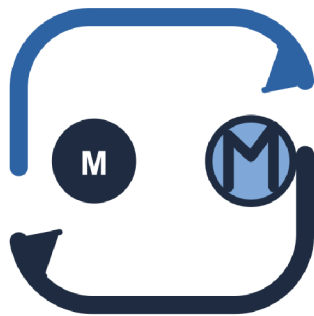


MLOps et Industrialisation des modèles IA

TP 3

Servir et promouvoir un modèle

Du modèle au système ML traçable, validé et exploitable

**MLOPS**
GL-M326

TP · SOMMAIRE

Table des matières

Mission	3
Compétences visées	3
1. Point d'entrée TP2	3
2. Contrat d'inférence et API	4
3. Docker et CI	5
Extension produit : frontend métier	5
4. Registry, sélection et policy	6
5. Questions d'ingénierie	7
6. Rapport, preuves et archive	7



Modalité : travail individuel. Chaque étudiant réalise le TP dans sa propre copie, conserve ses preuves et remet un rapport ainsi qu'une archive personnels.

Mission

Le TP2 a produit un modèle accepté, ses métriques, un manifeste et une décision de quality gate. Le TP3 transforme ce livrable en service exploitable et met en place la décision de release :

CODE

```
TP2 ACCEPT → import/bootstrap → MLflow Registry → challenger  
→ CI + image Docker → promotion contrôlée → champion → TP4
```

Question directrice : comment servir un modèle accepté, prouver son identité, vérifier une image et décider de façon réversible si un challenger peut remplacer le champion ?

Le socle est obligatoire : FastAPI/Pydantic, pytest/httpx, MLflow Registry local SQLite, aliases `challenger` et `champion`, Docker/OCI, GitHub Actions, endpoint technique `/metrics` pour Prometheus et promotion/rollback auditable. Prometheus/Grafana et le dashboard MLOps appartiennent au TP4 ; ils ne doivent pas être embarqués dans l'application métier ni dans le starter TP3. Kubernetes, KServe, cloud Registry, TLS et authentification sont des extensions et ne sont pas requis.

Compétences visées

- reprendre un artefact réellement accepté par le TP2 sans réentraîner au démarrage ;
- définir et tester un contrat d'inférence Pydantic strict ;
- charger une version MLflow par alias et exposer son identité ;
- construire une image Docker non-root avec healthcheck ;
- reproduire les vérifications locales dans GitHub Actions ;
- distinguer éligibilité, classement, promotion et rollback ;
- conserver des preuves exploitables pour le TP4.

1. Point d'entrée TP2

Le parcours réel exige `reports/quality_gate_decision.json` avec `decision: ACCEPT`, `reports/metrics.json`, `reports/run_manifest.json`, le hash des données, `quality_gates/baseline_metrics.json` et `models/model.joblib`. Le manifeste doit aussi contenir `run_id`, `model_artifact` et les mêmes métriques que `reports/metrics.json`. Un modèle `REJECT` doit être refusé avant toute utilisation.

Le starter fournit deux commandes :

TERMINAL · BASH

```
make import-tp2 TP2_DIR=/chemin/vers/TP2
make bootstrap-reference
```

`import-tp2` est le parcours attendu. `bootstrap-reference` est uniquement un secours reproductible pour travailler sans livrable TP2 ; le rapport doit le mentionner explicitement. Après l'import ou le bootstrap :

TERMINAL · BASH

```
make validate-tp2
```

`make pipeline` ne réentraîne pas le modèle ; il vérifie les tests et le contrat d'entrée TP2.

L'import est prévalidé avant copie : un échec ne laisse pas un mélange de fichiers TP2 dans le workspace TP3. Le serving local applique les mêmes contrôles avant de charger le fichier pickle.

2. Contrat d'inférence et API

Implémentez et testez :

- `GET /health` : HTTP 200 si le modèle est chargé, HTTP 503 sinon ;
- `GET /model-info` : nom, alias, version, `run_id`, hash des données et quality gate ;
- `POST /predict` : prédiction, probabilité, nom, alias et version ;
- `GET /metrics` : compteurs, jauge de chargement et histogramme de latence.

Les mêmes opérations sont disponibles sous `/v1`. Les routes historiques sont conservées pour montrer une migration compatible ; le contrat versionné est exporté dans `student/contracts/openapi.json`. Chaque réponse porte `X-Request-ID`, et les erreurs de validation utilisent une enveloppe stable avec un code, un message et des détails.

Le service Registry résout par défaut :

CODE

```
models:/predictive-maintenance@champion
```

La configuration unique est `configs/service.yaml`. Le contrat Pydantic utilise `extra="forbid"` : champs inconnus, champs manquants et valeurs invalides sont rejetés en HTTP 422 avant l'appel au modèle.

La réponse minimale attendue est :

CODE

```
{  
  "prediction": 0,  
  "probability": 0.12,  
  "model_name": "predictive-maintenance",  
  "alias": "champion",  
  "version": "2"  
}
```

3. Docker et CI

Le Dockerfile doit installer l'export issu de `uv.lock`, séparer les couches dépendances/code, exécuter le service avec un utilisateur non-root, utiliser une commande `CMD` exec, déclarer un `HEALTHCHECK` et exclure bases, caches et secrets via `.dockerignore`.

TERMINAL · BASH

```
make docker-build  
make docker-run  
make docker-smoke
```

Le smoke test vérifie les quatre endpoints et supprime le conteneur même en cas d'échec. Les logs ne doivent contenir aucun réentraînement.

