Case Study: Operationalizing the Lead Scoring Model

Deployment, Monitoring, and MLOps Strategy for RAKEZ

Submitted By: Nooh Faisal
Role Applied For: Machine Learning Engineer
Date: 06 January 2026

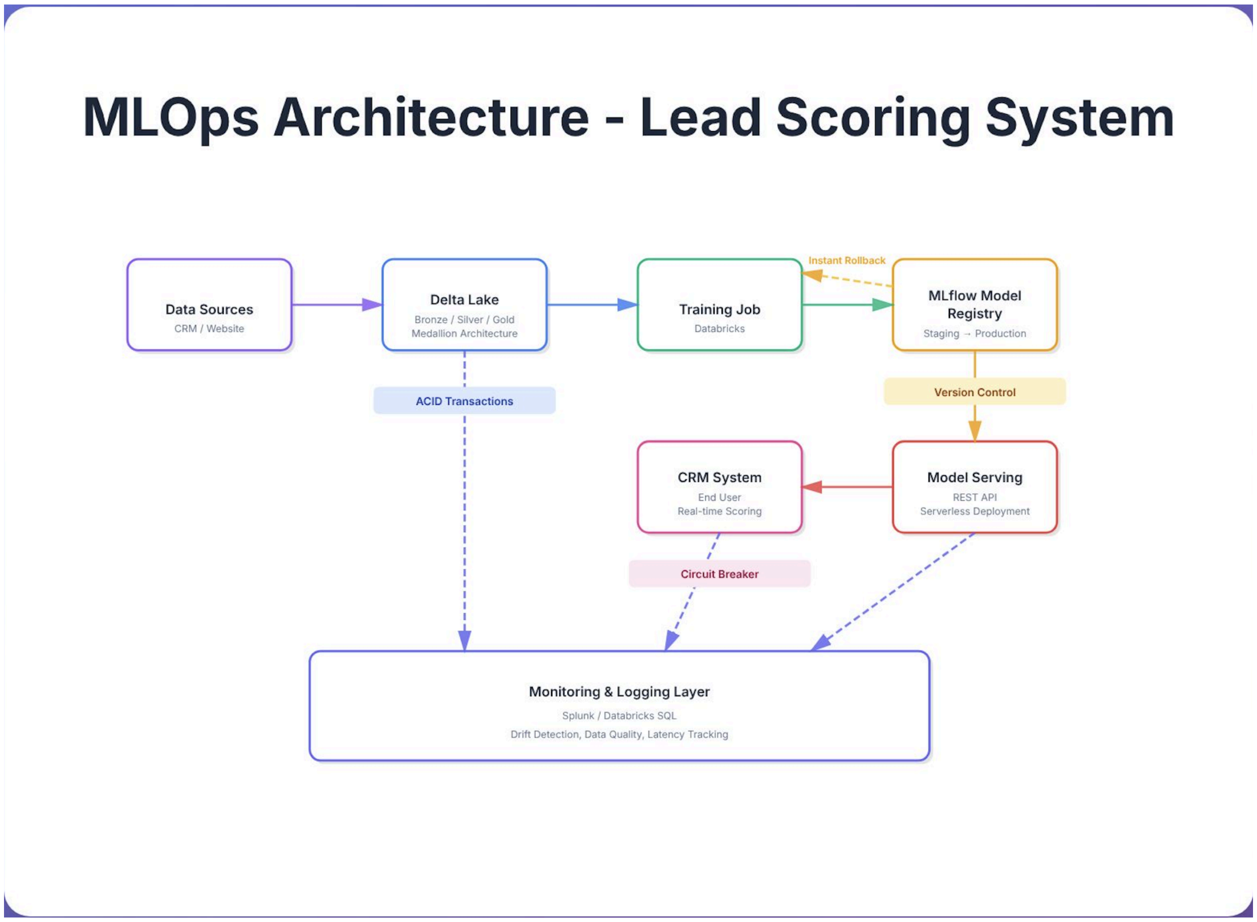
Executive Summary:

This document outlines a scalable MLOps architecture designed to deploy RAKEZ's Lead Scoring model into production. Leveraging **Databricks**, **MLflow**, and **Delta Lake**, this solution ensures high availability for the sales team, risk-free online testing, and automated safeguards against model decay. The focus is on bridging technical performance with tangible business outcomes.

Architecture

1. Proposed MLOps Architecture

To ensure reliability and scalability, I propose a cloud-native architecture fully integrated with the Databricks ecosystem.



