Take Home Test - MLOps Engineer

Goal

Build a small **ML or LLM inference service** with **production-grade MLOps**, delivered in one **GitHub repository**.

Core requirements:

- **Load balancing** on Kubernetes
- **CI/CD via GitHub Actions**
- **Observability with Prometheus + Grafana**
- **Pipeline orchestration & model tracking with Airflow, MLflow, or Kubeflow**

Functional Scope

Expose a minimal inference service:

POST /predict

- Input/output JSON schema documented
- Response includes: `prediction`, `latency_ms`, `model_version`

Any ML or LLM model can be used (sentiment classifier, summarizer, image classifier, etc.).

Required Capabilities

1) Load Balancer

- Kubernetes deployment with **≥2 replicas**
- Use **Ingress (NGINX/Traefik/Envoy)** or **Kong Gateway**
- Show **round-robin routing** or weighted load balancing
- Document **canary/blue-green rollout** and rollback procedure

2) Orchestration & Model Tracking (choose Airflow, MLflow, or Kubeflow)

- Implement a minimal pipeline:
  `data_fetch → train → evaluate → register → deploy`
- Log params, metrics, artifacts
- Register at least one model version and mark it for deployment
- Provide a **MODEL_CARD.md** (dataset, metrics, limitations, risks)

3) CI/CD (GitHub Actions only)

- `.github/workflows/ci.yml` → lint, unit tests (≥1 train, ≥1 API), build & push Docker image
- `.github/workflows/deploy-dev.yml` → auto deploy to **dev** on merge to `main` + smoke test (`/healthz`, `/predict`)
- `.github/workflows/promote-prod.yml` → manual dispatch, canary rollout → full prod promotion + smoke test
- Add **badges** in README for pipeline status

4) Observability (Grafana + Prometheus)

- Expose `/metrics` endpoint for Prometheus
- Grafana dashboard (JSON + screenshot) must show:
  - p95 latency
  - RPS
  - Error rate
  - CPU/memory usage
- Define at least one **alert rule** (e.g., error rate > 5%, p95 > 1s)

5) Traffic & Security

- Add **rate-limiting or key-auth** via Ingress/Kong
- Log all requests in **structured JSON**

6) State & Metadata

- Use **PostgreSQL** for logging request_id, model_version, latency, timestamp
- Provide migrations (DDL or Alembic)

7) Cost & Scalability Notes

- README must explain:
    - Autoscaling/HPA policy
    - Approximate cost per 1k requests
    - CPU/GPU right-sizing assumptions

8) Rollback

- Document **rollback commands** for Helm/K8s (e.g., Helm rollback, kubectl scale down old deployment)

Suggested Repo Structure

/app # FastAPI/Flask inference service

/train # training + pipeline integration

/pipelines # Airflow DAGs, MLflow projects, or Kubeflow pipelines

/deploy # Helm/K8s YAMLs, Ingress rules, Prometheus configs

/dashboards # Grafana JSON + screenshots

/sql # Postgres migrations

/tests # unit/integration tests

.github/workflows/

ci.yml

deploy-dev.yml

promote-prod.yml

MODEL_CARD.md

README.md

Dockerfile(s)

requirements.txt or pyproject.toml

Submission Guidelines

- Submit a **public GitHub repo** named:
  `mlops-takehome-<yourname>`
- Must include:
  - Source code, configs, Dockerfiles, Helm/YAMLs
  - `.github/workflows` for CI/CD
  - Grafana JSON + screenshots
  - Postgres migrations
  - `MODEL_CARD.md`
  - **README.md** with:
    - Setup/run instructions (local + K8s)
    - CI/CD explanation
    - Orchestration/pipeline steps
    - Load balancing and rollout instructions
    - Monitoring and alerts
    - Rollback steps

        - Cost/scalability notes
- Add **GitHub Actions badges** in README
- Include a **demo video (3–6 min)** showing:
    - Training pipeline run in Airflow/MLflow/Kubeflow
    - GitHub Actions workflows
    - K8s load-balanced deployment + rollout
    - Grafana dashboard in action
    - Canary/blue-green + rollback

Minimal Success Path

- FastAPI service with 2+ replicas behind Ingress (round-robin)
- Pipeline in **Airflow/MLflow/Kubeflow**: fetch→train→register
- GitHub Actions: CI → auto dev deploy → manual prod canary rollout
- Prometheus + Grafana with latency, RPS, errors, alert rule
- Postgres log table integrated into service
- Document rollout/rollback steps

Bonus: Agentic AI (Stretch Goal)

Implement one of the following:

- **Model Context Protocol (MCP) server**
    - Build an MCP server exposing model metadata (versions, health, metrics)
    - Integrate with orchestration/monitoring stack
- **Monitor an MCP server**
    - Add Grafana/Prometheus metrics for MCP requests/responses

    - Define alerts if MCP server fails health check

- **Agentic Workflow Integration**
    - Demonstrate an agent (e.g., LangChain/AutoGen) that queries MCP for deployment context (e.g., “which model is serving in prod?”)

Document this in README.

Stretch Goals (Extra Credit)

- GitHub Action for **k6/Locust load test**, upload results as artifact
- Auto-rollback triggered by Prometheus/Grafana alert
- Nightly pipeline with **Evidently drift detection** report
- OpenTelemetry tracing to Jaeger/Tempo