AYI NEDJIMI Consultants

# MLOps Open Source : MLflow, Kubeflow, ZenML : Guide Complet

Catégorie : Intelligence Artificielle  Lecture : 23 min  Publié le : 13/02/2026  Auteur : Ayi NEDJIMI

*Comparatif détaillé MLflow, Kubeflow et ZenML : experiment tracking, model registry, pipelines ML,. Thèmes : MLOps open source, pipeline ML.*

MLOps Open Source : MLflow, Kubeflow, ZenML : Guide Complet constitue un enjeu majeur pour les professionnels de la sécurité informatique et les équipes techniques. Comparatif détaillé MLflow, Kubeflow et ZenML : experiment tracking, model registry, pipelines ML,. Thèmes : MLOps open source, pipeline ML. Ce guide détaillé sur ia mlops open source mlflow propose une méthodologie structurée, des outils éprouvés et des recommandations opérationnelles directement applicables. L'objectif est de fournir aux praticiens — consultants, ingénieurs sécurité, administrateurs systèmes — les connaissances et les techniques nécessaires pour aborder ce sujet avec rigueur. Chaque section s'appuie sur des retours d'expérience terrain et intègre les évolutions les plus récentes du domaine. Les recommandations présentées sont adaptées aux environnements d'entreprise et tiennent compte des contraintes opérationnelles réelles.

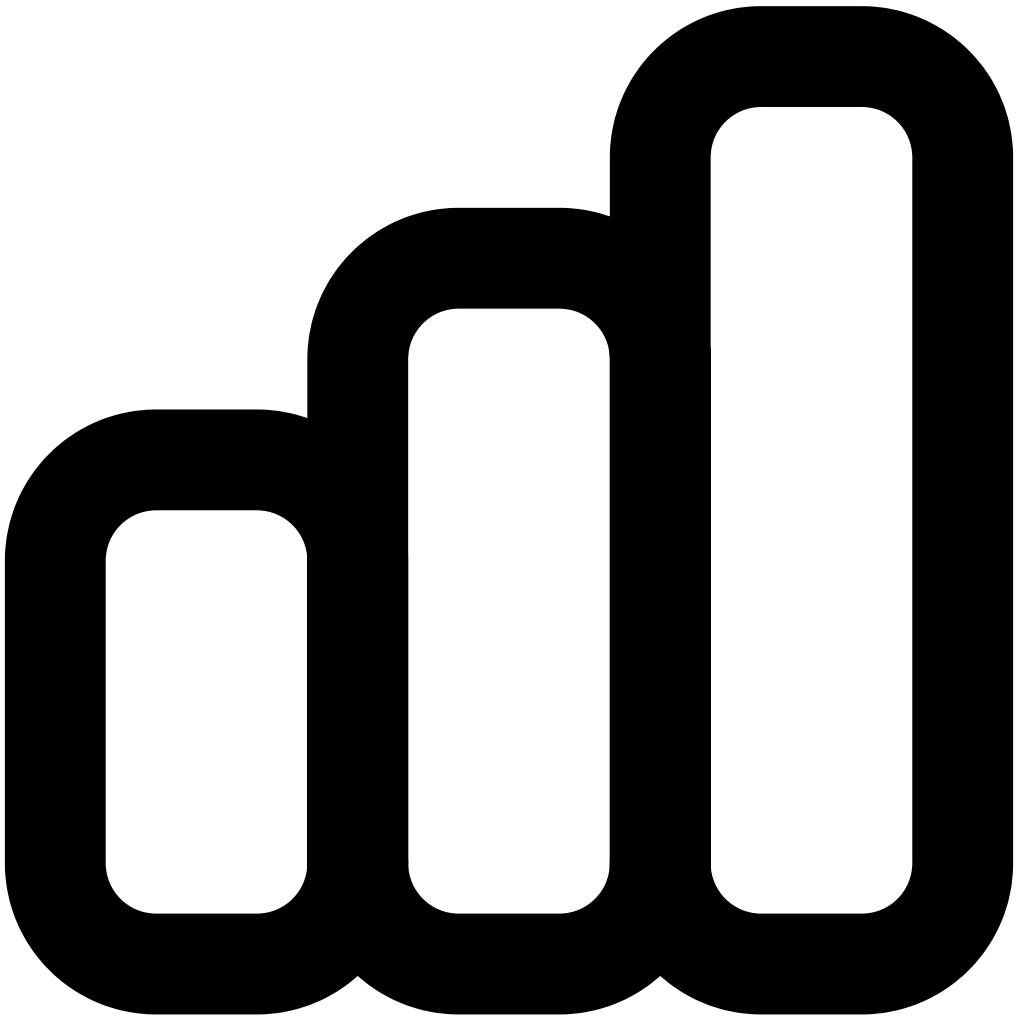
## Table des Matières



## 1 L'Essor du MLOps : Pourquoi Industrialiser le Machine Learning

Le **Machine Learning Operations** (MLOps) est devenu en quelques années une discipline incontournable pour toute organisation souhaitant passer du prototype au système d'IA en production. L'analogie avec le DevOps est éclairante mais insuffisante : là où le DevOps gère du code déterministe, le MLOps doit orchestrer du **code, des données et des modèles** — trois artefacts dont les cycles de vie sont découplés et dont les interactions génèrent une complexité combinatoire considérable. Selon le rapport « State of MLOps 2025 » publié par Weights & Biases et confirmé par les analyses de Gartner, **87 % des projets ML ne dépassent jamais le stade du prototype**. La raison principale n'est pas technique au sens algorithmique, mais

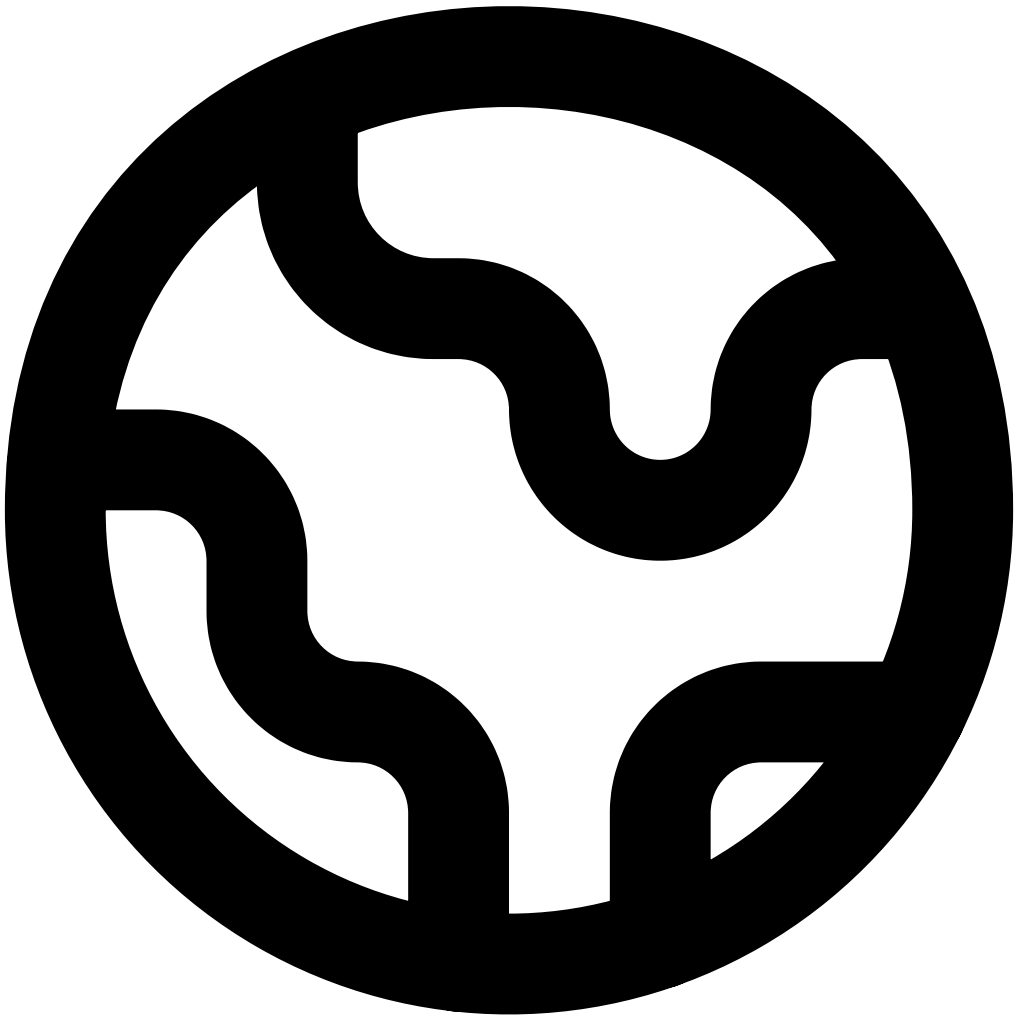
opérationnelle : absence de reproductibilité, dette technique liée aux notebooks, manque de versioning des données et des modèles, et impossibilité de monitorer la dérive des performances en production.



## Les défis du cycle de vie ML

Le cycle de vie d'un modèle de machine learning est fondamentalement plus complexe que celui d'une application logicielle classique. Il comprend au minimum sept étapes critiques : la **collecte et le nettoyage des données**, l'**ingénierie des features**, l'**entraînement et l'optimisation des hyperparamètres**, l'**évaluation et la validation**, le **packaging et le déploiement**, le **monitoring en production** et le **réentraînement**. Chacune de ces étapes implique des outils différents, des compétences variées (data engineers, ML engineers, platform engineers) et des artefacts spécifiques qui doivent être versionnés, tracés et reproductibles. La fameuse règle empirique de Google Research, publiée dans le papier fondateur « Hidden Technical Debt in Machine Learning Systems » (2015), reste plus pertinente que jamais : **le code du modèle ne**

**représente que 5 à 10 % du code total** d'un système ML en production. Le reste — pipelines de données, feature engineering, serving infrastructure, monitoring — constitue la véritable complexité opérationnelle.



## Le paysage open source en 2026

L'écosystème MLOps open source a connu une consolidation significative entre 2023 et 2026. Si des dizaines d'outils ont émergé durant la période d'hypercroissance 2020-2023, le marché s'est structuré autour de trois plateformes dominantes qui couvrent chacune un spectre fonctionnel large : **MLflow** (créé par Databricks), **Kubeflow** (issu de l'écosystème Google/Kubernetes) et **ZenML** (approche framework-agnostique et composable). À côté de ces trois piliers, des outils spécialisés comme **Metaflow** (Netflix), **ClearML**, **DVC** (Data Version Control), **Feast** (feature store) ou **Evidently** (monitoring) complètent l'écosystème. Le choix de la bonne stack MLOps dépend de multiples facteurs : la maturité ML de l'organisation, l'infrastructure existante (cloud,

on-premise, hybride), la taille de l'équipe, et surtout les priorités fonctionnelles — certaines organisations privilégient l'experiment tracking tandis que d'autres ont besoin en priorité d'un orchestrateur de pipelines robuste.

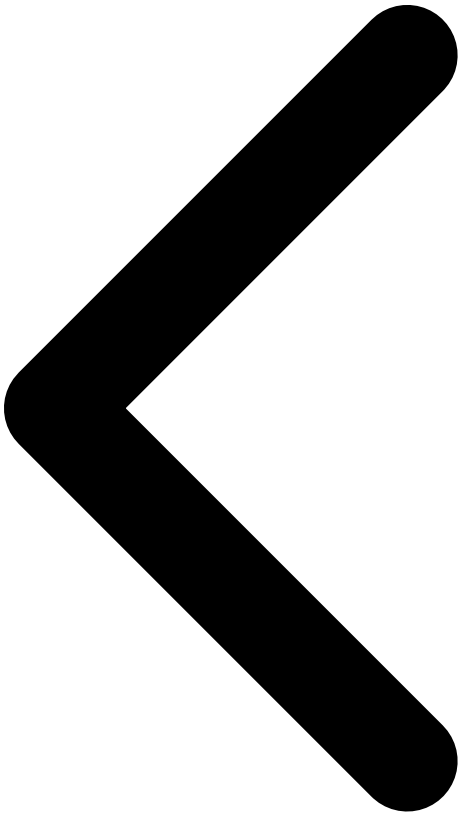
Vos pipelines de données d'entraînement sont-ils protégés contre l'empoisonnement ?

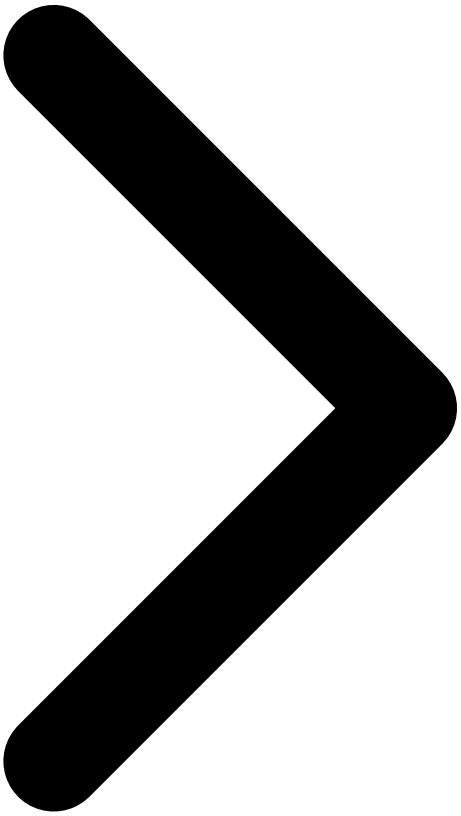
## MLOps et sécurité : un enjeu transversal

L'industrialisation des pipelines ML soulève des questions de sécurité fondamentales qui traversent l'ensemble du cycle de vie. Le **supply chain ML** — l'ensemble des dépendances, modèles pré-entraînés, datasets et librairies utilisés — constitue une surface d'attaque considérable. Les attaques par **data poisoning** (injection de données malveillantes dans les datasets d'entraînement), par **model poisoning** (compromission des poids via des modèles backdoorés sur Hugging Face) et par **inference manipulation** (prompt injection, extraction de modèle) nécessitent des contrôles de sécurité intégrés nativement dans la plateforme MLOps.

