Distributed Fraud Detection System – Documentation

# Overview

This project implements a distributed fraud detection system using two main components:  
- queue\_service: A RESTful microservice for message queue management. Handles creation, pushing, pulling, listing, and deletion of named queues, which serve as buffers between clients and high-performance processing services.  
- mpi\_prediction: A distributed machine learning prediction service, using MPI (Message Passing Interface) and multiple worker processes to perform parallel, high-throughput fraud predictions on transaction data.  
  
These services communicate via HTTP APIs, with all transactions and results flowing through the message queues.

# Architecture & Main Components

## queue\_service

Purpose: Manages in-memory message queues for both incoming transactions and outgoing prediction results.  
Endpoints:  
 • POST /api/v1/queues/{queue\_name}/push — Add a message to a queue.  
 • POST /api/v1/queues/{queue\_name}/pull — Remove and return the first message from a queue.  
 • GET /api/v1/queues — List all available queues.  
 • POST /api/v1/queues/{queue\_name} — Create a new queue.  
 • DELETE /api/v1/queues/{queue\_name} — Delete a queue and all messages.  
Features:  
 • Maximum queue length and persist interval are configurable.  
 • Handles all error cases (e.g., queue not found, empty queue, queue full).  
 • Queue state is periodically saved to persistent storage and restored on restart.  
 • Logs every client request and server response, including source, destination, headers, and message body.

## mpi\_prediction

Purpose: Performs high-performance, distributed fraud prediction on transactions pulled from the message queue.  
Operation:  
 • Loads a pre-trained RandomForest model at startup.  
 • Pulls up to P transactions (default 5, configurable) per batch from the transactions queue.  
 • Master process distributes requests to worker processes using MPI; each worker performs a prediction.  
 • Results are gathered and pushed to the results queue.  
 • Repeats continuously as new data arrives.  
Batching & Blocking: If fewer than P messages are available, processes available messages and waits (blocks) if queue is empty.  
Containerization: Docker Compose is provided for launching all services and multiple MPI workers.

# Setup & Usage

1. Build and start all services (requires Docker and Docker Compose):  
 docker compose build  
 docker compose up  
  
2. Submit a transaction for prediction:  
 curl -X POST "http://localhost:7500/api/v1/queues/transactions/push" \  
 -H "Content-Type: application/json" \  
 -d '{"transaction\_id": "test-001", "data": {"status": "submitted", "vendor\_id": 73, "amount": 229.88, "timestamp": 1700000000}}'  
  
3. List all queues:  
 curl http://localhost:7500/api/v1/queues  
  
4. Fetch a prediction result:  
 curl http://localhost:7500/api/v1/queues/results/pull

# Requirements & Features

Queue Service:  
 • API for push, pull, list, create, and delete operations  
 • Configurable queue length and persist interval  
 • Full error handling and logging  
 • Periodic persistent storage and recovery  
  
Prediction Service:  
 • Distributed batch prediction using MPI (default 5 workers)  
 • Pulls transactions and returns results via queue service  
 • Blocking behavior when queue is empty  
 • Configurable number of processors  
  
General:  
 • Fully containerized (Docker, Docker Compose)  
 • All client/server interactions are logged  
 • Submission includes all required files and this documentation

# Team Members

* Samir Ibrahimov
* Jaime Gracia Gil, IMC Krems, 2025