

# COELHO REALTIME

A KUBERNETES-NATIVE MLOPS PLATFORM THAT  
FUSES INCREMENTAL LEARNING WITH BATCH  
INTELLIGENCE

*TRANSACTION FRAUD DETECTION  
ESTIMATED TIME OF ARRIVAL  
E-COMMERCE CUSTOMER INTERACTIONS*

*INCREMENTAL MACHINE LEARNING  
BATCH MACHINE LEARNING  
DELTA LAKE SQL*

RAFAEL COELHO

# ABOUT THE PROJECT

---

COELHO RealTime is a demonstrative, Kubernetes-native MLOps (Machine Learning Operations) platform, built to demonstrate production-grade MLOps capabilities through simulated real-world data and scenarios.

It masterfully combines **Incremental Learning** with **Batch Intelligence** via a unique dual-pipeline, enabling both real-time model adaptation and robust batch training.

This platform addresses critical challenges in areas like **Transaction Fraud Detection**, **Estimated Time of Arrival**, and **E-commerce Customer Interactions**.

Orchestrated on K3d with end-to-end observability from **Prometheus**, **Grafana**, and **MLflow**, COELHO RealTime ensures the demonstration of accurate, transparent, and continuously optimized machine learning models for dynamic operational demands.



# PLATFORM ARCHITECTURE

## 01.

### INCREMENTAL & BATCH MACHINE LEARNING

The platform combines two complementary machine learning paradigms into a unified dual-pipeline architecture.

The Machine Learning Layer provides real-time model adaptation through River's incremental learning and robust batch training via Scikit-Learn and CatBoost.

The Data & Streaming Layer feeds these models through Apache Kafka event streaming and Apache Spark Structured Streaming, persisting all data as Delta Lake tables on MinIO.

The Application Services Layer exposes model training, predictions, and diagnostics through FastAPI, with Redis caching live models and MLflow tracking every experiment.

## 02.

### INFRASTRUCTURE AS CODE & MICROSERVICES

The entire platform — 15 containerized microservices — is provisioned declaratively through Terraform and orchestrated on Kubernetes via K3d.

The Container Orchestration Layer establishes the cluster foundation, while the GitOps & CI/CD Layer automates the complete delivery pipeline from GitLab CI builds through ArgoCD deployments with automated image updates.

The Observability & Monitoring Layer ensures operational transparency through Prometheus metrics collection, Grafana visualization, and Alertmanager with dependency-aware inhibition rules, all backed by PostgreSQL metadata persistence.

# 01.

## INCREMENTAL & BATCH MACHINE LEARNING

---



The platform combines two complementary machine learning paradigms into a unified dual-pipeline architecture.

River's incremental learning enables models to adapt in real-time as Kafka streams arrive, while Scikit-Learn and CatBoost deliver robust batch training on historical Delta Lake datasets.

Both pipelines share the same data, the same tracking (MLflow), and the same serving layer (FastAPI) — ensuring every model is continuously optimized whether learning from one event or one million rows.

COELHO  
REALTIME

# MAIN APPLICATIONS

---



## Transaction Fraud Detection

This demonstrative application showcases an adaptive MLOps platform that detects and mitigates fraud in real-time, simulating real-world scenarios to protect assets and maintain trust.



## Estimated Time of Arrival

This demonstrative application, simulating real-world data, provides highly accurate and dynamic ETAs by leveraging real-time data and predictive analytics, optimizing logistics and boosting customer satisfaction.



## E-Commerce Customer Interactions

A demonstrative application, this project simulates real-world e-commerce scenarios to showcase how personalized recommendations and dynamic content can transform customer journeys, driving engagement, fostering loyalty, and increasing conversion rates.

# MAIN APPLICATIONS



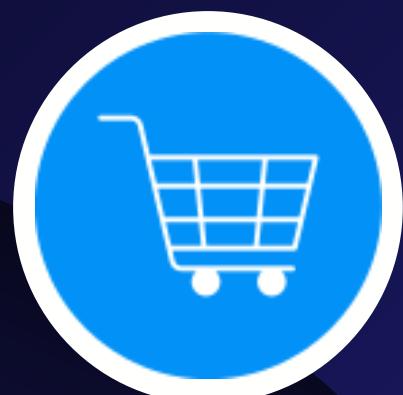
## TRANSACTION FRAUD DETECTION

- **Dual-Pipeline Model Deployment:** Showcases combining Incremental Learning (River) for real-time fraud pattern adaptation with Batch Intelligence (Scikit-Learn and CatBoost) for robust historical analysis.
- **Kafka-driven Data Ingestion:** Demonstrates real-time transaction data streaming using Apache Kafka producers.
- **FastAPI Prediction Service:** Exposes TFD model predictions via FastAPI endpoints for low-latency inference.
- **Comprehensive Observability:** Fraud detection metrics are monitored with Prometheus, visualized in Grafana, and model experiments are tracked in MLflow.



## ESTIMATED TIME OF ARRIVAL

- **Dual-Pipeline Model Deployment:** Utilizes Incremental Learning (River) for dynamic ETA adjustments based on live changes, and Batch Intelligence (Scikit-Learn and CatBoost) for foundational ETA model training.
- **Kafka-driven Data Streams:** Simulates and streams real-time data (e.g., traffic, weather) influencing ETAs via Apache Kafka.
- **FastAPI Prediction Service:** Provides ETA predictions through FastAPI endpoints.
- **Comprehensive Observability:** ETA model performance and system health are monitored with Prometheus, visualized in Grafana, and experiments are managed in MLflow.



## E-COMMERCE CUSTOMER INTERACTIONS

- **Dual-Pipeline Model Deployment:** Combines Incremental Learning (River) for adapting to changing customer preferences and behavior with Batch Intelligence (Scikit-Learn) for comprehensive historical behavioral analysis.
- **Kafka-driven User Data:** Simulates and streams real-time e-commerce interaction data via Apache Kafka producers.
- **FastAPI Prediction Service:** Serves ECCI-related predictions through FastAPI.
- **Comprehensive Observability:** Customer interaction model metrics are tracked using Prometheus, visualized in Grafana, and experiment management with MLflow.

# FOUNDATIONAL MLOPS COMPONENTS

---



## Incremental Machine Learning

Leveraging frameworks like River, this component enables models (TFD, ETA, ECCI) to continuously learn from real-time data streams. This ensures dynamic adaptation and sustained accuracy, effectively combating concept drift in evolving environments.



## Batch Machine Learning

This component establishes robust model baselines and deep insights by training on extensive historical datasets using Apache Spark, Scikit-Learn and CatBoost. It provides a powerful, scalable foundation for all TFD, ETA, and ECCI applications.



## Delta Lake SQL

Powered by Delta Lake and accessible via SQL on DuckDB, this foundational layer serves as the central, reliable, and consistent data repository. It seamlessly integrates and supports both high-throughput streaming (from Kafka) and large-scale batch processing for all project data.

02.

## INFRASTRUCTURE AS CODE & MICROSERVICES

---



- 1 - CONTAINER ORCHESTRATION LAYER
- 2 - GITOPS & CI/CD LAYER
- 3 - DATA & STREAMING LAYER
- 4 - APPLICATION SERVICES LAYER
- 5 - OBSERVABILITY & MONITORING LAYER

# 1 - CONTAINER ORCHESTRATION LAYER

---



## TERRAFORM

- Provisions K3d cluster with 1 server + 3 agent nodes through declarative HCL configuration
  - Manages Kubernetes, Helm, and K3d providers in a single state file, ensuring consistent cluster setup
  - Configures host volume mounts for persistent storage, enabling data survival across cluster rebuilds
  - **Why Terraform:** Eliminates OS-specific scripts, providing an agnostic Infrastructure-as-Code approach that works across Linux, Windows, and macOS without requiring translations
- 



## K3D

- Runs lightweight K3s distributions inside Docker containers, replicating production Kubernetes locally
  - Provides built-in container registry for local image distribution during CI/CD workflows
  - Supports multi-node clusters (1 server + 3 agents) with configurable NodePort mappings for service access
  - **Why K3d:** Enables a production-like Kubernetes environment on a single development machine with minimal resource overhead, bridging local development and cloud deployment
- 



## KUBERNETES

- Orchestrates all 15 containerized services with declarative manifests, managing scheduling, networking, and lifecycle
  - Handles service discovery via ClusterIP and external access via NodePort, connecting frontend to backend seamlessly
  - Enforces liveness/readiness probes, resource quotas, and automatic pod restart for self-healing operations
  - **Why Kubernetes:** The industry-standard orchestration platform that provides the reliability, scalability, and operational patterns required for a production-grade microservice architecture
- 



## DOCKER

- Containerizes all custom services (FastAPI, SvelteKit, Kafka Producers) with optimized single-stage Dockerfiles
  - Provides BuildKit-accelerated layer caching for rapid image rebuilds during development
  - Serves as the runtime foundation for K3d's Kubernetes nodes, running the entire cluster in containers
  - **Why Docker:** Standardizes every service into portable, reproducible containers – a prerequisite for Kubernetes orchestration and consistent behavior across development and production
-

# 2 - GITOPS & CI/CD LAYER

---



## GITLAB

- Ships a CI/CD-ready `.gitlab-ci.yml` that builds three Docker images (FastAPI, SvelteKit, Kafka Producers) via Docker-in-Docker, pluggable into any external GitLab instance
  - Tags every image with commit SHA and latest, pushing to the K3d local registry for full build traceability
  - Triggers automated builds on every master branch commit, bridging code changes to the deployment pipeline within seconds
  - **Why GitLab:** Portable pipeline definitions that plug into any GitLab instance, keeping the build-to-registry workflow environment-agnostic
- 



## ARGOCD

- Connects to an external ArgoCD via a pre-configured Application manifest declaring the complete Helm chart deployment for the coelho-realtime namespace
  - Auto-syncs cluster state to match Git with automated pruning, self-healing, and rollback across all 15 microservices
  - Paired with ArgoCD Image Updater, which watches the K3d registry for new images and commits updated tags back to Git
  - **Why ArgoCD:** Ready-to-apply GitOps manifests that transform any external ArgoCD into a fully declarative, automated deployment pipeline
- 



## SKAFFOLD

- Orchestrates local development with hot-reload file sync for Python and TypeScript directly into running pods
  - Manages port forwarding for 13 services simultaneously, exposing the complete platform on localhost
  - Intelligent caching and selective rebuilds based on file change detection
  - **Why Skaffold:** Eliminates the rebuild-redeploy cycle, enabling sub-second code changes in production-identical container environments
- 



## HELM

- Packages all 15 microservices into a single umbrella chart with seven subchart dependencies (MLflow, Redis, MinIO, PostgreSQL, kube-prometheus-stack, Kafka, Spark)
  - Declarative `values.yaml` centralizes configuration for every service: resource limits, ports, environment variables, and feature toggles
  - Conditional dependency enablement allows selective service deployment via simple boolean flags
  - **Why Helm:** The standard Kubernetes package manager that turns the entire platform into a single, versioned, reproducible deployment unit
-

# 3 - DATA & STREAMING LAYER

---



## APACHE KAFKA

- Central event backbone in KRaft mode (no ZooKeeper) across three topics: transaction\_fraud\_detection, estimated\_time\_of\_arrival, and e\_commerce\_customer\_interactions
- Fed by Python 3.13 producers generating realistic synthetic data (fraud patterns, geospatial deliveries, behavioral signals)
- Auto-creates topics with 3 partitions for parallel consumption by River incremental learners and Spark batch processors
- **Why Kafka:** Decouples data generation from consumption, letting the same stream feed both real-time and batch ML simultaneously



## APACHE SPARK

- Runs three Structured Streaming jobs (kafka\_to\_delta\_tfd.py, kafka\_to\_delta\_eta.py, kafka\_to\_delta\_ecommerce.py) consuming Kafka topics into Delta Lake tables
- Configured with Delta Lake 4.0.0 JARs and Hadoop-AWS connector for native MinIO S3 compatibility
- Deployed via Kubernetes ConfigMaps managed as part of the Helm chart lifecycle
- **Why Spark:** Transforms raw Kafka streams into queryable Delta Lake tables, powering batch ML training with Scikit-Learn and CatBoost



## DELTA LAKE

- Stores all streaming data as versioned, ACID-compliant tables on MinIO, supporting both streaming writes and batch reads
- Enables time-travel queries and schema enforcement for data consistency across ML workloads
- Queryable via DuckDB SQL through FastAPI endpoints, providing ad-hoc lakehouse access without Spark sessions
- **Why Delta Lake:** Unifies streaming and batch data into a single lakehouse format with ACID transactions, eliminating separate data stores



