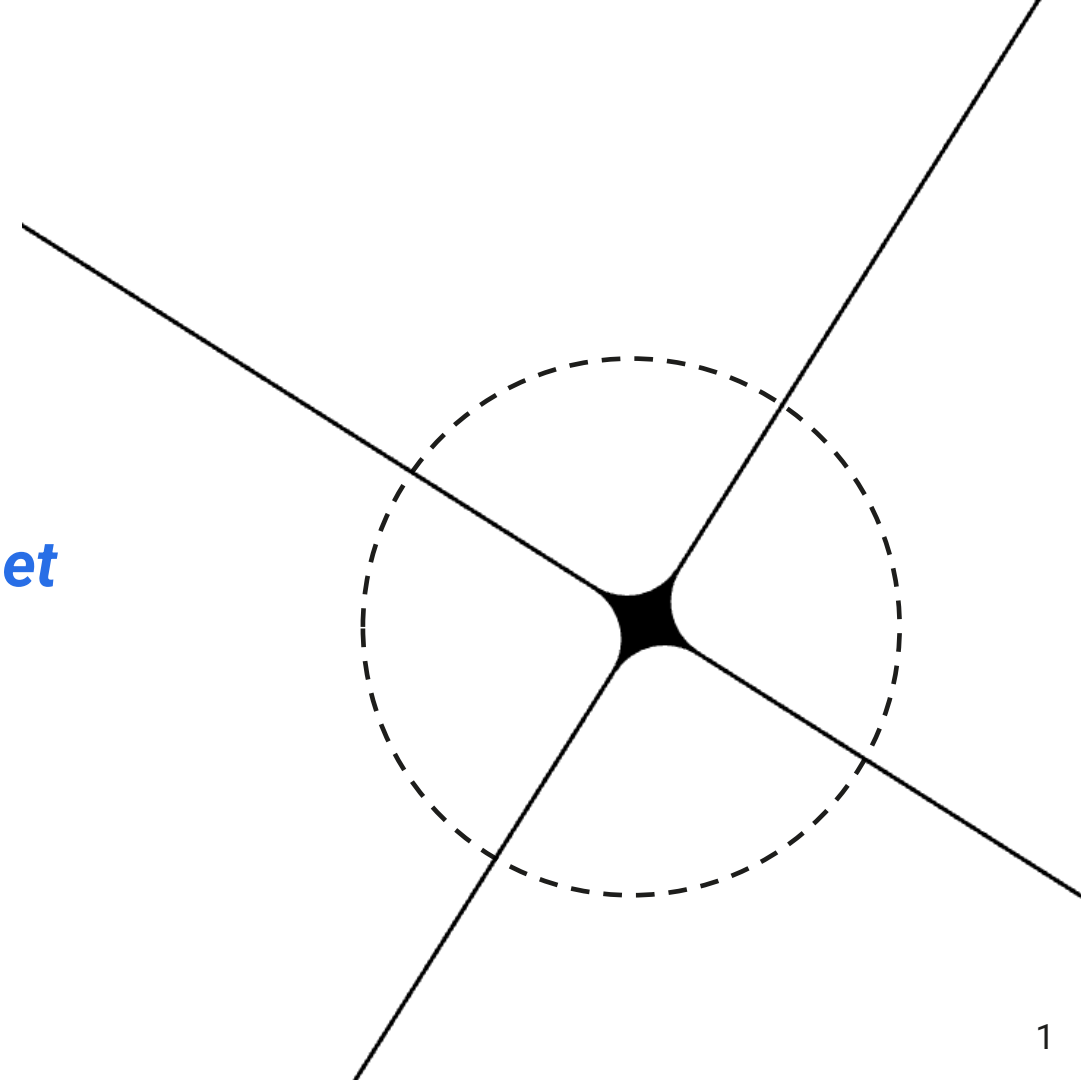
INSTITUT POLYTECHNIQUE DE PARIS

TELECOM SudParis

Monitoring, Observabilité et Détection du drift

Julien Romero

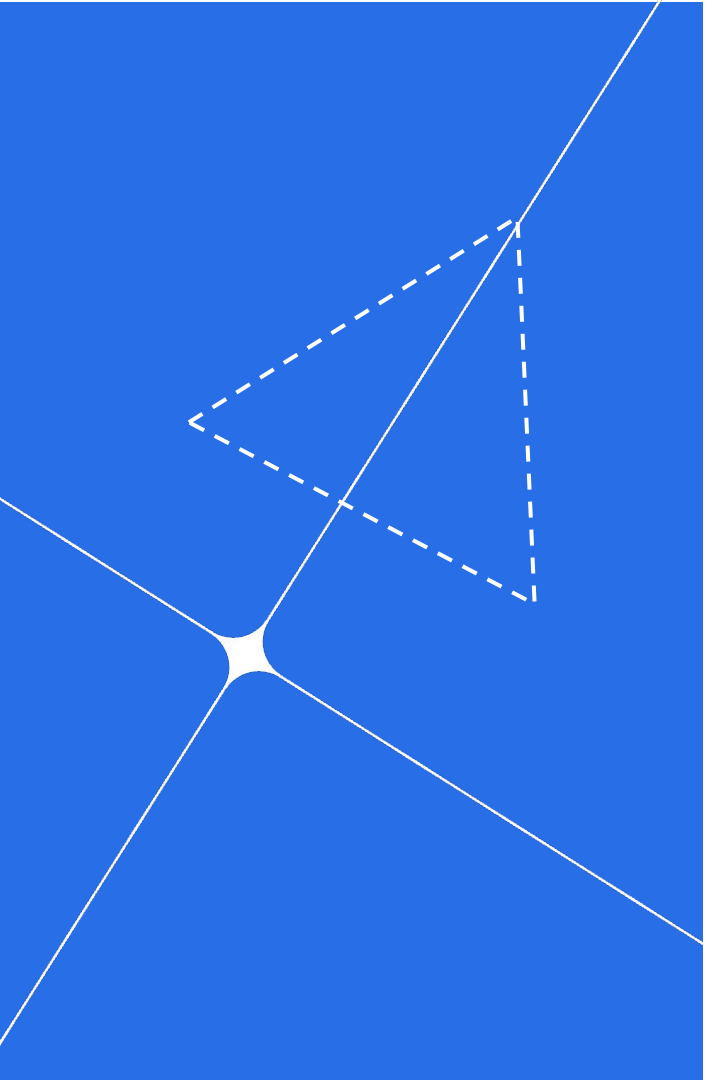


Failure Story (Failure First)

- Scénario réaliste
  - Le modèle fonctionnait le mois dernier
  - Aucun crash
  - Aucune exception
  - Aucune alerte
  - Les prédictions sont maintenant mauvaises
- Constat
  - Le système est vivant
  - Le monde a changé
  - Le modèle n’a pas été informé

Pourquoi c’est la partie la plus difficile du MLOps

- Échecs silencieux
  - Pas de stacktrace
  - Pas de bug visible
  - Dégradation progressive
- Feedback tardif
  - Labels disponibles avec retard
  - Impossible de valider immédiatement la qualité
- Problème système
  - Pas un problème d’algorithme
  - Pas un problème de code isolé
  - Interaction données × modèle × temps × infra



Rappels : La pipeline MLOps et où nous en sommes

ML System n’est pas uniquement le modèle