Les trois plateformes que nous comparons dans cet article offrent des niveaux de maturité différents sur ces aspects sécurité, un critère que nous évaluerons en détail dans chaque section.

**Point clé :** Le MLOps n'est pas un outil mais une **discipline d'ingénierie** qui combine des pratiques, des processus et des outils pour fiabiliser le cycle de vie ML. Le choix de la bonne plateforme open source est un levier d'accélération majeur, mais il doit s'inscrire dans une stratégie organisationnelle plus large incluant la gouvernance des données, la sécurité du pipeline et la montée en compétences des équipes.



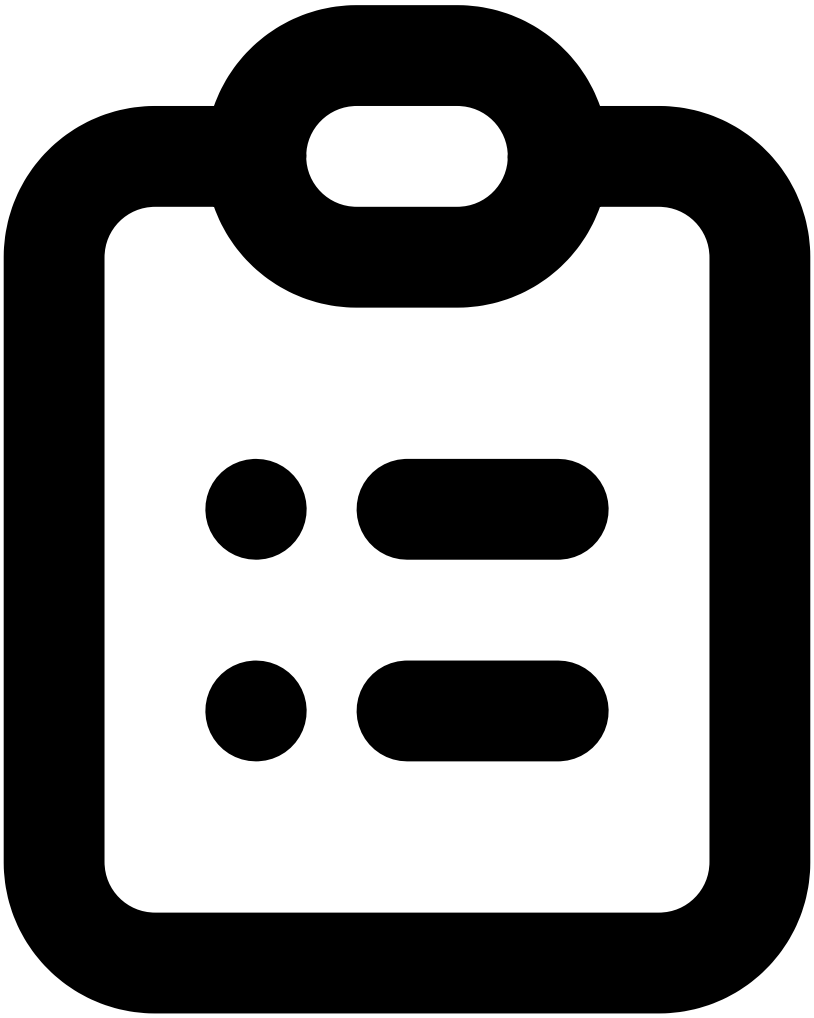


| Critere | Description | Niveau de risque |
|---|---|---|
| **Confidentialite** | Protection des donnees d'entrainement et des prompts | Eleve |
| **Integrite** | Fiabilite des sorties et detection des hallucinations | Critique |
| **Disponibilite** | Resilience du service et gestion de la charge | Moyen |
| **Conformite** | Respect du RGPD, AI Act et politiques internes | Eleve |

## 2 MLflow : Le Standard de Fait

**MLflow**, créé par Databricks en 2018 et placé sous la gouvernance de la Linux Foundation depuis 2024, s'est imposé comme la plateforme MLOps open source la plus largement adoptée au monde. Avec plus de **18 000 étoiles GitHub**, une communauté de plus de 800 contributeurs et une adoption dans plus de 75 % des entreprises du Fortune 500 utilisant du ML en production (selon le rapport Databricks 2025), MLflow est devenu le standard de facto pour l'experiment

tracking et le model registry. Son succès repose sur une philosophie pragmatique : fournir des **APIs simples et non-intrusives** qui s'intègrent dans n'importe quel workflow existant sans imposer une refonte complète de l'infrastructure.



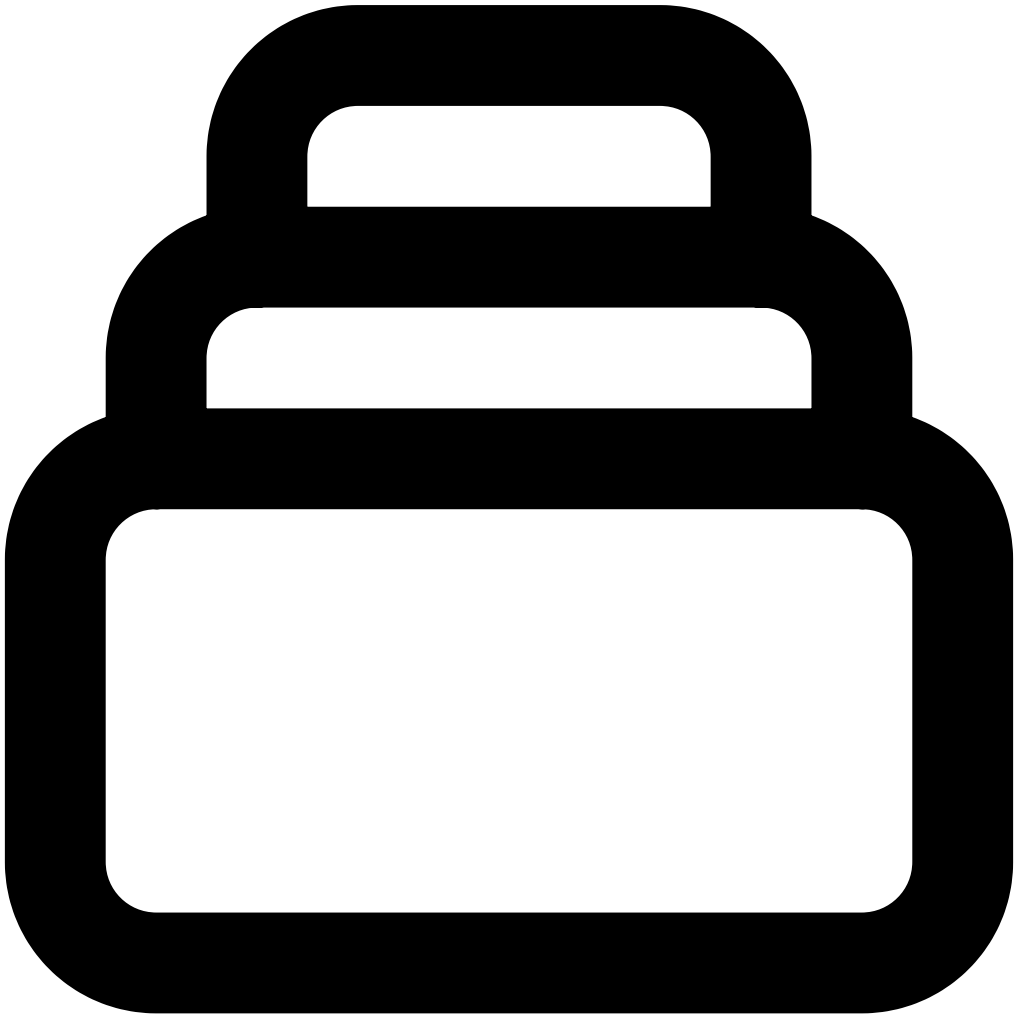
## MLflow Tracking : le coeur du système

Le composant **MLflow Tracking** constitue le pilier fondamental de la plateforme. Il permet de logger systématiquement les paramètres, les métriques, les artefacts et les tags de chaque exécution d'entraînement (appelée « run »). L'API Python est volontairement minimaliste : quelques lignes de code suffisent pour instrumenter n'importe quel script d'entraînement, qu'il utilise scikit-learn, PyTorch, TensorFlow, XGBoost ou n'importe quel autre framework. La fonction **autolog**, introduite en version 1.x et considérablement améliorée en version 2.x, permet même de capturer automatiquement les paramètres et métriques sans modifier le code existant — il suffit d'appeler `mlflow.autolog()` en début de script. Le serveur de tracking stocke les métadonnées dans un backend SQL (SQLite en local, PostgreSQL ou MySQL en production) et les

artefacts dans un stockage objet (S3, GCS, Azure Blob, ou même le système de fichiers local). L'interface web intégrée permet de comparer visuellement les runs, de filtrer par métriques et de naviguer dans les artefacts.

Python

```
import mlflow
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score

# Configuration du tracking server
mlflow.set_tracking_uri("http://mlflow-server:5000")
mlflow.set_experiment("fraud-detection-v3")

# Entraînement avec tracking automatique
with mlflow.start_run(run_name="rf-optimized"):
    params = {"n_estimators": 500, "max_depth": 12, "min_samples_split": 5}
    mlflow.log_params(params)

    model = RandomForestClassifier(**params)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)

    mlflow.log_metric("accuracy", accuracy_score(y_test, preds))
    mlflow.log_metric("f1_score", f1_score(y_test, preds))
    mlflow.sklearn.log_model(model, "model",
        registered_model_name="fraud-detector")
```



## Model Registry et déploiement

Le **MLflow Model Registry** fournit un registre centralisé pour gérer le cycle de vie des modèles, de l'expérimentation au déploiement en production. Chaque modèle enregistré dispose d'un versioning automatique, d'un système de **stages** (Staging, Production, Archived) permettant de gérer les transitions, et d'annotations (descriptions, tags) facilitant la gouvernance. La version 2.x de MLflow a introduit le concept d'**Aliases**, plus flexible que les stages traditionnels : un alias comme « champion » ou « challenger » peut pointer vers n'importe quelle version d'un modèle, permettant des stratégies de déploiement comme le canary release ou l'A/B testing. Le composant **MLflow Deployments** (anciennement MLflow Models) standardise le packaging des modèles au format **MLmodel**, un format portable qui encapsule le modèle, ses dépendances et sa signature d'entrée/sortie. Ce format unifié permet de déployer vers des cibles multiples : serveur REST local, Docker, Kubernetes (via Seldon ou KServe), AWS SageMaker, Azure ML ou Databricks Serving.

**Cas concret**
En 2024, des chercheurs de Cornell ont publié une étude démontrant l'empoisonnement de données d'entraînement de modèles de vision par ordinateur avec seulement 0.01% d'images malveillantes, suffisant pour créer des backdoors indétectables par les méthodes de validation standard.

## Forces et limites de MLflow

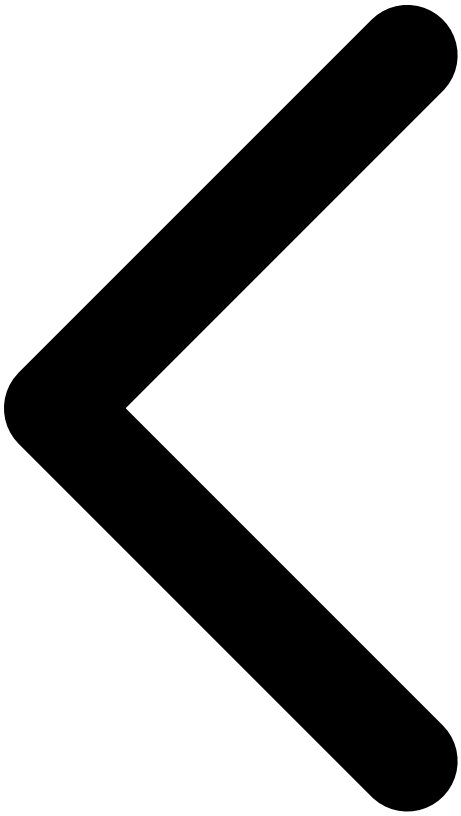
Les **forces majeures** de MLflow sont sa simplicité d'adoption (courbe d'apprentissage très faible, quelques minutes pour commencer), sa polyvalence (fonctionne avec n'importe quel framework ML, supporte Python, R, Java et REST), son écosystème d'intégrations très riche (plus de 30 flavors natifs pour les frameworks ML populaires) et sa flexibilité de déploiement (du laptop au cluster multi-cloud). En revanche, MLflow présente des **limites significatives**. L'orchestration de pipelines n'est pas son point fort : MLflow Recipes (anciennement Pipelines) reste limité par rapport à des orchestrateurs dédiés comme Airflow ou Prefect. La scalabilité de l'UI de tracking peut poser problème au-delà de dizaines de milliers de runs sans optimisation du backend. L'absence de gestion native du compute distribué signifie que MLflow ne sait pas nativement

provisionner des GPU ou gérer des jobs d'entraînement distribués — il faut coupler MLflow avec un orchestrateur externe (Kubernetes, Spark, Ray) pour ces cas d'usage. Enfin, la sécurité et le multi-tenancy sont des fonctionnalités de la version commerciale (Databricks), absentes de la version open source de base, ce qui constitue un frein pour les grandes organisations. Pour approfondir, consultez IA dans la Finance : Détection de Fraude Temps Réel et.