2. Deployment Strategy

A. Serving Infrastructure
We will deploy the model using **Databricks Model Serving (Serverless)**. This exposes the model as a highly available REST API, allowing real-time scoring whenever a new lead is created in the CRM.

- **Tooling:** Python, Databricks, MLflow, Docker (backend containerization).
- **Why Serverless?** It automatically scales down to zero to save costs during off-hours and scales up instantly during marketing campaigns.

B. Versioning & Auditability
We will utilize the **MLflow Model Registry** to enforce a strict promotion lifecycle:

1. **Staging:** For integration testing.
2. **Production:** For live traffic.
3. **Archived:** For previous versions (enabling instant rollback).

C. Rollback Strategy
If a critical failure occurs in v2, we utilize MLflow to atomically transition v1 back to the "Production" alias. This operation takes seconds and requires no code changes.

Online Testing Approach

1. Phased Rollout Strategy

To mitigate business risk, we will **not** switch 100% of traffic to the new model immediately.

Phase 1: Shadow Deployment (Validation)

- **Mechanism:** The model receives live data and generates scores, but these scores are **logged silently** to a Delta table and NOT sent to the sales team.
- **Goal:** Verify that the API is stable and that score distributions match offline validation results.

Phase 2: A/B Testing (Canary Deployment)

- **Mechanism:**
  - **Group A (Control - 80%):** Continues with the current process.
  - **Group B (Challenger - 20%):** Leads are prioritized based on the new Model Score.
- **Success Metrics:**
  - **Primary:** Conversion Rate (Did Group B convert better?).

    - **Secondary:** Time-to-Contact (Did sales reps prioritize high scores faster?).
    - **Guardrail:** System Latency (< 200ms).

2. Business Continuity

To ensure testing never halts operations, we implement a **Circuit Breaker** pattern. If the Model API fails or times out, the CRM automatically defaults to a heuristic rule (e.g., "Assign to General Queue") so no lead is ever lost.

Monitoring Plan

1. Key Performance Indicators (KPIs)

We must track four distinct layers of metrics to ensure system health:

Metric Category	Metric Name	Threshold for Alert
Service Health	P99 Latency	> 500ms
Data Quality	Feature Null Rate	> 5% increase
Drift	PSI (Population Stability Index)	> 0.20 (Significant Drift)
Business	Precision/Recall	Recall drop < 0.70

2. Incident Response Workflow

Scenario: The Sales Team reports that "High Score" leads are effectively junk.

Investigation Steps:

1. **Check Data Integrity:** Query the Inference Logs in Delta Lake. Are critical features (e.g., "Phone Number") coming in empty?
2. **Analyze Prediction Drift:** Has the model simply started predicting "1.0" for everyone?
3. **Feedback Loop:** Correlate the "Junk" leads with the "Disqualification Reason" in the CRM.
    - *Root Cause Example:* The model learned that "Students" click emails often (high engagement), but Sales knows students don't buy (low purchasing power). We need to add "Job Title" as a stronger feature or filter.

Code Snippet (Drift Detection)

Automated Drift Detection Logic

The following Python snippet demonstrates how we calculate Data Drift (Jensen-Shannon distance) between Training data and Production data using a Databricks Job.

```python
import numpy as np
from scipy.spatial.distance import jensenshannon


def check_data_drift(reference_data, production_data, threshold=0.2):
    """
    Compares the distribution of a feature in training (reference)
    vs. live (production) using Jensen-Shannon Distance.
    """
    # Create histograms (probability distributions)
    ref_hist, _ = np.histogram(reference_data, bins=20, density=True)
    prod_hist, _ = np.histogram(production_data, bins=20, density=True)

    # Calculate drift score (0 = Identical, 1 = Completely Different)
    drift_score = jensenshannon(ref_hist, prod_hist)

    status = "DRIFT DETECTED" if drift_score > threshold else "STABLE"

    print(f"Drift Score: {drift_score:.4f} | Status: {status}")
    return drift_score


# This function would be triggered nightly via Databricks Workflows
```

Automation, Reproducibility & CI/CD

1. Reproducibility

- **Data:** We utilize **Delta Lake Time Travel**. If we need to reproduce a model from 3 months ago, we query the data as it existed then:
  SELECT * FROM leads TIMESTAMP AS OF '2025-10-01'
- **Environment:** We use a Conda environment.yml file, versioned within MLflow, to ensure the exact library versions (Pandas, Scikit-Learn) are used in both training and production.