Le workflow `.github/workflows/mlops-ci.yml` doit exécuter :

CODE

```
uv sync --locked → ruff → pytest/JUnit → bootstrap de référence  
→ docker build → démarrage du conteneur → smoke test HTTP
```

La CI ne doit jamais promouvoir un modèle. Provoquez un échec contrôlé, identifiez l'étape bloquante, puis restaurez l'état nominal. La promotion reste une action de release explicite et auditable.

`make openapi` régénère le contrat depuis l'application. Une modification de l'API doit donc être accompagnée de la mise à jour de `contracts/openapi.json` et d'un test du contrat.

Extension produit : frontend métier

Une console métier optionnelle est disponible dans `products/predictive-maintenance/frontend/`. Elle consomme uniquement les routes `/v1` de l'API, affiche l'identité du champion et permet de lancer une prédiction. Elle n'est pas requise pour valider le starter TP3. Elle ne doit pas afficher les séries Prometheus, les alertes de drift ou les décisions de retraining : ces vues techniques seront fournies séparément par Grafana dans le TP4.

4. Registry, sélection et policy

Démarrez MLflow avec un backend SQLite local :

TERMINAL · BASH

```
make mlflow-server
```

La policy de référence est dans `configs/promotion_policy.yaml` :

CONFIGURATION · YAML

```
model_name: predictive-maintenance
aliases:
  challenger: challenger
  champion: champion
eligibility:
  min_f1: 0.64
  min_recall: 0.60
  max_latency_ms: 50
  require_quality_gate: true
ranking_metric: f1
promotion:
  minimum_f1_gain: 0.01
  minimum_recall_gain: -0.01
```

Exécutez :

TERMINAL · BASH

```
make register
make select
make promote
```

Enregistrez deux candidats `ACCEPT` issus de deux entrées TP2 distinctes. Après avoir promu le premier, sélectionnez le second pour obtenir une décision `PROMOTE` ou `KEEP_CHAMPION`. Le rollback doit cibler une version différente du champion courant ; cibler la même version est une erreur contrôlée, pas une preuve de rollback.

Les règles à démontrer sont les suivantes :

1. un gate `REJECT` interdit l'enregistrement ;
2. la sélection filtre d'abord l'éligibilité, puis classe les candidats ;
3. le challenger éligible n'est pas automatiquement champion ;
4. `KEEP_CHAMPION` ne modifie pas l'alias ;
5. `PROMOTE` déplace uniquement `champion` ;
6. le rollback réaffecte l'alias sans supprimer de version.

Les cinq artefacts sont obligatoires :

CODE

```
reports/registration.json
reports/model_selection.json
reports/promotion_decision.json
reports/rollback_decision.json
reports/release_manifest.json
```

Le fichier `reports/release_manifest.json` clôt le TP3 et devient l'élément de traçabilité remis au TP4. Il décrit la version effectivement portée par `champion`, après promotion éventuelle ou rollback ; cette version peut être différente du candidat de `registration.json`. Il doit reprendre sans ambiguïté le nom du modèle, l'alias, la version effective, le `run_id`, `data_sha256`, `quality_gate: ACCEPT` et l'image construite. Le handoff TP3 produit par l'import TP4 suit le contrat `tp3.handoff.v1` ; un champ absent ou un hash différent bloque la suite du parcours.

5. Questions d'ingénierie

Répondez dans le rapport, dans l'ordre, en reliant chaque réponse à une preuve réelle :

1. Pourquoi un fichier `model.joblib` ne constitue-t-il pas un contrat de service ?
2. Que protège Pydantic avant l'appel au modèle ?
3. Que rend Docker reproductible et que ne garantit-il pas ?
4. Pourquoi `CI PASS` ne signifie-t-il pas `PROMOTE` ?
5. Dans quel cas le quality gate passe-t-il mais le champion est conservé ?
6. Pourquoi un alias est-il préférable à un numéro de version codé en dur ?
7. Où placer une validation humaine ou métier dans la chaîne ?
8. Quelle preuve permettrait de diagnostiquer une régression entre deux versions ?

6. Rapport, preuves et archive

Créez l'unique rapport avec :

TERMINAL · BASH

```
make report
```

Le rapport doit couvrir la continuité TP2, le contrat API, Docker, la CI, l'enregistrement, la sélection, la promotion, le rollback, le contrat OpenAPI, `release_manifest.json`, Q1 à Q8 et un index des preuves. Ajoutez au moins treize captures réelles dans `reports/figures/` :

- installation et `make doctor` ;
- tests API ;
- réponses des trois endpoints ;
- erreur HTTP 422 ;

- image Docker ;
- conteneur Docker ;
- job CI vert ;
- job CI échoué ;
- version Registry ;
- décision de promotion ;
- rollback.

Chaque preuve doit montrer son contexte, sa commande ou requête, une légende et une limite d'interprétation. Le validateur final exige au moins six tests dans le rapport JUnit, un rapport vert, les quatre JSON Registry/promotion, le workflow CI, le Dockerfile non-root avec healthcheck, une décision cohérente, une preuve de rollback et l'absence de marqueurs [À COMPLÉTER].

Avant la remise :

TERMINAL · BASH

```
make validate-submission  
make package ARCHIVE=MLOps_TP3_Nom_Prenom.zip  
unzip -tq MLOps_TP3_Nom_Prenom.zip
```

L'archive finale doit contenir le rapport, les JSON de décision, le journal JUnit et les preuves. Les bases MLflow, caches, logs, secrets et archives imbriquées sont exclus. La commande `unzip -tq` vérifie l'intégrité physique ; elle ne remplace pas `make validate-submission`, qui vérifie le contenu.

L'archive exclut `.venv`, caches, bases MLflow locales, fichiers temporaires et secrets. Le TP4 consommera directement `models:/predictive-maintenance@champion`, `/model-info` et le contrat de `/predict`.