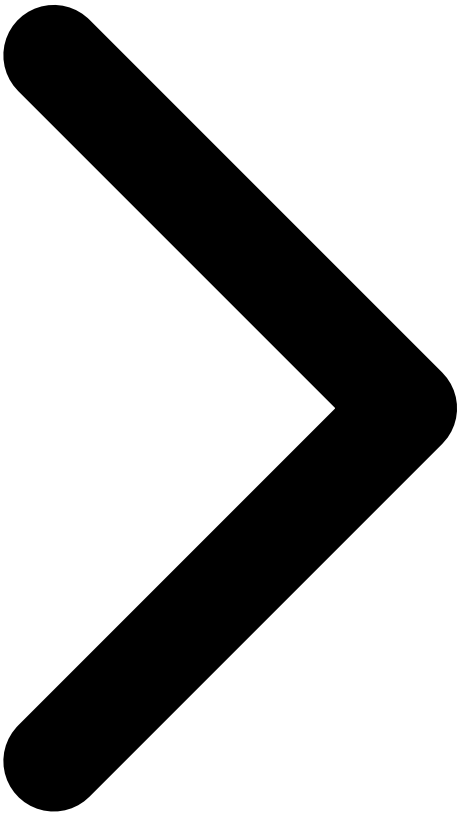
Votre organisation est-elle prête à faire face aux attaques basées sur l'IA ?

Figure 1 — Stack MLOps open source : positionnement de MLflow, Kubeflow et ZenML dans l'architecture de référence 2026

**Verdict MLflow :** MLflow est le choix idéal pour les équipes qui débutent en MLOps ou qui cherchent un **experiment tracker et model registry robuste** sans bouleverser leur infrastructure existante. Sa force est sa simplicité et son universalité. Sa limite est qu'il ne couvre pas nativement l'orchestration de pipelines complexes ni le compute distribué — il doit être combiné avec d'autres outils pour constituer une stack MLOps complète.

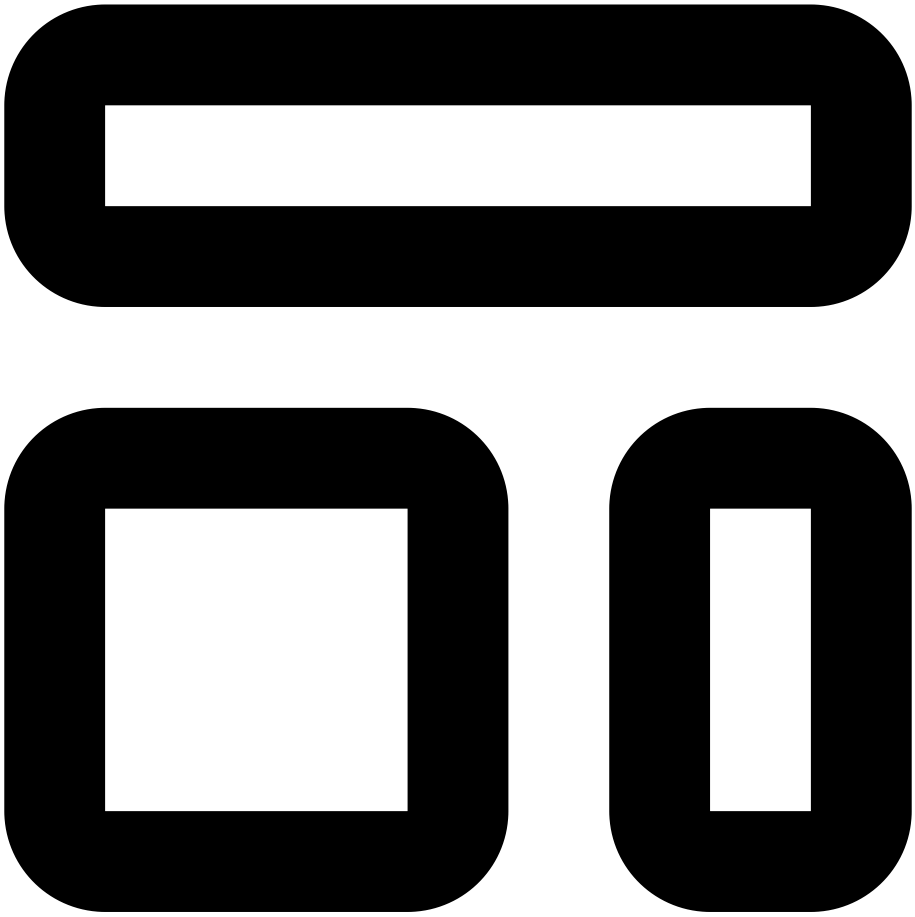


L'Essor du MLOps MLflow Kubeflow



## 3 Kubeflow : Le MLOps Cloud-Native

**Kubeflow**, lancé par Google en 2018 comme projet pour simplifier le déploiement de workflows ML sur Kubernetes, est devenu un projet CNCF (Cloud Native Computing Foundation) mature qui représente l'approche **cloud-native du MLOps**. Contrairement à MLflow qui est d'abord une librairie Python, Kubeflow est une **plateforme distribuée** composée de multiples composants déployés comme des microservices sur un cluster Kubernetes. Avec plus de 14 000 étoiles GitHub et une adoption significative par les grandes entreprises opérant des clusters Kubernetes (Google, Spotify, Bloomberg, Airbnb), Kubeflow s'adresse aux organisations qui disposent déjà d'une infrastructure Kubernetes et souhaitent capitaliser sur cet investissement pour leurs workloads ML.

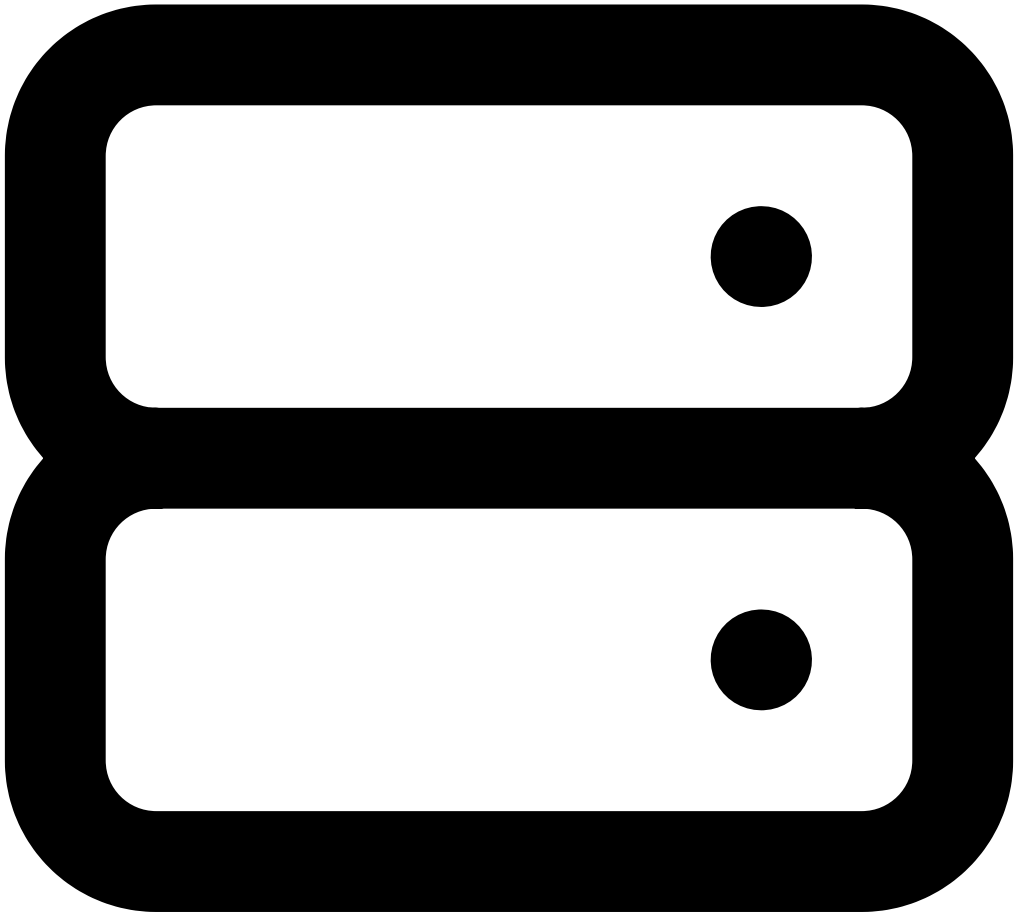


## Kubeflow Pipelines : l'orchestration ML avancée

**Kubeflow Pipelines** (KFP) est le composant le plus utilisé et le plus mature de l'écosystème. Il permet de définir des workflows ML sous forme de **graphes acycliques dirigés** (DAG) où chaque étape s'exécute dans un conteneur isolé sur Kubernetes. La version 2 de KFP, sortie en 2023 et stabilisée en 2025, a introduit une refonte majeure du SDK Python avec un référence déclaratif basé sur des décorateurs. Chaque composant est une fonction Python décorée avec `@dsl.component`, compilée en une image conteneur et orchestrée par Argo Workflows (le moteur d'exécution sous-jacent). L'avantage fondamental de cette approche est l'**isolation totale des environnements** : chaque étape du pipeline peut utiliser une image Docker différente avec ses propres dépendances, éliminant les conflits de versions — un problème récurrent dans les pipelines ML monolithiques. KFP gère nativement le **caching des étapes** (une étape avec les mêmes inputs n'est pas réexécutée), la gestion des artefacts intermédiaires et le versioning des pipelines.

Python

```
from kfp import dsl, compiler

@dsl.component(base_image="python:3.11-slim",
               packages_to_install=["pandas", "scikit-learn"])
def train_model(data_path: str, n_estimators: int) -> str:
    import pandas as pd
    from sklearn.ensemble import GradientBoostingClassifier
    import pickle

    df = pd.read_csv(data_path)
    X, y = df.drop("target", axis=1), df["target"]
    model = GradientBoostingClassifier(n_estimators=n_estimators)
    model.fit(X, y)

    model_path = "/tmp/model.pkl"
    with open(model_path, "wb") as f:
        pickle.dump(model, f)
    return model_path

@dsl.pipeline(name="training-pipeline")
def ml_pipeline(data_uri: str = "gs://bucket/data.csv"):
    train_task = train_model(data_path=data_uri, n_estimators=200)
    train_task.set_cpu_limit("4").set_memory_limit("16Gi")
    train_task.set_gpu_limit(1)  # Allocation GPU native K8s

compiler.Compiler().compile(ml_pipeline, "pipeline.yaml")
```



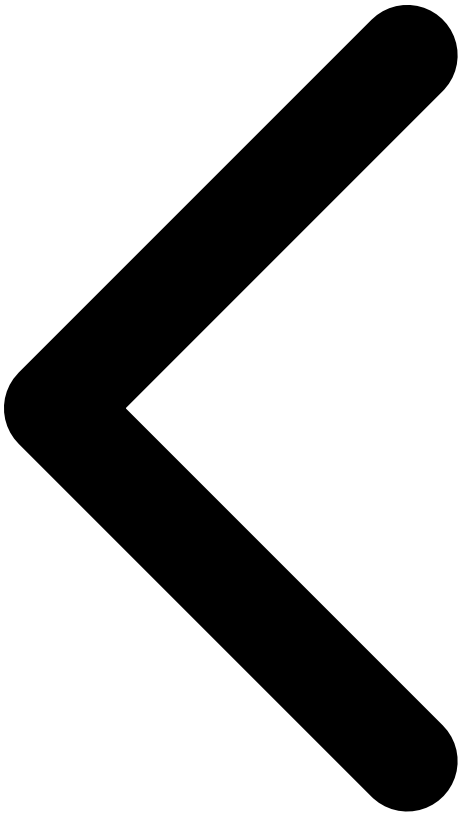
### KServe et Training Operators

**KServe** (anciennement KFServing) est le composant de serving de l'écosystème Kubeflow. Il fournit une couche d'abstraction serverless pour le déploiement de modèles sur Kubernetes, avec des fonctionnalités avancées comme l'**autoscaling basé sur le trafic** (scale-to-zero inclus), le **canary deployment** natif (routage progressif du trafic entre versions), les **transformers** (pré/post-processing des requêtes dans des conteneurs séparés) et le support multi-runtime (TensorFlow Serving, Triton Inference Server, ONNX Runtime, vLLM). KServe s'appuie sur Knative pour le scaling serverless et Istio pour le routage du trafic, ce qui ajoute de la puissance mais aussi de la complexité opérationnelle. Les **Training Operators** (TFJob pour TensorFlow, PyTorchJob pour PyTorch, MPIJob pour Horovod) permettent de lancer des entraînements distribués multi-GPU et multi-noeud directement sur Kubernetes, avec une gestion native des ressources GPU via le NVIDIA Device Plugin. L'intégration avec **Katib**, le composant d'AutoML de Kubeflow, permet d'automatiser l'optimisation des hyperparamètres en lançant des centaines d'essais en parallèle sur le cluster.