## MINIO

- Provides S3-compatible storage hosting two buckets: lakehouse for Delta Lake tables and mlflow-artifacts for experiment outputs
- Persists data via hostPath volume mount (/data/minio), ensuring survival across pod restarts and cluster rebuilds
- Exposes S3 API on port 9000, integrating seamlessly with Spark, MLflow, and any S3-compatible tooling
- **Why MinIO:** Production-grade S3 storage locally, enabling cloud-identical patterns without external dependencies or costs

# 4 - APPLICATION SERVICES LAYER

---



## FASTAPI

- Unified ML backend with three versioned API groups: /api/v1/incremental (River), /api/v1/batch (Scikit-Learn/CatBoost), /api/v1/sql (DuckDB queries)
- Manages training subprocesses with orjson serialization and Redis-backed model caching for low-latency predictions
- Instrumented with Prometheus metrics for request latency, throughput, and ML pipeline monitoring
- **Why FastAPI:** Single entry point unifying incremental learning, batch training, and data querying into one coherent async API surface



## SVELTEKIT

- Interactive frontend with SvelteKit 2.20, Vite 6.0, TailwindCSS 4.0, and Plotly.js for ML visualizations
- Dedicated UI sections for TFD, ETA, and ECCI with training controls, predictions, metrics dashboards, and SQL editors
- TypeScript strict mode with pre-bundled Vite caching for optimized container startup
- **Why SvelteKit:** Reactive framework delivering instant UI updates for real-time ML metrics with minimal bundle overhead



## REDIS

- Caches live incremental ML models in memory for sub-millisecond prediction serving with serialized River model state
- Manages training status flags for real-time frontend progress display (LIVE/FINISHED) per application
- ClusterIP service with configurable memory limits and eviction policies
- **Why Redis:** In-memory speed bridging model training (seconds) and prediction latency (milliseconds) in the incremental pipeline



## MLFLOW

- Tracks every experiment with parameters, metrics, and artifacts across both incremental and batch training pipelines
- Metadata in PostgreSQL, artifacts on MinIO S3 – persistent history and full reproducibility
- Tracking UI with run comparison and model registry integrated into the SvelteKit frontend
- **Why MLflow:** Complete ML lifecycle management making every training run auditable, comparable, and reproducible

# 5 - OBSERVABILITY & MONITORING LAYER

---



## PROMETHEUS

- Collects metrics from all 15 microservices via ServiceMonitors at 30-second intervals, centralizing operational and ML telemetry
- Deploys via kube-prometheus-stack (v80.6.0) with 800+ alerting rules covering infrastructure health and resource utilization
- Includes Node Exporter, kube-state-metrics, and Prometheus Operator for declarative monitoring
- **Why Prometheus:** Cloud-native monitoring standard providing pull-based collection, PromQL querying, and alerting at scale



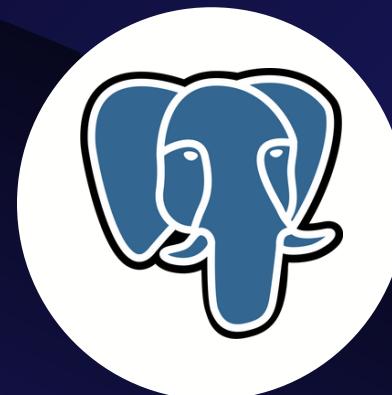
## GRAFANA

- Custom dashboards for ML pipeline throughput, API latency, and infrastructure health visualization
- Shares PostgreSQL backend with MLflow for unified metadata storage and reduced footprint
- Dashboard-as-code provisioning through Helm configuration for version-controlled monitoring
- **Why Grafana:** Transforms raw Prometheus metrics into actionable visual intelligence for rapid pipeline diagnosis



## ALERTMANAGER

- Severity-based alert routing with dependency-aware inhibition rules preventing alert storms from cascading failures
- Configured with multi-channel notification for ML pipeline failures and infrastructure degradation
- Integrates with Prometheus rules for proactive incident detection before user impact
- **Why Alertmanager:** Converts detected anomalies into actionable notifications, surfacing issues before manual discovery



## POSTGRESQL

- Shared relational backend for MLflow experiment tracking and Grafana dashboard/user storage
- Persists via hostPath mount (/data/postgresql) ensuring metadata survival across restarts and rebuilds
- ClusterIP service with configurable resource limits and connection pooling
- **Why PostgreSQL:** ACID-compliant relational storage for persistent, queryable metadata across ML tracking and monitoring

# PYTHON ECOSYSTEM

## 1 - ML FRAMEWORKS



River



Scikit-Learn



CatBoost



Imbalanced-learn



PySpark



MLflow

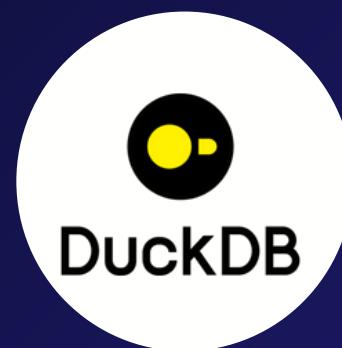
## 2 - DATA & STORAGE



Pandas



NumPy



DuckDB

## 3 - VISUALIZATION & DIAGNOSTICS



YellowBrick



Scikit-plot



UMAP

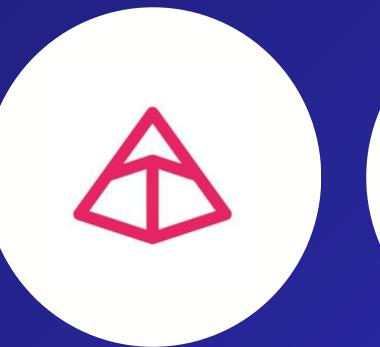


NLTK

## 4 - WEB FRAMEWORK



FastAPI

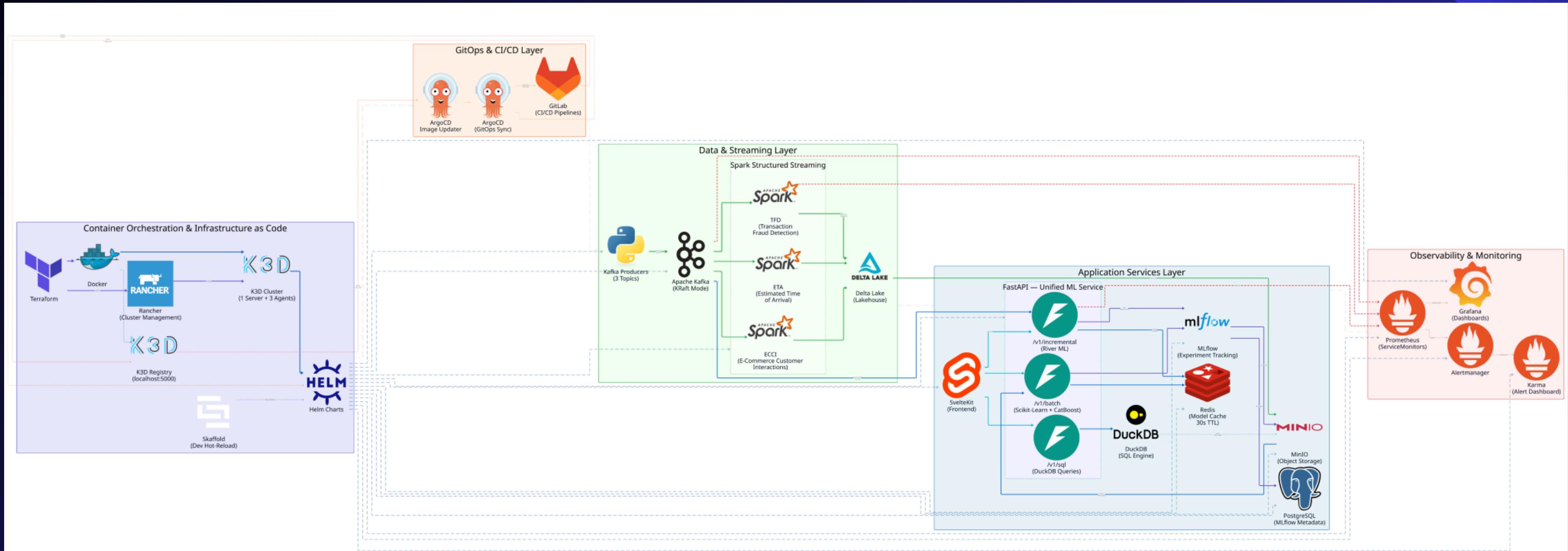


Pydantic



Redis

# PLATFORM ARCHITECTURE



# THE PLATFORM IN ACTION

---

A visual walkthrough of the COELHO RealTime platform — from its three demonstrative applications (Transaction Fraud Detection, Estimated Time of Arrival, and E-Commerce Customer Interactions) to its microservices infrastructure (Kafka, Spark, MLflow, Grafana, ArgoCD) — showcasing Incremental Machine Learning, Batch Machine Learning, and Delta Lake SQL capabilities running end-to-end on Kubernetes.



# TRANSACTION FRAUD DETECTION

---

Incremental Machine Learning  
Batch Machine Learning  
Delta Lake SQL

- **Incremental ML model:** Adaptive Random Forest Classifier (River)
- **Batch ML model:** CatBoost Classifier

# TRANSACTION FRAUD DETECTION - INCREMENTAL MACHINE LEARNING

COELHO  
RealTime

Home Applications Services

## Transaction Fraud Detection

Real-time fraud detection using streaming ML

Incremental ML Batch ML Delta Lake SQL

Real-time ML Training (1) Toggle to start processing live data

MLflow (2) Adaptive Random Forest Classifier (River)

### Prediction Result

MLflow FINISHED Run: 140e8c97 Started: 2026-01-25 02:37:42 (5)

#### Transaction Details

Predict (3) Randomize

Amount (1-6000) 3939.3 Account Age (0-1825) 2203 Currency AUD

Date 01/25/20: Time 2:33 PM Merchant ID (1-200) 142

Category travel Type deposit Payment paval

Latitude (29.5-30.1) 29.950574 Longitude (-95.8--95) -95.598269 Browser Chrome

OS Android CVV Provided Billing Address Match

Classification FRAUD % Fraud 77.80% Not Fraud 22.20%

Fraud Probability 50% 25% 75% 0% 100%

Transaction ID: txn\_dc9d7acb3067  
User ID: user\_8910  
IP Address: 135.191.128.29

## 1 Real-time ML training toggle

Activates or deactivates the continuous, real-time adaptation of the TFD model, enabling it to learn incrementally from incoming data streams.

## 2 MLflow model page - experiment run

Provides a summary of a specific incremental learning experiment for TFD on MLflow service page, detailing logged parameters, metrics, and associated artifacts within MLflow.

## 3 Prediction button

Initiates an immediate fraud prediction using the TFD model's current, incrementally updated state, demonstrating low-latency inference.

## 4 Prediction tab

Displays the interface where real-time fraud prediction outputs are presented.

## 5 MLflow experiment run informations

Offers status info (LIVE | FINISHED), MLflow run ID, and the datetime the last real-time ML model training was executed.

