AutoML FraudShield V1.1: Master Configuration Guide

This guide captures the final, successful state of the repository, including all code necessary for infrastructure provisioning, data ingestion, pipeline automation, and API deployment.

1. Project Directory Structure

Plaintext

```
fraudshield-v1/
├── README.md
├── api/
│   ├── app/
│   │   ├── main.py
│   │   └── services/
│   │       └── feature_store_client.py
│   └── Dockerfile
│   └── requirements.txt
├── data/
│   └── generate_mock_data.py   # Script to generate sample_transactions.csv
├── features/
│   ├── feature_definitions.py
│   ├── ingest_features.py
│   └── variables.py
└── infra/
    └── terraform/
        └── envs/
            └── dev/
                ├── main.tf
                └── variables.tf
└── pipelines/
    └── training/
        └── pipeline_definition.py
```

---

2. Infrastructure Files (`infra/terraform/envs/dev`)

2.1. `variables.tf`

Terraform

```
variable "project_id" {
  description = "The GCP Project ID where resources will be deployed."
  type        = string
}
```

```
variable "region" {
  description = "The default GCP region for resources (e.g., us-central1)."
  type        = string
  default     = "us-central1"
}

variable "env" {
  description = "The environment name (e.g., dev, prod)."
  type        = string
  default     = "dev"
}
```

2.2. `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"])
  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]
}

resource "google_vertex_ai_featurestore" "featurestore" {
  name     = "fraudshield_feature_store_${var.env}"
  region   = var.region
  online_serving_config {
    fixed_node_count = 1
  }
  depends_on = [google_project_service.enabled_apis]
}
```

---

3. Data & Feature Engineering Files

3.1. `data/generate_mock_data.py`

```
Python
import pandas as pd
import numpy as np
import random
from datetime import datetime, timedelta

NUM_TRANSACTIONS = 5000
NUM_CUSTOMERS = 100
NUM_TERMINALS = 50
START_DATE = datetime(2024, 1, 1)

def generate_data():
```

```
    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
        # SCENARIO A: High Amount Spike
        if amount > 800 and random.random() < 0.8:
            is_fraud = 1
        # SCENARIO C: The "Busted Terminal" (TERM_0013 has high fraud rate)
        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)

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

3.2. `features/variables.py`

```
Python
PROJECT_ID = "fraudshield-479419"
REGION = "us-central1"
ENV = "dev"
FEATURE_STORE_ID = f"fraudshield_feature_store_{ENV}"
API_ENDPOINT = f"{REGION}-aiplatform.googleapis.com"
```

3.3. `features/feature_definitions.py`

(The script that provisions Entity Types and Features in Vertex AI.)

Python
```
from google.cloud import aiplatform
import variables

def create_feature_store_resources():
    aiplatform.init(project=variables.PROJECT_ID, location=variables.REGION)
    fs_name = variables.FEATURE_STORE_ID

    # 1. Define Entity Types
    entity_types = {"customers": "Customer entity", "cards": "Credit card entity"}
    for entity_name, description in entity_types.items():
        try:
            aiplatform.EntityType.create(
                featurestore_name=fs_name,
                entity_type_id=entity_name,
                description=description
            )
        except Exception as e:
            if "already exists" not in str(e): raise e

    # 2. Define Features
    customer_features = {"txn_count_7d": "INT64", "txn_amount_sum_7d": "DOUBLE",
"avg_ticket_30d": "DOUBLE"}
    batch_create_features("customers", fs_name, customer_features)
    card_features = {"txn_count_7d": "INT64", "txn_amount_sum_7d": "DOUBLE"}
    batch_create_features("cards", fs_name, card_features)

def batch_create_features(entity_name, fs_name, feature_dict):
    fs = aiplatform.Featurestore(featurestore_name=fs_name)
    et = fs.get_entity_type(entity_type_id=entity_name)
    feature_configs = {name: {"value_type": dtype} for name, dtype in feature_dict.items()}
    try:
        et.batch_create_features(feature_configs=feature_configs).wait()
    except Exception as e:
        if "already exists" not in str(e): raise e

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

3.4. `features/ingest_features.py`

(The script that calculates BQ window functions and ingests data into the Online Store.)

Python

```
from google.cloud import bigquery
from google.cloud import aiplatform
import variables

def calculate_and_ingest():
    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)

    # --- SQL Query 1: CUSTOMER FEATURES ---
    cust_query = f"""
    CREATE OR REPLACE TABLE `{variables.PROJECT_ID}.fraudshield.features_customers`
AS
    SELECT customer_id, timestamp as feature_timestamp,
        COUNT(*) OVER(PARTITION BY customer_id ORDER BY UNIX_SECONDS(timestamp)
RANGE BETWEEN 604800 PRECEDING AND CURRENT ROW) as txn_count_7d,
        SUM(amount) OVER(PARTITION BY customer_id ORDER BY
UNIX_SECONDS(timestamp) RANGE BETWEEN 604800 PRECEDING AND CURRENT
ROW) as txn_amount_sum_7d,
        AVG(amount) OVER(PARTITION BY customer_id ORDER BY
UNIX_SECONDS(timestamp) RANGE BETWEEN 2592000 PRECEDING AND CURRENT
ROW) as avg_ticket_30d
    FROM `{variables.PROJECT_ID}.fraudshield.transactions`
    """
    bq_client.query(cust_query).result()

