Complete MLOps Infrastructure Rebuild Plan

Phase 1: Core Infrastructure Setup (Day 1-2)

1.1 S3 Buckets Recreation

```bash
# Create main data bucket
aws s3 mb s3://loan-mlops-data-bucket --region ap-south-1

# Create model artifacts bucket
aws s3 mb s3://loan-mlops-models-bucket --region ap-south-1

# Create MLflow bucket
aws s3 mb s3://loan-mlops-mlflow-bucket --region ap-south-1

# Set proper permissions
aws s3api put-bucket-policy --bucket loan-mlops-data-bucket --policy '{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::365021531163:role/eks-node-role"},
    "Action": ["s3:GetObject", "s3:PutObject", "s3:DeleteObject"],
    "Resource": "arn:aws:s3:::loan-mlops-data-bucket/*"
  }]
}'
```

1.2 Use Existing EKS Cluster

```bash
```

```bash
# Connect to your existing cluster
aws eks update-kubeconfig --region ap-south-1 --name loan-eks-simple

# Check current cluster status
kubectl get nodes
kubectl get namespaces

# Add GPU node group to existing cluster for LLM
eksctl create nodegroup \
  --cluster loan-eks-simple \
  --region ap-south-1 \
  --name gpu-nodes \
  --node-type g4dn.xlarge \
  --nodes 1 \
  --nodes-min 0 \
  --nodes-max 2 \
  --node-labels workload=llm-inference \
  --node-taints nvidia.com/gpu=true:NoSchedule
```

Phase 2: Monitoring Stack Setup (Day 2-3)

2.1 Prometheus Installation

```yaml
```

```yaml
# prometheus-values.yaml
prometheus:
  prometheusSpec:
    storageSpec:
      volumeClaimTemplate:
        spec:
          storageClassName: gp3
          accessModes: ["ReadWriteOnce"]
          resources:
            requests:
              storage: 50Gi
    retention: 30d
    resources:
      requests:
        memory: 2Gi
        cpu: 1
      limits:
        memory: 4Gi
        cpu: 2

alertmanager:
  alertmanagerSpec:
    storage:
      volumeClaimTemplate:
        spec:
          storageClassName: gp3
          accessModes: ["ReadWriteOnce"]
          resources:
            requests:
              storage: 10Gi

grafana:
  persistence:
    enabled: true
    storageClassName: gp3
    size: 10Gi

# Install with Helm
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm install kube-prometheus-stack prometheus-community/kube-prometheus-stack \
  --namespace monitoring --create-namespace \
  -f prometheus-values.yaml
```

2.2 MLflow Setup

yaml

yaml

```yaml
# mlflow-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mlflow-server
  namespace: loan-default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mlflow-server
  template:
    metadata:
      labels:
        app: mlflow-server
    spec:
      containers:
      - name: mlflow
        image: python:3.9
        command: ["/bin/bash"]
        args: ["-c", "pip install mlflow boto3 psycopg2-binary && mlflow server --backend-store-uri postgresql://mlflow:pa
        ports:
        - containerPort: 5000
        env:
        - name: AWS_ACCESS_KEY_ID
          valueFrom:
            secretKeyRef:
              name: aws-credentials
              key: access-key-id
        - name: AWS_SECRET_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              name: aws-credentials
              key: secret-access-key
        - name: AWS_DEFAULT_REGION
          value: ap-south-1

---
apiVersion: v1
kind: Service
metadata:
  name: mlflow-service
  namespace: loan-default
spec:
  selector:
    app: mlflow-server
```

```yaml
  ports:
  - port: 5000
    targetPort: 5000
  type: LoadBalancer
```

Phase 3: Fixed ML Pipeline Deployment (Day 3-4)