- Un système ML en production
  - Modèle ≠ cœur du système
  - Composantes indissociables :
    - Data
    - Code
    - Infrastructure
    - Temps
- Idée clé
  - Le temps est un paramètre caché
  - Ce qui était vrai hier peut être faux aujourd’hui

Pipeline MLOps complet

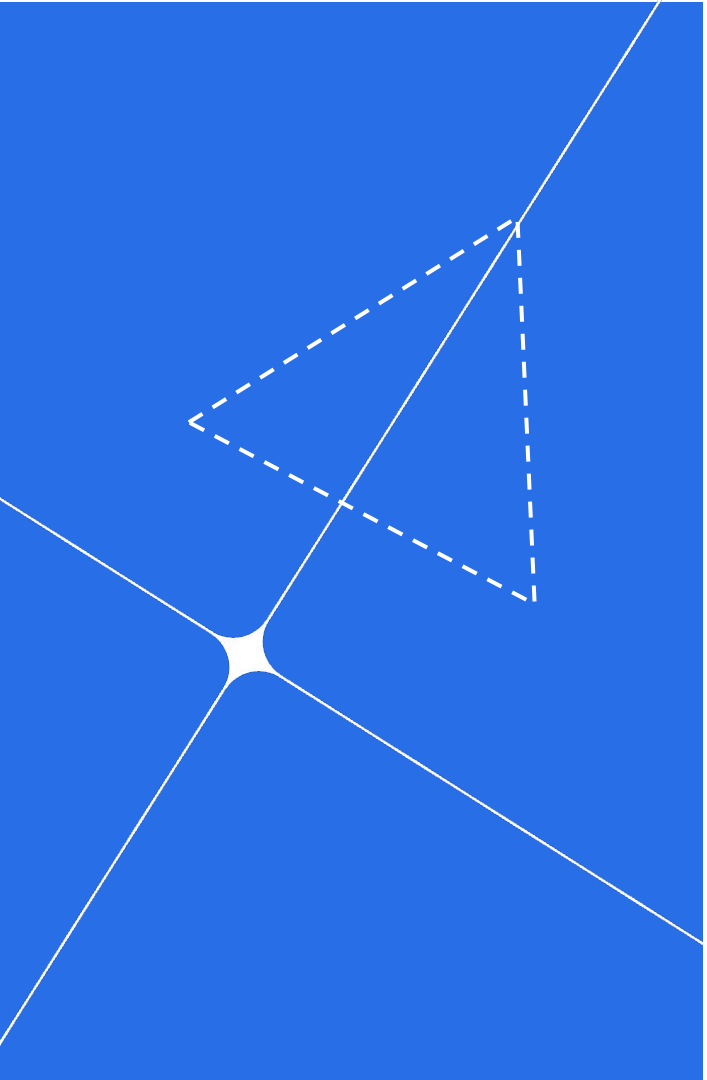
- Tout après le déploiement est souvent ignoré
- Pourtant, c’est là que les systèmes cassent

Où nous en sommes dans le cours

- Déjà construit
  - API déployée
  - Feature Store opérationnel
  - Modèles entraînés et servis
  - Pipelines reproductibles
- Manque critique
  - Pas de visibilité
  - Pas de feedback
  - Pas de signal d’alerte

“Production” : définition concrète

- Production, ici, signifie
  - Des utilisateurs appellent l’API
  - Les données changent en continu
  - Les décisions sont automatisées
  - Les erreurs ont un coût réel
- Conséquence
  - Ne pas observer le système = accepter le risque