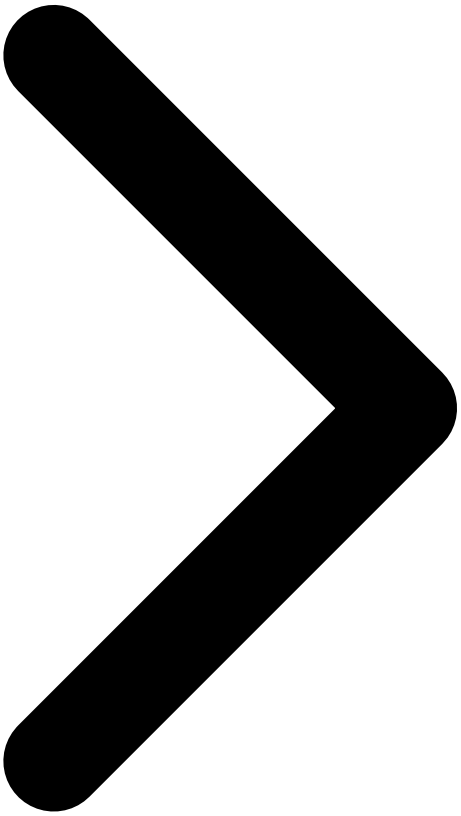
### Forces et limites de Kubeflow

Les **forces majeures** de Kubeflow sont sa scalabilité (hérite de la scalabilité de Kubernetes), son isolation d'environnement parfaite (chaque étape dans son conteneur), sa gestion native des GPU et du compute distribué, son multi-tenancy intégré (via les namespaces Kubernetes et les profils Kubeflow) et son serving serverless avancé via KServe. C'est la plateforme la plus complète pour les organisations à **grande échelle**. En revanche, Kubeflow souffre de **limites importantes**. La complexité d'installation et d'opération est considérable : un déploiement complet nécessite Kubernetes, Istio, Knative, cert-manager, et de nombreux CRDs — comptez plusieurs jours de travail pour un cluster de production. La courbe d'apprentissage est raide, particulièrement pour les data scientists qui ne sont pas familiers avec les concepts Kubernetes. Le debugging des pipelines échoués nécessite de naviguer dans les logs de pods Kubernetes, une compétence qui sort du périmètre classique d'un data scientist. Enfin, l'experiment tracking natif de Kubeflow est moins mature que celui de MLflow — de nombreuses organisations utilisent d'ailleurs MLflow pour le tracking tout en orchestrant leurs pipelines avec Kubeflow.

**Verdict Kubeflow :** Kubeflow est le choix optimal pour les organisations qui opèrent déjà **Kubernetes en production** et qui ont besoin de scalabilité, de multi-tenancy et de gestion GPU avancée. Il excelle pour les pipelines ML complexes à grande échelle. Sa complexité opérationnelle le réserve toutefois aux équipes disposant de **compétences Kubernetes solides** — il est surdimensionné pour les petites équipes ou les phases d'exploration.

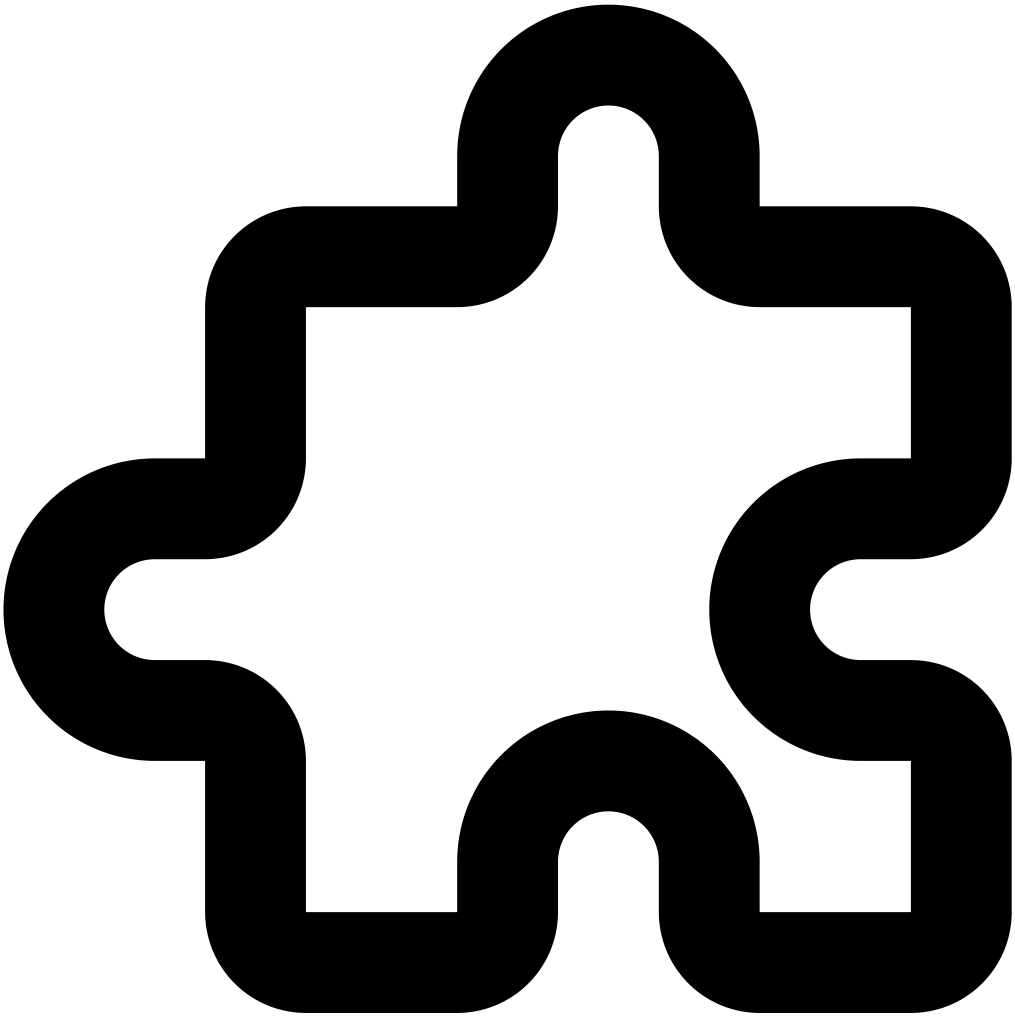


MLflow Kubeflow ZenML et Alternatives



## 4 ZenML et les Alternatives Émergentes

**ZenML**, fondé en 2021 par une équipe d'anciens ingénieurs ML de Google et Stripe, propose une approche radicalement différente du MLOps : au lieu de fournir une plateforme monolithique, ZenML se positionne comme un **framework d'orchestration composable** qui permet de connecter n'importe quel outil de l'écosystème ML via des abstractions standardisées. La philosophie de ZenML repose sur le concept de « stacks » : une stack est un ensemble de composants interchangeables (orchestrateur, artifact store, experiment tracker, model deployer, etc.) que l'on assemble selon ses besoins. Vous pouvez commencer avec une stack locale (orchestration séquentielle, filesystem local) et migrer progressivement vers une stack de production (Kubeflow comme orchestrateur, S3 comme artifact store, MLflow comme tracker, Seldon comme deployer) — le code du pipeline reste identique.

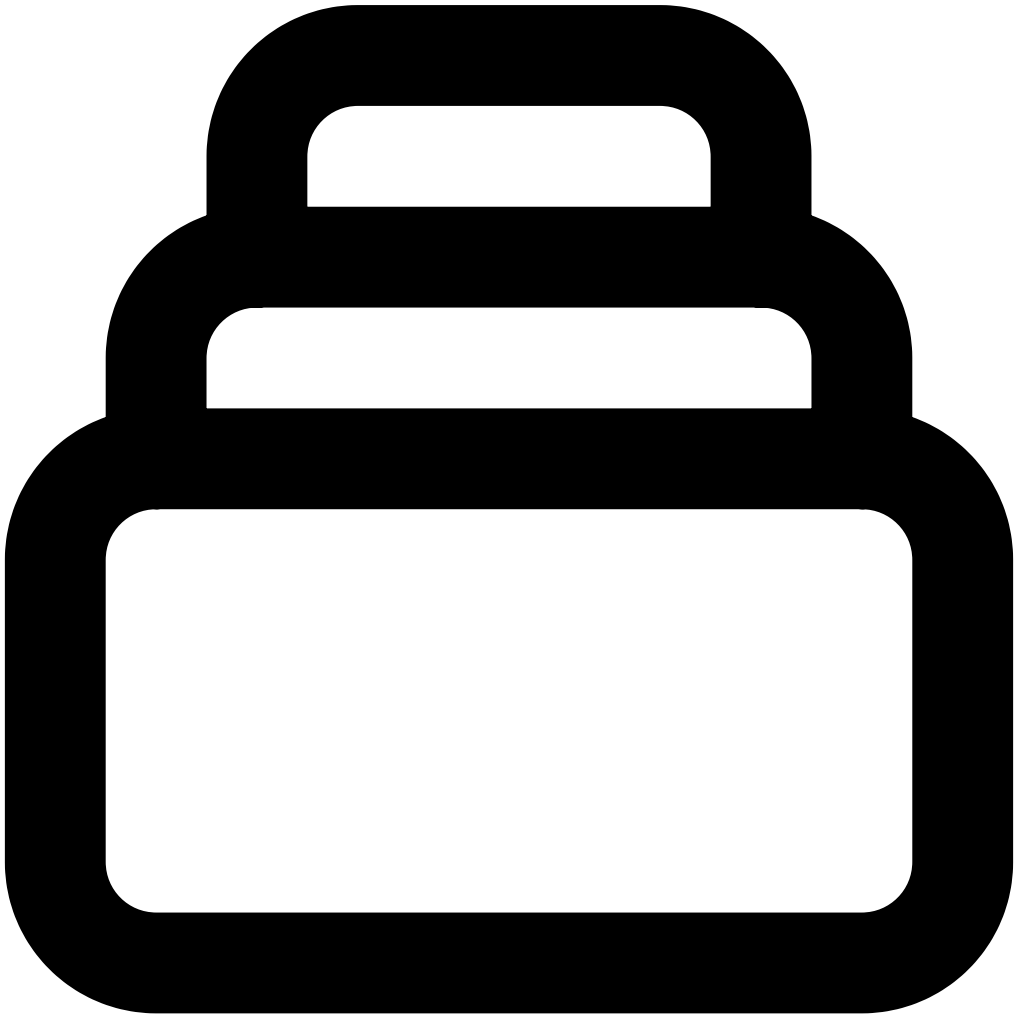


### L'architecture composable de ZenML

L'innovation clé de ZenML est son système de **stack components** qui abstrait chaque couche de l'infrastructure MLOps. En février 2026, ZenML propose plus de **50 intégrations** officielles couvrant les principales catégories : orchestrateurs (Airflow, Kubeflow, Tekton, Vertex AI, SageMaker, local), artifact stores (S3, GCS, Azure Blob, local), experiment trackers (MLflow, Weights & Biases, Neptune, Comet), model deployers (Seldon, BentoML, KServe, MLflow), annotateurs (Label Studio, Prodigy) et alerters (Slack, Discord, PagerDuty). Le SDK Python de ZenML est élégant et Pythonic : chaque étape de pipeline est un simple décorateur `@step`, et le pipeline est assemblé avec `@pipeline`. Le typage fort des entrées/sorties avec des annotations Python standard permet à ZenML de valider automatiquement la compatibilité des étapes et de sérialiser les artefacts de manière transparente. Pour approfondir, consultez la documentation officielle ZenML.

Python

```
from zenml import pipeline, step
from zenml.client import Client

@step
def load_data() -> pd.DataFrame:
    return pd.read_parquet("s3://bucket/training-data.parquet")

@step
def train_model(data: pd.DataFrame) -> sklearn.base.BaseEstimator:
    X, y = data.drop("target", axis=1), data["target"]
    model = GradientBoostingClassifier(n_estimators=300)
    model.fit(X, y)
    return model

@step
def evaluate(model: sklearn.base.BaseEstimator, data: pd.DataFrame) -> float:
    preds = model.predict(data.drop("target", axis=1))
    return accuracy_score(data["target"], preds)

@pipeline
def training_pipeline():
    data = load_data()
    model = train_model(data)
    score = evaluate(model, data)

# Exécution locale
training_pipeline()

# Migration vers Kubeflow : changer la stack, pas le code
# zenml stack set production-k8s
```



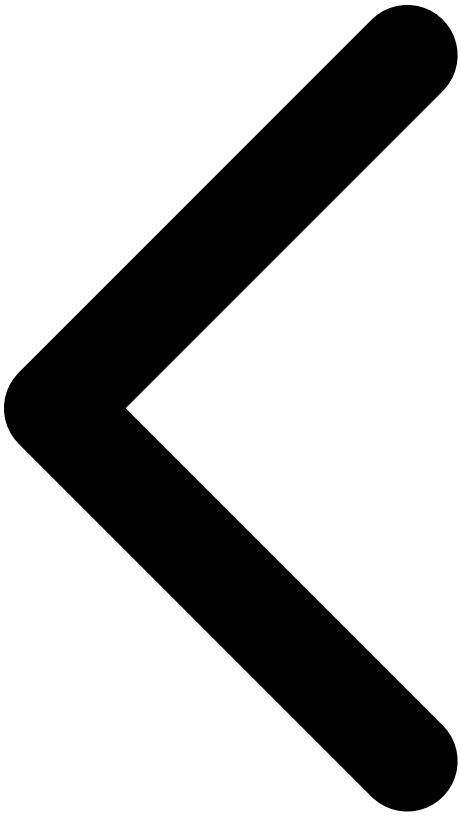
### Metaflow, ClearML et Prefect

Au-delà des trois plateformes principales, plusieurs alternatives méritent une attention particulière. **Metaflow**, créé par Netflix et open-sourcé en 2019, excelle dans la gestion des pipelines de données et ML à grande échelle. Sa force réside dans sa gestion native du versioning des données (chaque étape produit un artefact versionné automatiquement), son intégration transparente avec AWS (S3, Batch, Step Functions) et son approche « human-centric » qui minimise le boilerplate. Metaflow gère nativement le **compute scaling** : une étape peut passer de l'exécution locale à un cluster AWS Batch de 100 instances avec une simple annotation `@batch(cpu=16, memory=32000, gpu=1)`. **ClearML** (anciennement Trains) se positionne comme une alternative complète à MLflow avec un focus sur la simplicité et l'auto-hébergement. Son agent d'exécution permet de transformer n'importe quelle machine en worker d'entraînement, et son système de data versioning intégré évite d'avoir recours à un outil séparé comme DVC. **Prefect**, bien que principalement un orchestrateur de workflows génériques (concurrent d'Airflow), est de plus en plus utilisé pour les pipelines ML grâce à son API Pythonic,