3.1 Corrected Model Training Job

```python
```

```python
# train-balanced-models.py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, precision_score, recall_score, classification_report
from sklearn.utils.class_weight import compute_class_weight
import mlflow
import boto3
import pickle

def train_balanced_model():
    # Load data from S3
    s3 = boto3.client('s3')
    s3.download_file('loan-mlops-data-bucket', 'loan_default_dataset.csv', 'data.csv')
    df = pd.read_csv('data.csv')

    # Prepare features and target
    X = df.drop(['default'], axis=1)  # Adjust column name as needed
    y = df['default']

    print(f"Dataset shape: {X.shape}")
    print(f"Default rate: {y.mean():.3%}")

    # Stratified split to maintain class balance
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=42
    )

    with mlflow.start_run(run_name="balanced-model-training"):

        # Train control model (baseline)
        control_model = RandomForestClassifier(
            n_estimators=100,
            max_depth=8,
            random_state=42,
            class_weight='balanced'  # KEY: Handle class imbalance
        )
        control_model.fit(X_train, y_train)

        # Train treatment model (improved)
        treatment_model = RandomForestClassifier(
            n_estimators=200,
            max_depth=10,
            random_state=42,
            class_weight='balanced',
```

```
    min_samples_split=5,
    min_samples_leaf=2
)
treatment_model.fit(X_train, y_train)

# Evaluate both models properly
def evaluate_model(model, name):
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    # Find optimal threshold for F1
    thresholds = np.arange(0.1, 0.9, 0.01)
    f1_scores = []
    for thresh in thresholds:
        pred_thresh = (y_proba >= thresh).astype(int)
        f1_scores.append(f1_score(y_test, pred_thresh))

    optimal_threshold = thresholds[np.argmax(f1_scores)]
    y_pred_optimal = (y_proba >= optimal_threshold).astype(int)

    # Calculate final metrics
    f1 = f1_score(y_test, y_pred_optimal)
    precision = precision_score(y_test, y_pred_optimal)
    recall = recall_score(y_test, y_pred_optimal)

    print(f"\n{name} Model Results:")
    print(f"Optimal Threshold: {optimal_threshold:.3f}")
    print(f"F1 Score: {f1:.3f}")
    print(f"Precision: {precision:.3f}")
    print(f"Recall: {recall:.3f}")

    # Log to MLflow
    mlflow.log_param(f"{name}_optimal_threshold", optimal_threshold)
    mlflow.log_metric(f"{name}_f1_score", f1)
    mlflow.log_metric(f"{name}_precision", precision)
    mlflow.log_metric(f"{name}_recall", recall)

    return f1, optimal_threshold, model

# Evaluate both models
control_f1, control_threshold, _ = evaluate_model(control_model, "control")
treatment_f1, treatment_threshold, _ = evaluate_model(treatment_model, "treatment")

# Determine winner based on F1 score
if treatment_f1 > control_f1:
    winner = "treatment"
    winner_f1 = treatment_f1
```

```python
            winner_threshold = treatment_threshold
            winner_model = treatment_model
        else:
            winner = "control"
            winner_f1 = control_f1
            winner_threshold = control_threshold
            winner_model = control_model

        print(f"\nWinner: {winner.upper()}")
        print(f"Winner F1 Score: {winner_f1:.3f}")

        # Save winner model
        with open('winner_model.pkl', 'wb') as f:
            pickle.dump({
                'model': winner_model,
                'threshold': winner_threshold,
                'model_type': winner
            }, f)

        # Upload to S3
        s3.upload_file('winner_model.pkl', 'loan-mlops-models-bucket', f'{winner}_model.pkl')

        # Log final results
        mlflow.log_param("winning_model", winner)
        mlflow.log_metric("winner_f1_score", winner_f1)
        mlflow.log_artifact('winner_model.pkl')

        return winner, winner_f1

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

3.2 Direct Deployment Job

```python
```

```python
# deploy-winner.py
import boto3
import pickle
import mlflow
from datetime import datetime
import os

def deploy_winner_model():
    # Download winner model from S3
    s3 = boto3.client('s3')

    # Determine winner from MLflow or environment
    winner = os.getenv('WINNING_MODEL', 'treatment')  # From previous job

    s3.download_file('loan-mlops-models-bucket', f'{winner}_model.pkl', 'winner_model.pkl')

    with open('winner_model.pkl', 'rb') as f:
        model_data = pickle.load(f)

    model = model_data['model']
    threshold = model_data['threshold']

    with mlflow.start_run(run_name=f"direct-deployment-{datetime.now().strftime('%Y%m%d')}"):

        print(f"Deploying {winner} model directly")
        print(f"Optimal threshold: {threshold:.3f}")

        # Create model serving endpoint
        deployment_config = {
            'model_name': f'{winner}_loan_model',
            'model_version': '1.0.0',
            'threshold': threshold,
            'deployment_time': datetime.now().isoformat(),
            'business_validated': True
        }

        # Log deployment
        mlflow.log_param("deployment_type", "direct_winner")
        mlflow.log_param("model_deployed", winner)
        mlflow.log_param("optimal_threshold", threshold)
        mlflow.set_tag("deployment_status", "active")

        # Here you would add actual deployment commands:
        # kubectl apply -f model-serving-deployment.yaml
        # Update ingress controllers
        # Configure load balancers
```

```python
        print(f"Model {winner} deployed successfully")
        print("No retraining needed - using validated A/B test winner")

        return deployment_config

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

Phase 4: LLM Integration (Day 4-5)