Observabilité : les concepts clefs

Monitoring vs Observability

- **Monitoring**
  - Questions prédéfinies
  - Seuils fixes
  - Réponses connues à l'avance
  - "Est-ce que X dépasse Y ?"
- **Observabilité**
  - Capacité à poser de nouvelles questions
  - Compréhension du système via ses signaux
  - Investigation a posteriori
  - "Pourquoi ça a changé ?"
- Idée clé
  - Monitoring est inclu dans l'observabilité

Observabilité dans les systèmes ML

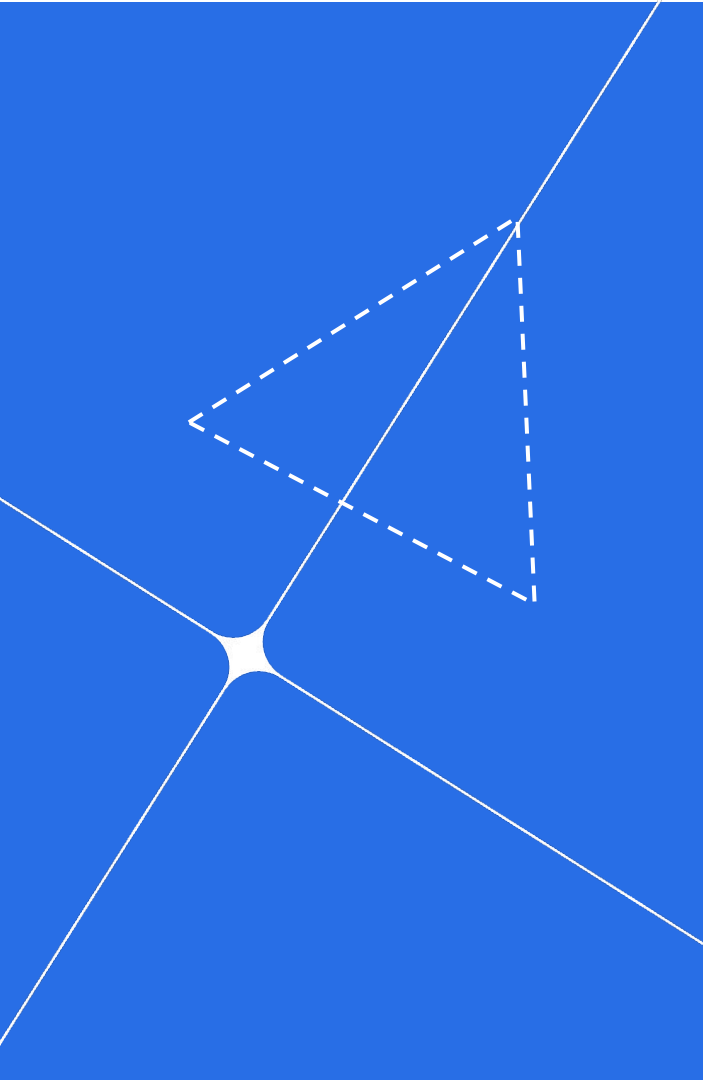
- Pourquoi le monitoring logiciel classique ne suffit pas
  - Le code peut être correct
  - L'infrastructure peut être saine
  - L'API peut répondre normalement
- Mais
  - Les données changent
  - Les distributions évoluent
  - Le modèle se dégrade sans erreur explicite
- Nouveaux modes de panne
  - Data drift
  - Prediction drift
  - Concept drift (souvent invisible)

Les 3 piliers de l'observabilité

- **Métriques**
  - Valeurs numériques dans le temps
  - Agrégables, requêtables
  - Base de l'automatisation
- **Logs**
  - Événements discrets
  - Contexte, erreurs, décisions
  - Essentiels pour le debug
- **Traces**
  - Propagation d'une requête
  - Latence distribuée
  - Hors périmètre ici

Périmètre de ce cours

- Ce que nous allons utiliser
  - Métriques avec Prometheus
  - Dashboards avec Grafana
  - Drift avec Evidently
- Logs
  - API FastAPI (erreurs, requêtes)
  - Prefect (exécution des flows)
  - Utilisation ciblée, pas de log engineering avancé
- Ce que nous ne couvrons pas
  - Tracing distribué
  - APM complet



Ce qui peut mal se passer

Modèle de monitoring en couches

- Système ML = empilement de couches
  1. **Infrastructure**
  2. **API / Application**
  3. **Données**
  4. **Modèle**
- Principe
  a. Une couche peut être saine
  b. Une autre peut être défaillante
  c. Les symptômes ne sont pas toujours visibles au même niveau

Défaillances d'infrastructure

- Exemples
  - CPU saturé
  - Mémoire insuffisante
  - I/O disque lent
  - Problèmes réseau
- Caractéristiques
  - Non spécifiques au ML
  - Mais impactent directement l'inférence
  - Souvent déjà monitorés par l'infra
- Risque
  - Latence artificielle
  - Timeouts
  - Effets en cascade

Défaillances au niveau de l'API

- Signaux typiques
  - Pics de latence
  - Augmentation du taux d'erreur (4xx / 5xx)
  - Variation brutale du trafic
- Causes possibles
  - Charge inattendue
  - Mauvaise gestion des cas limites
  - Dépendances externes lentes
- Point clé
  - Une API "up" peut être fonctionnellement dégradée