    # --- SQL Query 2: CARD FEATURES ---
    card_query = f"""
    CREATE OR REPLACE TABLE `{variables.PROJECT_ID}.fraudshield.features_cards` AS
    SELECT card_id, timestamp as feature_timestamp,
        COUNT(*) OVER(PARTITION BY card_id ORDER BY UNIX_SECONDS(timestamp)
RANGE BETWEEN 604800 PRECEDING AND CURRENT ROW) as txn_count_7d,
        SUM(amount) OVER(PARTITION BY card_id ORDER BY UNIX_SECONDS(timestamp)
RANGE BETWEEN 604800 PRECEDING AND CURRENT ROW) as txn_amount_sum_7d
    FROM `{variables.PROJECT_ID}.fraudshield.transactions`
    """
    bq_client.query(card_query).result()

    # --- Ingestion ---
    # 1. Ingest Customers
    fs.get_entity_type("customers").ingest_from_bq(
        feature_ids=["txn_count_7d", "txn_amount_sum_7d", "avg_ticket_30d"],
feature_time="feature_timestamp",
```

```
        bq_source_uri=f"bq://{variables.PROJECT_ID}.fraudshield.features_customers",
entity_id_field="customer_id"
    )
    # 2. Ingest Cards
    fs.get_entity_type("cards").ingest_from_bq(
        feature_ids=["txn_count_7d", "txn_amount_sum_7d"], feature_time="feature_timestamp",
        bq_source_uri=f"bq://{variables.PROJECT_ID}.fraudshield.features_cards",
entity_id_field="card_id"
    )

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

---

4. Training Pipeline File (`pipelines/training`)

4.1. `pipeline_definition.py`

(The final 2-step pipeline, merging data extraction into the training component.)

```
Python
from kfp import dsl
from kfp import compiler
import os
from google_cloud_pipeline_components.v1.bigquery import BigqueryQueryJobOp

PROJECT_ID = "fraudshield-479419"
REGION = "us-central1"
BUCKET_NAME = f"fraudshield-artifacts-dev-{PROJECT_ID}"
PIPELINE_ROOT = f"gs://{BUCKET_NAME}/pipeline_root"

# --- Component 1: Train Model (Merged Extraction + Training) ---
@dsl.component(
    base_image="python:3.9",
    packages_to_install=["pandas", "xgboost", "scikit-learn", "google-cloud-bigquery",
"db-dtypes", "pyarrow"]
)
def train_xgboost_model(project_id: str, region: str, metrics: dsl.Output[dsl.Metrics], model:
dsl.Output[dsl.Model]):
    import pandas as pd
    import xgboost as xgb
    from google.cloud import bigquery
    from sklearn.model_selection import train_test_split
```

```
    from sklearn.metrics import roc_auc_score, average_precision_score

    client = bigquery.Client(project=project_id, location=region)

    # 1. Define and Execute SQL JOIN Query
    query = f"""
    SELECT
        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.timestamp = c.feature_timestamp
    JOIN `{project_id}.fraudshield.features_cards` d
        ON t.card_id = d.card_id AND t.timestamp = d.feature_timestamp
    """
    df = client.query(query).to_dataframe()

    # 2. Train and Evaluate
    X = df.drop(columns=["is_fraud"])
    y = df["is_fraud"]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model_xgb = xgb.XGBClassifier(objective="binary:logistic", eval_metric="logloss",
use_label_encoder=False, n_estimators=100)
    model_xgb.fit(X_train, y_train)

    y_probs = model_xgb.predict_proba(X_test)[:, 1]
    metrics.log_metric("roc_auc", roc_auc_score(y_test, y_probs))
    metrics.log_metric("pr_auc", average_precision_score(y_test, y_probs))

    # 3. Save Artifact
    model.metadata["framework"] = "xgboost"
    model_path = os.path.join(model.path, "model.bst")
    model_xgb.save_model(model_path)

# --- Component 2: Register Model (Custom SDK Wrapper) ---
@dsl.component(
    base_image="python:3.9",
    packages_to_install=["google-cloud-aiplatform"]
)
def register_model(project_id: str, region: str, model: dsl.Input[dsl.Model], display_name: str,
serving_image: str):
    from google.cloud import aiplatform
    aiplatform.init(project=project_id, location=region)
```

```
    aiplatform.Model.upload(
        display_name=display_name,
        artifact_uri=model.uri.replace("/model.bst", ""),
        serving_container_image_uri=serving_image,
        sync=True
    )

# --- Pipeline Definition ---
@dsl.pipeline(name="fraudshield-training-pipeline")
def fraudshield_pipeline(project_id: str = PROJECT_ID, region: str = REGION):
    train_task = train_xgboost_model(project_id=project_id, region=region)
    register_task = register_model(
        project_id=project_id, region=region, model=train_task.outputs["model"],
        display_name="fraudshield-xgb-v1",
serving_image="us-docker.pkg.dev/vertex-ai/prediction/xgboost-cpu.1-6:latest"
    ).after(train_task)

if __name__ == "__main__":
    # Omitted submission logic for manifest brevity
    pass
```