4.1 LLM Model Deployment

```yaml
```

```yaml
# llm-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: llama-inference
  namespace: loan-default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: llama-inference
  template:
    metadata:
      labels:
        app: llama-inference
    spec:
      nodeSelector:
        workload: llm-inference
      tolerations:
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
      containers:
      - name: llama-server
        image: vllm/vllm-openai:latest
        command: ["python", "-m", "vllm.entrypoints.openai.api_server"]
        args:
        - "--model"
        - "meta-llama/Llama-3.1-8B-Instruct"
        - "--host"
        - "0.0.0.0"
        - "--port"
        - "8000"
        - "--tensor-parallel-size"
        - "1"
        ports:
        - containerPort: 8000
        resources:
          requests:
            nvidia.com/gpu: 1
            memory: "16Gi"
            cpu: "4"
          limits:
            nvidia.com/gpu: 1
            memory: "24Gi"
            cpu: "8"
```

```yaml
env:
- name: HUGGING_FACE_HUB_TOKEN
  valueFrom:
    secretKeyRef:
      name: hf-token
      key: token
```

Phase 5: Simplified Workflow Creation

5.1 New GitHub Actions Workflow

```yaml
```

```yaml
# .github/workflows/improved-ml-pipeline.yml
name: Improved ML Pipeline

on:
  workflow_dispatch:
  push:
    branches: [main]

env:
  AWS_REGION: ap-south-1
  ECR_REGISTRY: 365021531163.dkr.ecr.ap-south-1.amazonaws.com
  EKS_CLUSTER_NAME: loan-eks-simple
  K8S_NAMESPACE: loan-default

jobs:
  train-and-validate:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-python@v4
      with:
        python-version: '3.9'

    - name: Install dependencies
      run: |
        pip install scikit-learn pandas numpy mlflow boto3

    - name: Train balanced models
      run: |
        python train-balanced-models.py

    outputs:
      winning_model: ${{ steps.training.outputs.winner }}
      f1_score: ${{ steps.training.outputs.f1_score }}

  deploy-winner:
    needs: train-and-validate
    runs-on: ubuntu-latest
    if: ${{ needs.train-and-validate.outputs.f1_score > 0.3 }}
    steps:
    - name: Deploy winning model
      run: |
        python deploy-winner.py
        kubectl apply -f model-serving-config.yaml

    outputs:
```

```yaml
      deployment_status: ${{ steps.deploy.outputs.status }}

  setup-monitoring:
    needs: deploy-winner
    runs-on: ubuntu-latest
    steps:
    - name: Configure monitoring
      run: |
        kubectl apply -f monitoring-config.yaml
        # Update Grafana dashboards
        # Configure alerts
```

Phase 6: Step-by-Step Execution Plan

Day 1: Infrastructure

1. **Recreate S3 buckets** with proper permissions
2. **Setup EKS cluster** with GPU nodes
3. **Install NVIDIA device plugin**
4. **Create namespaces** (`loan-default`, `monitoring`)

Day 2: Monitoring Stack

1. **Deploy Prometheus** with persistent storage
2. **Setup Grafana** with loan-specific dashboards
3. **Configure AlertManager** for model performance alerts
4. **Test monitoring endpoints**

Day 3: Model Pipeline

1. **Deploy corrected training job** with class balancing
2. **Validate models achieve F1 > 0.3** before deployment
3. **Setup direct deployment** (skip champion retraining)
4. **Test model serving endpoints**

Day 4: LLM Integration

1. **Deploy Llama 3.1 8B** on GPU nodes
2. **Create LLM API service** for loan analysis
3. **Integrate with existing model predictions**
4. **Setup LLM monitoring**

Day 5: Integration & Testing

1. **End-to-end pipeline testing**
2. **Performance validation**
3. **Monitoring dashboard configuration**
4. **Documentation and handoff**

Key Changes from Previous Architecture

✅ What's Fixed

- **Direct winner deployment** (no broken retraining)
- **Proper class imbalance handling**
- **F1 score validation** before deployment
- **Simplified decision flow**
- **LLM integration for enhanced analysis**

❌ What's Removed

- Champion model retraining step
- Contradictory safety check gates
- Grafana deployment overrides
- Complex promotion logic

Expected Outcomes

Model Performance:

- F1 scores: 0.3-0.7 (vs previous 0.0028)
- Recall: 50-80% (actually catching defaults)
- Business value: Immediate capture upon deployment

Operational Benefits:

- Faster deployment cycles
- Predictable performance
- Clear decision flow
- Enhanced insights with LLM integration

This rebuild focuses on fixing the fundamental class imbalance issue while creating a clean, maintainable MLOps pipeline that actually delivers business value.