Défaillances au niveau des données

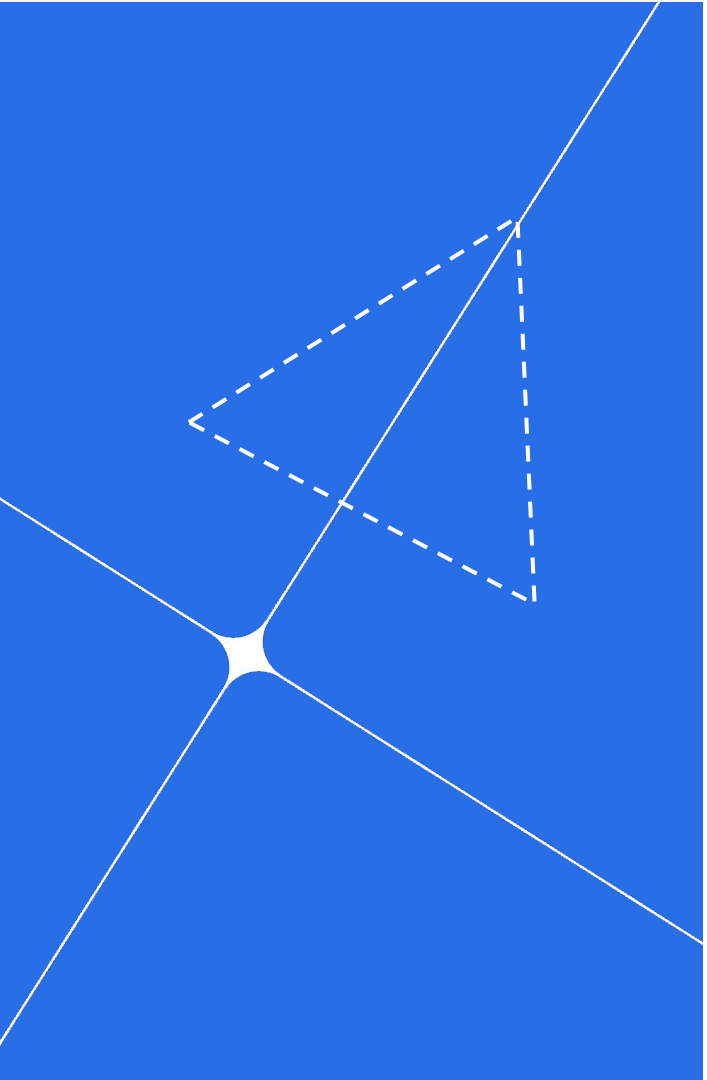
- Exemples courants
  - Valeurs manquantes
  - Changements de distribution
  - Colonnes nouvelles ou supprimées
  - Types incohérents
- Problème majeur
  - Peu ou pas d’erreurs visibles
  - Effets silencieux sur le modèle
- Impact
  - Features incorrectes
  - Décalage training / inference

Défaillances au niveau du modèle

- Formes de dérive
  - **Prediction drift**
    - Les sorties changent dans le temps
  - **Target drift**
    - La distribution des labels évolue
    - Observée avec retard
  - **Concept drift**
    - Relation entre les entrées et les sorties change
    - Très difficile à détecter directement

Pourquoi monitorer la performance est difficile

- Contraintes structurelles
  - Labels disponibles tardivement
  - Ground truth bruitée ou incomplète
  - Pas de feedback instantané
- Conséquence
  - Impossible de s’appuyer uniquement sur l’accuracy
  - Besoin de signaux indirects
  - La dérive/drift devient un proxy de dégradation



Métriques et Prometheus

Pourquoi commencer par les métriques

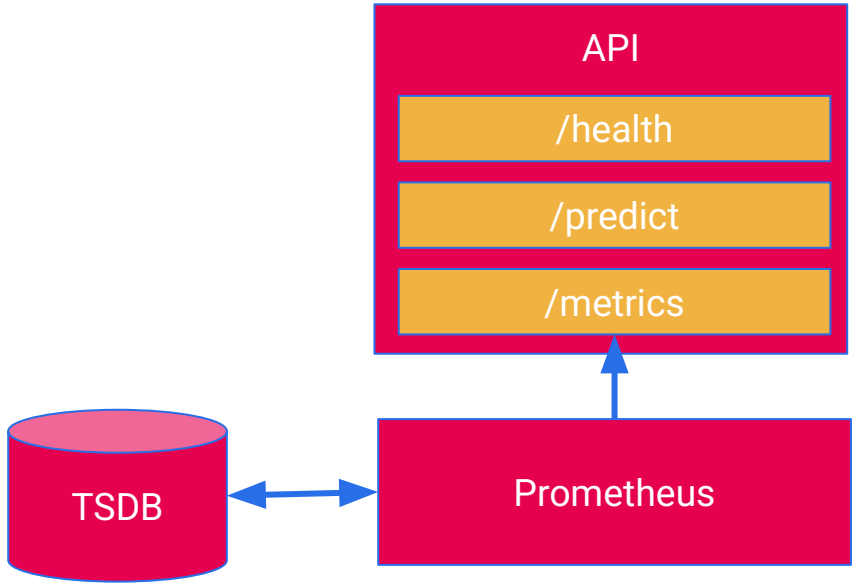
- Avantages
  - Peu coûteuses
  - Collecte continue
  - Faciles à automatiser
- Rôle
  - Premier signal de dégradation
  - Base de toute observabilité
  - Déclencheur d’investigation