## 6 Prediction results

Presents the output of the TFD model's real-time fraud predictions

COELHO  
REALTIME

# TRANSACTION FRAUD DETECTION - INCREMENTAL MACHINE LEARNING

**COELHO RealTime**

**Transaction Fraud Detection**  
Real-time fraud detection using streaming ML

**Incremental ML** (selected), **Batch ML**, **Delta Lake SQL**

**Real-time ML Training**  
Toggle to start processing live data  
**Adaptive Random Forest Classifier (River)**, **MLflow**

**Transaction Details**

**Predict**, **Randomize**

Amount (1-6000): 3939.3, Account Age (0-1825): 2203, Currency: AUD

Date: 01/25/20, Time: 2:33 PM, Merchant ID (1-200): 142

Category: travel, Type: deposit, Payment: daval

Latitude (29.5-30.1): 29.950574, Longitude (-95.8--95): -95.598269, Browser: Chrome

OS: Android, CVV: Provided, Billing: Address Match

Transaction ID: txn\_dc9d7acb3067, User ID: user\_8910, IP Address: 135.191.128.29

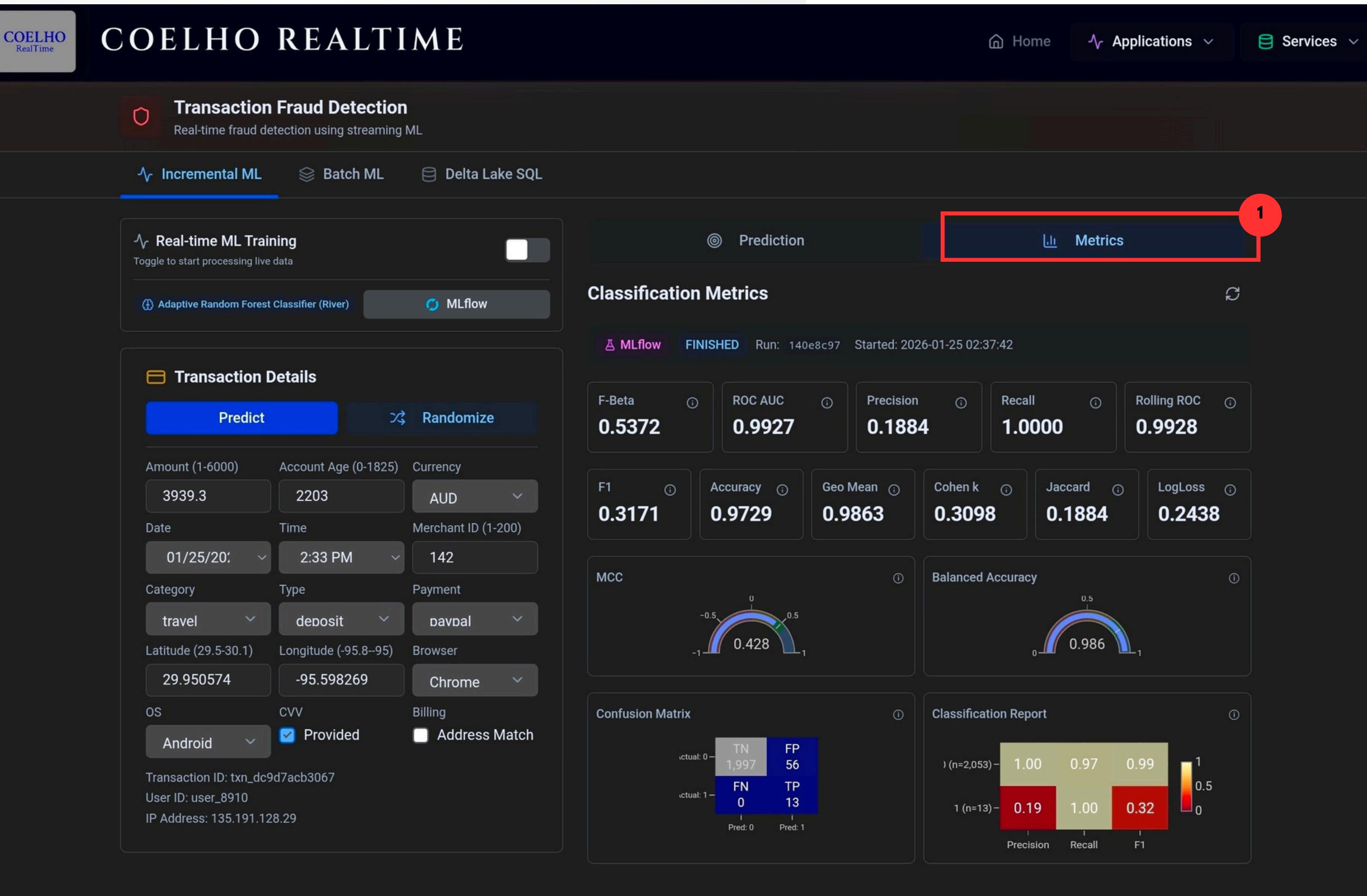
**Prediction**

**Metrics** (highlighted with a red box and a red circle with the number 1)

**Classification Metrics**

MLflow FINISHED Run: 140e8c97 Started: 2026-01-25 02:37:42

F-Beta: 0.5372	ROC AUC: 0.9927	Precision: 0.1884	Recall: 1.0000	Rolling ROC: 0.9928	
F1: 0.3171	Accuracy: 0.9729	Geo Mean: 0.9863	Cohen k: 0.3098	Jaccard: 0.1884	LogLoss: 0.2438
MCC: 0.428	Balanced Accuracy: 0.986				
Confusion Matrix:	Classification Report:				
Actual: 0 Pred: 0: TN 1,997, FP 56 Actual: 1 Pred: 0: FN 0, TP 13	(n=2,053) 1.00, 0.97, 0.99 (n=13) 0.19, 1.00, 0.32				



## 1 Metrics tab

Presents the real-time metrics and incremental learning status of the TFD model.

## Metrics (River library)

- Recall
- Precision
- F1
- F-Beta
- Accuracy
- Balanced Accuracy
- Matthews Correlation Coefficient
- Geometric Mean
- Cohen Kappa
- Jaccard
- ROC-AUC
- Rolling ROC-AUC
- Log Loss
- Confusion Matrix
- Classification Report

**COELHO**  
**REALTIME**

# TRANSACTION FRAUD DETECTION - BATCH MACHINE LEARNING

The screenshot shows the COELHO REALTIME Transaction Fraud Detection interface. The main navigation bar includes 'Home', 'Applications', and 'Services'. The 'Batch ML' tab is selected. On the left, there's a sidebar for 'MLflow Run' (CatBoost Classifier) and 'Batch ML Training' (Model trained and ready). The 'Data' section shows 'Max Rows' set to 10000. Below this is the 'Transaction Details' section with fields for Amount, Date, Category, Latitude, OS, and Transaction ID. At the bottom, there are user details like User ID and IP Address. The central area features a 'Prediction' tab (highlighted with a red box), an 'MLflow experiment run information' card (containing run ID, model, start time, and samples), a 'Prediction Result' card (showing a medium risk score of 56.5%), and a 'Prediction button' (highlighted with a red box). A large red box highlights the 'Prediction' tab and its associated components.

- 1 Train button
- 2 MLflow run ID - ML model
- 3 MLflow
- 4 Batch ML Training
- 5 Prediction button
- 6 Prediction
- 7 MLflow experiment run information
- 8 Prediction results

## 1 Train button

Initiates the batch machine learning training process for the TFD model.

## 2 MLflow run ID - ML model

Dropdown displaying the MLflow run IDs associated with batch-trained TFD model the user can pick after training new Batch ML models.

## 3 MLflow run ID page

Navigates to or provides detailed information within MLflow about a particular experiment run.

## 4 Batch ML Training options

Provides the user the options of training new Batch ML models in real-time per maximum number of rows or per percentage of rows from the original TFD dataset.

## 5 Prediction button

Triggers a prediction using the currently selected or trained batch ML model.

## 6 Prediction tab

Displays the interface or section where batch prediction results for the TFD model are presented.

## 7 MLflow experiment run informations

Offers detailed status information, run ID, and other metadata for an MLflow experiment run.

## 8 Prediction results

Presents the output of the TFD model's real-time fraud predictions.

# TRANSACTION FRAUD DETECTION - BATCH MACHINE LEARNING

The screenshot shows the COELHO REALTIME Transaction Fraud Detection interface. At the top, there's a navigation bar with 'COELHO RealTime' logo, 'Home', 'Applications', and 'Services'. Below the navigation, the main title is 'Transaction Fraud Detection' with a subtitle 'Real-time fraud detection using streaming ML'. There are tabs for 'Incremental ML', 'Batch ML' (which is selected), and 'Delta Lake SQL'. On the left, there's a sidebar with 'MLflow Run' (CatBoost Classifier, 8 runs, ID: 1e49c74eb2674380aa66c27e3676342c), 'Batch ML Training' (Model trained and ready, Train button), and 'Data' (Max Rows: 10000). The main area has two tabs: 'Prediction' and 'Metrics' (highlighted with a red box and circled with a red number 1). Under 'Metrics', there are tabs for 'Overview' (highlighted with a red box and circled with a red number 2), 'Classification', 'Features', 'Target', and 'Diagnostics'. The 'Classification Metrics' section contains four primary metrics: Recall (1.0000), Precision (0.6296), F1 Score (0.7727), and F2 (β=2) (0.8947). Below that are ranking metrics: ROC-AUC (0.9987) and Avg Precision (0.8199). The 'Secondary Metrics' section includes Accuracy (0.9950), Balance... (0.9975), MCC (0.7915), Cohen K... (0.7703), Jaccard (0.6296), and G-Mean (0.9975). The 'Calibration Metrics' section includes Log Loss (0.6249), Brier Score (0.2160), D<sup>2</sup> Log Loss (-11.7565), and D<sup>2</sup> Brier (-24.6257). At the bottom, transaction details like Amount (3939.3), Account Age (2203), Currency (AUD), Date (01/25/20), Time (2:33 PM), Merchant ID (142), Category (travel), Type (deposit), Payment (paypal), Latitude (29.950574), Longitude (-95.598269), Browser (Chrome), OS (Android), CSV, Billing (Address Match checked), and Transaction ID (txn\_dc9d7acb3067) are listed.

## 1 Metrics tab

Presents various metrics of the TFD model. Includes some tabs with a lot of visual metrics from Scikit-Learn, Scikit-Plot and YellowBrick libraries.

## 2 Classification metrics tabs

- **Overview:** Displays key MLflow metrics, including primary, ranking, secondary, and calibration scores.
- **Classification:** Provides various visualizations to assess classification model performance.
- **Features:** Offers visual tools for analyzing input features and their properties.
- **Target:** Presents visualizations related to the target variable's distribution and characteristics.
- **Diagnostics:** Contains visualizations for model selection, debugging, and advanced diagnostics.

## Metrics (Scikit-Learn library)

- |                     |                                    |
|---------------------|------------------------------------|
| • Recall            | • Matthews Correlation Coefficient |
| • Precision         | • Cohen Kappa                      |
| • F1 Score          | • Jaccard                          |
| • F2 (β=2)          | • Geometric Mean                   |
| • ROC-AUC           | • Log Loss                         |
| • Average Precision | • Brier Score                      |
| • Accuracy          | • D <sup>2</sup> Log Loss          |
| • Balanced Accuracy | • D <sup>2</sup> Brier             |

COELHO  
REALTIME

# TRANSACTION FRAUD DETECTION - BATCH MACHINE LEARNING

The screenshot shows the COELHO REALTIME platform interface for Transaction Fraud Detection. The top navigation bar includes links for Home, Applications, and Services. The main header "COELHO REALTIME" is on the left, and the sub-header "Transaction Fraud Detection" is in the center. Below the header, there are three tabs: Incremental ML, Batch ML (which is selected), and Delta Lake SQL. On the left side, there's a sidebar with "MLflow Run" (CatBoost Classifier) showing 8 runs, "Batch ML Training" (Model trained and ready), and "Data" (Max Rows: 10000). The main content area has two sections: "Prediction" and "Metrics". The "Metrics" section is active, displaying a "Classification Metrics" card with tabs for Overview, Classification (selected), Features, Target, and Diagnostics. A red circle with the number 1 highlights the "Classification" tab. Below it, a red box highlights the "Classification Report – YellowBrick" card, which displays a "CatBoostClassifierYellowbrickWrapper Classification Report" heatmap. The heatmap shows precision, recall, f1 score, and support values for Fraud and Non-Fraud categories. A red circle with the number 2 highlights this card. At the bottom of the page, transaction details are listed, including Predict, Randomize, and various input fields like Amount, Date, Category, and OS.

