AutoML FraudShield – V2 SRS + TDD + Deployment Manifest

0. Overview & Goals

Project Name: AutoML FraudShield V2
Role: AI/ML Engineer · MLOps Lead · AI Architect

V1 Recap:
V1 delivered a working fraud detection service on GCP with:

- Terraform-provisioned infra (BigQuery, GCS, Feature Store)
- Vertex AI Pipelines training an XGBoost model
- Model registration via custom SDK wrapper
- A FastAPI scoring service on Cloud Run using online Feature Store lookups

V2 Goal:
Turn FraudShield into a **production-grade, explainable, self-monitoring system** by adding:

1. **Model Intelligence & Explainability** (SHAP, feature validation).
2. **Safer Deployment & Serving** (Vertex Endpoint, shadow/canary, externalized config).
3. **Monitoring & Automation** (prediction logging, drift detection, auto retraining, human dashboard).

---

1. Scope & Requirements (SRS)

1.1 In-Scope (V2)

1. Add SHAP-based explainability to the scoring path.

2. Integrate feature validation into the training pipeline.
3. Replace GCS-based model loading in the API with serving via Vertex AI Endpoint.
4. Introduce shadow/canary deployment support (champion vs challenger).
5. Externalize key risk thresholds & config.
6. Implement streaming prediction logging (Pub/Sub → BigQuery).
7. Implement drift & performance monitoring with auto retrain triggers.
8. Provide a basic human dashboard for monitoring & manual retrain requests.

1.2 Out-of-Scope (V2)

- Full multi-tenant support and tenant-specific models.
- Complex, UI-driven approval workflows (we'll assume a simple "click-to-trigger retrain" in the dashboard).
- Advanced AutoML or model ensembles (we'll stay with XGBoost as the primary model).

1.3 Functional Requirements

1.3.1 Model Intelligence & Explainability (FR-E)

FR-E1 – SHAP Explanations per Prediction

- The scoring API (`POST /v1/score`) MUST compute and return SHAP-style feature contribution scores for each prediction.
- At minimum, the top N (e.g., N = 5) features by absolute contribution SHOULD be returned with each response.
- For tree models (XGBoost), TreeSHAP MUST be used for correctness and speed.

FR-E2 – Feature Validation in Training Pipeline

- The training pipeline MUST perform feature validation before model training using a data validation library (e.g., Great Expectations or Pandera).
- Validation MUST include:
    - Schema checks (column presence & types).
    - Missing value thresholds per feature.
    - Basic statistical sanity checks (mean, std, min/max vs baseline).
- If validation fails, the pipeline MUST stop and record the failure reason in logs and pipeline metadata.

1.3.2 Enhanced Deployment & Serving (FR-D)

FR-D1 – Shadow/Canary Deployment Support

- The deployment process MUST support running a **champion** and **challenger** model simultaneously via Vertex AI Endpoint traffic splitting or routing rules.
- It MUST support directing 5–10% of traffic to the challenger for evaluation while the champion continues serving the majority.
- Shadow results SHOULD be logged for comparison but not used for real-time decisions.

FR-D2 – Vertex Endpoint-based Serving

- The scoring service MUST call a Vertex AI Endpoint to score requests instead of loading `model.bst` directly from GCS at runtime.
- Model versions MUST be deployed to the endpoint via CI/CD and selectable via traffic splitting (champion/challenger).
- The API MUST remain backwards compatible: same `POST /v1/score` contract as V1.

FR-D3 – Externalized Risk Configuration

- Risk band thresholds and other critical scoring parameters (e.g., “high risk ≥ 0.6”) MUST be loaded from a configuration source, NOT hardcoded.

- Acceptable sources for V2:
    - Environment variables, or
    - A GCS-hosted JSON/YAML config file, or
    - Secret Manager (for sensitive values).
- The service SHOULD refresh configuration on startup and periodically (or on-demand).

1.3.3 Monitoring & Automation (FR-M)

FR-M1 – Real-Time Prediction Logging

- For every prediction, the scoring API MUST:
    - Publish a normalized prediction record to a Pub/Sub topic (e.g., `fraudshield-predictions`) including:
        - Request features (post-feature-engineering vector or a stable subset)
        - Prediction score
        - Model version ID
        - SHAP summary (e.g., top N features)
    - A downstream pipeline MUST write these records to BigQuery (`fraudshield.predictions_log_v2`) with low lag.