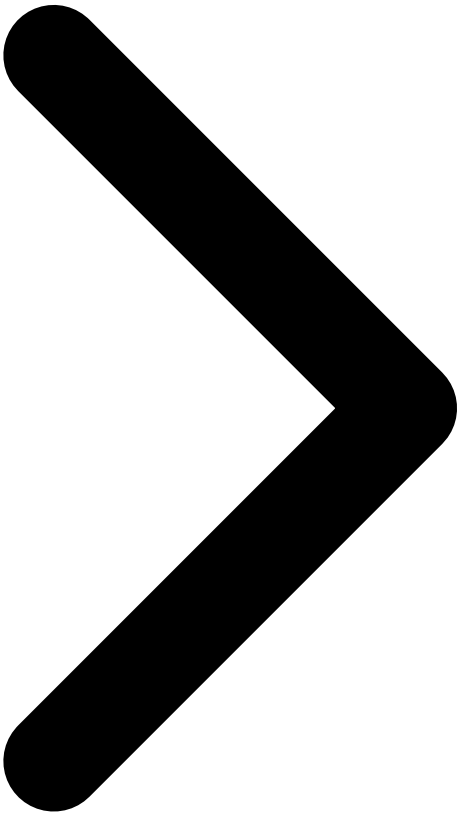
sa gestion avancée des retry et des dépendances, et son excellente observabilité. Sa version 3.x, sortie en 2025, a introduit des primitives spécifiques pour le ML comme le caching d'artefacts et les triggers basés sur des métriques de drift.

Figure 2 — Comparatif radar des features MLflow, Kubeflow et ZenML sur 8 critères clés

**Verdict ZenML :** ZenML est le choix optimal pour les équipes qui veulent une **plateforme composable et cloud-agnostique** sans se verrouiller avec un seul vendor ou orchestrateur. Sa capacité à abstraire l'infrastructure tout en restant flexible en fait un excellent compromis entre la simplicité de MLflow et la puissance de Kubeflow. Sa jeunesse relative (communauté plus petite, écosystème moins testé en battle conditions) est sa principale limite.

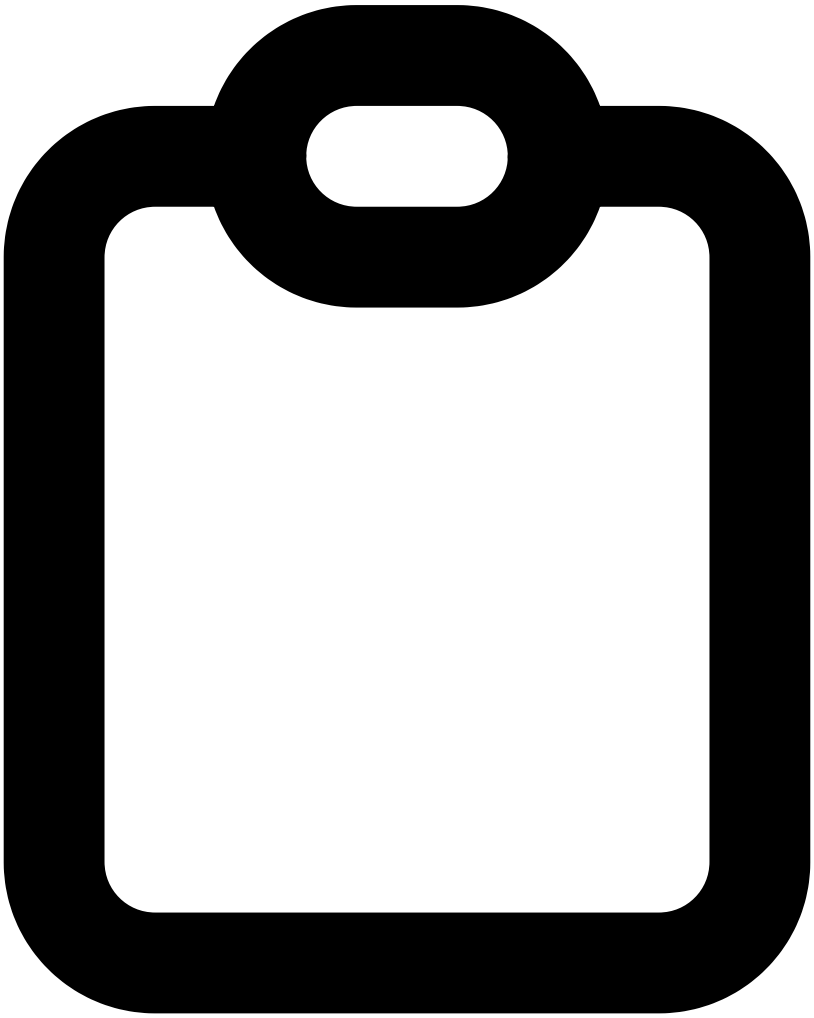


Kubeflow ZenML et Alternatives Tracking & Registry



## 5 Experiment Tracking et Model Registry

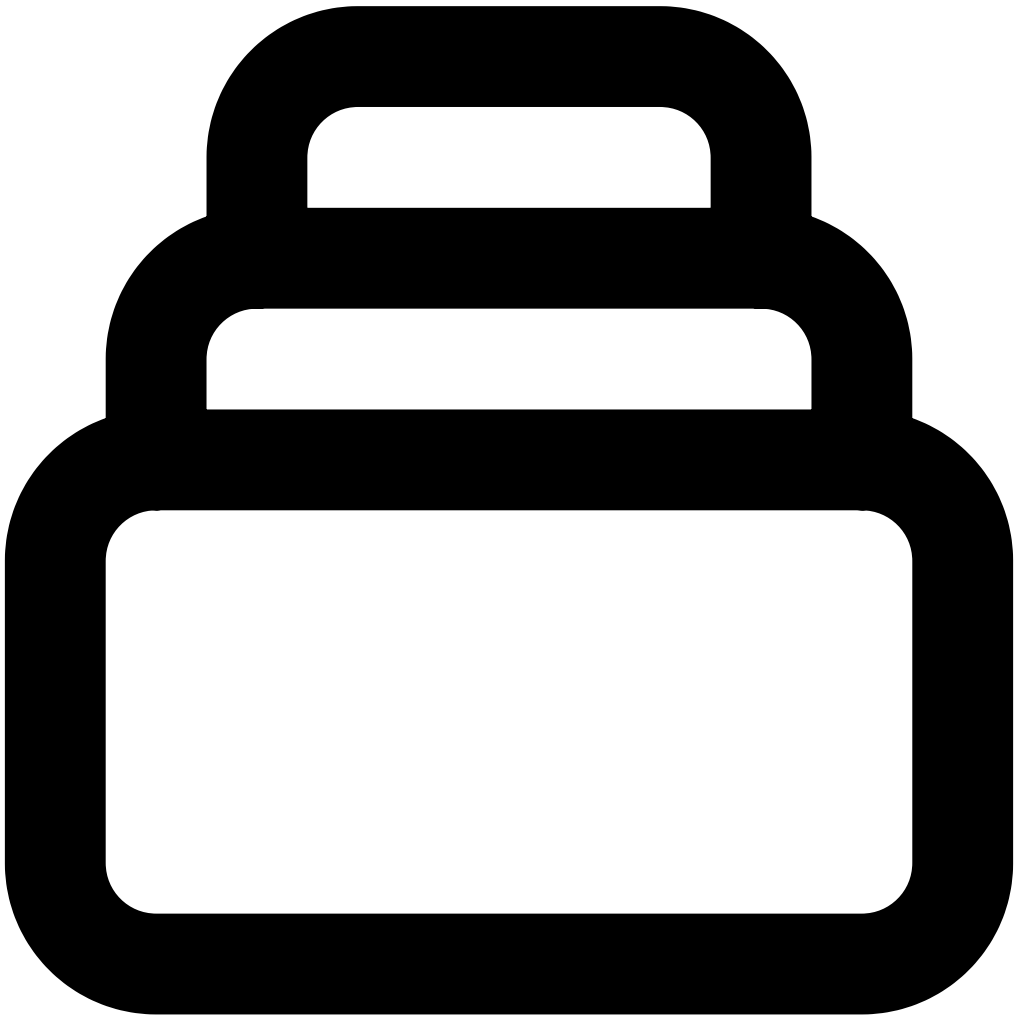
L'**experiment tracking** et le **model registry** sont les deux fonctionnalités fondamentales que toute stack MLOps doit fournir. L'experiment tracking résout le problème de la reproductibilité — chaque entraînement doit être traçable avec ses paramètres, métriques, données d'entrée et artefacts de sortie. Le model registry résout le problème de la gestion du cycle de vie des modèles — quelle version est en production, quelle version est en staging, quand a eu lieu la dernière mise à jour, qui a approuvé le déploiement. Ces deux fonctionnalités sont si critiques qu'elles déterminent souvent le choix de la plateforme MLOps principale.



### Comparatif détaillé du tracking

**MLflow Tracking** reste la référence en matière d'experiment tracking open source. Ses avantages sont nombreux : API minimaliste (trois lignes pour commencer), autolog pour la majorité des frameworks ML, UI web complète avec comparaison visuelle des runs, recherche par métriques avec syntaxe SQL-like, et stockage flexible (du SQLite local à PostgreSQL/MySQL en production). La limite principale est la scalabilité de l'UI au-delà de 50 000 runs sans indexation fine du backend. **Kubeflow** s'appuie sur **ML Metadata** (MLMD), un composant de TensorFlow Extended (TFX) qui enregistre les métadonnées de chaque exécution dans une base SQL. MLMD est puissant pour la traçabilité (lineage complet artefact-par-artefact), mais son API est de plus bas niveau et son UI est moins intuitive que celle de MLflow. **ZenML** ne propose pas son propre experiment tracker mais s'intègre nativement avec MLflow, Weights & Biases, Neptune et Comet ML. Cette approche « bring your own tracker » est cohérente avec sa philosophie composable mais signifie que le choix et la configuration du tracker restent à la charge de l'utilisateur. **ClearML** se distingue par son tracking « zero-config » : il suffit d'ajouter

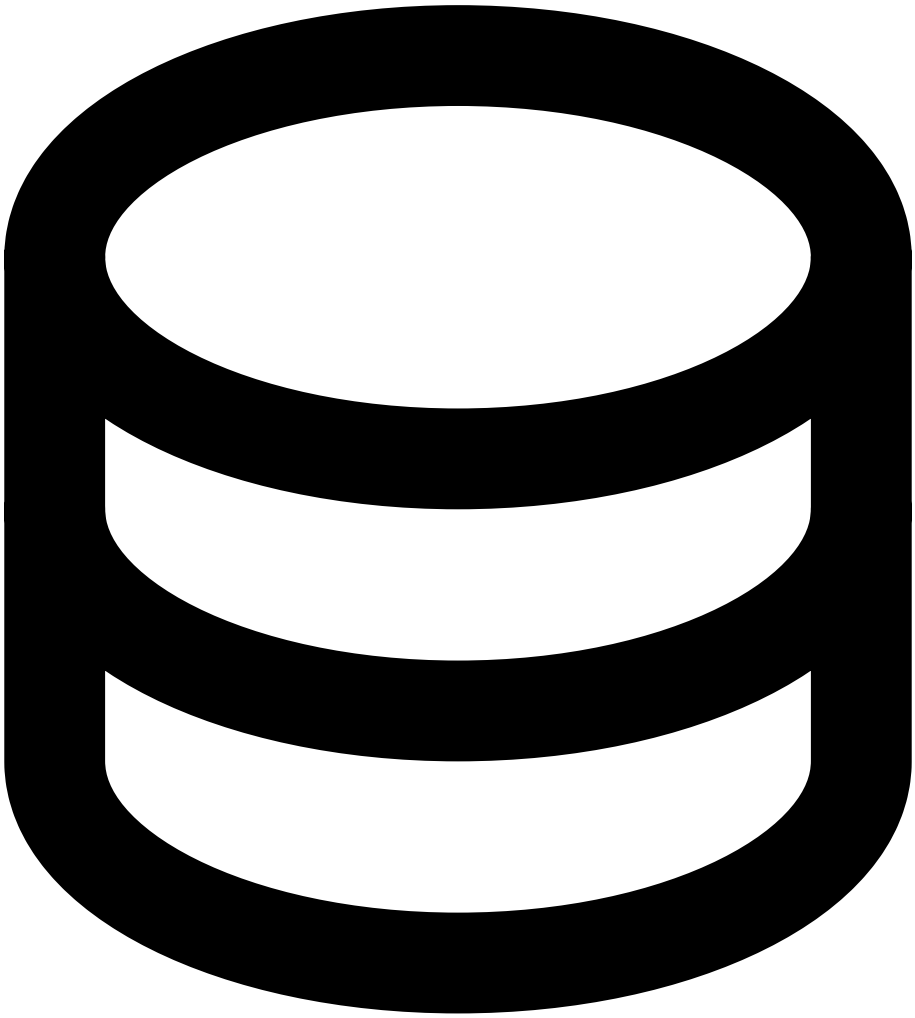
`from clearml import Task; task = Task.init()` en début de script pour capturer automatiquement les métriques, les graphiques Matplotlib, les hyperparamètres et même les packages installés — sans aucune modification du code d'entraînement existant.