Modèle mental de Prometheus

- Prometheus est une application qui va nous permettre de suivre et gérer des métriques
- Principes clés
  - Pull-based
    - Prometheus vient chercher les métriques (et pas l'inverse)
  - Time series
    - Valeurs indexées par le temps
  - Targets stateless
    - L'application n'a pas de mémoire Prometheus
- Conséquence
  - Simplicité côté application
  - Robustesse du système

Architecture Prometheus

- Flux conceptuel
  - L'API expose un endpoint /metrics
  - Prometheus scrape périodiquement
  - Les données sont stockées dans une TSDB
    - (Time Serie Database)
- Caractéristiques
  - Historique conservé
  - Pas de push depuis l'application
  - Tolérant aux redémarrages



Types de métriques

- Counter
  - Compteur monotone
  - Ex : nombre de requêtes
- Gauge
  - Valeur instantanée
  - Ex : mémoire utilisée
- Histogram
  - Distribution de valeurs
  - Ex : latence des requêtes
- Summary
  - Similarité avec histogram
  - Moins utilisé

Métriques typiques d'une API ML

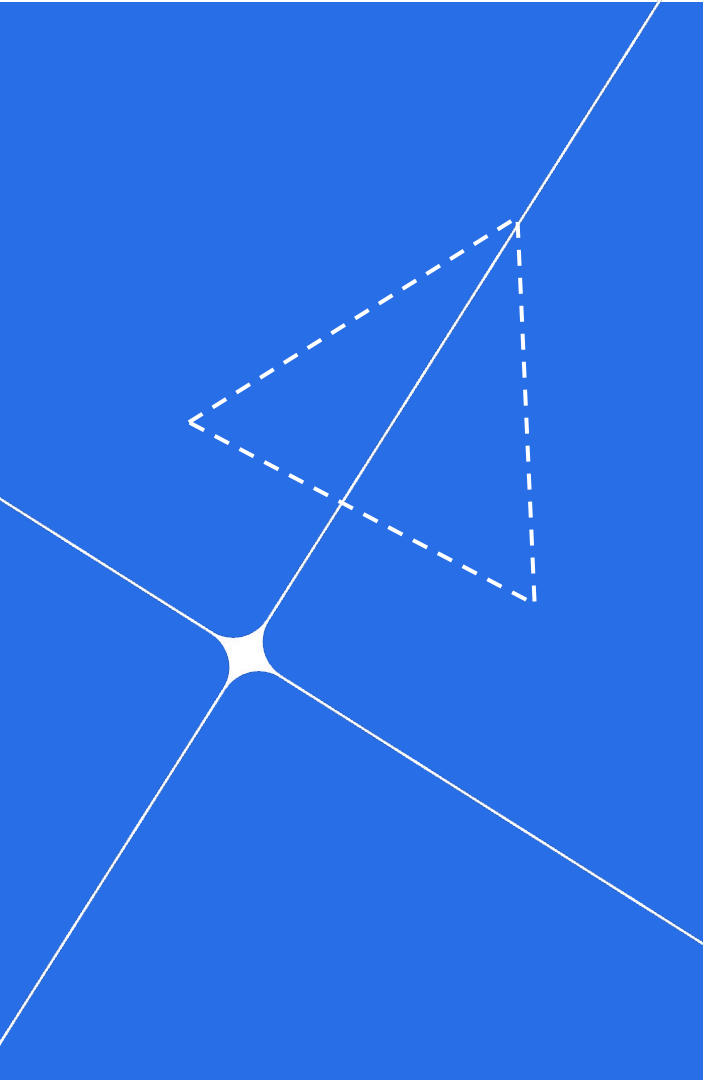
- Volume
  - Nombre de requêtes
  - Requêtes par seconde
- Latence
  - Temps de réponse
  - p50 / p95 / p99 (percentile)
- Erreurs
  - Codes 4xx / 5xx
  - Taux d'échec
- Idée clé
  - Pas encore de métriques "qualité modèle"

Labels Prometheus : puissance et danger

- Les labels dans Prometheus permettent caractériser la chose en train d'être mesurée
  - Ex: api_request_duration_seconds peut avoir une valeur par stage (extract, transform, load)
- Utilité
  - Ajouter des dimensions
    - path
    - method
    - status
- Risque majeur
  - Explosion de cardinalité
  - Coût mémoire
  - Requêtes lentes
- Règle pratique
  - Labels finis
  - Pas de valeurs dynamiques (user_id, timestamps)

Lire du PromQL

- PromQL = langage de requête de Prometheus
- Fonctions essentielles
  - rate(counter[window])
    - Dérivée temporelle
  - histogram_quantile(q, ...)
    - Estimation des percentiles
- Objectif pédagogique
  - Comprendre ce que montre un dashboard
  - Adapter légèrement une requête existante
  - Pas d’écriture complexe



Instrumentaliser une API ML

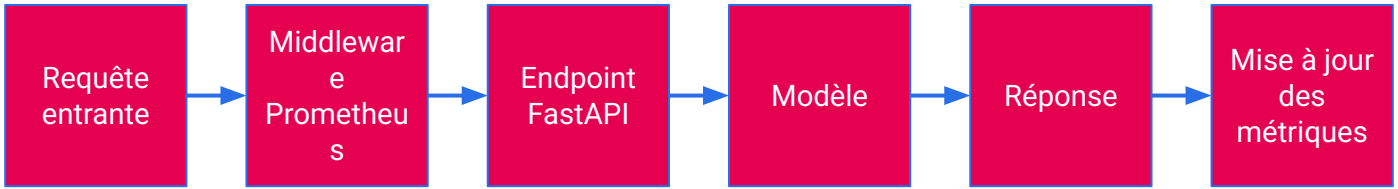
Où vivent les métriques

- Localisation
  - Directement dans le service d’inférence
  - Au plus proche de la requête réelle
- Pourquoi ici
  - Vision fidèle de la production
  - Pas de proxy
  - Pas d’approximation
- Principe
  - Ce qui n’est pas mesuré ici n’existe pas