## Classification Metrics tabs

1

Overview, Classification, Features, Target, Diagnostics

## Classification metrics tabs options

2

- Classification
  - Confusion Matrix
  - Classification Report
  - ROC AUC Curve
  - Precision-Recall Curve
  - Class Prediction Error
  - Discrimination Threshold
  - Calibration Curve
  - DET Curve
  - ROC Curve
  - KS Statistic
  - Cumulative Gain
  - Lift Curve
- Features
  - Rank 1D
  - Rank 2D
  - PCA Decomposition
  - Manifold
  - Parallel Coordinates
  - RadViz
  - Joint Plot
  - Partial Dependence
  - Decision Boundary
  - PCA 2D Projection
  - PCA Component Variance
  - Feature Importances

### Target

- Class Balance
- Feature Correlation (Mutual Info)
- Feature Correlation (Pearson)
- Balanced Binning Reference

### Diagnostics

- Feature Importances
- Cross-Validation Scores
- Validation Curve
- Learning Curve
- Recursive Feature Elimination
- Dropping Curve

COELHO  
REALTIME

# TRANSACTION FRAUD DETECTION - DELTA LAKE SQL

The screenshot shows the COELHO REALTIME web application interface. At the top, there's a navigation bar with 'COELHO' and 'RealTime' on the left, and 'Home', 'Applications', and 'Services' on the right. Below the navigation is a section titled 'Transaction Fraud Detection' with a subtitle 'Real-time fraud detection using streaming ML'. There are three tabs: 'Incremental ML', 'Batch ML', and 'Delta Lake SQL', with 'Delta Lake SQL' being the active tab. The main area contains a 'SQL Editor' where the query 'SELECT \* FROM data LIMIT 100' is entered, and a 'Query Results' table showing transaction data. The results table has columns: timestamp, amount, currency, merchant\_id, product\_category, and train. The first few rows of data are:

timestamp	amount	currency	merchant_id	product_category	train
2026-01-25T20:36:16.607468+00:00	50.94	USD	merchant_79	luxury_items	pay
2026-01-25T20:36:19.425698+00:00	410.02	USD	merchant_76	luxury_items	def
2026-01-25T20:36:19.956102+00:00	134.35	USD	merchant_126	gambling	wit
2026-01-25T20:36:22.327767+00:00	306.36	USD	merchant_16	travel	def
2026-01-25T20:36:24.058332+00:00	188.02	USD	merchant_36	electronics	wit
2026-01-25T20:36:27.955165+00:00	273.99	USD	merchant_51	services	pay
2026-01-25T20:36:28.646752+00:00	350.2	USD	merchant_77	electronics	trai
2026-01-25T20:36:32.747613+00:00	385.31	USD	merchant_21	clothing	pur
2026-01-25T20:36:37.812369+00:00	436.82	USD	merchant_65	groceries	trai
2026-01-25T20:36:41.739019+00:00	392.15	USD	merchant_123	gambling	pur

At the bottom of the editor, there's a blue 'Run' button with a play icon. The entire interface is set against a dark background.

## 1 SQL Editor

Interface for writing and editing SQL queries against the TFD Delta Lake.

## 2 Query run button

Executes the SQL query in the editor against the TFD Delta Lake.

## 3 Search box, rows and latency

Displays search, allowing user to filter specific values over the entire result, total rows returned, and query execution time.

## 4 Query results with filter

Presents the results of the SQL query with values filtered on the filter box.

# ESTIMATED TIME OF ARRIVAL

---

Incremental Machine Learning  
Batch Machine Learning  
Delta Lake SQL

- **Incremental ML model:** Adaptive Random Forest Regressor (River)
- **Batch ML model:** CatBoost Regressor

# ESTIMATED TIME OF ARRIVAL - INCREMENTAL MACHINE LEARNING

The screenshot shows the COELHO REALTIME application interface with the 'Estimated Time of Arrival' module selected. The interface is divided into several sections:

- Real-time ML Training:** A section with a toggle switch (1) to start processing live data, currently off. Below it is a dropdown for the MLflow experiment (2).
- Trip Details:** A form (3) containing fields for Driver ID, Vehicle ID, Weather, Date, Time, Origin and Destination coordinates, Rating, Temperature, Traffic Factor, and Weather/Driver/Incident factors. It includes a 'Predict' button.
- Prediction:** A dark card (4) showing the ETA prediction process.
- Metrics:** A section showing various performance metrics.
- MLflow experiment run information:** A card (5) showing the status as 'FINISHED', run ID 'cb8d6ac3', and start time '2026-01-25 02:35:33'.
- ETA - Prediction:** A large card (6) displaying the estimated travel time in minutes and seconds: 'Minutes 53.0' and 'Seconds 3180'.
- Map with Origin and Destination:** A map card (7) showing a path from Cypress to Houston, with markers for the origin and destination.

## 1 Real-time ML training toggle

Activates or deactivates the continuous, real-time adaptation of the ETA model, enabling it to learn incrementally from incoming data streams.

## 2 MLflow model page - experiment run

Provides a summary of a specific incremental learning experiment for ETA on MLflow service page, detailing logged parameters, metrics, and associated artifacts within MLflow.

## 3 Prediction button

Initiates an immediate ETA prediction using the ETA model's current, incrementally updated state, demonstrating low-latency inference.

## 4 Prediction tab

Displays real-time estimated time of arrival (ETA) predictions from the incremental machine learning model, offering immediate and continuously updated insights.

## 5 MLflow experiment run informations

Offers status info (LIVE | FINISHED), MLflow run ID, and the datetime the last real-time ML model training was executed.

## 6 Prediction results

Presents the output of the ETA model's real-time predictions.

## 7 Map with Origin and Destination

Illustrates the real-time or predicted path of an entity, clearly marking its starting point (Origin) and its projected endpoint (Destination) to support dynamic ETA calculations.

# ESTIMATED TIME OF ARRIVAL - INCREMENTAL MACHINE LEARNING

The screenshot shows the COELHO REALTIME application interface. At the top left is the COELHO RealTime logo. The main header is "COELHO REALTIME". On the right are navigation links: "Home", "Applications", and "Services". Below the header, there's a section titled "Estimated Time of Arrival" with a subtitle "Predict delivery times with high accuracy". It includes tabs for "Incremental ML" (selected), "Batch ML", and "Delta Lake SQL". A "Real-time ML Training" section has a toggle switch and a "MLflow" button. The "Prediction" section shows a "Metrics" tab highlighted with a red box and a red circle containing the number "1". The "Regression Metrics" section displays various performance metrics:

MAE	RMSE	R <sup>2</sup> Score	Rolling MAE
331.9464	545.2323	0.9221	308.9953
MSE	RMSLE	SMAPE	Time Rolling MAE
297278.2455	0.1267	8.8275	331.9464
R <sup>2</sup> Score	MAPE		
0.9221	9.06%		

The "Trip Details" section contains input fields for Driver ID, Vehicle ID, Weather, Date, Time, Vehicle Type, Origin and Dest Coordinates, Rating, Temp C, Traffic Factor, Weather Factor, Driver Factor, and Incident (s). At the bottom, it shows Trip ID: trip\_f5c815efca4c, Estimated Distance: 11.74 km, and Initial Estimated Travel Time: 705 s.

## 1 Metrics tab

Presents the real-time metrics and incremental learning status of the ETA model.

## Metrics (River library)

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- Mean Squared Error (MSE)
- R-Squared ( $R^2$ )
- Symmetric MAPE (SMAPE)
- Root Mean Squared Log Error (RMSLE)
- Rolling MAE
- Rolling RMSE
- Time Rolling MAE

# ESTIMATED TIME OF ARRIVAL - BATCH MACHINE LEARNING

The screenshot shows the COELHO REALTIME application interface with the following sections:

- COELHO REALTIME**: Top navigation bar.
- Home**, **Applications**, **Services**: Top right navigation.
- Estimated Time of Arrival**: Main title with subtitle "Predict delivery times with high accuracy".
- MLflow**: Subtitle under Estimated Time of Arrival.
- Batch ML**: Selected tab.
- Delta Lake SQL**: Other tab option.
- MLflow Run**: Sub-section showing "CatBoost Regressor" and run ID **7a843e6b387a41838261b1dea8269c76**.
- Batch ML Training**: Sub-section showing "Model trained and ready".
- Data**: Sub-section with "Max Rows" dropdown set to **10000**.
- Trip Details**: Sub-section with "Predict" button highlighted by a red circle **5**.
- Origin and Destination**: Map view showing two locations with a route line.
- ETA - Prediction**: Sub-section displaying travel time results:
  - Seconds**: **4330**
  - Minutes**: **72.1**
- MLflow**: Sub-section showing "CatBoost Regressor", "Run: 7a843e6b", and "Started: 2026-01-23 18:56:58".
- Prediction**: Sub-section showing "Metrics".
- Metrics**: Sub-section showing "Metrics".

## 1 Train button

Initiates the batch machine learning training process for the ETA model.

## 2 MLflow run ID - ML model

Dropdown displaying the MLflow run IDs associated with batch-trained ETA model the user can pick after training new Batch ML models.

## 3 MLflow run ID page

Navigates to or provides detailed information within MLflow about a particular experiment run.

## 4 Batch ML Training options

Provides the user the options of training new Batch ML models in real-time per maximum number of rows or per percentage of rows from the original ETA dataset.

## 5 Prediction button

Triggers a prediction using the currently selected or trained batch ML model.

## 6 Prediction tab

Displays the interface or section where batch prediction results for the ETA model are presented.

## 7 MLflow experiment run informations

Offers detailed status information, run ID, and other metadata for an MLflow experiment run.

## 8 Prediction results

Presents the output of the ETA model's real-time predictions.

# ESTIMATED TIME OF ARRIVAL - BATCH MACHINE LEARNING

The screenshot shows the COELHO REALTIME web application interface. At the top, there's a navigation bar with 'COELHO RealTime' logo, 'Home', 'Applications', and 'Services'. Below the header, there's a section titled 'Estimated Time of Arrival' with a sub-section 'Incremental ML', 'Batch ML' (which is selected), and 'Delta Lake SQL'. On the left, there's a 'MLflow Run' card for a 'CatBoost Regressor' run (7a843e6b387a41838261b1dea8269c76) and a 'Batch ML Training' section indicating a model is trained and ready. On the right, the 'Metrics' tab is selected in the 'Prediction' section, which displays various metrics. A red box highlights the 'Metrics' tab itself (labeled 1). Another red box highlights the tabs in the sub-section (labeled 2), which include 'Overview', 'Regression', 'Features', 'Target', and 'Diagnostics'. A third red box highlights the 'Training Data: 10,000 rows (80% train / 20% test split)' section (labeled 3).

## 1 Metrics tab

Presents various metrics of the ETA model. Includes some tabs with a lot of visual metrics from Scikit-Learn, Scikit-Plot and YellowBrick libraries.

## 2 Regression metrics tabs

- **Overview:** Displays key MLflow metrics, including primary, ranking, secondary, and calibration scores.
- **Regression:** Provides various visualizations to assess regression model performance.
- **Features:** Offers visual tools for analyzing input features and their properties.
- **Target:** Presents visualizations related to the target variable's distribution and characteristics.
- **Diagnostics:** Contains visualizations for model selection, debugging, and advanced diagnostics.

## 3 Training data info

Displays the total amount of rows used to train the Batch ML model

## Metrics (Scikit-Learn library)

- |  |                                     |
|--|-------------------------------------|
| • Mean Absolute Error (MAE)                        | • Mean Squared Error (MSE)          |
| • Root Mean Squared Error (RMSE)                   | • Median Absolute Error (Median AE) |
| • Mean Absolute Percentage Error (MAPE)            | • Max Error                         |
| • Symmetric Mean Absolute Percentage Error (SMAPE) | • D2 Absolute Error Score           |
| • R2 Score   | • D2 Pinball Score                  |
| • Explained Variance Score                         | • D2 Tweedie Score                  |

COELHO  
REALTIME

# ESTIMATED TIME OF ARRIVAL - BATCH MACHINE LEARNING

The screenshot shows the COELHO REALTIME platform interface. At the top, there's a navigation bar with 'COELHO RealTime' logo, 'Home', 'Applications', and 'Services'. Below the navigation, the main title is 'Estimated Time of Arrival' with the subtitle 'Predict delivery times with high accuracy'. There are three tabs: 'Incremental ML', 'Batch ML' (which is selected), and 'Delta Lake SQL'. On the left, there's a 'MLflow Run' section for 'CatBoost Regressor' (5 runs) and a 'Batch ML Training' section where a model has been trained. On the right, the 'Metrics' tab is active, showing a 'Regression Metrics' section for 'MLflow CatBoost Regressor' (Run: 7a843e6b, Started: 2026-01-23 18:56:58). A red circle labeled '1' highlights the 'Metrics' tab. A red box labeled '2' highlights the 'Residuals Plot – YellowBrick' visualization, which displays 'Residuals for CatBoostRegressorYellowbrickWrapper Model' with Train  $R^2 = 0.084$  and Test  $R^2 = 0.082$ . The plot shows residuals on the y-axis (ranging from -10000 to 4000) against Predicted Value (x-axis, 4300 to 4700) and Distribution (right y-axis, 0 to 4000). Below the plot, trip details are listed: Driver ID (3664), Vehicle ID (853), Weather (Clouds), Date (01/26/2024), Time (10:41 PM), Vehicle Type (Van), Origin Lat (29.963463), Origin Lon (-95.765509), Dest Lat (30.051225), Dest Lon (-95.789062), Hour (10), Rating (4.5), Temp C (20.8). Trip ID: trip\_f5c815efca4c, Estimated Distance: 11.74 km, Initial ETA: 705 s (11.8 min).