## Model Registry : versioning et gouvernance

Le **MLflow Model Registry** est le plus mature des registres open source. Il offre le versioning automatique des modèles, les transitions de stage (Staging/Production/Archived), les alias flexibles (champion/challenger), les annotations et descriptions, les webhooks pour l'automatisation (déclencher un pipeline de tests quand un modèle passe en Staging), et l'intégration native avec le tracking. L'API REST permet d'intégrer le registry dans des pipelines CI/CD standard. **Kubeflow** ne dispose pas d'un model registry natif aussi complet — l'écosystème recommande typiquement d'utiliser MLflow Model Registry ou un registry d'images OCI (comme Harbor) pour stocker les modèles conteneurisés. **ZenML** propose depuis sa version 0.50+ un **Model Control Plane** qui va au-delà du simple registry : il associe un modèle non

seulement à ses artefacts et versions, mais aussi aux pipelines qui l'ont produit, aux données d'entraînement utilisées, et aux déploiements actifs. Cette vision « model-centric » du lineage est plus riche que celle de MLflow et facilite l'audit et la conformité réglementaire.

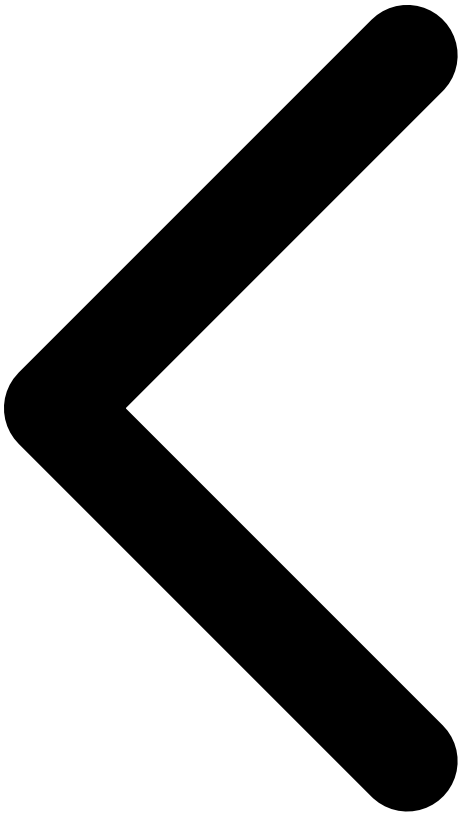


## Versioning des données et lineage

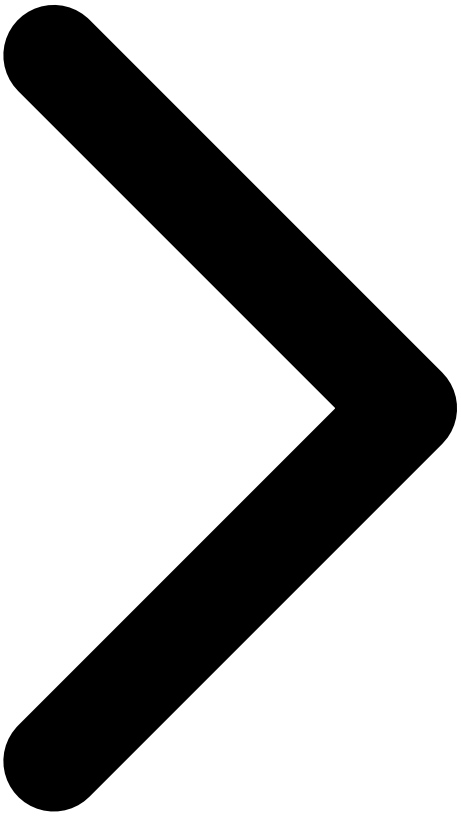
Un aspect souvent négligé du MLOps est le **versioning des données**. Un modèle en production n'a de valeur que si l'on peut retracer exactement les données qui ont servi à l'entraîner. **DVC** (Data Version Control) est l'outil de référence pour le versioning des datasets : il fonctionne comme Git mais pour les fichiers volumineux, en stockant les données dans un remote (S3, GCS) et les métadonnées dans le repo Git. L'intégration DVC + MLflow permet un lineage complet : le hash DVC du dataset est loggé comme paramètre du run MLflow, reliant ainsi un modèle à sa version exacte de données. **LakeFS**, une alternative émergente, propose un système de branches et de commits Git-like directement sur le data lake (S3 compatible), permettant de créer des « branches de données » pour l'expérimentation sans dupliquer les fichiers. ZenML

intègre nativement le concept d'**artifact versioning** : chaque sortie d'étape de pipeline est automatiquement versionnée, hachée et stockée dans l'artifact store configuré, avec un lineage complet traçable via le dashboard ZenML.

**Recommandation :** Pour l'experiment tracking, commencez avec **MLflow Tracking** — c'est le standard qui offre le meilleur ratio fonctionnalités/complexité. Pour le model registry, MLflow Model Registry convient à 90 % des cas ; envisagez le ZenML Model Control Plane si vous avez des exigences fortes en lineage et audit réglementaire. Pour le versioning des données, **DVC** est indispensable dès que vos datasets dépassent quelques centaines de Mo.

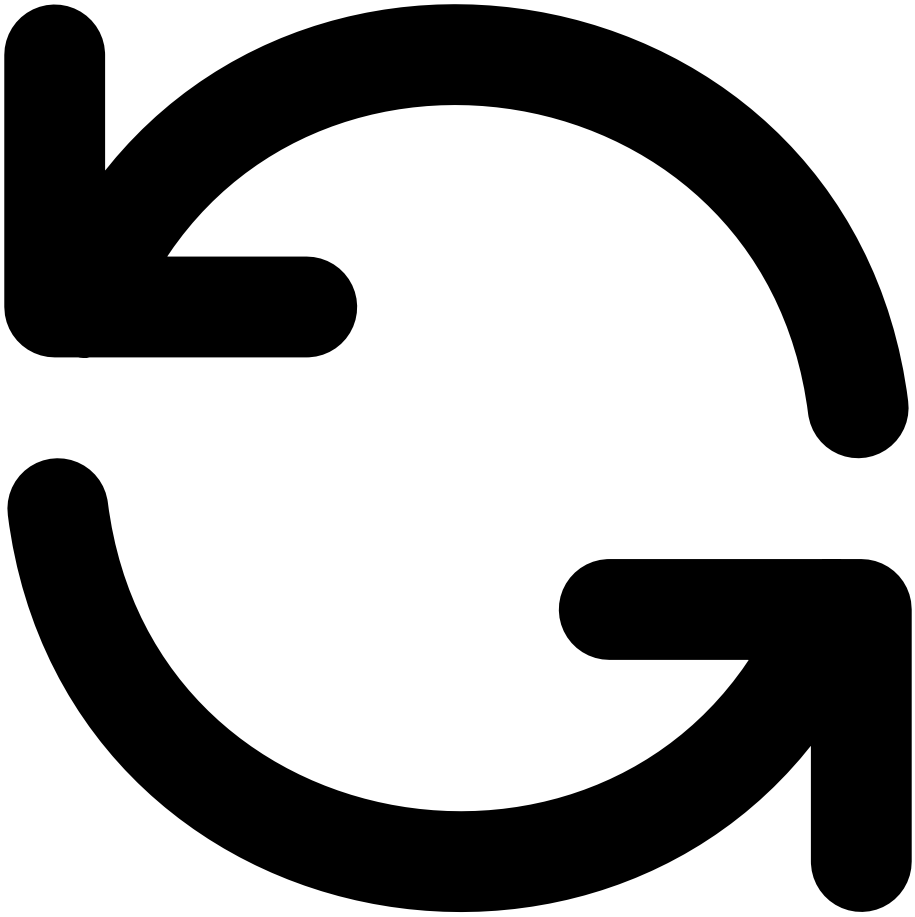


ZenML et Alternatives Tracking & Registry Pipelines en Production



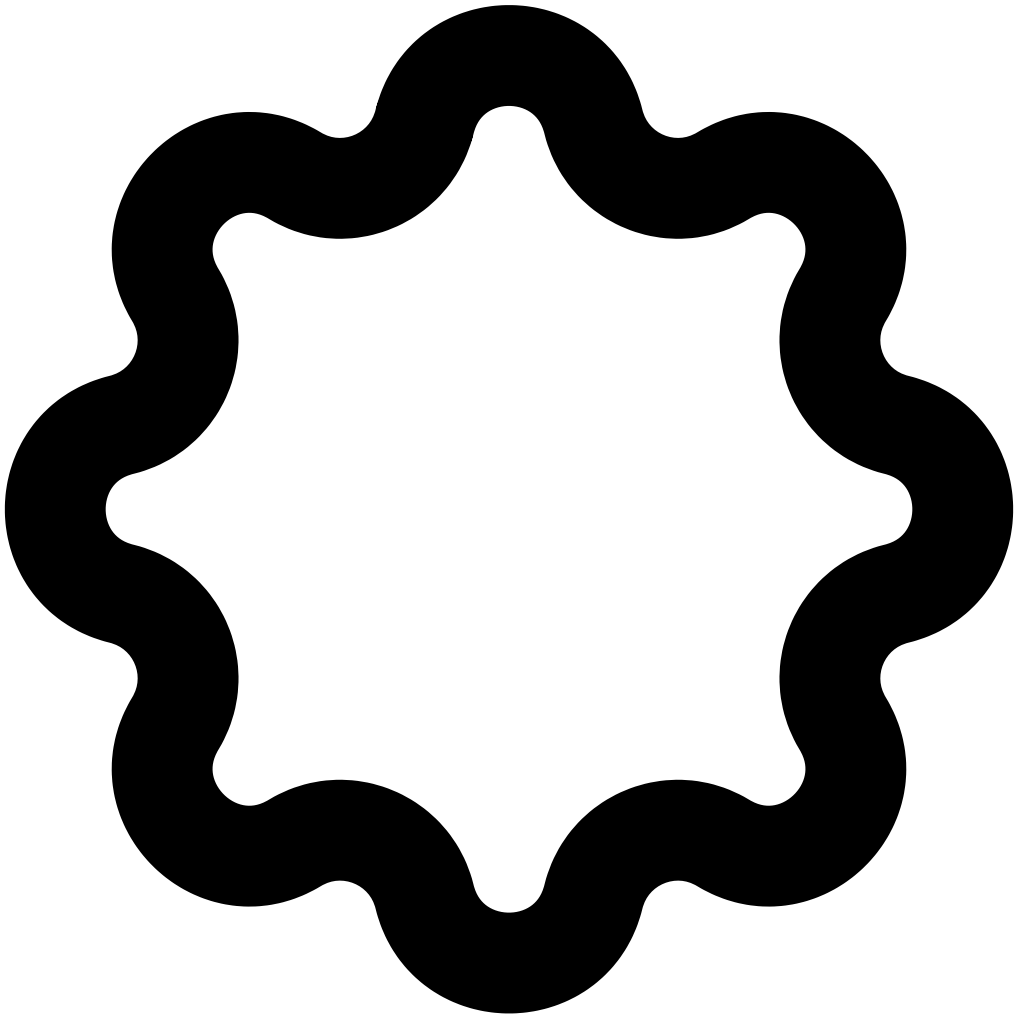
## 6 Pipelines ML en Production

Le passage d'un notebook d'expérimentation à un **pipeline ML de production** est la transition la plus critique — et la plus difficile — du cycle de vie MLOps. Un pipeline de production doit être reproductible, testable, monitorable, scalable et résilient aux pannes. Il ne s'agit plus de lancer un script manuellement, mais d'orchestrer une séquence de tâches interdépendantes qui s'exécutent de manière automatisée, avec des mécanismes de retry, de validation et d'alerte. Les trois plateformes que nous comparons proposent des approches différentes pour résoudre ce défi, chacune avec ses compromis entre simplicité et puissance. Pour approfondir, consultez OWASP Top 10 LLM 2025 : Risques et Remediations.



### Orchestration : Airflow, Argo, Prefect, Dagster

Le choix de l'**orchestrateur** est une décision architecturale fondamentale. **Apache Airflow**, le vétéran de l'orchestration, reste le choix le plus répandu avec plus de 35 000 étoiles GitHub et une adoption massive dans l'industrie. Sa force est son écosystème de 2 000+ opérateurs (connexions vers quasiment tous les services cloud et bases de données existants) et sa maturité opérationnelle éprouvée. Sa limite pour le ML est son modèle d'exécution basé sur des DAG Python statiques, peu adapté aux workflows dynamiques où le graphe dépend des résultats intermédiaires. **Argo Workflows**, utilisé en interne par Kubeflow Pipelines, est l'orchestrateur Kubernetes-native le plus performant : chaque étape est un pod Kubernetes, avec une gestion native des GPU, des volumes et du parallélisme. **Prefect** (version 3.x) et **Dagster** représentent la nouvelle génération d'orchestrateurs conçus dès l'origine pour le ML : ils supportent les workflows dynamiques, le caching intelligent des résultats intermédiaires, les retry granulaires et une observabilité fine de chaque exécution. Dagster introduit le concept innovant de « software-defined assets » qui modélise les pipelines non pas comme des tâches à exécuter, mais comme des **artefacts de données à produire** — un schéma particulièrement adapté au ML.