Instrumentation FastAPI : concept

- Mécanisme
  - Middleware interceptant les requêtes
  - Instrumentation transversale
  - Aucune logique métier modifiée
- Mesures automatiques
  - Compteurs de requêtes
  - Latence par requête
  - Codes de réponse
- Avantage
  - Faible effort
  - Faible risque
  - Couverture globale

Cycle de vie d'une requête

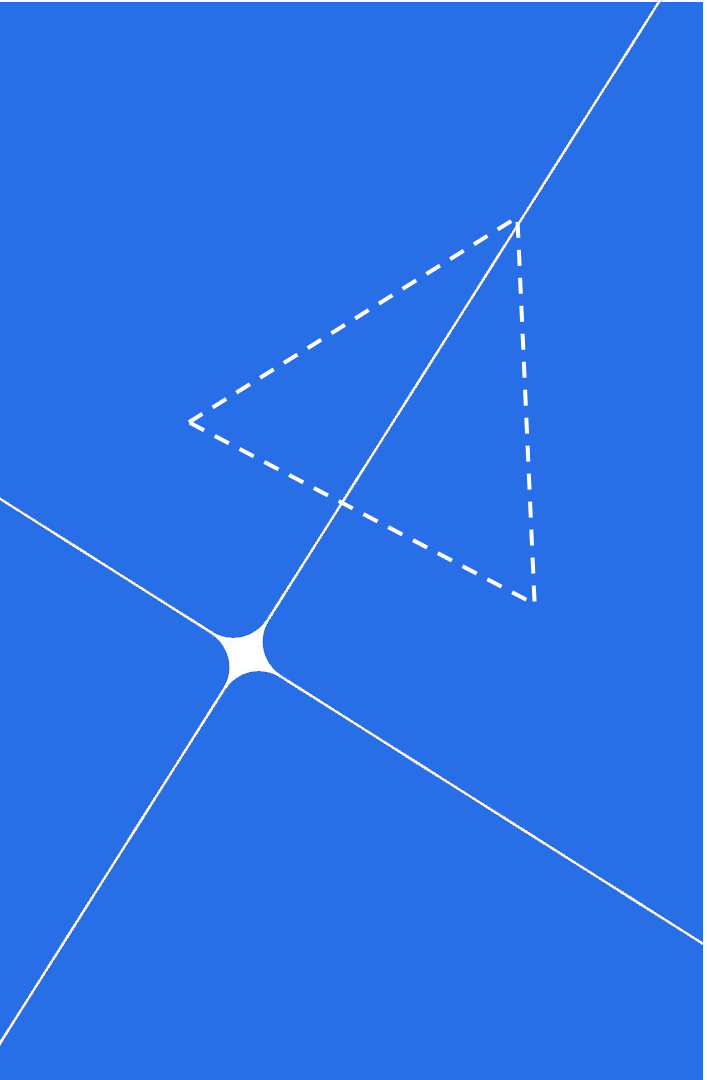


Idée clé

- La mesure est orthogonale au code ML
- Le modèle reste un composant pur

Ce que nous ne mesurons pas encore

- Absent volontairement
  - Qualité du modèle
  - Performance métier
  - Valeur business
- Pourquoi
  - Labels retardés
  - Données partielles
  - Nécessite une autre couche de monitoring
- Transition
  - Besoin de signaux indirects => drift



Grafana

Pourquoi les dashboards comptent

- Un dashboard est une interface permettant de visualiser un ensemble de données facilement
- Raison humaine
  - Les humains raisonnent visuellement
  - Les tendances sont plus parlantes que des valeurs brutes
- En production
  - Un point isolé ne dit rien
  - Une évolution dans le temps révèle un problème
- Objectif
  - Comprendre l’état du système en quelques secondes

Modèle mental de Grafana

- Grafana est une suite logicielle pour créer des Dashboards
- Briques fondamentales
  - Data source
    - Origine des métriques (Prometheus)
  - Panel
    - Une visualisation = une question
  - Dashboard
    - Ensemble cohérent de panels
- Principe
  - Grafana ne calcule rien
  - Il visualise et agrège ce qui existe déjà

