Deliverables

1) **Git repo (source code)**

```
https://github.com/Mohammed-af/image-pipeline.git
```

2) Bonus Challenges

Infrastructure as Code & Deployment:

- docker-compose.yml (Kafka, MinIO, Milvus, Postgres, Rust backend, automation producer)

- Kustomize.yaml (Kustomize manifests)

    Base configuration with all resources
    Environment-specific overlays (dev/staging/prod)
    Patches for resource limits and storage

3) Setup Guide:

All steps to build, configure, and run are below.

1- Clone Repository

```
git clone https://github.com/Mohammed-af/image-pipeline.git
```

2- Python Environment Setup

```
# Create virtual environment
python3 -m venv pipeline_env

# Activate environment
# Linux/macOS:
source pipeline_env/bin/activate
# Windows:
pipeline_env\Scripts\activate

# Install dependencies
pip install --upgrade pip
pip install huggingface_hub requests pillow pymilvus
```

3- Download Models

```
# Create directories
mkdir -p tritonserver/model_repository/{SigLIP_image/1,SigLIP_text/1}

pip install huggingface_hub

# Download models
hf download AbdullahMubark/siglib-models \
  --include "SigLIP_image/1/model.onnx" "SigLIP_text/1/model.onnx" \
  --local-dir ./tritonserver/model_repository


rm -r ./tritonserver/model_repository/.cache

rm -rf ./temp_models
```

4- Configure Models

Create configuration files for GPU mode:

```
# Image model config
cat > tritonserver/model_repository/SigLIP_image/config.pbtxt << 'EOF'
name: "SigLIP_image"
platform: "onnxruntime_onnx"
max_batch_size: 0
input [{
  name: "input0"
  data_type: TYPE_FP32
  dims: [-1, 3, 384, 384]
}]
output [{
  name: "output0"
  data_type: TYPE_FP32
  dims: [-1, 1152]
}]
instance_group [{
  count: 1
  kind: KIND_GPU
  gpus: [0]
}]
EOF

# Text model config
cat > tritonserver/model_repository/SigLIP_text/config.pbtxt << 'EOF'
name: "SigLIP_text"
platform: "onnxruntime_onnx"
max_batch_size: 0
input [{
  name: "input"
  data_type: TYPE_INT64
  dims: [-1, 64]
}]
output [{
  name: "output"
  data_type: TYPE_FP32
  dims: [-1, 1152]
}]
instance_group [{
```

```
  count: 1
  kind: KIND_GPU
  gpus: [0]
}]
EOF
```

For CPU mode, replace `KIND_GPU` with `KIND_CPU` and remove the `gpus: [0]` line.

5- Start Model Services

```
# Create network for model services
docker network create ml-network

# Start Triton Inference Server (GPU)
docker run -d --name triton-server \
  --gpus=1 \
  --network ml-network \
  -p 8000-8002:8000-8002 \
  -v $(pwd)/../tritonserver/model_repository:/models \
  nvcr.io/nvidia/tritonserver:24.07-py3 \
  tritonserver \
  --model-repository=/models \
  --model-control-mode=poll \
  --repository-poll-secs=5

# For CPU mode, remove --gpus=1

# Build FastAPI service
cd model-serving
docker build -t crossmodal-model-serving .

# Start FastAPI
docker run -d --name model-serving-app \
  --network ml-network \
  -p 8003:8003 \
  -e TRITON_SERVER_URL=triton-server:8001 \
  crossmodal-model-serving


cd ..
```

7- Start Pipeline Infrastructure

```
# Start all services
cd image-pipeline
docker-compose up -d

# Check service status
docker ps
```

8- Process Images

```
# Run the Automation
docker-compose run --rm python-producer
```

Complete Results Verification

1. PostgreSQL - Check All Records

```
# Connect to PostgreSQL and see everything
docker exec -it postgres psql -U postgres -d imagedb

# Once connected, run these queries:
\x on  -- Enable expanded display
SELECT * FROM image_records ORDER BY processed_at DESC;
\q     -- Exit

# Or run directly:
docker exec postgres psql -U postgres -d imagedb -c "
SELECT
    filename,
    substring(minio_path, 1, 50) as minio_path,
    substring(milvus_id, 1, 20) as milvus_id,
    substring(text_milvus_id, 1, 20) as text_milvus_id,
    embedding_dim,
    file_size,
    processed_at
FROM image_records
ORDER BY processed_at DESC;"
```

2. MinIO - Check Stored Images

```
# Access MinIO Console
# Open browser: http://localhost:9001
# Login: minioadmin / minioadmin123
# Navigate to Buckets > images
```

3. Milvus - Check Stored Vectors

```
docker run -d --name attu -p 8080:3000 -e
MILVUS_URL=host.docker.internal:19530 zilliz/attu:latest

#Then access the GUI at: http://localhost:8080
```