Implementation Plan - PetroStream Data Pipeline

Goal: Build a Serverless Real-Time Oil & Gas Data Lake on AWS with ML-based Anomaly Detection and Power BI Dashboarding.

Important Note regarding CI/CD:

- **What it manages (Code)**: The pipeline logic (Python scripts, Dockerfile, Terraform).
- **What it does NOT manage (Data)**: The actual Petrobras Parquet files. CI/CD updates the *machinery* that processes the data, but the data flows through it continuously.

1. Project Architecture

Data Flow

1. **Source**: Local Parquet Files (`/Petrobras Data/`).
2. **Ingestion (Producer)**: Local Python script streams data to AWS Kinesis.
3. **Processing (ML Consumer)**:
    - **Docker Container**: Python application running the ML model.
    - **AWS ECS (Fargate)**: Serverless container orchestration.
    - **Application Load Balancer (ALB)**: Distributes incoming traffic.
    - **Auto-Scaling**: ECS Service scales up/down based on CPU/Memory usage.
    - **Model**: **Isolation Forest** (Unsupervised Anomaly Detection).
        - *Training*: Locally on **Mac M4**.
        - *Deployment*: Docker Image pushed to **Amazon ECR**.
4. **Visualization (Hybrid)**:
    - **Custom Web App**: Built with **Streamlit (Python)** for real-time operational views.
    - **BI Tool**: **Power BI** for executive dashboards and historical reporting.
    - **Hosting**: Streamlit on ECS Fargate.
    - **Backend**:
        - Streamlit: Queries Athena via `boto3`.
        - Power BI: Connects to Athena via ODBC Driver.
    - **Features**:
        - Real-time Anomaly Dashboard (Streamlit).
        - Historical Pressure/Temperature Charts (Streamlit + Power BI).
        - Well Status Map.

---

2. SQL Usage (Yes, we use SQL!)

We will use **Standard SQL (Presto/Trino dialect)** in **Amazon Athena**.

- **Why?**: S3 stores files (Parquet), but we need to *query* them like a database. Athena allows us to write SQL to scan these files.
- **What we will do with SQL**:
  1. **Create Tables**: Define the schema of our Parquet files in the Glue Catalog (this can be automatic or manual SQL `CREATE EXTERNAL TABLE`).
  2. **Analyze Data**:
     - `SELECT * FROM well_data WHERE anomaly_flag = 1;` (Find all anomalies).
     - `SELECT well_id, AVG(pressure) FROM well_data GROUP BY well_id;` (Aggregations).
  3. **Power BI Integration**: Power BI sends SQL queries to Athena behind the scenes to generate your charts.

---

3. Local vs. AWS: The Plan

This project is a **Hybrid Workflow**. Here is exactly what runs where:

Local Environment (Your Mac M4)

1. **Data Source**: The Petrobras Parquet files live here.
2. **Model Training**:
   - We write a Python script (`train_model.py`) using `scikit-learn`.
   - We read the Parquet files locally.
   - We train the **Isolation Forest** model on your M4 chip.
   - We save the trained model to a file (`model.joblib`).
3. **Infrastructure as Code (Terraform)**:
   - You write `.tf` files locally.
   - You run `terraform apply` locally to tell AWS what to build.
4. **Data Producer**:
   - A Python script (`producer.py`) runs on your Mac.
   - It reads local data and "replays" it to the cloud (AWS Kinesis) as if it were happening live.
5. **Visualization (Power BI)**: runs on your Mac, connecting to AWS.

AWS Cloud Environment

1. **Ingestion (Kinesis Data Streams)**: Receives the data stream from your Mac.
2. **Compute (AWS Lambda)**:
   - Runs the *inference* code (anomaly detection).
   - Downloads the model (`model.joblib`) from S3.

    - Scales automatically to handle the data volume.
3. **Storage (S3)**:
    - Stores the raw data (Parquet) and the trained model file.
4. **Catalog (Glue)**: Keeps track of the data schema (columns, types).
5. **Query Engine (Athena)**: Executes the SQL queries on the S3 data.

---

4. detailed Execution Steps (Documentation)

Phase 1: Infrastructure Setup (Terraform)

1. **Setup**: Configure Terraform provider (`aws`) and backend.
2. **Storage**: Create S3 buckets for `raw-data`, `curated-data`, and `athena-results`.
3. **Streaming**: Create Kinesis Data Stream (`petrostream-ingest`) and Firehose (`petrostream-delivery`).
4. **Security**: Create IAM Roles to allow Kinesis to talk to Firehose, and Firehose to talk to S3.

Phase 2: Machine Learning (Local Training)

1. **Development**: Write `ml/train_model.py`.
2. **Training**: Run the script on Mac to produce `model.joblib`.
3. **Deployment**: Upload `model.joblib` to the S3 bucket created in Phase 1.

Phase 3: Stream Processing (Docker + ECS)

1. **Containerize**: Create a `Dockerfile` for the consumer app.
2. **Registry**: Create an AWS ECR repository and push the Docker image.
3. **Orchestration**:
    - Create an **ECS Cluster** (Fargate).
    - Create a **Task Definition** (CPU/Memory specs).
    - Create an **ECS Service** with **Auto-Scaling** (e.g., scale out if CPU > 70%).
4. **Networking**:
    - Create an **Application Load Balancer (ALB)**.
    - (Note: For Kinesis processing, ALB is optional as Kinesis “pushes” to consumers, but we can expose an API endpoint on the container to justify the ALB).

Phase 4: Data Producer (Simulation)

1. **Script**: Write `producer/producer.py`.

2. **Logic**: Read local Parquet -> Convert to JSON/Bytes -> `kinesis.put_record()`.