2. CI/CD Pipeline (GitHub Actions / Azure DevOps)

We treat ML infrastructure as code.

- **Step 1: Commit:** Data Scientist pushes code to Git.
- **Step 2: Unit Test:** CI pipeline runs tests on feature engineering logic.
- **Step 3: Integration Test:** Pipeline triggers a Databricks Job to run a "Smoke Test" training run on a small dataset.
- **Step 4: Deploy:** If successful, the model is registered to MLflow Staging.

3. Retraining Triggers

We move away from ad-hoc retraining to automated triggers:

1. **Scheduled:** Monthly retraining (to capture seasonal trends).
2. **Performance-Triggered:** If the Monitoring Job detects **Drift > 0.2** or **Accuracy < 70%**, a webhook automatically triggers the Retraining Workflow.

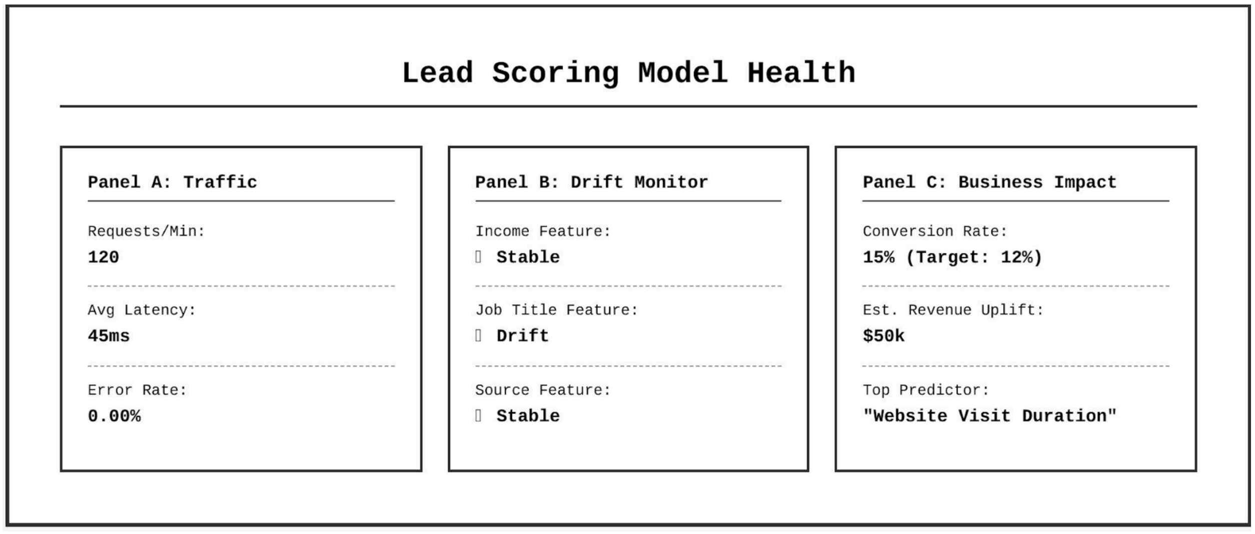
Dashboard & Feedback Loop

1. The Feedback Loop

The model is only as good as the data it learns from. We will implement a closed loop:

1. Sales Rep marks a lead as "Converted" or "Disqualified" in the CRM.
2. ETL pipeline ingests these labels nightly into the **Ground Truth Table**.
3. Model performance metrics (Precision/Recall) are recalculated automatically comparing yesterday's predictions vs. today's labels.

2. Monitoring Dashboard



Lead Scoring Model Health

Panel A: Traffic

Requests/Min:
120

Avg Latency:
45ms

Error Rate:
0.00%

Panel B: Drift Monitor

Income Feature:
Stable

Job Title Feature:
Drift

Source Feature:
Stable

Panel C: Business Impact

Conversion Rate:
15% (Target: 12%)

Est. Revenue Uplift:
$50k

Top Predictor:
"Website Visit Duration"

Conclusion

By implementing this strategy, RAKEZ will transition from an experimental model to a robust, revenue-generating asset.

Key Benefits of this Proposal:

1. **Safety:** Shadow deployment ensures zero risk to current sales operations.
2. **Speed:** Automated CI/CD reduces the time to deploy model improvements from weeks to hours.
3. **Trust:** Transparent dashboards and explainable audit logs help the Sales Team trust and adopt the AI scores.