## 1 Regression Metrics tabs

Overview, Regression, Features, Target, Diagnostics

## 2 Regression metrics tabs options

- Regression
  - Residuals Plot
  - Prediction Error
  - Prediction Error (Actual vs Predicted)
  - Prediction Error (Residuals)
- Features
  - Rank 1D
  - Rank 2D
  - PCA Decomposition
  - Manifold
  - Joint Plot
  - Partial Dependence
  - PCA 2D Projection
  - PCA Component Variance
- Target
  - Feature Correlation (Mutual Info)
  - Feature Correlation (Pearson)
  - Balanced Binning Reference
- Diagnostics
  - Feature Importances
  - Cross-Validation Scores
  - Validation Curve
  - Learning Curve
  - Recursive Feature Elimination
  - Dropping Curve

# ESTIMATED TIME OF ARRIVAL - DELTA LAKE SQL

The screenshot shows the COELHO REALTIME interface with the Delta Lake SQL tab selected. The interface includes:

- SQL Editor:** A text input field containing the query `SELECT * FROM data LIMIT 100`. (Red box 1)
- Run button:** A blue button labeled "Run". (Red box 2)
- Query Results:** A table displaying the results of the query. The columns are `trip_id`, `driver_id`, `vehicle_id`, and `timestamp`. The table has 100 rows. (Red box 3)
- Filter results...**: A search bar at the top of the results table. (Red box 4)

## 1 SQL Editor

Interface for writing and editing SQL queries against the ETA Delta Lake.

## 2 Query run button

Executes the SQL query in the editor against the ETA Delta Lake.

## 3 Search box, rows and latency

Displays search, allowing user to filter specific values over the entire result, total rows returned, and query execution time.

## 4 Query results with filter

Presents the results of the SQL query with values filtered on the filter box.

# E-COMMERCE CUSTOMER INTERACTIONS

---

Incremental Machine Learning  
Batch Machine Learning  
Delta Lake SQL

- Incremental ML model: DBSTREAM (River)
- Batch ML model: KMEANS (Scikit-learn)

# E-COMMERCE CUSTOMER INTERACTIONS - INCREMENTAL MACHINE LEARNING

The screenshot shows the COELHO REALTIME interface with several key components highlighted by red boxes and numbered circles:

- Real-time ML training toggle (1)**: A switch to start processing live data.
- MLflow experiment run (2)**: An experiment named "DBSTREAM Clustering (River)" is shown as FINISHED.
- Prediction button (3)**: A button to initiate an immediate prediction.
- Prediction tab (4)**: Displays real-time ECCI predictions.
- Customer Location (5)**: A map showing a customer's location at (30.0240, -95.2090) in KINGWOOD, Texas.
- Predicted Cluster (6)**: The customer is assigned to Cluster 2.
- Cluster Behavior (7)**: A bar chart showing event\_type distribution: search (~100), page\_view (~50), add\_to\_cart (~20), purchase (~10), leave (~5).
- Annotations (8)**: A note explaining the cluster assignment: "This customer interaction was assigned to Cluster 2. Clusters represent groups of similar customer behaviors based on their browsing patterns, device usage, and purchase activities."

## 1 Real-time ML training toggle

Activates or deactivates the continuous, real-time adaptation of the ECCI model, enabling it to learn incrementally from incoming data streams.

## 2 MLflow model page - experiment run

Provides a summary of a specific incremental learning experiment for ECCI on MLflow service page, detailing logged parameters, metrics, and associated artifacts within MLflow.

## 3 Prediction button

Initiates an immediate ECCI prediction using the ECCI model's current, incrementally updated state, demonstrating low-latency inference.

## 4 Prediction tab

Displays real-time ECCI predictions from the incremental machine learning model, offering immediate and continuously updated insights into customer interactions.

## 5 MLflow experiment run informations

Offers status info (LIVE | FINISHED), MLflow run ID, and the datetime the last real-time ML model training was executed.

## 6 Prediction results

Presents the output of the ECCI model's real-time predictions.

## 7 Map with customer location

Illustrates customer locations and their interaction journeys on a map, providing spatial insights into e-commerce customer behavior.

## 8 Cluster behavior analysis

Illustrates the distribution of a selected customer interaction feature (e.g., event type, product category) within the predicted cluster for the sample user, revealing the characteristic behaviors of that customer segment.

# E-COMMERCE CUSTOMER INTERACTIONS - INCREMENTAL MACHINE LEARNING

The screenshot shows the COELHO REALTIME E-Commerce Customer Interactions dashboard. At the top, there's a navigation bar with 'COELHO RealTime' logo, 'Home', 'Applications', and 'Services'. Below the header, there are tabs for 'Incremental ML' (selected), 'Batch ML', and 'Delta Lake SQL'. On the left, there's a sidebar for 'Real-time ML Training' with a toggle switch and buttons for 'DBSTREAM Clustering (River)' and 'MLflow'. The main area is titled 'E-Commerce Customer Interactions' with a subtitle 'Customer segmentation and behavior analysis'. It features a 'Customer Interaction' section with a 'Predict' button, dropdowns for 'Browser', 'Device', 'OS', 'Event Type', 'Category', 'Price', 'Date', 'Time', 'Product ID', 'Latitude', 'Longitude', 'Quantity', 'Time (s)', 'Sequence', and 'Referrer' (linkedin.com). At the bottom, there are logs for 'Customer ID', 'Event ID', 'Page URL', 'Search Query', and 'Session ID'. The central part of the dashboard is the 'Metrics' tab, which is highlighted with a red box and a red circle containing the number '1'. This tab is part of a 'Clustering Metrics' section. The 'Metrics' tab shows 'MLflow FINISHED' status. Below it are several metrics: Silhouette (0.0034), Rolling Silhouette (0.0021), N Clusters (4), N Micro-clusters (4), Silhouette Score (0.0034), Rolling Silhouette (0.0021), Time Rolling Silhouette (0.0034), Silhouette Score (0.0034), Cluster Statistics (4 Clusters, 4 Micro-clusters), and a progress bar for Silhouette Score at 0.0034.

## 1 Metrics tab

Presents the real-time metrics and incremental learning status of the ECCI model.

## Metrics (River library)

- Silhouette
- Silhouette Score
- Rolling Silhouette
- Time Rolling Silhouette
- Number of Clusters
- Number of Micro-clusters

# E-COMMERCE CUSTOMER INTERACTIONS - INCREMENTAL MACHINE LEARNING

The screenshot shows the COELHO REALTIME E-commerce Customer Interactions dashboard. On the left, there's a sidebar with various filters and details about a customer interaction. The main area has three tabs: 'Analytics' (selected), 'Samples per cluster', and 'Feature Distribution'. The 'Analytics' tab displays aggregated statistics across all clusters. The 'Samples per cluster' tab shows a bar chart of interaction samples per cluster. The 'Feature Distribution' tab shows a grouped bar chart of feature distributions across different clusters.

**Analytics tab**

This tab provides an in-depth view of aggregated statistics across all identified customer clusters, allowing for comprehensive analysis of customer segments and their evolving characteristics.

**Samples per cluster**

Visualizes the real-time distribution of customer interaction samples across different clusters, showing the relative size and density of each identified customer segment.

**Feature Distribution**

Displays the distribution of values for a selected feature (e.g., event type, product category) across all customer clusters, highlighting the distinguishing attributes and behavioral patterns of each segment.

① This tab shows aggregated statistics across all clusters. Use the feature selector to explore how different attributes are distributed across customer segments.

## 1 Analytics tab

Provides an in-depth view of aggregated statistics across all identified customer clusters, allowing for comprehensive analysis of customer segments and their evolving characteristics.

## 2 Samples per cluster

Visualizes the real-time distribution of customer interaction samples across different clusters, showing the relative size and density of each identified customer segment.

## 3 Feature Distribution

Displays the distribution of values for a selected feature (e.g., event type, product category) across all customer clusters, highlighting the distinguishing attributes and behavioral patterns of each segment.

COELHO  
REALTIME

# E-COMMERCE CUSTOMER INTERACTIONS - BATCH MACHINE LEARNING

The screenshot shows the COELHO REALTIME platform interface for E-Commerce Customer Interactions. The main navigation bar includes Home, Applications, and Services. The main content area has tabs for Incremental ML, Batch ML (selected), and Delta Lake SQL. The Batch ML tab displays:

- MLflow Run:** KMeans (Scikit-Learn). A dropdown menu (2) lists runs: ★ 04a0ab06ae94427f9491b6e0332e44a6.
- Batch ML Training:** Model trained and ready. A "Train" button (1).
- Data:** Max Rows (4) set to 10000 rows.
- Prediction:** A "Predict" button (5).
- Customer Location:** A map showing a location in Deer Park, Pasadena, and La Porte. The coordinates are (29.6810, -95.1200).
- Predicted Cluster:** Cluster ID 1 (8).
- Cluster Behavior:** A histogram showing event\_type distribution.
- Note:** This customer interaction was assigned to Cluster 1. Clusters represent groups of similar customer behaviors based on their browsing patterns, device usage, and purchase activities.

## 1 Train button

Initiates the batch machine learning training process for the ECCI model.

## 2 MLflow run ID - ML model

Dropdown displaying the MLflow run IDs associated with batch-trained ECCI model the user can pick after training new Batch ML models.

## 3 MLflow run ID page

Navigates to or provides detailed information within MLflow about a particular experiment run.

## 4 Batch ML Training options

Provides the user the options of training new Batch ML models in real-time per maximum number of rows or per percentage of rows from the original ECCI dataset.

## 5 Prediction button

Triggers a prediction using the currently selected or trained batch ML model.

## 6 Prediction tab

Displays the interface or section where batch prediction results for the ECCI model are presented.

## 7 MLflow experiment run informations

Offers detailed status information, run ID, and other metadata for an MLflow experiment run.

## 8 Prediction results

Presents the output of the ECCI model's batch machine learning predictions.

# E-COMMERCE CUSTOMER INTERACTIONS - BATCH MACHINE LEARNING

The screenshot shows the COELHO REALTIME interface for E-Commerce Customer Interactions. At the top, there's a navigation bar with 'COELHO RealTime' logo, 'Home', 'Applications', and 'Services'. Below the header, there are tabs for 'Incremental ML', 'Batch ML' (which is selected), and 'Delta Lake SQL'. On the left, there's a sidebar for 'MLflow Run KMeans (Scikit-Learn)' with a dropdown showing '04a0ab06ae94427f9491b6e0332e44a6'. Under 'Batch ML Training', it says 'Model trained and ready' with a 'Train' button. On the right, the main area has a title 'E-Commerce Customer Interactions' and 'Customer segmentation and behavior analysis'. It features two tabs: 'Prediction' and 'Metrics' (highlighted with a red box and number 1). Below this is a section titled 'Clustering Metrics' with tabs 'Overview' (highlighted with a red box and number 2), 'Clustering', 'Features', 'Target', 'Diagnostics', and 'Text'. The 'Overview' tab displays various clustering metrics: Silhouette (0.1585), Calinski-Harabasz (1553), Davies-Bouldin (2.3054), and N Clusters (2). The 'Secondary Metrics' section shows Inertia (WCSS) (95214), Rolling Silhouette (0.0000), and Time Rolling Silhouette (0.0000). The bottom section shows Silhouette Score (0.1585) and Cluster Statistics (2 Clusters, 95,214 Inertia).

## 1 Metrics tab

Presents various metrics of the ECCI model. Includes some tabs with a lot of visual metrics from Scikit-Learn, Scikit-Plot and YellowBrick libraries.

## 2 Clustering metrics tabs

- Overview:** Displays key MLflow metrics, including primary, ranking, secondary, and calibration scores.
- Clustering:** Provides various visualizations to assess clustering model performance.
- Features:** Offers visual tools for analyzing input features and their properties.
- Target:** Presents visualizations related to the target variable's distribution and characteristics.
- Diagnostics:** Contains visualizations for model selection, debugging, and advanced diagnostics.
- Text:** Visualizations and metrics for Textual Features, specifically for the search\_query field, including analyses such as word frequency distributions, search query clustering (t-SNE/UMAP), word dispersion, word correlation, and part-of-speech tagging, all aimed at understanding customer search behavior and intent.

