The Lead Engineer Handbook

Infrastructure, Pipelines & MLOps Architecture

Project: Customer Behavior Analysis for Subscription Retention
Dataset: Alibaba User Behavior (RL)

Your Master Plan

1. **Phase 1 (Completed):** Infrastructure Setup (Docker, Airflow, MinIO, MLflow).
2. **Phase 2 (Completed):** The ETL Pipeline (Ingestion & Feature Engineering).
3. **Phase 3 (Current):** MLOps Orchestration (Automating the Team's Models).
4. **Phase 4 (Future):** Model Serving (FastAPI/Deployment).

Status: CONFIDENTIAL TEAM DOCUMENT
Date: February 2026

1. System Architecture (The "Factory")

Use this section to write the "Methodology" part of your final report.

We have built a **Lambda Architecture** simulation using a containerized stack. This ensures reproducibility, scalability, and separation of concerns.

1.1 The Components

- **Orchestrator (Airflow):** The central nervous system. It schedules ETL jobs and triggers model training. We use the `CeleryExecutor` to allow parallel task processing.
- **Data Lake (MinIO):** An S3-compatible object storage.
  - **Bronze Layer (`raw-data`):** Holds the original Alibaba `.txt` file.
  - **Silver Layer (`processed-data`):** Holds cleaned Parquet files ready for ML.
  - **Gold Layer (`models`):** Stores the serialized `.pkl` model artifacts.
- **Experiment Tracking (MLflow):** Logs metrics (Accuracy, MSE) and versions the models.
- **Database (Postgres):** Handles metadata for Airflow and MLflow.

1.2 The Network

All services run inside a custom Docker Network (`customer-churn-mlops_default`), allowing them to communicate via service names (e.g., `minio:9000`) while exposing ports to the host machine (MacBook) for development.

2. The Data Pipelines (Completed Work)

You have successfully engineered a pipeline to handle "Big Data" (6GB Log File).

2.1 Pipeline 1: Ingestion (`alibaba_dag.py`)

Challenge: The source file used non-standard delimiters (`;`) and contained nested lists within CSV cells. Loading it entirely into RAM caused `SIGTERM` crashes.

Solution: implemented a **Chunking Strategy**.

- Read 5,000 rows at a time.
- Parsed nested strings into float aggregates.
- Utilized `gc.collect()` to force memory release.
- Uploaded chunks to MinIO as Parquet (Columnar storage) for faster downstream reading.

2.2 Pipeline 2: Feature Engineering (`feature_engineering_dag.py`)

Goal: Transform raw logs into RL-ready States and Rewards.

Transformations Implemented:

1. **Reward Calculation:** $R = (Spending \times 1.0) + (Clicks \times 0.5)$.
2. **State Normalization:** Scaled `purchase_power` to [0-1].
3. **Terminal Flag:** Mapped Column 15 to `done` signal.

3. Phase 3: Automating the Team (The "Integration")

Your friends have written their training scripts in `src/regression` and `src/classification`. Currently, they run these manually. **Your job is to make Airflow run them automatically.**

3.1 Step 1: The "Environment-Aware" Code

You must ensure their scripts can run BOTH on their laptops (Localhost) and inside Docker (MinIO container).

The Logic you added to their code:

```
if os.getenv('AIRFLOW_HOME'):
    MINIO_ENDPOINT = "minio:9000"
else:
    MINIO_ENDPOINT = "localhost:9000"
```

3.2 Step 2: The Training DAG (`training_pipeline.py`)

Create this DAG in `dags/`. It imports their Python classes and executes them.

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
import sys

# Critical: Add src to path so we can import friend's code
sys.path.append('/opt/airflow/src')

def train_regression_wrapper():
    from regression.value_predictor import train
    train()

def train_classification_wrapper():
    from classification.churn_predictor import train
    train()

with DAG('model_training_pipeline', schedule_interval=None, start_date=datetime
    (2024,1,1), catchup=False) as dag:

    t1 = PythonOperator(
        task_id='train_regression',
        python_callable=train_regression_wrapper
    )

    t2 = PythonOperator(
        task_id='train_classification',
        python_callable=train_classification_wrapper
    )

    t1 >> t2
```

3.3 Step 3: Handling Dependencies

Problem: Airflow Docker image doesn't have `sklearn` or `minio` installed by default. **Solution:** You patched the `docker-compose.yaml` (or created a custom Dockerfile) to install `requirements.txt` on startup.

4. Phase 4: Model Serving (The "Product")

To get the "Extra Points," we don't just train models; we use them.

4.1 Task: Build the API (`src/api/main.py`)

You will build a lightweight FastAPI server that loads the models from MinIO and answers questions.

Blueprint code for later:

```python
from fastapi import FastAPI
import joblib
from minio import Minio

app = FastAPI()

# Load models on startup
client = Minio(...)
client.fget_object("models", "value_model.pkl", "value_model.pkl")
model = joblib.load("value_model.pkl")

@app.post("/predict/value")
def predict_value(features: list):
    prediction = model.predict([features])
    return {"expected_revenue": prediction[0]}
```

5. The Lead Engineer's Checklist

You are the gatekeeper. Before the final submission (Mar 9), verify these:

- ☐ **Data Integrity:** Are the Parquet files in MinIO actually readable? (Run `inspect_data.py`).
- ☐ **Code Hygiene:** Did anyone push a `.csv` to GitHub? (Check `.gitignore`).
- ☐ **Pipeline Stability:** Does the `feature_engineering_pipeline` finish without `SIGTERM`?
- ☐ **Integration:** Do the models trained in Airflow allow `BaseModel.load()` without error?
- ☐ **Documentation:** Does the README explain how to run `docker-compose up`?

Mission Control Status:

- Infrastructure: **ONLINE**
- Data Pipeline: **OPERATIONAL**
- MLOps Pipeline: **IN PROGRESS**