Point de vue opérateur

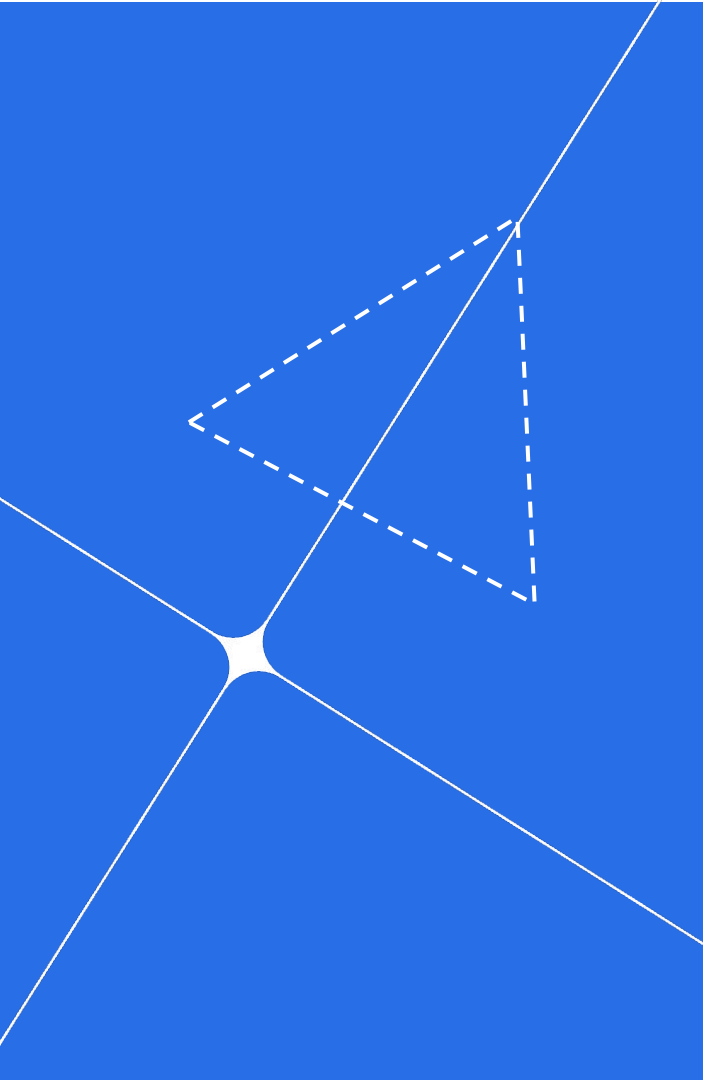
- Questions typiques
  - “Le système est-il sain ?”
  - “Quelque chose a-t-il changé récemment ?”
- Ce qu’on ne demande pas ici
  - “Pourquoi exactement ?”
  - “Quel feature est responsable ?”
- Rôle du dashboard
  - Détection rapide
  - Pas diagnostic complet

Dashboards essentiels

- Vue trafic
  - Nombre de requêtes
  - Requêtes par seconde
  - Variations anormales
- Vue latence
  - Temps de réponse
  - Percentiles
- Vue erreurs
  - Taux d’erreurs
  - Codes de statut

Lire un histogramme de latence

- Percentiles
  - p50 : cas “normal”
  - p95 : utilisateurs pénalisés
  - p99 : pires cas
- Idée clé
  - La moyenne cache les problèmes
  - La latence de queue (tail latency) dégrade l’expérience
- Signal faible mais critique
  - p95/p99 augmentent avant les erreurs visibles



Des métriques au drift

Les métriques sont nécessaires mais insuffisantes

- Situation fréquente
  - L'API répond
  - La latence est stable
  - Le taux d'erreur est faible
- Mais
  - Les prédictions sont mauvaises
  - Les décisions métier sont dégradées
- Conclusion
  - Un système peut être **techniquement sain**
  - Et **fonctionnellement faux**

Qu’est-ce que le drift ?

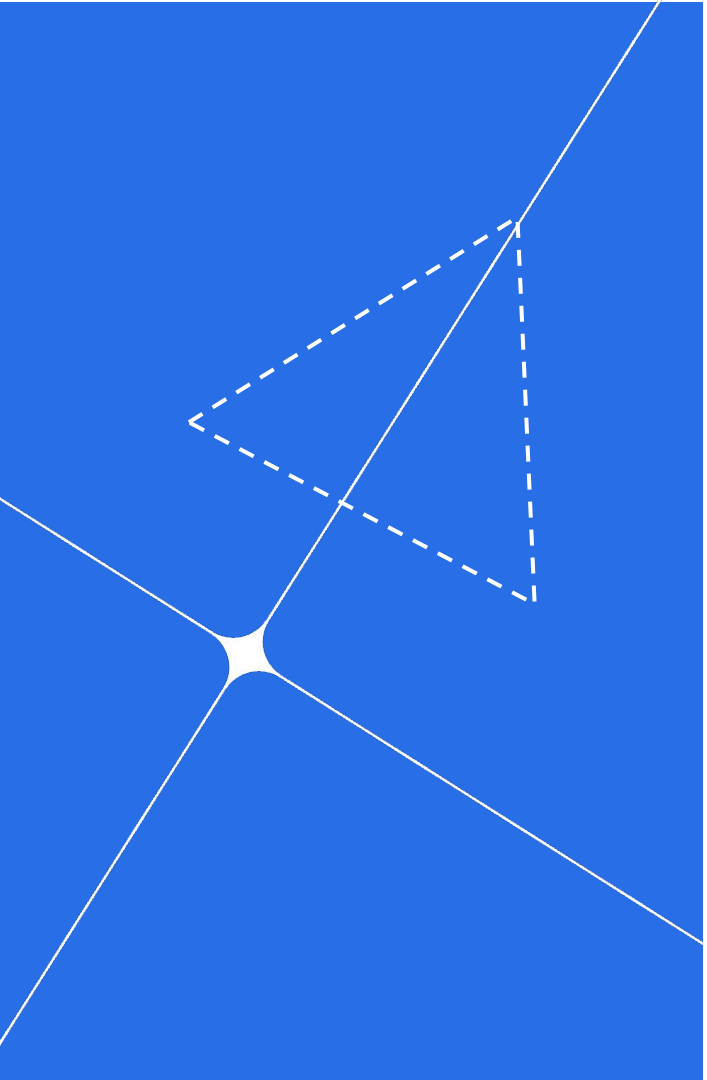
- Idée centrale
  - Le monde change
  - Les données évoluent
  - Les hypothèses du modèle deviennent invalides
- Conséquence
  - Le modèle ne généralise plus
  - Les performances chutent progressivement
- Caractère inévitable
  - Le drift n’est pas une exception
  - C’est une propriété des systèmes réels