---

5. API Serving Files (`api/`)

5.1. `api/requirements.txt`

```
Plaintext
fastapi==0.95.1
uvicorn==0.22.0
google-cloud-aiplatform==1.35.0
google-cloud-storage==2.10.0
xgboost==1.6.2
pandas==1.5.3
scikit-learn==1.2.2
pydantic==1.10.7
numpy<2.0.0
```

5.2. `api/Dockerfile`

```
Dockerfile
FROM python:3.9-slim

# Install system dependencies (Critical for XGBoost)
RUN apt-get update && apt-get install -y \
```

```
    libgomp1 \
    && rm -rf /var/lib/apt/lists/*

# Install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY app/ /app/app/

# Environment variable for Cloud Run
ENV PORT=8080

# Run the application
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]
```

5.3. `api/app/services/feature_store_client.py`

(Note: Uses `FeaturestoreOnlineServingServiceClient` for low-latency lookup.)

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

PROJECT_ID = "fraudshield-479419"
REGION = "us-central1"
FEATURE_STORE_ID = "fraudshield_feature_store_dev"

class FeatureStoreClient:
    def __init__(self):
        api_endpoint = f"{REGION}-aiplatform.googleapis.com"
        self.client = FeaturestoreOnlineServingServiceClient(
            client_options={"api_endpoint": api_endpoint}
        )
        self.fs_path =
f"projects/{PROJECT_ID}/locations/{REGION}/featurestores/{FEATURE_STORE_ID}"

    def get_customer_features(self, customer_id: str):
        return self._get_features(entity_type="customers", entity_id=customer_id,
feature_ids=["txn_count_7d", "txn_amount_sum_7d", "avg_ticket_30d"])
```

```
    def get_card_features(self, card_id: str):
        return self._get_features(entity_type="cards", entity_id=card_id,
feature_ids=["txn_count_7d", "txn_amount_sum_7d"])

    def _get_features(self, entity_type, entity_id, feature_ids):
        entity_type_path = f"{self.fs_path}/entityTypes/{entity_type}"
        selector = FeatureSelector(id_matcher=IdMatcher(ids=feature_ids))

        try:
            response = self.client.read_feature_values(
                request=ReadFeatureValuesRequest(entity_type=entity_type_path,
entity_id=entity_id, feature_selector=selector)
            )

            result = {}
            for i, feature_meta in enumerate(response.header.feature_descriptors):
                val = 0.0
                if response.entity_view.data[i].value.double_value:
                    val = response.entity_view.data[i].value.double_value
                elif response.entity_view.data[i].value.int64_value:
                    val = float(response.entity_view.data[i].value.int64_value)

                result[feature_meta.id.split("/")[-1]] = val

            return result
        except Exception:
            return {f: 0.0 for f in feature_ids}
```

5.4. `api/app/main.py`

(The FastAPI app that combines features and serves the prediction.)

Python

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import pandas as pd
import xgboost as xgb
import os
from app.services.feature_store_client import FeatureStoreClient
from google.cloud import storage

app = FastAPI(title="AutoML FraudShield API", version="1.0")
model = None
fs_client = None
```

```
MODEL_GCS_URI = os.environ.get("MODEL_ARTIFACT_URI", "")

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

@app.on_event("startup")
def load_resources():
    global model, fs_client

    fs_client = FeatureStoreClient()

    if MODEL_GCS_URI:
        try:
            client = storage.Client()
            bucket_name = MODEL_GCS_URI.split("/")[2]
            blob_path = "/".join(MODEL_GCS_URI.split("/")[3:])

            bucket = client.bucket(bucket_name)
            blob = bucket.blob(blob_path)
            blob.download_to_filename("model.bst")

            model = xgb.Booster()
            model.load_model("model.bst")
        except Exception as e:
            print(f"❌ Failed to load model: {e}")

@app.post("/v1/score")
def score_transaction(txn: TransactionRequest):
    if not model:
        raise HTTPException(status_code=503, detail="Model not loaded or training in progress")

    cust_feats = fs_client.get_customer_features(txn.customer_id)
    card_feats = fs_client.get_card_features(txn.card_id)

    # Feature order MUST match training: [amount, cust_count, cust_sum, cust_avg, card_count,
card_sum]
    vector = [
        txn.amount,
        cust_feats.get("txn_count_7d", 0),
```

```
    cust_feats.get("txn_amount_sum_7d", 0.0),
    cust_feats.get("avg_ticket_30d", 0.0),
    card_feats.get("txn_count_7d", 0),
    card_feats.get("txn_amount_sum_7d", 0.0)
]

dmatrix = xgb.DMatrix([vector])
prob = model.predict(dmatrix)[0]

risk_band = "LOW"
if prob >= 0.6:
    risk_band = "HIGH"
elif prob >= 0.2:
    risk_band = "MEDIUM"

return {"transact
```