FR-M2 – Drift & Performance Monitoring

- A scheduled monitoring job MUST:
    - Compute input feature drift metrics (e.g., PSI, Jensen-Shannon divergence, or similar) vs. the baseline training window.
    - Compute performance metrics (ROC AUC, PR AUC, etc.) over the last N days of predictions that have acquired labels.
- Thresholds for drift and performance degradation MUST be defined and configurable.

FR-M3 – Automatic Retrain Trigger

- When drift or performance metrics exceed thresholds:
    - The monitoring job MUST trigger the training pipeline (e.g., via Vertex AI Pipelines SDK, Cloud Function, or Cloud Scheduler → HTTP).
    - Retrain events MUST be logged with reason (e.g., “PSI > 0.2 on amount”).

FR-M4 – Human-in-the-Loop Dashboard

- A lightweight web dashboard (Streamlit, or similar) MUST:
    - Display key metrics: model performance, drift statistics, volume of predictions, and retrain events.
    - Allow an authorized user to manually trigger a retrain (e.g., press “Retrain Now” → call pipeline launcher).
    - Show current champion/challenger versions and traffic split.

1.4 Non-Functional Requirements (NFR)

- **NFR-1 – Latency:** V2 scoring latency SHOULD stay within p95 < 250 ms despite Vertex Endpoint hop and SHAP computation (SHAP can be computed locally using a parallel artifact).
- **NFR-2 – Reliability:** Drift & monitoring jobs MUST fail gracefully and log clearly; they MUST NOT block scoring.
- **NFR-3 – Observability:** All major steps (scoring, logging, drift, retrain triggers) MUST include trace IDs and structured logs.
- **NFR-4 – Backwards Compatibility:** V2 MUST keep the same scoring endpoint signature, with explanations added in a non-breaking way (e.g., an additional `explanations` field).

---

2. High-Level Architecture (TDD – System View)

2.1 Architecture Changes from V1 → V2

V1 major elements:

- Training: Vertex AI Pipeline → XGBoost → Model Registry → artifact in GCS.
- Serving: FastAPI on Cloud Run loading `model.bst` from GCS and calling Feature Store.

V2 enhancements:

1. **Serving via Vertex AI Endpoint**
   - Training pipeline registers model and deploys it to a Vertex Endpoint.
   - Scoring service calls Vertex Endpoint instead of loading model locally.
2. **Explainability Layer**
   - Training pipeline creates and saves SHAP explainer configuration and a “background dataset” snapshot in GCS.
   - Scoring service loads a local copy of the model artifact and background dataset to run TreeSHAP locally for explanations, while still using Vertex Endpoint for the primary score. (Champion score from endpoint is ground truth; SHAP is from a synced local copy.)
3. **Prediction Logging Pipeline**
   - Scoring service → Pub/Sub `fraudshield-predictions` → Dataflow or Cloud Run → BigQuery `predictions_log_v2`.
4. **Monitoring & Retrain Control Loop**
   - Scheduled job (Cloud Scheduler → Cloud Run/Vertex Pipeline) reads from BigQuery to compute drift & performance metrics.
   - When thresholds are breached, job triggers training pipeline.
5. **Dashboard**

- Streamlit app reading from:
  - BigQuery (metrics and logs).
  - Maybe Firestore / GCS config for current model versions & thresholds.

2.2 System Components (V2)

1. **Training Pipeline (Vertex AI Pipelines – V2)**
   - Components:
     - `load_data_and_features`
     - `validate_features` (new)
     - `train_xgboost_model`
     - `evaluate_model`
     - `register_and_deploy_model` (deploy to Vertex Endpoint)
     - `export_shap_artifacts` (background dataset + feature map)
2. **Scoring API (FastAPI on Cloud Run – V2)**
   - Responsibilities:
     - Fetch online features (as in V1).
     - Call Vertex Endpoint for primary score.
     - Compute SHAP values using a local copy of the model + background dataset.
     - Build response (score + risk band + top-N SHAP explanations).
     - Publish prediction record to Pub/Sub for logging.