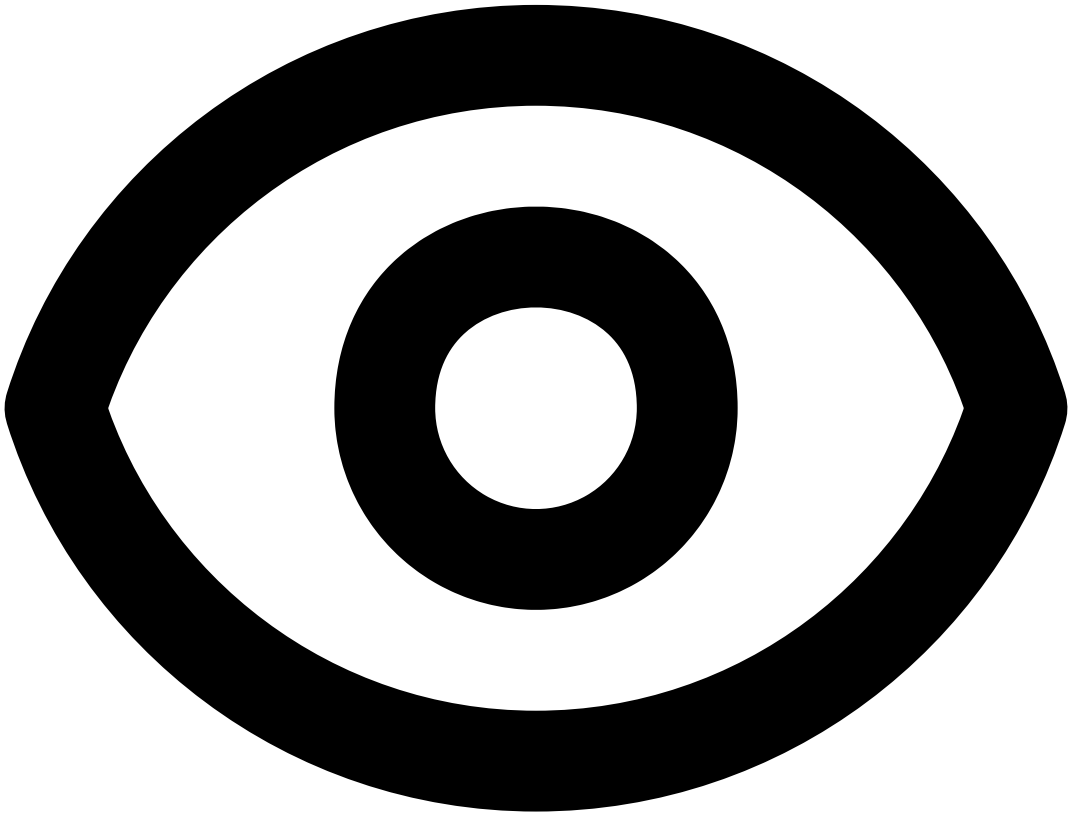


## CI/CD pour le Machine Learning

L'intégration du ML dans les pratiques **CI/CD** requiert d'adapter les pipelines classiques de livraison logicielle aux spécificités du ML. Un pipeline CI/CD ML comprend typiquement trois couches : le **CI/CD du code** (tests unitaires, linting, revue de code — identique au logiciel classique), le **CI/CD des données** (validation des schémas, détection d'anomalies, contrôle de qualité avec Great Expectations ou Pandera) et le **CI/CD des modèles** (entraînement automatisé, évaluation sur un dataset de test, comparaison avec le modèle en production, déploiement conditionnel). Ce dernier niveau est spécifique au ML et nécessite des outils dédiés. Le pattern le plus mature, implémentable avec n'importe laquelle des trois plateformes, est le **model validation gate** : avant tout déploiement, un pipeline automatisé compare les métriques du nouveau modèle avec celles du modèle en production sur un ensemble de test représentatif. Le déploiement n'est déclenché que si le nouveau modèle dépasse un seuil de performance configurable. GitHub Actions, GitLab CI et Tekton sont les moteurs CI/CD les plus utilisés pour orchestrer ces validations.

YAML

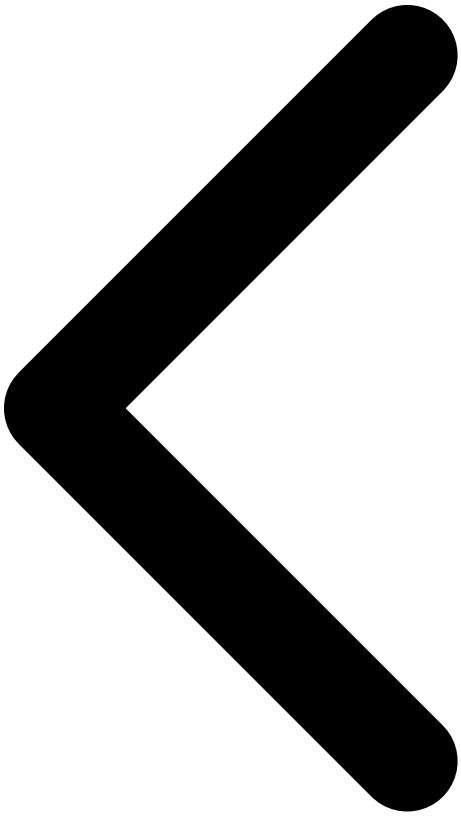
```
# GitHub Actions - Pipeline CI/CD ML avec validation gate
name: ML Model CI/CD
on:
  push:
    paths: ['models/**', 'data/**', 'pipelines/**']

jobs:
  train-and-validate:
    runs-on: [self-hosted, gpu]
    steps:
      - uses: actions/checkout@v4
      - name: Train model
        run: |
          python pipelines/train.py \
            --experiment-name "ci-${{ github.sha }}" \
            --tracking-uri ${{ secrets.MLFLOW_URI }}

      - name: Validate against production
        run: |
          python pipelines/validate.py \
            --candidate "models/candidate" \
            --champion "models/production" \
            --threshold-accuracy 0.02 \
            --threshold-latency-ms 50

      - name: Deploy if validated
        if: success()
        run: |
          mlflow models serve -m "models:/fraud-detector/candidate" \
            --port 8080 --enable-mlserver
```



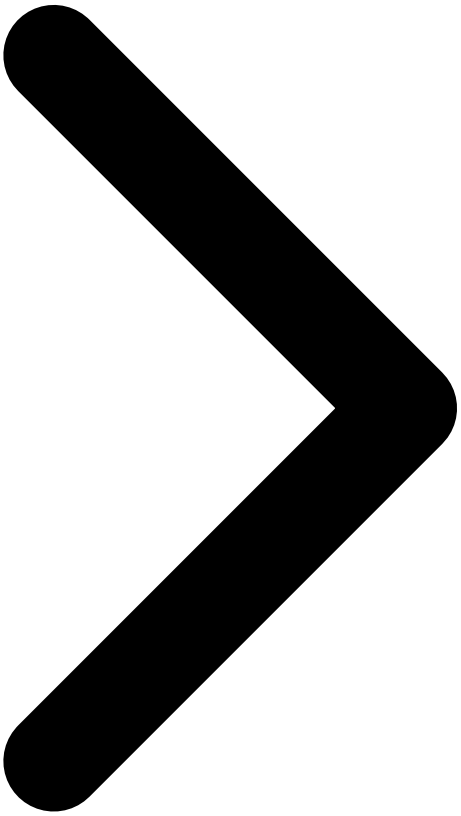
## Monitoring et détection de drift

Le **monitoring en production** est la couche la plus souvent négligée du MLOps, alors qu'elle est essentielle pour maintenir la qualité du système dans le temps. Un modèle ML en production se dégrade inévitablement à cause du **data drift** (les distributions des données d'entrée évoluent par rapport aux données d'entraînement), du **concept drift** (la relation entre les features et la cible change) et de la **dégradation technique** (bugs dans le preprocessing, changements d'API en amont). **Evidently AI** est l'outil de référence open source pour la détection de drift et le monitoring de la qualité des données et des prédictions. Il génère des rapports HTML détaillés et peut s'intégrer dans des pipelines automatisés via ses tests programmables. **Whylogs** (WhyLabs) propose une approche de profiling statistique léger qui génère des résumés compacts des distributions de données, permettant de monitorer des flux de millions d'événements sans stocker les données brutes. **NannyML** se distingue par sa capacité à estimer la performance du modèle **sans les labels réels** (méthode CBPE — Confidence-Based Performance Estimation), particulièrement utile quand le feedback de vérité terrain est décalé dans le temps (fraude détectée des semaines après la transaction).

**Architecture recommandée :** Pour une stack MLOps de production complète, combinez **Airflow ou Prefect** pour l'orchestration, **MLflow** pour le tracking et le registry, **DVC** pour le versioning des données, **Great Expectations** pour la validation de données, **Evidently** pour le monitoring de drift et **GitHub Actions** pour le CI/CD. ZenML peut servir de couche d'abstraction unificatrice pour connecter tous ces composants.

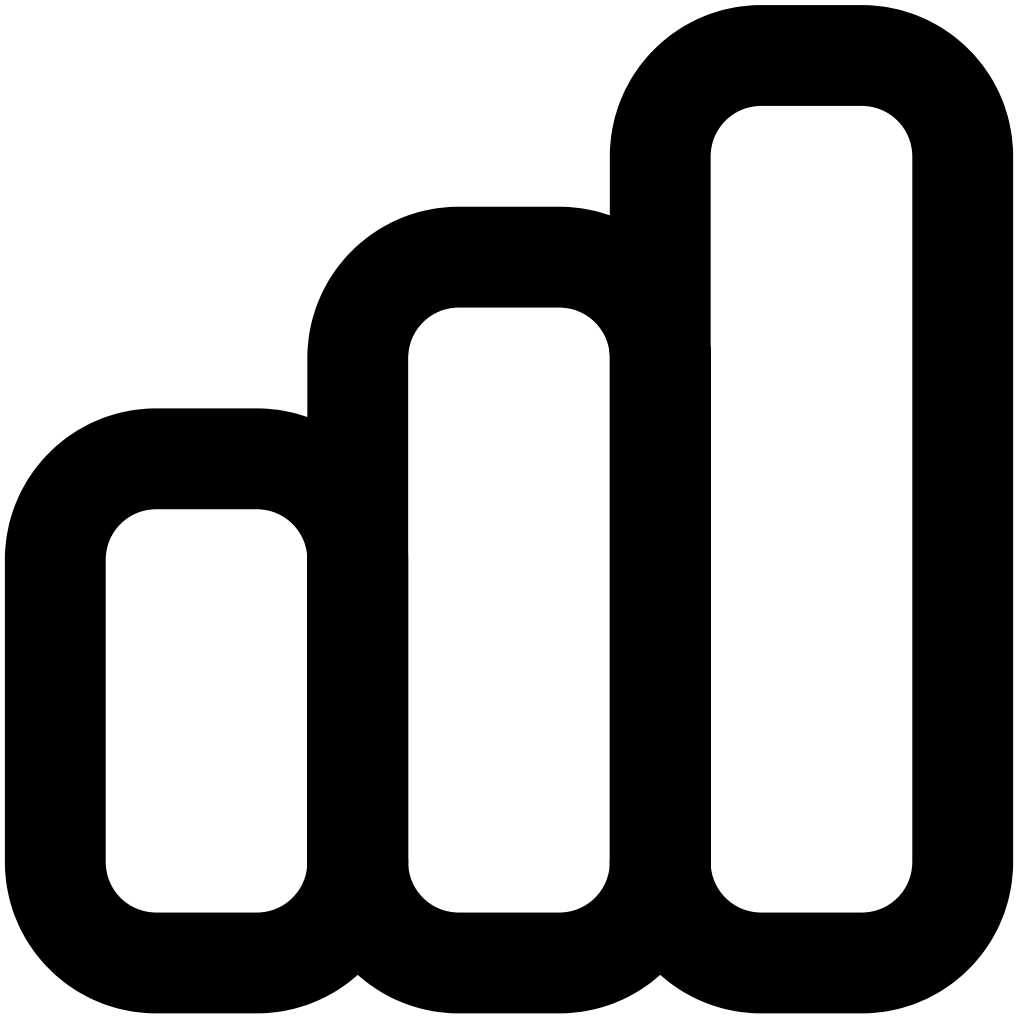


Tracking & Registry Pipelines en Production Choisir sa Stack



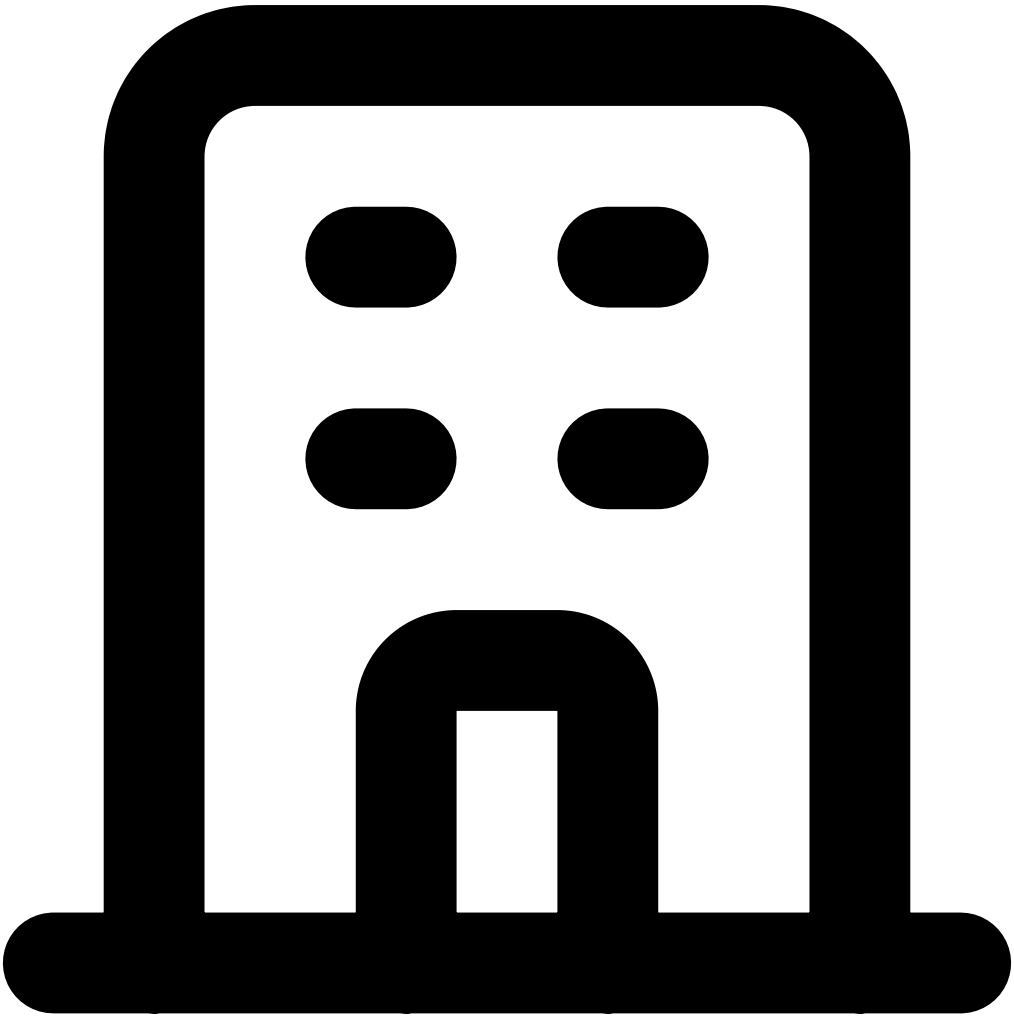
## 7 Choisir sa Stack MLOps : Arbre de Décision

