Project: Chicago Restaurant Health Inspection Machine Learning Pipeline

This project develops an end to end machine learning pipeline based on the Chicago Open Portal Food Inspection Database. The use case for the pipeline is as follows: ***A public health inspector has completed an inspection of a restaurant and has to decide whether it should be a pass, pass with condition, or fail. The decision needs to be consistent with general historical practice. The inspector needs a decision support tool that accepts data collected from the on-site inspection and makes a recommendation about pass, pass with condition, or fail that is consistent with historical inspection decisions.*** To support this, a cloud-native machine learning pipeline that collects, cleans, transforms, and stores data, supports exploratory data analysis, machine learning training and prediction, and inspection report generation has been created. The design architecture is a containerized set of microservices implemented in the Google Cloud Platform.

The pipeline is implemented in google cloud run and has the following microservices (see Appendix A): (1) Extractor: Fetches food inspection data from the Chicago Open Data API and stores raw JSON files in GCS; (2) Cleaner: Cleans raw data by removing invalid or redundant rows and standardizing formats. Outputs cleaned NDJSON and PARQUET files to GCS; (3) Loaders: Loads cleaned Parquet and NDJSON files into BigQuery tables; (4) Trigger: Coordinates workflow execution — receives external triggers, initiates extractor, cleaner, and loader steps by sending POST requests; (5) Pipeline Controller: UI to manually trigger pipeline components, monitor service health, and inspect file and bucket status; (6) EDA Dashboard: Explores and visualizes data trends and distributions from BigQuery; (7) ML Training / Service: Trains prediction models and exposes a REST endpoint for running predictions; (8) ML Dashboard: Displays prediction results, model performance metrics, and enables export of risk scoring reports for decision support.

Pipeline Evaluation Summary

Advantages:
- Modular architecture with microservices enables independent deployment and scaling.
- CI/CD with GitHub Actions ensures automated, reproducible deployments.
- Cloud Run provides auto-scaling and low overhead for container orchestration.
- Data lake + BigQuery combo offers flexible storage and fast analytics.
- Streamlit dashboards simplify UI development for non-technical users.

Disadvantages:
- Cloud Run limitations (e.g., cold starts, monitoring overhead) can impact latency.
- Manual orchestration dependencies (e.g., Trigger) may be brittle without retries or queueing.
- Git workflows not implemented for all services.
- Grafana dashboards not implemented for all services.
- Limited fault tolerance and recovery.

Future Improvement Areas

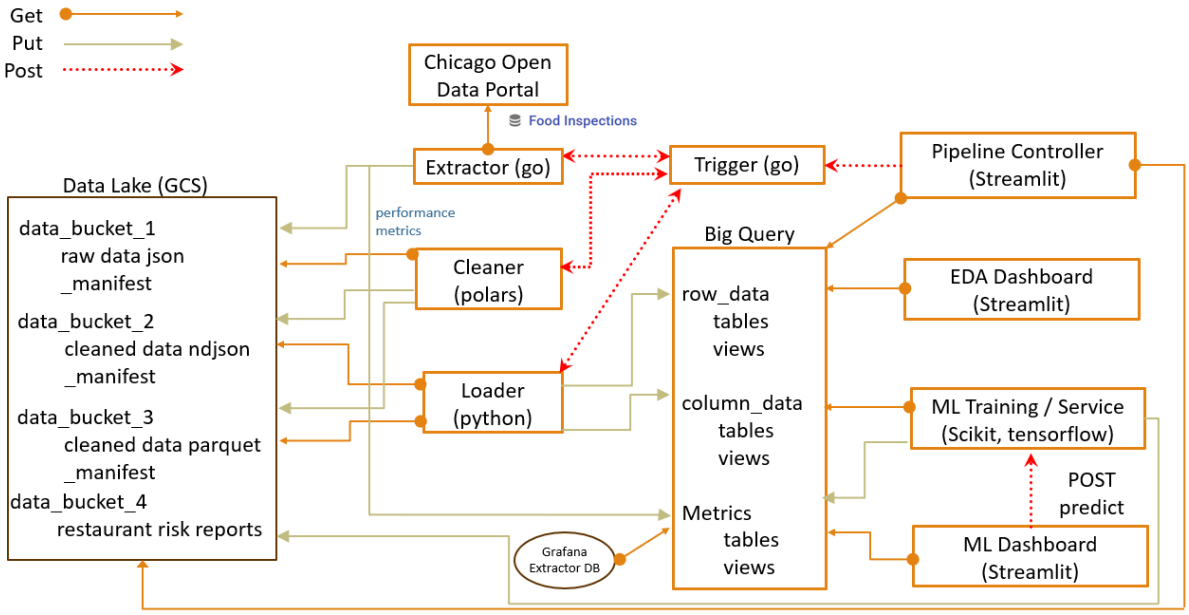
1. Orchestration and Retry Logic
   - Current Issue: Trigger directly calls services without retries or status tracking.
   - Improvement: Use Google Cloud Workflows or Cloud Tasks for reliable, observable orchestration.
2. Logging and Alerting
   - Current Issue: Logs are fragmented and there's no automated alerting.
   - Improvement: Centralize logs with Google Cloud Logging and set up alerting in Grafana.
4. Standardized DevOps
   - Current Issue: CI/CD and monitoring are inconsistently implemented across services.
   - Improvement: Use GitHub reusable workflows and Grafana provisioning to standardize deployment and dashboards.

Distinction from Peer Projects

This project was developed not simply as a course assignment, but as part of my preparation to teach an advanced-level course in data pipeline engineering for senior undergraduates and graduate students. Professional practice was foremost in my mind, and every effort was made to produce a system that reflects real-world engineering standards.

This work delivers a full-stack, cloud-native pipeline with modular services, end-to-end orchestration, automated deployment, and integrated monitoring. The use case is oriented toward decision support, and the system architecture is designed to be extensible, observable, and close to production-ready. As such, I think it represents a more comprehensive and practice-driven approach to data engineering.

Appendix A Pipeline Schematic



Appendix B Pipeline Technologies Used

Artifact Registry: Stores and manages Docker container images built and deployed by GitHub Actions.

Docker: Used to containerize microservices for consistent, portable deployment across environments.

GitHub: Hosts the project codebase and manages version control; also triggers CI/CD via GitHub Actions.

Go (Golang): Used to implement fast, lightweight services like the Extractor and Trigger.

Google BigQuery: Cloud data warehouse that stores structured inspection and modeling data for querying and reporting.

Google Cloud Platform (GCP): Primary cloud provider for infrastructure, deployment, storage, and analytics.

Google Cloud Run: Deploys and scales containerized microservices automatically without server management.

Google Cloud Storage (GCS): Used as a data lake for storing raw JSON, cleaned NDJSON, Parquet files, and model output reports.

Grafana: Monitors pipeline performance metrics (e.g., request volume, latency, failures) using visual dashboards.

Makefile: Automates local development and deployment tasks such as building, testing, and launching services.

Python: Core language for data processing, cleaning, loading, and ML services in the pipeline.

REST API: Enables HTTP-based communication between services (e.g., Trigger POSTs to Extractor/Cleaner/Loader).

Scikit-learn: Provides ML model training and inference tools used in the prediction service.

SQL: Used in BigQuery to query and transform structured inspection data and model results.