3. **Prediction Logging Pipeline**
    - Pub/Sub `fraudshield-predictions` → (Dataflow / Cloud Run consumer) → BigQuery `predictions_log_v2`.
4. **Monitoring Job**
    - Scheduled Cloud Run job:
        - Queries training baseline distribution.
        - Queries recent prediction data window.
        - Computes drift metrics.
        - Computes performance metrics (using labels, if available).
        - Logs results & triggers retrain if thresholds breached.
5. **Dashboard (Streamlit)**
    - Views:
        - Model performance & drift overview.
        - Prediction volume charts.
        - Retrain history (with reasons).
        - Button to manually trigger a retrain call.

---

3. Detailed Design (TDD – Component-Level)

3.1 Training Pipeline – V2 Spec

Step 1: Load Data & Features

- As in V1 (BigQuery transactions + offline Feature Store join).

- Output: training DataFrame (features + label), metadata (feature list, training window).

Step 2: Feature Validation (FR-E2)

- Implement a `validate_features` component using Great Expectations or Pandera:
    - Validate presence & type of all required features.
    - Check missing value percentage per feature (e.g., < 5% allowed).
    - Check range constraints where appropriate (e.g., amount >= 0).
    - Compare basic stats to baseline stored in a `feature_stats.json` (v1 or initial run).
- On failure:
    - Mark pipeline as FAILED.
    - Persist validation report in GCS and pipeline metadata.

Step 3: Train XGBoost Model

- Train on validated data.
- Use class weights or sample weighting if needed.
- Save model artifacts:
    - `model.bst`
    - `feature_map.json` (list of feature names in order)

Step 4: Evaluate Model

- Compute metrics:
    - ROC AUC, PR AUC
    - F1 @ default threshold, maybe custom thresholds.

- Export metrics to a JSON artifact and log to Vertex Experiments.

Step 5: Register & Deploy Model (FR-D2, FR-D1)

- Register model with Model Registry.
- Deploy to Vertex Endpoint:
    - If endpoint does not exist, create it.
    - If exists:
        - Keep current "champion" model.
        - Deploy new model as "challenger" with small traffic fraction (e.g., 10%).
- Output: model resource name, endpoint ID, champion & challenger versions, traffic split.

Step 6: SHAP Artifacts (FR-E1)

- Generate a small "background dataset" (e.g., 1000 random non-fraud transactions) for SHAP.
- Save:
    - `background_data.parquet`
    - `feature_map.json`
- These will be used by the scoring service to compute SHAP values offline.

3.2 Scoring API – V2 Spec

Endpoint: `POST /v1/score` (unchanged path & core request)

Request: same schema as V1 (transaction fields + IDs).

Internal Steps:

1. **Config Load (FR-D3)**
    - On startup:
        - Load risk thresholds and flags from:
            - env vars, or
            - GCS-hosted `config/risk_config.json`.
    - Optionally reload periodically.
2. **Feature Retrieval**
    - Use Vertex Feature Store online API as in V1.
    - Assemble feature vector in the same order as `feature_map.json`.
3. **Model Scoring via Vertex Endpoint (FR-D2)**
    - Construct prediction request payload and call Vertex Endpoint.
    - Get back fraud probability and model version metadata.
4. **SHAP Computation (FR-E1)**
    - Load:
        - Local `model.bst` (synced artifact)
        - `background_data.parquet`
    - Use tree SHAP (e.g., `shap.TreeExplainer`) to compute:
        - Per-feature contributions for this sample.
    - Extract top N features by absolute contribution.
5. **Risk Banding (FR-D3)**
    - Use externally loaded thresholds to assign:

        - `LOW`, `MEDIUM`, `HIGH`.

6. **Prediction Logging (FR-M1)**
    - Build log record with:
        - transaction & entity IDs
        - feature vector or summarized features
        - score, risk band
        - model version
        - top-N SHAP explanations
    - Publish to Pub/Sub `fraudshield-predictions`.

Response

```
{
  "transaction_id": "tx_123",
  "score": 0.82,
  "risk_band": "HIGH",
  "model_version": "fraudshield-xgb-v2",
  "explanations": [
    {"feature": "txn_count_7d", "contribution": 0.15},
    {"feature": "txn_amount_sum_7d", "contribution": 0.11},
    {"feature": "avg_ticket_30d", "contribution": 0.09}
  ]
}
```

7.