## Metrics (Scikit-Learn library)

- |  |   |
|--|---|
| <ul style="list-style-type: none"><li>Silhouette</li><li>Calinski-Harabasz</li><li>Davies-Bouldin</li><li>N Clusters</li></ul> | <ul style="list-style-type: none"><li>Inertia (WCSS)</li><li>Rolling Silhouette</li><li>Time Rolling Silhouette</li></ul> |
|--|---|

COELHO  
REALTIME

# E-COMMERCE CUSTOMER INTERACTIONS - BATCH MACHINE LEARNING

The screenshot shows the COELHO REALTIME platform interface for E-Commerce Customer Interactions. The top navigation bar includes links for Home, Applications, and Services. The main menu has tabs for Incremental ML, Batch ML (which is selected), and Delta Lake SQL. The Batch ML section displays an MLflow Run for KMeans (Scikit-Learn) with a run ID of 04a0ab06ae94427f9491b6e0332e44a6. Below this, there's a Batch ML Training section indicating a model is trained and ready, with a 'Train' button. On the left, a Customer Interaction panel allows filtering by various parameters like Browser, Device, OS, Event Type, Category, Price, Date, Time, Product ID, Latitude, Longitude, Coords, Quantity, Time, Sequence, Referrer, Customer ID, Event ID, and Session ID. The central area is titled 'Clustering Metrics' and shows a 'Metrics' tab selected. A sub-menu bar at the top of this section includes Overview, Clustering, Features, Target, Diagnostics, and Text. A dropdown menu under 'Clustering' is highlighted with a red box and labeled 'Intercluster Distance – YellowBrick'. Below it is a scatter plot titled 'KMeans Intercluster Distance Map (via MDS)' showing data points on PC1 and PC2 axes, with clusters labeled 0 and 1.

## 1 Clustering Metrics tabs

Overview, Clustering, Features, Target, Diagnostics, Text

## 2 Clustering metrics tabs options

- Clustering

- K-Elbow
- Silhouette
- Intercluster Distance
- Elbow Curve

- Features

- Rank 1D
- Rank 2D
- PCA Decomposition
- Manifold
- Parallel Coordinates
- RadViz
- Joint Plot
- Partial Dependence
- Decision Boundary
- PCA 2D Projection
- PCA Component Variance
- Feature Importances

- Target

- Cluster Distribution
- Feature Correlation (Mutual Info)
- Feature Correlation (Pearson)
- Balanced Binning Reference

- Diagnostics

- Feature Importances
- Cross-Validation Scores
- Validation Curve
- Learning Curve
- Recursive Feature Elimination
- Dropping Curve

- Text

- Frequency Distribution
- t-SNE Visualization
- UMAP Visualization
- Dispersion Plot
- Word Correlation
- POS Tag Distribution

COELHO  
REALTIME

# E-COMMERCE CUSTOMER INTERACTIONS - DELTA LAKE SQL

The screenshot shows the COELHO REALTIME platform interface for E-Commerce Customer Interactions. The top navigation bar includes links for Home, Applications, and Services. The main menu has tabs for Incremental ML, Batch ML, and Delta Lake SQL, with Delta Lake SQL selected and highlighted in blue. The left sidebar contains sections for SQL Editor, Templates, Sample Data, and Purchases. The main content area displays the results of a SQL query in a table format.

**SQL Editor:** A red box highlights the SQL Editor section on the left, which contains the following SQL code:

```
SELECT product_category, COUNT(*) as count
FROM data GROUP BY product_category ORDER
BY count DESC
```

**Query run button:** A red box highlights the "Run" button at the bottom of the SQL Editor section, with a red circle labeled "2" indicating its location.

**Search box, rows and latency:** A red box highlights the "Filter results..." search box and the row count "13/13" and latency "3795ms" in the top right corner of the results table, with a red circle labeled "3" indicating its location.

**Query results with filter:** A red box highlights the results table itself, with a red circle labeled "4" indicating its location. The table lists product categories and their counts:

product_category	count
-	795090
Electronics	416800
Grocery & Gourmet Food	416746
Beauty & Personal Care	416685
Fashion & Apparel	416359
Computers	416001
Home & Garden	415810
Health & Household	415773
Books	415755
Toys & Games	415715
Sports & Outdoors	415696
Pet Supplies	415534
Automotive	415177

## 1 SQL Editor

Interface for writing and editing SQL queries against the ECCI Delta Lake.

## 2 Query run button

Executes the SQL query in the editor against the ECCI Delta Lake.

## 3 Search box, rows and latency

Displays search, allowing user to filter specific values over the entire result, total rows returned, and query execution time.

## 4 Query results with filter

Presents the results of the SQL query with values filtered on the filter box.

# THE MICROSERVICES ENGINE

---

A closer look at the microservices powering the COELHO RealTime platform — from real-time observability with Prometheus and Grafana, to experiment tracking with MLflow, event streaming with Apache Kafka, and the complete infrastructure that keeps every component running seamlessly on Kubernetes.



FastAPI



Spark



MLflow



MinIO



Prometheus



Grafana



Alertmanager



Karma

# MICROSERVICES - HOW TO ACCESS

The screenshot shows the COELHO REALTIME platform interface. At the top, there's a navigation bar with a logo on the left, followed by 'Home', 'Applications' (which has a red circle with '1' over it), and 'Services'. A red box highlights the 'Services' dropdown. To its right is a sidebar titled 'EXTERNAL SERVICES' containing links to various microservices: FastAPI, Spark, MLflow, MinIO, Prometheus, Grafana, Alertmanager, and Karma. Each service entry includes a small icon and a copy icon. Below the sidebar, there's a section for 'ML Applications' with a sub-section for 'Production-ready machine learning use cases'. At the bottom, there are three large buttons with arrows pointing right, each containing a shield icon, a clock icon, and a shopping cart icon.

COELHO REALTIME

COELHO  
RealTime

# COELHO REALTIME

Enterprise-grade Real-Time Machine Learning Platform combining streaming data processing, incremental learning, and batch model training in a unified Kubernetes-native architecture.

3 ML Projects    2 ML Paradigms    10+ Microservices    100% Cloud Native

## ML Applications

Production-ready machine learning use cases

## 1 Services dropdown on navbar

The Services dropdown on the navbar provides quick access to all external microservices running on the platform. Clicking it reveals a menu listing 7 services – FastAPI, Spark, MLflow, MinIO, Prometheus, Grafana, and Alertmanager – each linking directly to its web UI in a new browser tab.

## 2 Microservices options

- **FastAPI** – localhost:8000/docs
- **Spark** – localhost:4040
- **MLflow** – localhost:5001
- **MinIO** – localhost:9001
- **Prometheus** – localhost:9090
- **Grafana** – localhost:3000
- **Alertmanager** – localhost:9094
- **Karma**: localhost:8088

COELHO  
REALTIME

# MICROSERVICES - FASTAPI

**Unified ML Service - COELHO RealTime** 1.0.0 OAS 3.1  
[/openapi.json](#)

Unified API for Incremental ML (River) and Batch ML (Scikit-Learn/CatBoost).

### Endpoints

- Incremental ML ( /api/v1/incremental )**
  - Real-time streaming ML with River library
  - Kafka consumer integration
  - Live model predictions during training
- Batch ML ( /api/v1/batch )**
  - Batch training with CatBoost/Scikit-Learn
  - YellowBrick visualizations
  - MLflow model versioning
- Delta Lake SQL ( /api/v1/sql )**
  - SQL queries against Delta Lake tables
  - DuckDB and Polars query engines
  - Table schema inspection

**default**

- GET** /metrics Metrics
- GET** / Root
- GET** /health Health

**Incremental ML (River)**

- GET** /api/v1/incremental/health Health
- GET** /api/v1/incremental/status Get Status
- POST** /api/v1/incremental/switch-model Switch Model
- GET** /api/v1/incremental/training-status/{project\_name} Get Training Status
- POST** /api/v1/incremental/predict Predict
- POST** /api/v1/incremental/model-available Check Model Available
- POST** /api/v1/incremental/mlflow-metrics Get Mlflow Metrics
- POST** /api/v1/incremental/report-metrics Get Report Metrics

## 1 FastAPI – Important REST API Endpoints

Incremental ML (/api/v1/incremental) – 16 endpoints

- [GET] /health – Health check
- [GET] /status – Current training status
- [POST] /switch-model – Start/stop incremental training
- [POST] /predict – Real-time prediction (live Redis or MLflow model)
- [POST] /mlflow-metrics – Experiment metrics with caching
- [POST] /page-init – Combined page initialization
- [POST] /sample – Random dataset sample via DuckDB

Batch ML (/api/v1/batch) – 20 endpoints

- [GET] /health – Health check
- [GET] /status – Training progress with live updates
- [POST] /switch-model – Start/stop batch training
- [POST] /predict – Prediction using MLflow models
- [POST] /mlflow-runs – List all experiment runs
- [POST] /init – Page initialization with all data
- [POST] /visualization/metric/yellowbrick – YellowBrick visualizations
- [POST] /visualization/metric/scikit-learn – Scikit-Learn visualizations
- [POST] /visualization/metric/scikitplot – Scikit-plot visualizations

Delta Lake SQL (/api/v1/sql) – 3 endpoints

- [POST] /query – Execute SQL against Delta Lake (SELECT only)
- [POST] /schema – Table schema and metadata
- [POST] /total-rows – Total row count

# MICROSERVICES - SPARK

Spark Master at spark://coelho-realtime-spark-master-0.coelho-realtime-spark-headless.coelho-realtime.svc.cluster.local:7077						
URL: spark://coelho-realtime-spark-master-0.coelho-realtime-spark-headless.coelho-realtime.svc.cluster.local:7077						
Workers: 1 Alive, 0 Dead, 0 Decommissioned, 0 Unknown						
Cores in use: 8 Total, 6 Used						
Memory in use: 3.0 GiB Total, 1536.0 MiB Used						
Resources in use:						
Applications: 3 Running, 24 Completed						
Drivers: 0 Running (0 Waiting), 0 Completed (0 Killed, 0 Failed, 0 Error, 0 Relaunching)						
Status: ALIVE (Environment, Log)						
<b>Workers (1)</b>						
Worker Id	Address	State	Cores	Memory		
worker-20260126235202-10.42.0.106-45349	10.42.0.106:45349	ALIVE	8 (6 Used)	3.0 GiB (1536.0 MiB Used)		
<b>Running Applications (3)</b>						
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User
app-20260127023438-0026 (kill)	KafkaToDelta-ECommerceCustomerInteractions	2	512.0 MiB		2026/01/27 02:34:38	root
app-20260127023438-0025 (kill)	KafkaToDelta-EstimatedTimeOfArrival	2	512.0 MiB		2026/01/27 02:34:38	root
app-20260127023438-0024 (kill)	KafkaToDelta-TransactionFraudDetection	2	512.0 MiB		2026/01/27 02:34:38	root
<b>Completed Applications (24)</b>						
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User
app-20260127021321-0021	KafkaToDelta-ECommerceCustomerInteractions	2	512.0 MiB		2026/01/27 02:13:21	root
app-20260127021322-0022	KafkaToDelta-TransactionFraudDetection	2	512.0 MiB		2026/01/27 02:13:22	root
app-20260127021322-0023	KafkaToDelta-EstimatedTimeOfArrival	2	512.0 MiB		2026/01/27 02:13:22	root
app-20260127015433-0020	KafkaToDelta-EstimatedTimeOfArrival	2	512.0 MiB		2026/01/27 01:54:33	root
app-20260127015433-0018	KafkaToDelta-TransactionFraudDetection	2	512.0 MiB		2026/01/27 01:54:33	root
app-20260127015433-0019	KafkaToDelta-ECommerceCustomerInteractions	2	512.0 MiB		2026/01/27 01:54:33	root
app-20260127013721-0016	KafkaToDelta-EstimatedTimeOfArrival	2	512.0 MiB		2026/01/27 01:37:21	root
app-20260127013721-0015	KafkaToDelta-TransactionFraudDetection	2	512.0 MiB		2026/01/27 01:37:21	root
app-20260127013722-0017	KafkaToDelta-ECommerceCustomerInteractions	2	512.0 MiB		2026/01/27 01:37:22	root
app-20260127011310-0013	KafkaToDelta-ECommerceCustomerInteractions	2	512.0 MiB		2026/01/27 01:13:10	root
app-20260127011310-0012	KafkaToDelta-TransactionFraudDetection	2	512.0 MiB		2026/01/27 01:13:10	root
app-20260127011310-0014	KafkaToDelta-EstimatedTimeOfArrival	2	512.0 MiB		2026/01/27 01:13:10	root
app-20260127005700-0009	KafkaToDelta-EstimatedTimeOfArrival	2	512.0 MiB		2026/01/27 00:57:00	root
app-20260127005701-0011	KafkaToDelta-TransactionFraudDetection	2	512.0 MiB		2026/01/27 00:57:01	root
app-20260127005701-0010	KafkaToDelta-ECommerceCustomerInteractions	2	512.0 MiB		2026/01/27 00:57:01	root
app-20260127003704-0007	KafkaToDelta-EstimatedTimeOfArrival	2	512.0 MiB		2026/01/27 00:37:04	root
app-20260127003704-0006	KafkaToDelta-ECommerceCustomerInteractions	2	512.0 MiB		2026/01/27 00:37:04	root
app-20260127003704-0008	KafkaToDelta-TransactionFraudDetection	2	512.0 MiB		2026/01/27 00:37:04	root
app-20260127001818-0004	KafkaToDelta-ECommerceCustomerInteractions	2	512.0 MiB		2026/01/27 00:18:18	root
app-20260127001817-0003	KafkaToDelta-TransactionFraudDetection	2	512.0 MiB		2026/01/27 00:18:17	root
app-20260127001818-0005	KafkaToDelta-EstimatedTimeOfArrival	2	512.0 MiB		2026/01/27 00:18:18	root
app-20260126235417-0002	KafkaToDelta-EstimatedTimeOfArrival	2	512.0 MiB		2026/01/26 23:54:17	root

