OPSWAT MLOPS Engineer Test

Approx 4 hours

Objective

The goal of this technical test is to build and deploy a minimal, production-ready **MLOps batch pipeline** that encodes input text into vector embeddings using a HuggingFace-compatible model (model provided). The service must process long texts by splitting them into manageable chunks, returning both the chunked input and corresponding embeddings. All embeddings and original inputs must be stored in a versioned S3 bucket for auditability.

The solution should be containerized, deployed on **AWS** (or any other cloud solution) using **Terraform** (or any other IAC tool) as Infrastructure as Code (IaC), and must include monitoring for invalid inputs. Additionally, the system should track metadata about each run (e.g., time, chunk counts, model version), support configurable parameters. Bonus points for CI/CD and robustness features like retry handling. Given files:

- model.joblib – HuggingFace-compatible encoder
- requirements.txt – Base dependencies (can be extended, python3.11 used)
- data.txt – Input data for embedding

Tasks

1. Model & Embedding Logic

- Use the given model that supports both Russian and English with the given embedder interface (model attached, info MODEL)
- Load the model using the provided interface:

```
import joblib
# Load the model globally (replace 'path_to_model.joblib' with your actual path or object)
MODEL_PATH_OR_OBJECT = "path_to_model.joblib" # or pass the model object directly

if isinstance(MODEL_PATH_OR_OBJECT, str):
  embedder = joblib.load(MODEL_PATH_OR_OBJECT)
else:
  embedder = MODEL_PATH_OR_OBJECT

def process_text_to_embedding(text: str):
   """
   Encodes a single string using the globally loaded model and returns the     embedding(s).
"""
   embedding = embedder.encode([text])
   return embedding.tolist()

embeddings = process_text_to_embedding("your text here")
```

- If the input text exceeds the model's max sequence length, split it into appropriate chunks and encode each chunk separately. **Encodable length is 2048**

- Encode each chunk separately, and return:

```
{"embeddings": {
        "chunks": [
            {"text": "string part 1","vector": [/* embedding vector for part 1 */]},
            {"text": "string part 1", "vector": [/* embedding vector for part 2 */]}   ]
        }
}
```

2. Batch Pipeline

- Process all inputs in **data.txt,** line by line.
- Skip and log invalid inputs (i.e., not "valid"strings *Not valid if full of numbers, emoji etc.*).
- Store results for each input in a structured **.json** format.

3. Storage in S3 (with versioning)

- Save all embeddings and corresponding chunked inputs into a versioned S3 bucket.
- Suggested structure: **s3://your-bucket/embeddings/2025-07-28_13-45-02.json**
- Ensure S3 versioning is enabled via Terraform.

4. Run Metadata Logging

- For each pipeline execution, save a metadata JSON file in S3, e.g.:

```
{
 "timestamp": "2025-07-28T13:45:02Z",
 "input_file": "data.txt",
 "num_inputs": 20,
 "num_chunks": 42,
 "model_version": "model_v1",
 "pipeline_version": "commit abc123"
}
```

5. Configuration Management

- Store all configurable parameters (input file path, chunk size, model path, S3 bucket name) in a config.yaml or .env file.
- The pipeline must dynamically load config values on runtime.

6. Containerization

- Containerize the pipeline using Docker.
- Ensure the image is minimal, secure, and suitable for production (e.g., python:3.11-slim, non-root user).

7. AWS Deployment (Minimal, other provider and tools are good as well)

- Use Terraform to provision:
  - S3 bucket (with versioning)
  - Execution environment (e.g., ECS Fargate Task, EC2, or Lambda)
  - Scheduling via EventBridge (run the task daily or manually)
- Optional: use Terraform modules and remote state (S3 + DynamoDB)

8. Monitoring

- Implement minimal monitoring for the deployed service (e.g., CloudWatch logs, basic health check, or alert on failure).

9. Failure Handling

- Ensure that invalid inputs do not stop the pipeline.
- Bonus: implement simple retry logic for S3 upload errors.

What to provide

- Please submit a ZIP archive or a repository with a similar structure:

```
mlops-embedding-pipeline/
│
├── src/                  # Source code
│   ├── main.py               # Main entry for the batch pipeline
│   ├── embedder.py           # Model loading, chunking, embedding logic
│   ├── pipeline.py           # Orchestrates the full pipeline
│   ├── storage.py            # Handles S3 upload (with retry if implemented)
│   ├── metadata.py           # Generates run metadata
│   └── utils.py              # Helpers for chunking, validation, etc.
│
├── config.yaml               # All configurable parameters
├── requirements.txt          # Python dependencies
├── Dockerfile                # Minimal production-ready container
├── docker-compose.yml        # Optional for local testing
│
├── terraform/                # Infrastructure as Code
│   ├── main.tf               # Provisions S3, ECS/Lambda, EventBridge
│   ├── variables.tf
│   ├── outputs.tf
│   └── backend.tf            # Optional: remote state setup
│
├── data.txt                  # Provided input text
├── model.joblib              # Provided HuggingFace-compatible model
│
├── output/                   # Example results and logs (must be provided)
│   ├── embeddings_2025-08-07.json    # Sample output file with embeddings
│   ├── metadata_2025-08-07.json      # Sample metadata log for run
│   └── logs_2025-08-07.txt           # Log output of a successful run
│
├── .github/                  # Optional: CI/CD
│   └── workflows/ci.yml
│
├── README.md                 # Instructions for setup, usage, deployment, and test
└── .env or .env.example      # Example environment variable file
```