3.3 Monitoring & Retraining

Prediction Logging Pipeline:

- **Input:** Pub/Sub `fraudshield-predictions`

- **Processor:** Dataflow or Cloud Run subscriber:
    - Deserialize JSON log records.
    - Normalize schema, add ingestion timestamps.
    - Write to BigQuery `fraudshield.predictions_log_v2`.

Monitoring Job:

- Cloud Run job launched via Cloud Scheduler (e.g., hourly or daily).
- Steps:
    1. Read baseline training distribution stats from `feature_stats.json` and training set snapshot.
    2. Query last N days (e.g., 7/30) from `predictions_log_v2`.
    3. Compute:
        - Drift metric per key feature (PSI, KL, or simplified).
        - Performance metrics (only where labels are present).
    4. Compare metrics against config thresholds.
    5. If thresholds exceeded:
        - Call `launch_training_pipeline_v2()` (via HTTP or SDK).
        - Log alert + reason in BigQuery `fraudshield.monitoring_events`.

3.4 Dashboard

- **Tech:** Streamlit (simple to build, matches your other projects).
- **Data Sources:**

    - BigQuery `predictions_log_v2` (for metrics charts).
    - BigQuery `monitoring_events` (for drift/retrain history).
    - Maybe a config endpoint or file for current model versions/traffic split.

Views:

1. **Overview**
    - Cards for:
        - Current champion version
        - Challenger version (if any)
        - Last retrain time & reason
        - Current average score / risk mix
2. **Performance & Drift**
    - Time series of ROC AUC / PR AUC (where labels exist).
    - Drift metrics per key feature.
3. **Retraining Control**
    - Table of recent retrain events.
    - Button: “Trigger Retrain Now” → calls pipeline launcher.

---

4. Deployment Manifest (DM – Infra & Environments)

4.1 New / Updated Resources

In addition to V1 resources:

Vertex AI:

- Vertex Endpoint `fraudshield-endpoint` with:
  - Champion deployment
  - Challenger deployment
  - Traffic split config

Pub/Sub:

- Topic: `fraudshield-predictions`

BigQuery:

- Table: `fraudshield.predictions_log_v2`
- Table: `fraudshield.monitoring_events`

Cloud Run:

- Service: `fraudshield-monitoring-job-{env}`
- Service: `fraudshield-dashboard-{env}` (Streamlit)

Cloud Scheduler:

- Job: `fraudshield-monitoring-schedule-{env}` → triggers monitoring job

GCS:

- `gs://fraudshield-config-{env}/risk_config.json`
- `gs://fraudshield-model-artifacts-{env}/v2/...` (for SHAP background + model copy)

Terraform modules from V1 can be extended with:

- `pubsub_predictions`
- `vertex_endpoint` (with traffic_split vars)
- `monitoring_job`
- `dashboard_service`

---

5. Repository / Directory Structure (V2 Additions)

Starting from the V1 layout, V2 adds/extends:

```
fraudshield/
├── docs/
│   ├── FraudShield_SRS_TDD_DM_v1.md
│   └── FraudShield_V2_SRS_TDD_DM.md      # <-- this file
├── config/
│   └── risk_config.example.json          # sample thresholds
├── pipelines/
│   └── training/
│       ├── pipeline_definition_v2.py     # V2 pipeline with
validation + SHAP + deploy
│       ├── components/
│       │   ├── validate_features_component.py
│       │   ├── export_shap_artifacts_component.py
│       │   └── deploy_to_vertex_endpoint_component.py
├── api/
│   ├── app/
│   │   ├── main.py                       # updated to call Vertex
Endpoint + SHAP
│   │   ├── services/
│   │   │   ├── feature_store_client.py
│   │   │   ├── model_endpoint_client.py  # new: calls Vertex Endpoint
│   │   │   ├── shap_explainer.py         # new: local SHAP
computation
│   │   │   └── config_loader.py          # new: risk thresholds, etc.
```

```
├── monitoring/
│   ├── monitoring_job.py                   # drift + performance
computation
│   └── launch_monitoring_job.py
├── logging_pipeline/
│   ├── subscriber_main.py                  # Pub/Sub → BigQuery
│   └── Dockerfile
└── dashboard/
    ├── app.py                              # Streamlit dashboard
    └── requirements.txt
```