Le choix de la bonne stack MLOps est une décision stratégique qui doit prendre en compte la **maturité ML de l'organisation**, l'infrastructure existante, la taille et les compétences de l'équipe, et les besoins fonctionnels prioritaires. Il n'existe pas de solution universelle : la meilleure stack est celle qui s'adapte à votre contexte et qui peut évoluer avec vos besoins. Voici un arbre de décision structuré pour guider ce choix.



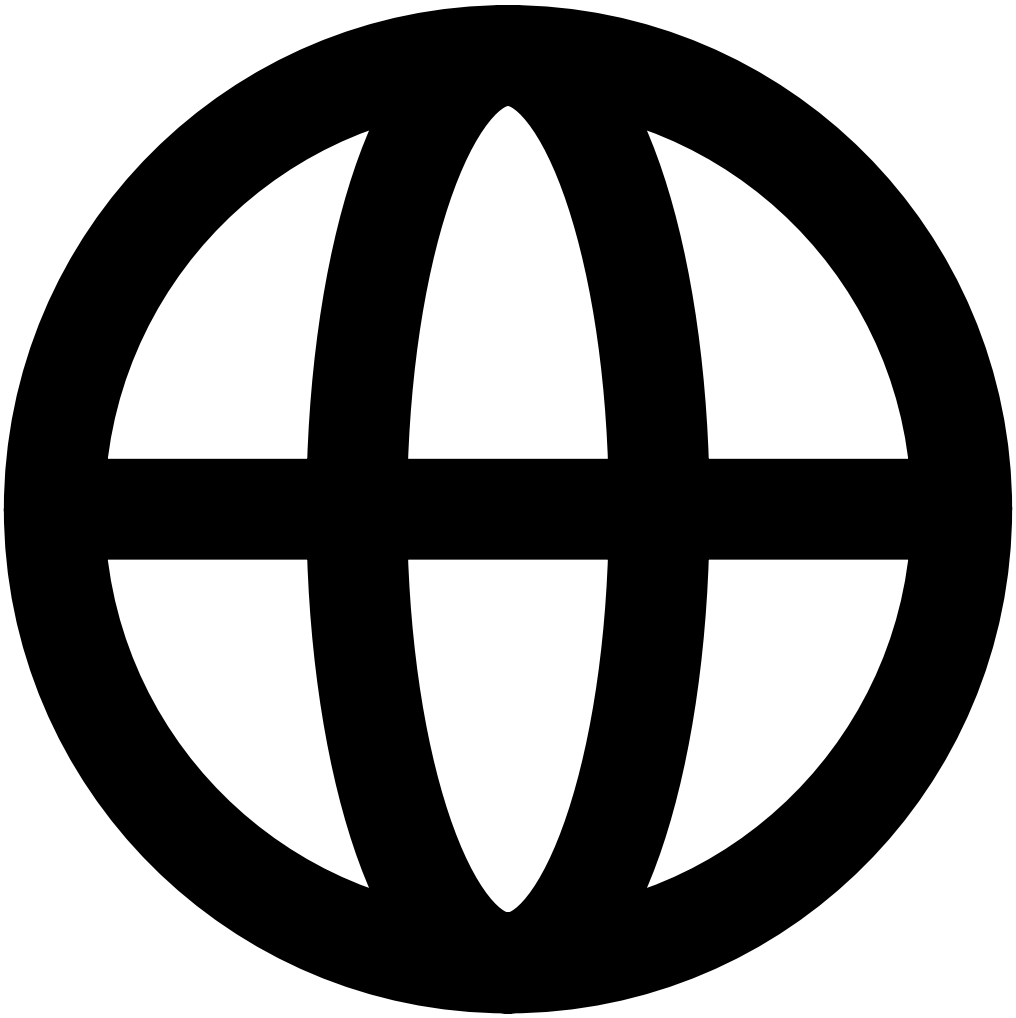
### Niveau 1 : Débutant en MLOps (1-5 data scientists)

Pour les équipes qui débutent en MLOps, la priorité est de mettre en place les **fondations** sans complexité excessive. La stack recommandée est : **MLflow** (experiment tracking + model registry), **DVC** (versioning des données et des modèles), **Git** (versioning du code) et un **serveur de déploiement simple** (Docker + API REST via MLflow serve ou BentoML). L'orchestration peut rester manuelle dans un premier temps (exécution de scripts via cron ou CI/CD basique). Cette stack se déploie en quelques heures, ne nécessite pas Kubernetes et permet de résoudre les problèmes les plus critiques : la reproductibilité des expériences et la traçabilité des modèles. Le coût d'infrastructure est minimal — une VM t3.medium (2 vCPU, 4 Go RAM) à 30 $/mois suffit pour héberger le tracking server MLflow et le backend PostgreSQL pour une petite équipe. L'investissement principal est le temps de formation des data scientists aux bonnes pratiques MLflow (environ une journée).



### Niveau 2 : Intermédiaire (5-20 ML engineers)

Les équipes intermédiaires ont besoin d'**automatiser les pipelines** et de gérer plusieurs modèles en production simultanément. Deux voies sont possibles. La **voie ZenML** : ZenML comme framework unificateur, avec MLflow pour le tracking, Airflow ou Prefect comme orchestrateur, et un model deployer au choix (Seldon, BentoML, KServe). Cette voie est recommandée si l'équipe souhaite rester **cloud-agnostique** et pouvoir changer d'orchestrateur ou de cloud provider sans réécrire les pipelines. La **voie MLflow-centric** : MLflow comme composant central, complété par Prefect ou Dagster pour l'orchestration, Great Expectations pour la validation des données et Evidently pour le monitoring. Cette voie est plus simple à mettre en oeuvre mais crée un couplage plus fort avec l'écosystème MLflow/Databricks. Dans les deux cas, l'introduction d'un **feature store** (Feast) et d'outils de monitoring (Evidently, Whylogs) devient nécessaire dès que plus de 3-5 modèles tournent en production simultanément.



### Niveau 3 : Avancé / Entreprise (20+ ML engineers)

Les grandes organisations avec des dizaines de ML engineers, des centaines de modèles en production et des exigences de sécurité et conformité strictes ont besoin de la puissance complète de **Kubeflow**. La stack recommandée est : **Kubeflow Pipelines** pour l'orchestration (avec Argo Workflows), **KServe** pour le serving (avec autoscaling et canary deployments), **MLflow** pour le tracking et le registry (déployé sur le cluster Kubernetes), **Feast** pour le feature store, **Katib** pour l'AutoML et l'optimisation des hyperparamètres, et **Istio** pour la sécurité réseau (mTLS, RBAC). Cette stack nécessite une équipe **platform engineering dédiée** de 2-5 personnes pour l'opérer et la maintenir. Le coût d'infrastructure est significatif — comptez 5 000 à 20 000 $/mois pour un cluster Kubernetes de production avec GPU — mais il est largement justifié par la productivité des équipes ML et la fiabilité des déploiements. Les managed services comme **Google Vertex AI** (basé sur Kubeflow), **AWS SageMaker** ou **Azure ML** offrent une alternative qui réduit la charge opérationnelle au prix d'un vendor lock-in plus important.

## Critères de choix transversaux

Au-delà de la taille de l'équipe, plusieurs critères transversaux doivent guider le choix. La **sécurité** : Kubeflow offre le multi-tenancy et le RBAC les plus matures via Kubernetes ; MLflow OSS nécessite des solutions tierces (nginx auth, reverse proxy) ; ZenML propose des RBAC via son offre cloud (ZenML Pro). La **conformité réglementaire** (RGPD, AI Act) : ZenML avec son Model Control Plane offre le lineage le plus complet pour les audits ; MLflow nécessite des scripts custom pour extraire la traçabilité. L'**écosystème cloud** : si vous êtes sur GCP, Vertex AI (basé sur Kubeflow) est un choix naturel ; sur AWS, SageMaker avec MLflow est la combinaison dominante ; sur Azure, Azure ML intègre nativement MLflow. La **dette technique existante** : si vos équipes utilisent déjà des notebooks Jupyter et des scripts Python simples, MLflow est le chemin de moindre résistance ; si elles opèrent déjà des microservices sur Kubernetes, Kubeflow s'intègre naturellement. Enfin, la **stratégie long terme** : ZenML offre la plus grande flexibilité future grâce à son architecture composable, permettant de remplacer n'importe quel

composant sans réécrire le code des pipelines — un avantage stratégique considérable dans un écosystème qui évolue rapidement. Pour approfondir, consultez Gouvernance LLM et Conformité : RGPD, AI Act et Auditabilité.

**Erreur fréquente :** Ne choisissez pas la plateforme MLOps la plus puissante, mais celle qui correspond à votre **niveau de maturité actuel**. Déployer Kubeflow dans une équipe de 3 data scientists qui n'ont pas encore d'experiment tracking est un anti-pattern classique qui mène à l'abandon. Commencez simple (MLflow), ajoutez de la complexité incrémentalement (ZenML pour l'abstraction, Kubeflow quand le scale l'exige), et investissez d'abord dans les **pratiques** (versioning, testing, monitoring) avant de complexifier l'**infrastructure**.

**Synthèse : MLflow** = meilleur point d'entrée, standard de facto pour tracking et registry. **Kubeflow** = puissance maximale pour les grandes organisations sur Kubernetes. **ZenML** = meilleur compromis composabilité/simplicité pour les équipes qui veulent rester flexibles. La combinaison **MLflow + ZenML + l'orchestrateur de votre choix** constitue probablement la stack open source la plus polyvalente et pérenne en 2026.

## Besoin d'un accompagnement expert ?

Nos consultants en cybersécurité et IA vous accompagnent dans vos projets. Devis personnalisé sous 24h.

## Références et ressources externes

- vLLM — Moteur d'inférence LLM haute performance
- llama.cpp — Inférence LLM optimisée en C/C++
- MLflow — Plateforme open source de gestion du cycle de vie ML
- Kubernetes Docs — Documentation officielle Kubernetes
- HuggingFace Docs — Documentation de référence pour les modèles de ML

Pour approfondir ce sujet, consultez notre outil open-source ai-prompt-injection-detector qui facilite la détection des injections de prompt.

**Sources et références :** ArXiv IA · Hugging Face Papers

Articles connexes

- Llama 4, Mistral Large, Gemma 3 : Comparatif LLM Open Source

> Points clés à retenir
>
> - Table des Matières
> - 1 L'Essor du MLOps : Pourquoi Industrialiser le Machine Learning
> - 2 MLflow : Le Standard de Fait
> - 3 Kubeflow : Le MLOps Cloud-Native
> - 4 ZenML et les Alternatives Émergentes
> - 5 Experiment Tracking et Model Registry

## FAQ

### Qu'est-ce que MLOps Open Source ?

Le concept de MLOps Open Source est détaillé dans les premières sections de cet article, qui couvrent les fondamentaux, les enjeux et le contexte opérationnel. Pour un accompagnement sur ce sujet, contactez nos experts.

### Pourquoi MLOps Open Source est-il important en cybersécurité ?

La compréhension de MLOps Open Source permet aux équipes de sécurité d'améliorer leur posture défensive. Les sections « Table des Matières » et « 1 L'Essor du MLOps : Pourquoi Industrialiser le Machine Learning » détaillent les raisons de cette importance. Pour un accompagnement sur ce sujet, contactez nos experts.

### Comment mettre en œuvre les recommandations de cet article ?

Les recommandations pratiques sont détaillées tout au long de l'article, avec des commandes, des outils et des méthodologies éprouvées. La section « Conclusion » fournit une synthèse actionnable. Pour un accompagnement sur ce sujet, contactez nos experts.

## Conclusion

Cet article a couvert les aspects essentiels de Table des Matières, 1 L'Essor du MLOps : Pourquoi Industrialiser le Machine Learning, 2 MLflow : Le Standard de Fait. La mise en pratique de ces recommandations permet de renforcer significativement la posture de securite de votre organisation.

**Ayi NEDJIMI Consultants** — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr