AutoML FraudShield V2.0: Master Configuration Guide

Version: 2.0 (Production Ready) **Date:** December 2, 2025 **Scope:** Full Stack (Infra, Data, Pipeline, API, Monitoring, Dashboard)

---

1. Project Directory Structure

Plaintext

```
fraudshield-v2/
├── README.md
├── .gitignore
├── api/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── app/
│       ├── main.py
│       └── services/
│           ├── feature_store_client.py
│           ├── model_endpoint_client.py
│           └── shap_explainer.py
├── dashboard/
│   ├── app.py
│   └── requirements.txt
├── data/
│   └── generate_mock_data.py
├── features/
│   ├── ingest_features.py
│   └── variables.py
├── infra/
│   └── terraform/
│       └── envs/
│           └── dev/
│               ├── main.tf
│               └── variables.tf
├── logging_pipeline/
│   ├── setup_log_table.py
│   └── subscriber_main.py
├── monitoring/
```

```
│   └── monitoring_job.py
└── pipelines/
    └── training/
        ├── pipeline_definition_v2.py
        └── components/
            ├── deploy_model_component.py
            ├── export_shap_artifacts_component.py
            ├── train_model_component.py
            └── validate_features_component.py
```

---

2. Infrastructure (Terraform)

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

Terraform
```
variable "project_id" {
  description = "The GCP Project ID"
  type        = string
  default     = "fraudshield-v2-dev"
}

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"
  ])
  service            = each.key
  disable_on_destroy = false
}

# --- Core Resources ---
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_bigquery_dataset" "fraudshield" {
  dataset_id                 = "fraudshield"
  location                   = var.region
  delete_contents_on_destroy = 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]
}
```

```
# --- Feature Store ---
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]
}

# --- V2: Logging & Serving ---
resource "google_pubsub_topic" "predictions" {
  name       = "fraudshield-predictions"
  depends_on = [google_project_service.enabled_apis]
}

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

---

3. Data & Features

Location: `features/variables.py`

```
Python
PROJECT_ID = "fraudshield-v2-dev"
REGION = "us-central1"
ENV = "dev"
FEATURE_STORE_ID = f"fraudshield_feature_store_{ENV}"
```

Location: `data/generate_mock_data.py`

```
Python
import pandas as pd
import numpy as np
import random
import os
```

```
from datetime import datetime, timedelta

os.makedirs("data", exist_ok=True)
NUM_TRANSACTIONS = 5000
NUM_CUSTOMERS = 100
NUM_TERMINALS = 50
START_DATE = datetime(2024, 1, 1)

def generate_data():
    print(f"Generating {NUM_TRANSACTIONS} transactions...")
    customer_ids = [f"CUST_{i:04d}" for i in range(NUM_CUSTOMERS)]
    terminal_ids = [f"TERM_{i:04d}" for i in range(NUM_TERMINALS)]

    data = []
    for _ in range(NUM_TRANSACTIONS):
        tx_id = f"TXN_{random.randint(10000000, 99999999)}"
        customer_id = random.choice(customer_ids)
        terminal_id = random.choice(terminal_ids)
        days_offset = random.randint(0, 90)
        hour = int(np.random.normal(14, 4)) % 24
        tx_ts = START_DATE + timedelta(days=days_offset, hours=hour,
minutes=random.randint(0,59))
        amount = round(np.random.lognormal(3.5, 1.0), 2)

        is_fraud = 0
        if amount > 800 and random.random() < 0.8: is_fraud = 1
        if terminal_id == "TERM_0013" and random.random() < 0.5: is_fraud = 1

        data.append({
            "tx_id": tx_id, "customer_id": customer_id, "terminal_id": terminal_id,
            "tx_ts": tx_ts, "amount": amount, "is_fraud": is_fraud
        })

    df = pd.DataFrame(data)
    df.to_csv("data/sample_transactions.csv", index=False)
    print("Data saved to data/sample_transactions.csv")

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

Location: `features/ingest_features.py`

Python

```
from google.cloud import bigquery
from google.cloud import aiplatform
import pandas as pd
import variables

def calculate_and_ingest():
    print(f"Initializing for Project: {variables.PROJECT_ID}...")
    bq_client = bigquery.Client(project=variables.PROJECT_ID)
    aiplatform.init(project=variables.PROJECT_ID, location=variables.REGION)
    fs = aiplatform.Featurestore(featurestore_name=variables.FEATURE_STORE_ID)

    # 1. Load Raw Data
    table_id = f"{variables.PROJECT_ID}.fraudshield.transactions"
    try:
        bq_client.get_table(table_id)
        print("Table exists.")
    except Exception:
        print("Table not found. Uploading CSV...")
        df = pd.read_csv("data/sample_transactions.csv")
        df['tx_ts'] = pd.to_datetime(df['tx_ts'])
        job_config = bigquery.LoadJobConfig(write_disposition="WRITE_TRUNCATE",
schema=[bigquery.SchemaField("tx_ts", "TIMESTAMP")])
        bq_client.load_table_from_dataframe(df, table_id, job_config=job_config).result()

    # 2. Calculate Features (SQL)
    print("Calculating Features...")
    cust_query = f"""
        CREATE OR REPLACE TABLE
`{variables.PROJECT_ID}.fraudshield.features_customers` AS
        SELECT customer_id, tx_ts as feature_timestamp,
        COUNT(*) OVER(PARTITION BY customer_id ORDER BY UNIX_SECONDS(tx_ts)
RANGE BETWEEN 604800 PRECEDING AND CURRENT ROW) as txn_count_7d,
        SUM(amount) OVER(PARTITION BY customer_id ORDER BY UNIX_SECONDS(tx_ts)
RANGE BETWEEN 604800 PRECEDING AND CURRENT ROW) as txn_amount_sum_7d,
        AVG(amount) OVER(PARTITION BY customer_id ORDER BY UNIX_SECONDS(tx_ts)
RANGE BETWEEN 2592000 PRECEDING AND CURRENT ROW) as avg_ticket_30d
        FROM `{variables.PROJECT_ID}.fraudshield.transactions`
    """
    bq_client.query(cust_query).result()

    card_query = f"""
        CREATE OR REPLACE TABLE `{variables.PROJECT_ID}.fraudshield.features_cards` AS
        SELECT terminal_id as card_id, tx_ts as feature_timestamp,
```

```
        COUNT(*) OVER(PARTITION BY terminal_id ORDER BY UNIX_SECONDS(tx_ts)
RANGE BETWEEN 604800 PRECEDING AND CURRENT ROW) as txn_count_7d,
        SUM(amount) OVER(PARTITION BY terminal_id ORDER BY UNIX_SECONDS(tx_ts)
RANGE BETWEEN 604800 PRECEDING AND CURRENT ROW) as txn_amount_sum_7d
        FROM `{variables.PROJECT_ID}.fraudshield.transactions`
    """
    bq_client.query(card_query).result()

    # 3. Create Entities
    try: aiplatform.EntityType.create(featurestore_name=variables.FEATURE_STORE_ID,
entity_type_id="customers")
    except: pass
    try: aiplatform.EntityType.create(featurestore_name=variables.FEATURE_STORE_ID,
entity_type_id="cards")
    except: pass

    # 4. Create Features & Ingest
    print("Ingesting to Online Store...")
    cust_feats = {"txn_count_7d": "INT64", "txn_amount_sum_7d": "DOUBLE", "avg_ticket_30d":
"DOUBLE"}
    card_feats = {"txn_count_7d": "INT64", "txn_amount_sum_7d": "DOUBLE"}

    fs.get_entity_type("customers").batch_create_features(feature_configs={k: {"value_type": v}
for k,v in cust_feats.items()})
    fs.get_entity_type("cards").batch_create_features(feature_configs={k: {"value_type": v} for k,v
in card_feats.items()})

    fs.get_entity_type("customers").ingest_from_bq(
        feature_ids=list(cust_feats.keys()), feature_time="feature_timestamp",
        bq_source_uri=f"bq://{variables.PROJECT_ID}.fraudshield.features_customers",
entity_id_field="customer_id"
    )
    fs.get_entity_type("cards").ingest_from_bq(
        feature_ids=list(card_feats.keys()), feature_time="feature_timestamp",
        bq_source_uri=f"bq://{variables.PROJECT_ID}.fraudshield.features_cards",
entity_id_field="card_id"
    )
    print("Ingestion submitted.")

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

---

4. Training Pipeline

Location:
`pipelines/training/components/export_shap_artifacts_component.py`

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

@component(base_image="python:3.10", packages_to_install=["pandas", "numpy", "pyarrow"])
def export_shap_artifacts(training_data: Input[Dataset], shap_artifacts: Output[Artifact]):
    import pandas as pd
    import os
    import json

    df = pd.read_csv(training_data.path)
    background_df = df[df['is_fraud'] == 0].sample(n=min(1000, len(df)), random_state=42)

    excluded = ['transaction_id', 'customer_id', 'card_id', 'timestamp', 'tx_ts', 'is_fraud']
    features = [c for c in df.columns if c not in excluded]

    os.makedirs(shap_artifacts.path, exist_ok=True)
    background_df[features].to_parquet(os.path.join(shap_artifacts.path,
"background_data.parquet"), index=False)
    with open(os.path.join(shap_artifacts.path, "feature_map.json"), 'w') as f:
        json.dump(features, f)
```