3. **Run**: Start this script in a terminal on your Mac.

4. **Deploy**: Dockerize and deploy to ECS Fargate (same cluster as consumer).

Phase 5: Frontend Development (Streamlit)

1. **Setup**: Create `dashboard/app.py`.
2. **Components**:
   - *Dashboard*: Summary stats (Total Anomalies, Active Wells).
   - *Charts*: Interactive plots using **Altair** or **Plotly**.
   - *Status*: Live well status indicators.
3. **Integration**: Use `awswrangler` or `boto3` to fetch data from Athena.
4. **Unified Portal**: Add a page or link in Streamlit that opens the Power BI Executive Dashboard.
5. **Deploy**: Dockerize and deploy to ECS Fargate (same cluster as consumer).

Phase 6: Business Intelligence (Power BI)

1. **Setup**: Install AWS Athena ODBC Driver on local machine.
2. **Connection**: Configure Power BI to connect to the Athena Data Source.
3. **Dashboards**:
   - Create "Executive Overview" report.
   - Visualize long-term trends (Monthly/Yearly) that are heavier to compute in real-time.
4. **Publish Report**: (Optional) Share with stakeholders.

Phase 6: CI/CD Pipeline (GitHub Actions)

1. **Workflow**: Create `.github/workflows/deploy.yml`.
2. **Infrastructure**:
   - On push to `main`, run `terraform plan`.
   - (Manual Approval for `terraform apply` to control costs).
3. **Code**:
   - Lint Python code (`flake8`).
   - Build Docker images and push to ECR.
   - Update ECS Service with new image.

Phase 7: Project Teardown & Cost Management (CRITICAL)

1. **Budget Alerts**: Set up AWS Budgets (e.g., alert at $10).

2. **Destroy**: Run `terraform destroy` to delete all resources (Kinesis, ECS, Lambda, Glue).

3. **S3 Cleanup**: Manually empty S3 buckets (Terraform won't delete non-empty buckets by default).

4. **Result**: $0 cost when not in use.
5. **Result**: $0 cost when not in use.

Phase 8: The "Project Switch" (Easy Operations)

To make this easy, we will create a `Makefile` or Shell Scripts to act as your **Single Switch**:

- `./project_up.sh`:
  - Terraform Apply (Infrastructure).
  - Docker Build & Push (Code).
  - Deploy to ECS.
- `./project_down.sh`:
  - Empty S3 Buckets.
  - Terraform Destroy.
  - **Result**: complete shutdown to save money.

Phase 9: Cost Optimization (How we stay under $120)

1. **Spot Instances**: We will use **Fargate Spot** (Recommended).
   - *Correction*: Spot instances only stop the *compute* (the program running).
   - *Data Safety*: Your data is stored in **S3** and **Kinesis**. It is 100% safe even if the Spot instance stops.
2. **Retention Policies**:
   - **S3**: **Keep Data Indefinitely** (Standard Class).
   - **CloudWatch Logs**: Expire logs after 1 day (Metadata only).
3. **Kinesis Shards**: Use **Provisioned Mode (1 Shard)** or **On-Demand**.
4. **Lambda**: Tune memory to minimum needed (e.g., 256MB).
5. **Clean Up**: The `project_down.sh` script is your ultimate safety net.

---

5. Dataset Analysis (The Gist)

1. Data Source: Petrobras 3W Dataset

This is the **Petrobras 3W Dataset**, a benchmark for anomaly detection in oil & gas. It contains real (`DRAWN`) and realistic simulated (`SIMULATED`) offshore well data.

2. File Types

- **`DRAWN_*.parquet`**: Data from **real** offshore wells. These represent actual operational scenarios but are often harder to find in large quantities (hence "rare events").

- `SIMULATED_*.parquet`: Synthetic data generated to mimic real wells, often used to augment the dataset for training since real anomaly data is scarce.

3. Data Structure (Schema)

From inspecting `SIMULATED_00001.parquet`, the data is **Time-Series Sensor Data** with the following key columns:

- **Timestamp**: Examples show data logged every second (e.g., `2018-10-06 03:57:02`).
- **Sensors (The "Features")**:
  - `P-PDG`: **Pressure** at the Permanent Downhole Gauge (Bottom-hole pressure).
  - `P-TPT`: **Pressure** at the Temperature-Pressure Transducer (upstream of the choke).
  - `T-TPT`: **Temperature** at the Temperature-Pressure Transducer.
  - `P-MON-CKP`: Pressure upstream of the production choke.
  - `T-JUS-CKP`: Temperature downstream of the production choke.
  - (And many others like `QGL` for Gas Lift Flow Rate).
- **Target Variables (The "Labels")**:
  - `class`: The classification of the event.
    - `0`: **Normal Operation**.
    - `1, 2, 3...`: **Specific Anomalies** (e.g., Abrupt Increase of BSW, Spurious Closure of DHSV, Severe Slugging).
  - `state`: The operational state of the well (e.g., Steady State, Transient).

4. Volume

- **Rows**: The simulated file I checked has ~**59,781 rows** (about 16 hours of data at 1-second intervals).
- **Columns**: 29 columns total (Sensors + Labels + Timestamps).

5. Why This is Perfect for your Project

- **Realism**: You are using industry-standard sensor names (`P-PDG`, `T-TPT`).
- **Pipeline Ready**: The `class` column allows us to instantly verify if our **Lambda Anomaly Detector** is working (we can detect an anomaly and verify if `class != 0`).
- **Parquet Format**: High-performance format ready for S3/Athena.

---

6. User Review Required

- **Ready**: This documented plan is saved.
- **Status**: Waiting for permission to start Phase 1.