AutoML FraudShield V3.1: Master Configuration Guide

Version: 3.1 (Production Hardened) **Date:** December 3, 2025 **Scope:** Full Stack (Streaming Infra, Hybrid CPR, API Orchestration, Vertex AI)

---

1. Project Directory Structure

Plaintext

```
fraudshield-v3/
├── README.md
├── .gitignore
├── api/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── app/
│   │   ├── main.py                # [HARDENED] Endpoint Pinning & Orchestration
│   │   └── services/
│   │       └── feature_store_client.py # [HARDENED] Streaming Fetch + Batch TODOs
├── features/
│   ├── create_schema.py
│   ├── variables.py
├── infra/
│   └── terraform/
│       └── envs/
│           └── dev/
│               ├── main.tf          # V3 Infrastructure
│               └── variables.tf
├── models/
│   ├── train_hybrid.py
│   ├── ensemble_cpr/
│   │   ├── Dockerfile
│   │   ├── predictor.py           # Hybrid Ensemble Logic
│   │   └── requirements.txt
├── pipelines/
│   └── training/
│       ├── pipeline_definition_v3.py
│       └── components/
│
```

```
│           └── train_hybrid_model_component.py # [HARDENED] Metric Logging & IsoForest
Notes
├── streaming/
│   ├── pipeline.py                # [HARDENED] Event Time, Validation, Safe Aggregation
│   ├── setup.py
│   ├── requirements.txt
│   ├── generate_stream.py
│   └── run_dataflow.ps1
└── deploy_cpr.py
```

---

2. Infrastructure (Terraform)

Location: `infra/terraform/envs/dev/variables.tf`

Terraform
```
variable "project_id" {
  description = "The GCP Project ID"
  type        = string
  default     = "fraudshield-v3-dev-5320"
}

variable "region" {
  description = "GCP Region"
  type        = string
  default     = "us-central1"
}

variable "env" {
  description = "Environment"
  type        = string
  default     = "dev"
}
```

Location: `infra/terraform/envs/dev/main.tf`

Terraform
```
terraform {
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = ">= 4.0.0"
    }
  }
}

provider "google" {
  project = var.project_id
  region  = var.region
}

# --- Service APIs ---
resource "google_project_service" "enabled_apis" {
  for_each = toset([
    "aiplatform.googleapis.com", "bigquery.googleapis.com",
    "storage.googleapis.com", "artifactregistry.googleapis.com",
    "run.googleapis.com", "cloudbuild.googleapis.com",
    "pubsub.googleapis.com", "dataflow.googleapis.com"
  ])
  service            = each.key
  disable_on_destroy = false
}

# --- Storage & Artifacts ---
resource "google_storage_bucket" "artifacts" {
  name                        = "fraudshield-artifacts-${var.env}-${var.project_id}"
  location                    = var.region
  force_destroy               = true
  uniform_bucket_level_access = true
  depends_on                  = [google_project_service.enabled_apis]
}

resource "google_storage_bucket" "dataflow_temp" {
  name                        = "fraudshield-dataflow-temp-${var.env}-${var.project_id}"
  location                    = var.region
  force_destroy               = true
  uniform_bucket_level_access = true
  depends_on                  = [google_project_service.enabled_apis]
}

resource "google_artifact_registry_repository" "repo" {
  location      = var.region
  repository_id = "fraudshield-repo"
  format        = "DOCKER"
  depends_on    = [google_project_service.enabled_apis]
}
```

```
# --- Pub/Sub (V3 Ingestion) ---
resource "google_pubsub_topic" "raw_events" {
  name       = "fraudshield-raw-events"
  depends_on = [google_project_service.enabled_apis]
}

resource "google_pubsub_subscription" "raw_events_sub" {
  name  = "fraudshield-raw-sub"
  topic = google_pubsub_topic.raw_events.name
  message_retention_duration = "604800s" # 7 days
}

# --- Vertex AI (Feature Store & Endpoint) ---
resource "google_vertex_ai_featurestore" "featurestore" {
  name     = "fraudshield_feature_store_${var.env}"
  region   = var.region
  labels   = { env = var.env }
  online_serving_config {
    fixed_node_count = 1
  }
  depends_on = [google_project_service.enabled_apis]
}

resource "google_vertex_ai_endpoint" "primary" {
  name         = "fraudshield-hybrid-endpoint"
  display_name = "fraudshield-hybrid-endpoint"
  location     = var.region
  depends_on   = [google_project_service.enabled_apis]
}

# --- IAM for Dataflow ---
resource "google_service_account" "dataflow_sa" {
  account_id   = "fraudshield-dataflow-sa"
  display_name = "Dataflow Service Account for FraudShield V3"
}

resource "google_project_iam_member" "dataflow_worker" {
  project = var.project_id
  role    = "roles/dataflow.worker"
  member  = "serviceAccount:${google_service_account.dataflow_sa.email}"
}

resource "google_project_iam_member" "dataflow_admin" {
  project = var.project_id
  role    = "roles/dataflow.admin"
  member  = "serviceAccount:${google_service_account.dataflow_sa.email}"
}

resource "google_project_iam_member" "dataflow_vertex" {
  project = var.project_id
  role    = "roles/aiplatform.user"
  member  = "serviceAccount:${google_service_account.dataflow_sa.email}"
}
```

---

3. Streaming Engine (Hardened)

Location: `streaming/pipeline.py` *Includes fixes for Event Time, Validation, Safe Aggregation, and Retry TODOs.*

```
Python
import argparse
import json
import logging
from datetime import datetime
import apache_beam as beam
from apache_beam.options.pipeline_options import PipelineOptions, GoogleCloudOptions,
StandardOptions
from apache_beam.transforms.trigger import AfterWatermark, AfterProcessingTime,
AccumulationMode
from apache_beam.transforms.window import SlidingWindows
from google.cloud import aiplatform

# --- Configuration ---
PROJECT_ID = "fraudshield-v3-dev-5320"
REGION = "us-central1"
FEATURE_STORE_ID = "fraudshield_feature_store_dev"
SUBSCRIPTION_ID = f"projects/{PROJECT_ID}/subscriptions/fraudshield-raw-sub"

ALLOWED_LATENESS_SECONDS = 300  # 5 minutes
WINDOW_SIZE_SECONDS = 600       # 10 minutes
WINDOW_PERIOD_SECONDS = 60      # 1 minute

class ValidateEvent(beam.DoFn):
    """Filter out malformed events to prevent crashes."""
    def process(self, element):
        try:
            record = json.loads(element.decode("utf-8"))
            required = ["amount", "card_id", "tenant_id", "timestamp"]
            if all(k in record for k in required):
                yield record
            else:
                logging.warning(f"Dropping malformed record: {record}")
                # TODO: Route to Dead Letter Queue (DLQ) here
        except Exception:
            logging.error("Dropping non-JSON record")

class ParseAndTimestamp(beam.DoFn):
    """Attach Event Time to the element for correct Windowing."""
    def process(self, record):
        try:
            dt = datetime.fromisoformat(record["timestamp"].replace("Z", "+00:00"))
            unix_ts = dt.timestamp()
            # Wrap in TimestampedValue so WindowInto uses THIS time, not processing time
            yield beam.window.TimestampedValue(record, unix_ts)
        except Exception as e:
            logging.error(f"Timestamp parsing error: {e}")

class ExtractKey(beam.DoFn):
    """Composite Keying for Multi-tenancy."""
    def process(self, element):
        key = f"{element['tenant_id']}#{element['card_id']}"
        yield (key, element)

class WriteToFeatureStore(beam.DoFn):
    def setup(self):
        aiplatform.init(project=PROJECT_ID, location=REGION)
        self.fs = aiplatform.Featurestore(featurestore_name=FEATURE_STORE_ID)
        self.entity = self.fs.get_entity_type("cards")

    def process(self, element, window=beam.DoFn.WindowParam):
        key, agg = element
        _, card_id = key.split("#")
        ts = window.end.to_utc_datetime()

        # TODO: Implement exponential backoff retry for transient network failures
        try:
            self.entity.write_feature_values(
                entity_id=card_id,
                feature_values={
```

```
                "txn_count_10m": int(agg["count"]),
                "txn_sum_10m": float(agg["sum"])
            },
            feature_time=ts
        )
        logging.info(f"Updated {key}: {agg}")
    except Exception as e:
        logging.error(f"FS Write Failure {key}: {e}")

def run():
    parser = argparse.ArgumentParser()
    parser.add_argument("--job_name", required=True)
    parser.add_argument("--runner", default="DataflowRunner")
    parser.add_argument("--temp_location", required=True)
    args, beam_args = parser.parse_known_args()

    options = PipelineOptions(beam_args)
    options.view_as(StandardOptions).streaming = True
    # Ensure we use the timestamp from the data, not the arrival time
    options.view_as(StandardOptions).allow_unsafe_triggers = True

    with beam.Pipeline(options=options) as p:
        (
            p
            | "Read" >> beam.io.ReadFromPubSub(subscription=SUBSCRIPTION_ID)
            | "Validate" >> beam.ParDo(ValidateEvent())
            | "AddTimestamp" >> beam.ParDo(ParseAndTimestamp())
            | "KeyByCard" >> beam.ParDo(ExtractKey())
            | "Window" >> beam.WindowInto(
                SlidingWindows(WINDOW_SIZE_SECONDS, WINDOW_PERIOD_SECONDS),
                trigger=AfterWatermark(early=AfterProcessingTime(10),
late=AfterProcessingTime(1)),
                accumulation_mode=AccumulationMode.ACCUMULATING,
                allowed_lateness=ALLOWED_LATENESS_SECONDS
            )
            # Safe Aggregation to prevent division-by-zero or empty window crashes
            | "Aggregate" >> beam.CombinePerKey(
                lambda elements: {
                    "count": len(elements),
                    "sum": sum(e["amount"] for e in elements) if elements else 0.0
                }
            )
            | "Write" >> beam.ParDo(WriteToFeatureStore())
        )
```

```
if __name__ == "__main__":
    run()
```

---

4. API Orchestration (Hardened)

Location: `api/app/services/feature_store_client.py` *Includes TODO for Batch Optimization.*

Python

```
from google.cloud import aiplatform
from google.cloud.aiplatform_v1 import FeaturestoreOnlineServingServiceClient,
ReadFeatureValuesRequest
from google.cloud.aiplatform_v1.types import FeatureSelector, IdMatcher

class FeatureStoreClient:
    def __init__(self, project_id, region, fs_id):
        self.fs_path = f"projects/{project_id}/locations/{region}/featurestores/{fs_id}"
        api_endpoint = f"{region}-aiplatform.googleapis.com"
        self.client = FeaturestoreOnlineServingServiceClient(client_options={"api_endpoint":
api_endpoint})

    def get_streaming_features(self, card_id: str):
        """
        Fetches real-time velocity features for a card.
        TODO: Optimize for high-throughput by implementing `streaming_read_feature_values`
        or `read_feature_values` with multiple entity IDs in a single batch call.
        """
        entity_type_path = f"{self.fs_path}/entityTypes/cards"
        selector = FeatureSelector(id_matcher=IdMatcher(ids=["txn_count_10m",
"txn_sum_10m"]))

        try:
            response = self.client.read_feature_values(
                request=ReadFeatureValuesRequest(
                    entity_type=entity_type_path,
                    entity_id=card_id,
                    feature_selector=selector
                )
            )
            data = response.entity_view.data
            result = {"txn_count_10m": 0, "txn_sum_10m": 0.0}
            for feature in data:
                fid = feature.id.split("/")[-1]
                if fid == "txn_count_10m": result[fid] = feature.value.int64_value
                elif fid == "txn_sum_10m": result[fid] = feature.value.double_value
            return result
        except Exception as e:
            print(f"Error fetching features: {e}")
            return {"txn_count_10m": 0, "txn_sum_10m": 0.0}
```

Location: `api/app/main.py` *Includes Endpoint Pinning safeguards.*

Python
```
import os
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from google.cloud import aiplatform
from app.services.feature_store_client import FeatureStoreClient

app = FastAPI(title="FraudShield V3: Real-Time Hybrid API")

PROJECT_ID = "fraudshield-v3-dev-5320"
REGION = "us-central1"
FEATURE_STORE_ID = "fraudshield_feature_store_dev"
ENDPOINT_NAME = "fraudshield-hybrid-endpoint"

fs_client = None
endpoint = None

class TransactionRequest(BaseModel):
    transaction_id: str
    tenant_id: str
    card_id: str
    amount: float

@app.on_event("startup")
def startup_event():
    global fs_client, endpoint
    fs_client = FeatureStoreClient(PROJECT_ID, REGION, FEATURE_STORE_ID)
    aiplatform.init(project=PROJECT_ID, location=REGION)

    # TODO: Explicitly pin endpoint_id or match model version metadata.
    # Current logic takes the first found endpoint, which risks hitting a "Challenger" model.
    endpoints = aiplatform.Endpoint.list(filter=f'display_name="{ENDPOINT_NAME}"')
    if endpoints:
        endpoint = endpoints[0]
    else:
        print("WARNING: Endpoint not found.")

@app.post("/v3/score")
def score(txn: TransactionRequest):
    if not endpoint: raise HTTPException(status_code=503, detail="Endpoint unavailable")

    velocity = fs_client.get_streaming_features(txn.card_id)
    vector = [txn.amount, velocity["txn_count_10m"], velocity["txn_sum_10m"]]

    try:
        prediction = endpoint.predict(instances=[vector])
        return {
            "transaction_id": txn.transaction_id,
            "risk_assessment": prediction.predictions[0],
            "features": velocity
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Prediction failed: {e}")
```

---

5. Training Pipeline (Hardened)

Location: `pipelines/training/components/train_hybrid_model_component.py`
Includes Metric Logging and Stability Notes.

Python
```
import kfp.dsl as dsl
from kfp.dsl import component, Input, Output, Dataset, Metrics

@component(base_image="python:3.10", packages_to_install=["pandas", "xgboost",
"scikit-learn", "joblib"])
def train_hybrid_model(training_data: Input[Dataset], artifact_uri: str, metrics: Output[Metrics]):
    import pandas as pd
    import xgboost as xgb
    import joblib
    import os
    from sklearn.ensemble import IsolationForest

    df = pd.read_csv(training_data.path)
    X = df[["amount", "txn_count_10m", "txn_sum_10m"]]
```

```python
    y = df['is_fraud']

    # 1. Supervised XGBoost
    model_xgb = xgb.XGBClassifier(objective="binary:logistic", n_estimators=50)
    model_xgb.fit(X, y)
    model_xgb.save_model("model.bst")
    os.system(f"gsutil cp model.bst {os.path.join(artifact_uri, 'model.bst')}")

    # 2. Unsupervised Isolation Forest
    # NOTE: Isolation Forest is brittle on small/undiverse windows.
    # Future V4: Replace with Embedding-based anomaly detection or AutoEncoder.
    model_iso = IsolationForest(contamination=0.01, random_state=42)
    model_iso.fit(X[y==0])
    joblib.dump(model_iso, "isolation_forest.joblib")
    os.system(f"gsutil cp isolation_forest.joblib {os.path.join(artifact_uri, 'isolation_forest.joblib')}")

    # TODO: Calculate and Log Drift Stats (PSI/KL Divergence) and Thresholds
    metrics.log_metric("models_trained", 2)
```

---

6. Hybrid Intelligence Container

Location: `models/ensemble_cpr/predictor.py` *(Logic to load and ensemble the two models)*

Python
```python
import os
import joblib
import numpy as np
import xgboost as xgb
from google.cloud.aiplatform.prediction.predictor import Predictor

class CprPredictor(Predictor):
    def __init__(self):
        self.xgb_model = None
        self.iso_model = None

    def load(self, artifacts_uri: str):
        xgb_path = os.path.join(artifacts_uri, "model.bst")
        self.xgb_model = xgb.XGBClassifier()
        self.xgb_model.load_model(xgb_path)
        iso_path = os.path.join(artifacts_uri, "isolation_forest.joblib")
        self.iso_model = joblib.load(iso_path)
```

```python
def predict(self, instances):
    inputs = np.array(instances)
    # Thread A: Supervised Probability
    prob_xgb = self.xgb_model.predict_proba(inputs)[:, 1]

    # Thread B: Unsupervised Anomaly Score (Normalized 0-1)
    raw_iso = self.iso_model.decision_function(inputs)
    prob_iso = 1 - ((raw_iso + 1) / 2)
    prob_iso = np.clip(prob_iso, 0, 1)

    # Ensemble: 80% Supervised, 20% Unsupervised
    final_scores = (0.8 * prob_xgb) + (0.2 * prob_iso)

    results = []
    for score in final_scores:
        band = "LOW"
        if score > 0.7: band = "HIGH"
        elif score > 0.3: band = "MEDIUM"
        results.append({"score": float(score), "risk_band": band})

    return {"predictions": results}
```