Location: `pipelines/training/components/validate_features_component.py`

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

@component(base_image="python:3.10", packages_to_install=["pandas"])
def validate_features(training_data: Input[Dataset], validation_metrics: Output[Metrics]):
    import pandas as pd
    df = pd.read_csv(training_data.path)
    for col in df.columns:
        if df[col].isnull().sum() / len(df) > 0.05:
            raise ValueError(f"Feature '{col}' failed validation (too many nulls).")
```

Location: `pipelines/training/components/train_model_component.py`

Python

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

@component(base_image="python:3.10", packages_to_install=["pandas", "xgboost",
"scikit-learn"])
def train_xgboost_model(training_data: Input[Dataset], model_artifact: Output[Model],
metrics_artifact: Output[Metrics]):
    import pandas as pd
    import xgboost as xgb
    import os
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import roc_auc_score, average_precision_score

    df = pd.read_csv(training_data.path)
    excluded = ['transaction_id', 'customer_id', 'card_id', 'timestamp', 'tx_ts', 'is_fraud']
    features = [c for c in df.columns if c not in excluded]

    X = df[features]
    y = df['is_fraud']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train on Values (Numpy) to avoid schema conflicts in serving
    model = xgb.XGBClassifier(objective="binary:logistic", use_label_encoder=False,
n_estimators=100)
    model.fit(X_train.values, y_train)

    probs = model.predict_proba(X_test.values)[:, 1]
    metrics_artifact.log_metric("roc_auc", roc_auc_score(y_test, probs))
    metrics_artifact.log_metric("pr_auc", average_precision_score(y_test, probs))

    model_artifact.metadata["framework"] = "xgboost"
    os.makedirs(model_artifact.path, exist_ok=True)
    model.save_model(os.path.join(model_artifact.path, "model.bst"))
```

Location: `pipelines/training/components/deploy_model_component.py`

```
Python
from kfp.dsl import component, Input, Model

@component(base_image="python:3.10", packages_to_install=["google-cloud-aiplatform"])
def deploy_model_to_endpoint(project_id: str, region: str, model: Input[Model], endpoint_name:
str, display_name: str, serving_container: str):
    from google.cloud import aiplatform
    aiplatform.init(project=project_id, location=region)
```

```
    uploaded_model = aiplatform.Model.upload(display_name=display_name,
artifact_uri=model.uri.replace("/model.bst", ""), serving_container_image_uri=serving_container)
    endpoints = aiplatform.Endpoint.list(filter=f'display_name="{endpoint_name}"')
    endpoint = endpoints[0] if endpoints else
aiplatform.Endpoint.create(display_name=endpoint_name)
    endpoint.deploy(model=uploaded_model, deployed_model_display_name=display_name,
machine_type="n1-standard-2", traffic_percentage=100)
```

Location: `pipelines/training/pipeline_definition_v2.py`

Python

```
from kfp import dsl, compiler
from components.validate_features_component import validate_features
from components.export_shap_artifacts_component import export_shap_artifacts
from components.train_model_component import train_xgboost_model
from components.deploy_model_component import deploy_model_to_endpoint

PROJECT_ID = "fraudshield-v2-dev"
REGION = "us-central1"
BUCKET = f"fraudshield-artifacts-dev-{PROJECT_ID}"
PIPELINE_ROOT = f"gs://{BUCKET}/pipeline_root"

@dsl.component(base_image="python:3.10", packages_to_install=["pandas",
"google-cloud-bigquery", "db-dtypes"])
def extract_bq_to_dataset(project_id: str, query: str, dataset: dsl.Output[dsl.Dataset]):
    from google.cloud import bigquery
    bigquery.Client(project=project_id).query(query).to_dataframe().to_csv(dataset.path,
index=False)

@dsl.pipeline(name="fraudshield-training-pipeline-v2")
def fraudshield_pipeline_v2(project_id: str = PROJECT_ID, region: str = REGION):
    query = f"""
        SELECT t.tx_id as transaction_id, t.customer_id, t.terminal_id as card_id, t.tx_ts as
timestamp, t.is_fraud, t.amount,
        c.txn_count_7d, c.txn_amount_sum_7d, c.avg_ticket_30d, d.txn_count_7d as
card_count_7d, d.txn_amount_sum_7d as card_sum_7d
        FROM `{project_id}.fraudshield.transactions` t
        JOIN `{project_id}.fraudshield.features_customers` c ON t.customer_id = c.customer_id
AND t.tx_ts = c.feature_timestamp
        JOIN `{project_id}.fraudshield.features_cards` d ON t.terminal_id = d.card_id AND t.tx_ts =
d.feature_timestamp
    """
    extract = extract_bq_to_dataset(project_id=project_id, query=query)
```

```
    validate_features(training_data=extract.outputs["dataset"])
    train = train_xgboost_model(training_data=extract.outputs["dataset"]).after(extract)
    export_shap_artifacts(training_data=extract.outputs["dataset"]).after(extract)
    deploy_model_to_endpoint(project_id=project_id, region=region,
model=train.outputs["model_artifact"], endpoint_name="fraudshield-endpoint",
display_name="fraudshield-xgb-v2",
serving_container="us-docker.pkg.dev/vertex-ai/prediction/xgboost-cpu.1-6:latest").after(train)

if __name__ == "__main__":
    compiler.Compiler().compile(pipeline_func=fraudshield_pipeline_v2,
package_path="fraudshield_pipeline_v2.json")
    from google.cloud import aiplatform
    aiplatform.init(project=PROJECT_ID, location=REGION)
    aiplatform.PipelineJob(display_name="fraudshield-v2-run",
template_path="fraudshield_pipeline_v2.json", pipeline_root=PIPELINE_ROOT,
enable_caching=False).submit()
```

---

5. Scoring API

Location: `api/requirements.txt`

```
Plaintext
fastapi==0.95.1
uvicorn==0.22.0
google-cloud-aiplatform==1.35.0
google-cloud-storage==2.10.0
google-cloud-pubsub==2.18.0
xgboost==1.6.2
pandas==1.5.3
scikit-learn==1.2.2
shap==0.41.0
pydantic==1.10.7
numpy<1.24.0
```

Location: `api/app/main.py`

```
Python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import os
import json
```

```
from google.cloud import pubsub_v1
from app.services.feature_store_client import FeatureStoreClient
from app.services.model_endpoint_client import ModelEndpointClient
from app.services.shap_explainer import ShapExplainer

app = FastAPI(title="AutoML FraudShield V2 API")
PROJECT_ID = "fraudshield-v2-dev"
REGION = "us-central1"
ENDPOINT_NAME = "fraudshield-endpoint"
PUBSUB_TOPIC = "fraudshield-predictions"
SHAP_ARTIFACT_URI = os.environ.get("SHAP_ARTIFACT_URI", "")

fs_client = None
endpoint_client = None
shap_explainer = None
publisher = None
topic_path = None

class TransactionRequest(BaseModel):
    transaction_id: str
    customer_id: str
    card_id: str
    amount: float
    timestamp: str

@app.on_event("startup")
def startup_event():
    global fs_client, endpoint_client, shap_explainer, publisher, topic_path
    fs_client = FeatureStoreClient()
    try:
        endpoint_client = ModelEndpointClient(PROJECT_ID, REGION, ENDPOINT_NAME)
    except: pass
    if SHAP_ARTIFACT_URI:
        parts = SHAP_ARTIFACT_URI.replace("gs://", "").split("/")
        shap_explainer = ShapExplainer(PROJECT_ID, parts[0], "/".join(parts[1:]))
        shap_explainer.load_artifacts()
    try:
        publisher = pubsub_v1.PublisherClient()
        topic_path = publisher.topic_path(PROJECT_ID, PUBSUB_TOPIC)
    except: pass

@app.post("/v1/score")
def score(txn: TransactionRequest):
    cust = fs_client.get_customer_features(txn.customer_id)
```

```
    card = fs_client.get_card_features(txn.card_id)
    features = [txn.amount, cust.get("txn_count_7d", 0), cust.get("txn_amount_sum_7d", 0.0),
cust.get("avg_ticket_30d", 0.0), card.get("txn_count_7d", 0), card.get("txn_amount_sum_7d",
0.0)]

    prediction = endpoint_client.predict(features)
    prob_fraud = prediction[1] if isinstance(prediction, list) else prediction

    risk = "LOW"
    if prob_fraud >= 0.6: risk = "HIGH"
    elif prob_fraud >= 0.2: risk = "MEDIUM"

    explanations = shap_explainer.explain(features) if shap_explainer else []

    if publisher and topic_path:
        log = {"transaction_id": txn.transaction_id, "score": prob_fraud, "risk_band": risk,
"explanations": explanations}
        publisher.publish(topic_path, json.dumps(log).encode("utf-8"))

    return {"transaction_id": txn.transaction_id, "score": prob_fraud, "risk_band": risk,
"explanations": explanations}
```

---

6. Monitoring & Automation

Location: `monitoring/monitoring_job.py`

```
Python
from google.cloud import bigquery
from google.cloud import aiplatform
import pandas as pd
import sys
import os
from kfp import compiler

training_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), '../pipelines/training'))
if training_dir not in sys.path: sys.path.append(training_dir)
from pipeline_definition_v2 import fraudshield_pipeline_v2

PROJECT_ID = "fraudshield-v2-dev"
REGION = "us-central1"
BQ_TABLE = f"{PROJECT_ID}.fraudshield.predictions_log_v2"
PIPELINE_ROOT = f"gs://fraudshield-artifacts-dev-{PROJECT_ID}/pipeline_root"
```

```
def monitor_drift():
    client = bigquery.Client(project=PROJECT_ID)
    df = client.query(f"SELECT score FROM `{BQ_TABLE}` ORDER BY timestamp
DESC").to_dataframe()

    split_idx = int(len(df) * 0.2)
    if split_idx == 0: split_idx = 1

    avg_recent = df.iloc[:split_idx]['score'].mean()
    avg_baseline = df.iloc[split_idx:]['score'].mean() if len(df) > split_idx else 0.5
    if avg_baseline == 0: avg_baseline = 0.001

    drift = abs((avg_recent - avg_baseline) / avg_baseline)
    print(f"Drift: {drift:.2%}")

    if drift > 0.20:
        print(">>> ALERT: Drift Detected. Triggering Retraining...")
        compiler.Compiler().compile(pipeline_func=fraudshield_pipeline_v2,
package_path="autotrain.json")
        aiplatform.init(project=PROJECT_ID, location=REGION)
        aiplatform.PipelineJob(display_name="fraudshield-autotrain",
template_path="autotrain.json", pipeline_root=PIPELINE_ROOT).submit()

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

---

7. Dashboard (Streamlit)

Location: `dashboard/app.py`

```
Python
import streamlit as st
from google.cloud import bigquery
from google.oauth2 import service_account
import pandas as pd
import plotly.express as px

PROJECT_ID = "fraudshield-v2-dev"
TABLE_ID = f"{PROJECT_ID}.fraudshield.predictions_log_v2"

def get_credentials():
```

```
    if "gcp_service_account" in st.secrets:
        return
service_account.Credentials.from_service_account_info(st.secrets["gcp_service_account"])
    return None

st.title("FraudShield Operations Center")
try:
    client = bigquery.Client(project=PROJECT_ID, credentials=get_credentials())
    query = f"SELECT transaction_id, score, risk_band, timestamp FROM `{TABLE_ID}` ORDER
BY timestamp DESC LIMIT 1000"
    df = client.query(query).to_dataframe(create_bqstorage_client=False)

    st.metric("Total Logs", len(df))
    st.dataframe(df.head(20))
except Exception as e:
    st.error(f"Error: {e}")
```