# MICROSERVICES - MLFLOW

mlflow 3.7.0 GitHub Docs 1

## Welcome to MLflow

### Get started

**Log traces**  
Trace LLM applications for debugging and monitoring.

**Run evaluation**  
Iterate on quality with offline evaluations and comparisons.

**Train models**  
Track experiments, parameters, and metrics throughout training.

**Register prompts**  
Manage prompt updates and collaborate across teams.

### Experiments

Name	Time created	Last modified	Description	Tags
Default	Invalid Date	Invalid Date	-	
Transaction Fraud Detection	01/19/2026, 10:31:18 AM	01/19/2026, 06:41:32 PM	-	
E-Commerce Customer Interactions	01/19/2026, 06:38:31 PM	01/19/2026, 06:38:31 PM	-	
Estimated Time of Arrival	01/19/2026, 06:38:30 PM	01/19/2026, 06:38:30 PM	-	

### Discover new features

**MLflow MCP server**  
Connect your coding assistants and AI applications to MLflow and automatically analyze your experiments and traces.

**Optimize prompts**  
Access the state-of-the-art prompt optimization algorithms such as MIPROv2, GEPA, through MLflow Prompt Registry.

**Agents-as-a-judge**  
Leverage agents as a judge to perform deep trace analysis and improve your evaluation accuracy.

**Dataset tracking**  
Track dataset lineage and versions and effectively drive the quality improvement loop.

## MLflow experiments

Each ML project maps to a dedicated MLflow experiment: Transaction Fraud Detection (binary classification), Estimated Time of Arrival (regression), and E-Commerce Customer Interactions (clustering).

Both Incremental (River) and Batch (Scikit-Learn/CatBoost) training runs are logged under their respective experiment, tracking parameters, metrics, and artifacts across model versions.

# MICROSERVICES - MLFLOW

The screenshot shows the MLflow UI for a 'CatBoostClassifier' experiment under 'Transaction Fraud Detection'. The interface is dark-themed.

**Top Navigation:** mlflow 3.7.0, GitHub, Docs.

**Breadcrumbs:** Transaction Fraud Detection > Runs > CatBoostClassifier

**Overview Tab:** Metrics (17), Parameters (60).

**Metrics Table (1):** A red box highlights this section. It includes a search bar for metrics and lists 17 metrics with their values and source (model). The metrics include preprocessing\_time\_seconds, recall\_score, precision\_score, f1\_score, fbeta\_score, accuracy\_score, balanced\_accuracy\_score, matthews\_corrcoef, cohen\_kappa\_score, jaccard\_score, roc\_auc\_score, average\_precision\_score, log\_loss, and brier\_score\_loss.

Metric	Value	Models
preprocessing_time_seconds	94.86727929115295	model
recall_score	0.9166666666666666	model
precision_score	0.049107142857142856	model
f1_score	0.09322033898305085	model
fbeta_score	0.20220588235294118	model
accuracy_score	0.786	model
balanced_accuracy_score	0.8505398110661269	model
matthews_corrcoef	0.18309714371605	model
cohen_kappa_score	0.07208269737754969	model
jaccard_score	0.04888888888888889	model
roc_auc_score	0.9031713900134952	model
average_precision_score	0.48358252164502163	model
log_loss	0.6599173599219417	model
brier_score_loss	0.23340218826303138	model

**About this run (2):** A red box highlights this section. It provides details about the experiment run, including creation time (01/24/2026, 11:52:12 PM), created by (root), experiment ID (2), status (Finished), run ID (9febb65a09f84d358c00ced498b56fb5), duration (14.5s), source (tfd.py), and registered prompts (none).

**Datasets:** None

**Tags:** training\_mode: batch, preprocessing: DuckDB SQL

**Registered models:** None

**Parameters (60):** A search bar for parameters.

1

2

2

**MLflow experiment run metrics**  
Metrics are logged in real time during training and persist after completion.  
For classification (TFD): F-beta, accuracy, precision, recall, F1 score.  
For regression (ETA): RMSE, MAE, R<sup>2</sup>.  
For clustering (ECCI): Silhouette score, inertia, number of clusters.

These metrics enable comparison across runs and determine which model is selected for predictions.

**MLflow experiment run info**

# MICROSERVICES - MLFLOW

≡ mlflow 3.7.0

Transaction Fraud Detection > Runs >

CatBoostClassifier

Overview Model metrics System metrics Traces Artifacts

visualizations/classifier/ConfusionMatrix.png 23.85KB

Path: s3://mlflow-artifacts/2/1e49c74eb2674380aa66c27e3676342c/artifacts/visualizations/classifier/ConfusionMatrix.png

↓

1

True Class

Non-Fraud

Fraud

99%

1%

0%

100%

Non-Fraud

Fraud

Predicted Class

Each run stores the trained model (pickle), feature encoders, training data (Parquet), and visualizations generated by YellowBrick, Scikit-Learn, and Scikit-plot (e.g., confusion matrix, ROC curve, residuals plot, silhouette analysis).

Artifacts are saved to MinIO (S3-compatible storage) and loaded on demand for predictions, model comparison, and visualization caching.

1

## MLflow experiment run artifacts

Each run stores the trained model (pickle), feature encoders, training data (Parquet), and visualizations generated by YellowBrick, Scikit-Learn, and Scikit-plot (e.g., confusion matrix, ROC curve, residuals plot, silhouette analysis).

Artifacts are saved to MinIO (S3-compatible storage) and loaded on demand for predictions, model comparison, and visualization caching.

# MICROSERVICES - MINIO

The screenshot shows the MinIO Object Browser interface. On the left is a sidebar with navigation links for User (Object Browser, Access Keys, Documentation) and Administrator (Buckets, Policies, Identity, Monitoring, Events, Configuration, License). The main area is titled "Object Browser" and contains a table of buckets. A red box highlights the first two rows of the table. A red circle with the number "1" points to the "mlflow-artifacts" bucket.

Name	Objects	Size	Access
lakehouse	336,564	12.6 GiB	R/W
mlflow-artifacts	748	145.5 MiB	R/W

1

## MinIO buckets

MinIO serves as the unified storage backend for the platform.

The mlflow-artifacts bucket stores all MLflow experiment data – trained models, encoders, training datasets, and visualizations.

The lakehouse bucket holds the Delta Lake tables consumed by Apache Spark and DuckDB, organized by project (TFD, ETA, ECCI).

Both buckets are accessed via the S3 API, making the setup portable to any S3-compatible cloud provider.

# MICROSERVICES - PROMETHEUS

The screenshot shows the Prometheus web interface. At the top, there are navigation links for 'Prometheus', 'Query' (which is selected), 'Alerts', and 'Status'. On the right side of the header are icons for 'Overview', 'Settings', and 'Documentation'. Below the header is a search bar containing the PromQL query: `>_ up{namespace="coelho-realtime"}`. To the right of the search bar is a blue 'Execute' button with a red circle containing the number '1' above it. Under the search bar, there are three tabs: 'Table' (selected), 'Graph', and 'Explain'. Below these tabs is a 'Evaluation time' dropdown. The main content area displays the results of the query, which consists of 20 rows of PromQL metrics. Each row includes a metric name, its value (all are '1'), and a timestamp. The metrics are categorized by service: MinIO, Kafka Producers, Spark Streaming, Node Exporter, MLflow, Alertmanager, FastAPI, Redis Metrics, Prometheus Stack, Kube State Metrics, and Config Reloader. At the bottom left is a '+ Add query' button.

Metric	Value	Timestamp
up{instance="coelho-realtime-minio.coelho-realtime", job="coelho-realtime-minio", namespace="coelho-realtime"}	1	2023-09-01T10:00:00Z
up{container="grafana", endpoint="http-web", instance="10.42.1.245:3000", job="coelho-realtime-grafana", namespace="coelho-realtime", pod="coelho-realtime-grafana-5f478675b8-rfhtj", service="coelho-realtime-grafana"}	1	2023-09-01T10:00:00Z
up{container="coelho-realtime-kafka-producers-container", endpoint="metrics", instance="10.42.1.243:8000", job="coelho-realtime-kafka-producers", namespace="coelho-realtime", pod="coelho-realtime-kafka-producers-577f7b7c86-sss8m", service="coelho-realtime-kafka-producers"}	1	2023-09-01T10:00:00Z
up{container="spark-streaming", endpoint="spark-ui-eta", instance="10.42.1.238:4041", job="coelho-realtime-spark-streaming", namespace="coelho-realtime", pod="coelho-realtime-spark-streaming-6b75cd7f7-vdfwt", service="coelho-realtime-spark-streaming"}	1	2023-09-01T10:00:00Z
up{container="node-exporter", endpoint="http-metrics", instance="172.23.0.5:9100", job="node-exporter", namespace="coelho-realtime", pod="coelho-realtime-prometheus-node-exporter-2vblp", service="coelho-realtime-prometheus-node-exporter"}	1	2023-09-01T10:00:00Z
up{container="spark-streaming", endpoint="spark-ui-ecci", instance="10.42.1.238:4042", job="coelho-realtime-spark-streaming", namespace="coelho-realtime", pod="coelho-realtime-spark-streaming-6b75cd7f7-vdfwt", service="coelho-realtime-spark-streaming"}	1	2023-09-01T10:00:00Z
up{container="mlflow", endpoint="http", instance="10.42.0.105:5000", job="coelho-realtime-mlflow", namespace="coelho-realtime", pod="coelho-realtime-mlflow-65b6d999f5-pm86f", service="coelho-realtime-mlflow"}	1	2023-09-01T10:00:00Z
up{container="spark-master", endpoint="http", instance="10.42.0.107:8080", job="coelho-realtime-spark-master-svc", namespace="coelho-realtime", pod="coelho-realtime-spark-master-0", service="coelho-realtime-spark-master-svc"}	1	2023-09-01T10:00:00Z
up{container="spark-streaming", endpoint="spark-ui-tfd", instance="10.42.1.238:4040", job="coelho-realtime-spark-streaming", namespace="coelho-realtime", pod="coelho-realtime-spark-streaming-6b75cd7f7-vdfwt", service="coelho-realtime-spark-streaming"}	1	2023-09-01T10:00:00Z
up{container="node-exporter", endpoint="http-metrics", instance="172.23.0.2:9100", job="node-exporter", namespace="coelho-realtime", pod="coelho-realtime-prometheus-node-exporter-gz4gg", service="coelho-realtime-prometheus-node-exporter"}	1	2023-09-01T10:00:00Z
up{endpoint="http", instance="10.42.1.242:8000", job="coelho-realtime-fastapi", namespace="coelho-realtime", pod="coelho-realtime-fastapi-6bbb9b9b88-9z7kt", service="coelho-realtime-fastapi"}	1	2023-09-01T10:00:00Z
up{container="metrics", endpoint="http-metrics", instance="10.42.1.241:9121", job="coelho-realtime-redis-metrics", namespace="coelho-realtime", pod="coelho-realtime-redis-master-0", service="coelho-realtime-redis-metrics"}	1	2023-09-01T10:00:00Z
up{container="kube-prometheus-stack", endpoint="https", instance="10.42.1.240:10250", job="coelho-realtime-kube-prome-operator", namespace="coelho-realtime", pod="coelho-realtime-kube-prome-operator-5548f495c8-djnq5", service="coelho-realtime-kube-prome-operator"}	1	2023-09-01T10:00:00Z
up{container="alertmanager", endpoint="http-web", instance="10.42.1.246:9093", job="coelho-realtime-kube-prome-alertmanager", namespace="coelho-realtime", pod="alertmanager-coelho-realtime-kube-prome-alertmanager-0", service="coelho-realtime-kube-prome-alertmanager"}	1	2023-09-01T10:00:00Z
up{container="metrics", endpoint="http-metrics", instance="10.42.0.112:9187", job="coelho-realtime-postgresql-metrics", namespace="coelho-realtime", pod="coelho-realtime-postgresql-0", service="coelho-realtime-postgresql-metrics"}	1	2023-09-01T10:00:00Z
up{container="minio", endpoint="http", instance="10.42.0.111:9000", job="coelho-realtime-minio", namespace="coelho-realtime", pod="coelho-realtime-minio-7cf4ff575d-gppfv", service="coelho-realtime-minio"}	1	2023-09-01T10:00:00Z
up{container="config-reloader", endpoint="reloader-web", instance="10.42.1.246:8080", job="coelho-realtime-kube-prome-alertmanager", namespace="coelho-realtime", pod="alertmanager-coelho-realtime-kube-prome-alertmanager-0", service="coelho-realtime-kube-prome-alertmanager"}	1	2023-09-01T10:00:00Z
up{container="kube-state-metrics", endpoint="http", instance="10.42.1.244:8080", job="kube-state-metrics", namespace="coelho-realtime", pod="coelho-realtime-kube-state-metrics-78d45896c6-7jbzk", service="coelho-realtime-kube-state-metrics"}	1	2023-09-01T10:00:00Z
up{container="config-reloader", endpoint="reloader-web", instance="10.42.1.247:8080", job="coelho-realtime-kube-prome-prometheus", namespace="coelho-realtime", pod="prometheus-coelho-realtime-kube-prome-prometheus-0", service="coelho-realtime-kube-prome-prometheus"}	1	2023-09-01T10:00:00Z
up{container="prometheus", endpoint="http-web", instance="10.42.1.247:9090", job="coelho-realtime-kube-prome-prometheus", namespace="coelho-realtime", pod="prometheus-coelho-realtime-kube-prome-prometheus-0", service="coelho-realtime-kube-prome-prometheus"}	1	2023-09-01T10:00:00Z

