- Conversion rate drop > 10%: Warning

3.6 Sales Team Complaint Investigation

Scenario: "Model scores are no longer helping prioritize leads effectively"

Investigation Workflow (6 Steps):

Immediate Response (1 hour)
- Check model health metrics
- Verify API functionality
- Review recent model changes
- Check data quality issues

Data Analysis (4 hours)
- Analyze conversion rates by score bucket
- Compare current vs historical performance
- Check for data drift (PSI, KL divergence)
- Review feature distributions

Model Performance Review (24 hours)
- Evaluate model metrics (AUC, Precision, Recall)
- Compare production vs shadow model
- Review calibration plots
- Check for concept drift

Root Cause Analysis
- **If Data Drift**: Investigate data source changes
- **If Concept Drift**: Business environment may have changed
- **If Model Issue**: Review training data and features
- **If Integration Issue**: Check CRM sync and data pipeline

Resolution
- **Data Issue**: Fix data pipeline, retrain model
- **Model Issue**: Retrain with updated data/features
- **Business Change**: Update model to reflect new patterns
- **Integration Issue**: Fix CRM sync mechanism

Communication & Prevention
- Document findings and resolution
- Update sales team with explanation
- Implement preventive measures
- Schedule follow-up review

Tools Used:
- Streamlit dashboard for metrics
- MLflow UI for model comparison
- Drift detection reports
- Conversion rate analysis

---

4. Automation, Reproducibility & Retraining

4.1 Reproducibility

Implementation: MLflow + Delta Lake + Git

Components:
- **MLflow**: Experiment tracking, model versioning, parameter logging
- **Delta Lake**: Data snapshots and versioning
- **Git**: Code and configuration version control

Data Snapshots:
- Training data stored in Delta Lake with timestamps
- Feature engineering code versioned
- Model artifacts stored in MLflow

4.2 CI/CD Workflow

Implementation: `04_ci_cd/github_actions.yaml`

Pipeline Stages:

Lint & Test
- Code formatting (Black)
- Linting (Flake8)
- Unit tests (Pytest)

Validate
- Notebook syntax validation
- Model validation checks

Deploy Staging
- Auto-deploy on develop branch
- Deploy notebooks to Databricks
- Trigger inference job

Deploy Production
- Manual approval required
- Canary deployment (10% → 50% → 100%)
- Monitor metrics
- Automatic rollback on failure

Model Registry Update
- Promote model from Staging to Production
- Update model tags and metadata

4.3 Retraining Strategy

Implementation: `02_notebooks/retraining_pipeline.py`

Triggers:

Drift-Triggered
- Automatic when PSI > 0.25
- Significant distribution shift detected

Scheduled
- Weekly for first 3 months
- Monthly thereafter

Manual
- Admin-initiated for special cases

Retraining Process:

Data Collection
- Latest 6 months of labeled data
- Minimum 10,000 records required
- Time-based train/test split

Model Training
- Hyperparameter optimization (Optuna, 50-100 trials)
- Time series cross-validation
- XGBoost/LightGBM models

Model Evaluation
- Must outperform production model
- Minimum improvement: +2% AUC OR +5% business KPI
- Shadow testing for 1-2 weeks

Model Promotion
- Register to MLflow Staging
- Manual review and approval
- Canary deployment
- Promote to Production

Rollback Mechanism:

- **Automatic**: Error rate > 2%, Latency > 50% degradation, Business KPI drop > 10%
- **Manual**: One-click rollback to previous version
- **History**: Maintains model registry history

---

5. Bonus Features

5.1 Monitoring Dashboard

Implementation: `05_dashboard/streamlit_dashboard.py`

Features:
- Real-time metrics visualization
- Drift detection charts (PSI by feature)
- Performance trends (latency, throughput)
- Business metrics (conversion rates by score bucket)
- Alert viewer
- Works in demo mode (mock data) and production mode

Tabs:
1. **Overview**: System metrics and performance trends
2. **Drift Detection**: PSI and distribution monitoring
3. **Performance**: Latency and throughput metrics
4. **Business Metrics**: Conversion rates and score distributions
5. **Alerts**: System alerts and notifications

5.2 CRM Integration

Implementation: `06_docs/detailed_readme.md` (CRM Integration Plan)

3-Phase Approach:

Phase 1: Batch Integration
- Scheduled job updates CRM scores daily
- Delta table as intermediary
- Low latency requirement

Phase 2: Real-time Integration
- FastAPI sends webhook on prediction
- CRM receives score immediately
- Higher latency requirement

Phase 3: API Integration
- CRM calls FastAPI directly
- On-demand scoring
- Highest latency requirement

Integration Methods:
- Webhook integration
- API integration
- Database integration (Delta tables)

5.3 Feedback Loop

Implementation: `06_docs/presentation_slides.md` (Slide 10)

- Sales team complaint investigation workflow
- Root cause analysis process
- Preventive measures implementation
- Regular stakeholder communication

Technical Implementation

Code Structure

```
rakez-lead-scoring-deployment/
■■■ 01_architecture/          # Architecture diagrams
■■■ 02_notebooks/             # Databricks notebooks
■■■ 03_api/                   # FastAPI application
■■■ 04_ci_cd/                 # CI/CD pipeline
■■■ 05_dashboard/             # Monitoring dashboard
■■■ 06_docs/                  # Documentation
```

Key Technologies

- **ML**: XGBoost, LightGBM, scikit-learn
- **Platform**: Databricks, MLflow

- **API**: FastAPI, Uvicorn
- **Dashboard**: Streamlit, Plotly
- **CI/CD**: GitHub Actions
- **Data**: Delta Lake, Spark

Conclusion

This solution provides a **complete, production-ready system** for deploying and monitoring RAKEZ's lead scoring model. All assessment requirements have been met:

■ **Deployment Strategy**: Complete with MLflow, FastAPI, and Databricks
■ **Online Testing**: Shadow deployment and A/B testing implemented
■ **Monitoring Plan**: Comprehensive drift detection, performance metrics, and alerting
■ **Automation & Retraining**: CI/CD pipeline and automated retraining
■ **Bonus Features**: Working dashboard, CRM integration, feedback loop

The solution is **enterprise-grade**, **well-documented**, and **ready for production deployment**.

Appendix

Files and Locations

- **Architecture Diagrams**: `01_architecture/`
- **Production Code**: `02_notebooks/`, `03_api/`
- **CI/CD Pipeline**: `04_ci_cd/github_actions.yaml`
- **Dashboard**: `05_dashboard/streamlit_dashboard.py`
- **Documentation**: `06_docs/`

Quick Start

```
# Setup
python setup.py

# Start services
python start.py

# Access
# - API: http://localhost:8000/docs
# - Dashboard: http://localhost:8501
```

End of Case Study