## PromQL query

Prometheus Query Language expression executed against the time-series database.

The up metric is a built-in indicator that returns 1 if a target is reachable and 0 if it is down, serving as the foundation for all service health alerts in the platform.

2

## PromQL query results

The query returns 20 monitored targets across the platform: FastAPI, Kafka Producers, Spark (Master + 3 Streaming jobs), MLflow, MinIO, Redis, PostgreSQL, Grafana, Alertmanager, Prometheus, Kube State Metrics, and Node Exporters — all reporting up = 1 (healthy).

These results feed the alerting rules that trigger notifications when any service goes down.

# MICROSERVICES - GRAFANA

Grafana

Home > Dashboards

Search... ctrl+k ? Sign in

## Dashboards

Create and manage dashboards to visualize your data

Search for dashboards and folders

Filter by tag Starred

Name Tags

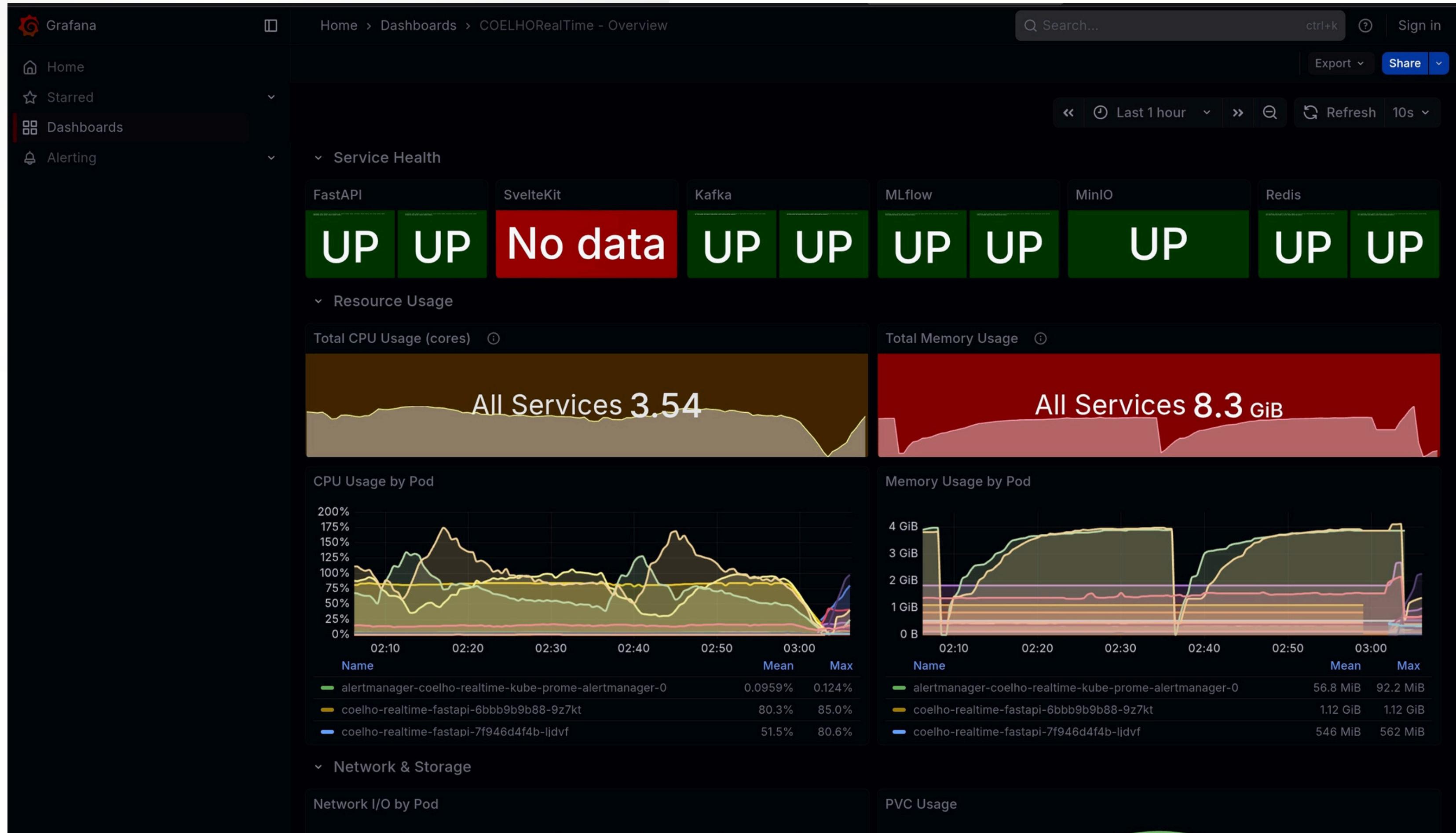
- Alertmanager / Overview alertmanager-mixin
- Apache Spark Cluster delta-lake spark streaming
- COELHOREalTime - FastAPI Application application coelho-realtime fastapi
- COELHOREalTime - Kafka coelho-realtime jmx kafka streaming
- COELHOREalTime - Kafka Producers coelho-realtime kafka producers
- COELHOREalTime - MinIO coelho-realtime minio s3 storage
- COELHOREalTime - ML Pipeline coelho-realtime fastapi kafka ml-pipeline mlflow
- COELHOREalTime - Overview coelho-realtime ml-pipeline overview
- COELHOREalTime - PostgreSQL coelho-realtime database postgresql
- COELHOREalTime - Redis cache coelho-realtime redis
- CoreDNS coredns dns
- Grafana Overview
- Kubernetes / API server kubernetes-mixin
- Kubernetes / Compute Resources / Multi-Cluster kubernetes-mixin
- Kubernetes / Compute Resources / Cluster kubernetes-mixin
- Kubernetes / Compute Resources / Namespace (Pods) kubernetes-mixin

1

## Grafana dashboards

The platform ships with 10 pre-built dashboards, automatically provisioned via Kubernetes ConfigMaps: COELHO RealTime Overview (platform-wide health), ML Pipeline (training and prediction metrics), FastAPI Detailed (API latency, throughput, errors), Kafka Producers (message rates, send latency), Kafka (broker health, partitions), Spark (master and workers), Spark Streaming (batch processing, records ingested), PostgreSQL (connections, deadlocks), Redis (memory, commands), and MinIO (storage, requests). All dashboards query Prometheus as their data source and update in real time.

# MICROSERVICES - GRAFANA



# MICROSERVICES - ALERTMANAGER

The screenshot shows the Alertmanager interface with a dark theme. At the top, there's a navigation bar with links for Alertmanager, Alerts, Silences, Status, Settings, and Help. A prominent "New Silence" button is located in the top right corner. Below the navigation, there are two tabs: "Filter" (selected) and "Group". On the right side of the header, there are filter options: "Receiver: All" and three checkboxes for "Silenced", "Inhibited", and "Muted". A "Custom matcher, e.g. env='production'" input field is present. In the center, there are two alert groups. The first group, under "null", has an alertname of "Watchdog", severity of "none", and 1 alert. The second group, also under "null", has an alertname of "InfoInhibitor", namespace of "coelho-realtime", severity of "none", and 4 alerts. At the bottom right of the interface, there's a "Silence" button with a plus sign.

Alertmanager receives firing alerts from Prometheus and handles routing, grouping, deduplication, and silencing.

Alerts are grouped by alertname, namespace, and severity, with critical alerts dispatched within 10 seconds and warnings batched at 10-minute intervals.

Inhibition rules prevent cascading noise — if Kafka goes down, all downstream Kafka alerts are suppressed; if PostgreSQL is unavailable, MLflow alerts are silenced.

The routing tree supports Slack, PagerDuty, Discord, email, and webhook receivers, all pre-configured and ready to enable.

# MICROSERVICES - KARMA

The screenshot shows the Karma dashboard interface with three alert cards:

- Watchdog** (severity: none):
  - Description: This is an alert meant to ensure that the entire alerting pipeline is functional. This alert is always firing, therefore it should always be firing in Alertmanager and always fire against a receiver. There are integrations with various notification mechanisms that send a notification when this alert is not firing. For example the "DeadMansSnitch" integration in PagerDuty.
  - Summary: An alert that should always be firing to certify that Alertmanager is working properly.
  - Timestamp: 22 minutes ago
  - Labels: cluster: coelho-realtime, prometheus: coelho-realtime/coelho-realtime-kube-prome-prometheus
  - Actions: runbook\_url
- CPUThrottlingHigh** (severity: info):
  - Description: 73.92% throttling of CPU in namespace coelho-realtime for container spark-worker in pod coelho-realtime-spark-worker-0 on cluster.
  - Summary: Processes experience elevated CPU throttling.
  - Timestamps: 5 minutes ago, 9 minutes ago
  - Labels: cluster: coelho-realtime, container: spark-worker, instance: 172.23.0.5:10250, pod: coelho-realtime-spark-worker-0, prometheus: coelho-realtime/coelho-realtime-kube-prome-prometheus, service: coelho-realtime-kube-prome-kubelet
  - Actions: runbook\_url
- InfoInhibitor** (severity: none):
  - Description: This is an alert that is used to inhibit info alerts. By themselves, the info-level alerts are sometimes very noisy, but they are relevant when combined with other alerts. This alert fires whenever there's a severity="info" alert, and stops firing when another alert with a severity of 'warning' or 'critical' starts firing on the same namespace. This alert should be routed to a null receiver and configured to inhibit alerts with severity="info".
  - Summary: Info-level alert inhibition.
  - Timestamps: 5 minutes ago, 9 minutes ago
  - Labels: alertstate: firing, container: spark-worker, pod: coelho-realtime-spark-worker-0, alertstate: pending, container: metrics, pod: coelho-realtime-postgresql-0
  - Actions: runbook\_url

**Karma** is a lightweight Alertmanager dashboard that provides real-time visibility into all firing, suppressed, and silenced alerts.

It connects to Alertmanager and offers label-based filtering, color-coded severity grouping, autocomplete search, and one-click silence creation — features that go beyond Alertmanager's built-in UI.

Karma enables fast alert triage during incidents by aggregating and deduplicating alerts across sources into a single overview.

“

COELHO REALTIME IS AN END-TO-END, CLOUD-NATIVE MACHINE LEARNING PLATFORM THAT UNIFIES REAL-TIME STREAMING, INCREMENTAL AND BATCH TRAINING, AND FULL OBSERVABILITY INTO A SINGLE KUBERNETES-NATIVE ARCHITECTURE — FROM KAFKA INGESTION TO MODEL SERVING, FULLY AUTOMATED AND PRODUCTION-READY.

”

# LET'S WORK TOGETHER

---



<https://rafaelcoelho1409.github.io/>



rafaelcoelho1409@hotmail.com  
coelhorafael@proton.me