Types de drift

- **Covariate drift**
  - Changement de distribution des features d'entrée
  - Ex : baisse du nombre de sessions après un changement produit
- **Prior probability drift**
  - Changement de la fréquence des labels
  - Ex : plus de churn après une hausse de prix
- **Concept drift**
  - Changement de la relation entre les entrées et les sorties
  - Ex : les mêmes signaux ne prédisent plus le churn

Clarification importante

- Point clé
  - Le concept drift n’est pas directement observable
- Pourquoi
  - Il nécessite des labels
  - Les labels arrivent tard
  - La causalité est difficile à établir
- En pratique
  - On observe des signaux indirects
  - Drift des features
  - Drift des prédictions
  - Drift du target (avec retard)



Détection du drift avec Evidently

Pourquoi Evidently

- Besoin
  - Comparer des jeux de données dans le temps
  - Quantifier les changements
  - Avoir des signaux exploitables
- Evidently fournit
  - Comparaison reference vs current
  - Tests statistiques automatiques
  - Métriques orientées ML, pas génériques
- Positionnement
  - Outil de diagnostic
  - Pas un outil de prédiction

Données de référence vs données courantes

- Cadre temporel
  - Données historiques stabilisées
  - Données récentes à analyser
- Exemple
  - Mois N : référence
  - Mois N+1 : courant
- Principe clé
  - Toujours comparer des périodes cohérentes
  - Même schéma, même définition de features

Ce que détecte réellement Evidently

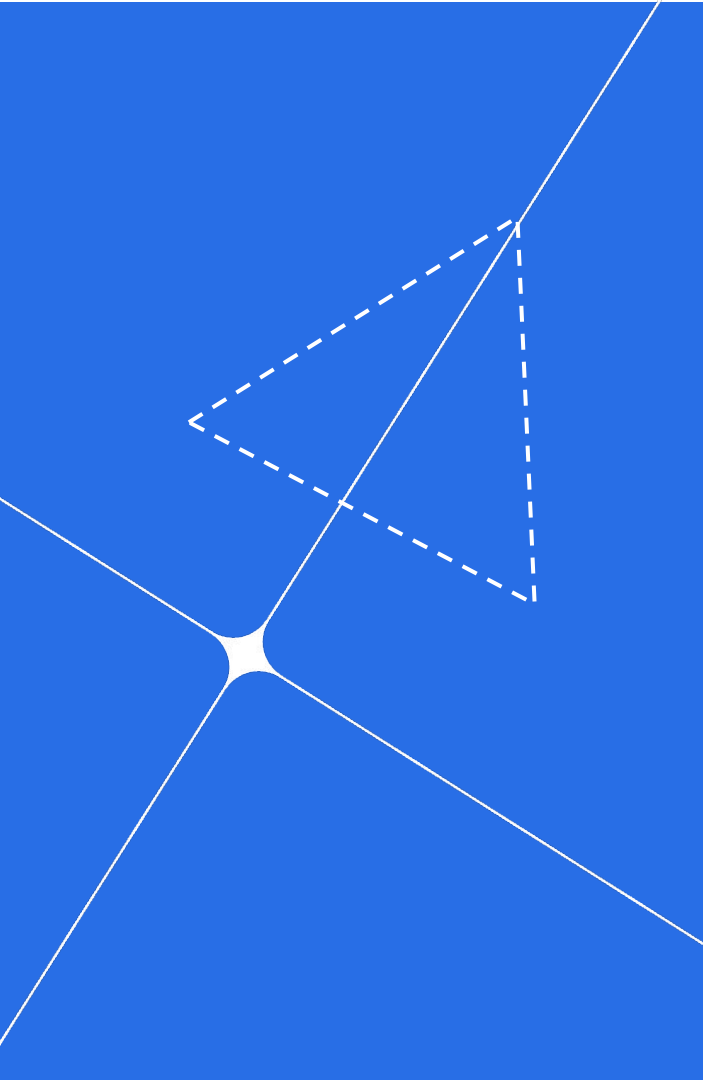
- Observables directs
  - Changement de distribution des features
  - Anomalies statistiques
- Si les labels sont disponibles
  - Changement de distribution du target
  - Proxy de dégradation de performance
- Non détecté directement
  - Concept drift réel
  - Causalité

Métriques de drift

- Niveau feature
  - Drift par colonne
  - Tests adaptés au type (numérique / catégoriel)
- Niveau dataset
  - Drift share
    - Proportion de features en drift
- Utilisation
  - Résumer un état complexe en un signal exploitable

Les seuils sont des heuristiques

- Règle arbitraire
  - 20% de features en drift => alerte / action
- Réalité industrielle
  - Dépend du domaine
  - Dépend du coût d’erreur
  - Dépend de la stabilité attendue
- Message clé
  - Les seuils ne sont pas universels
  - Ce sont des décisions système, pas statistiques



Le drift comme signal de décision

Drift n’entraîne pas forcément un réentraînement automatique

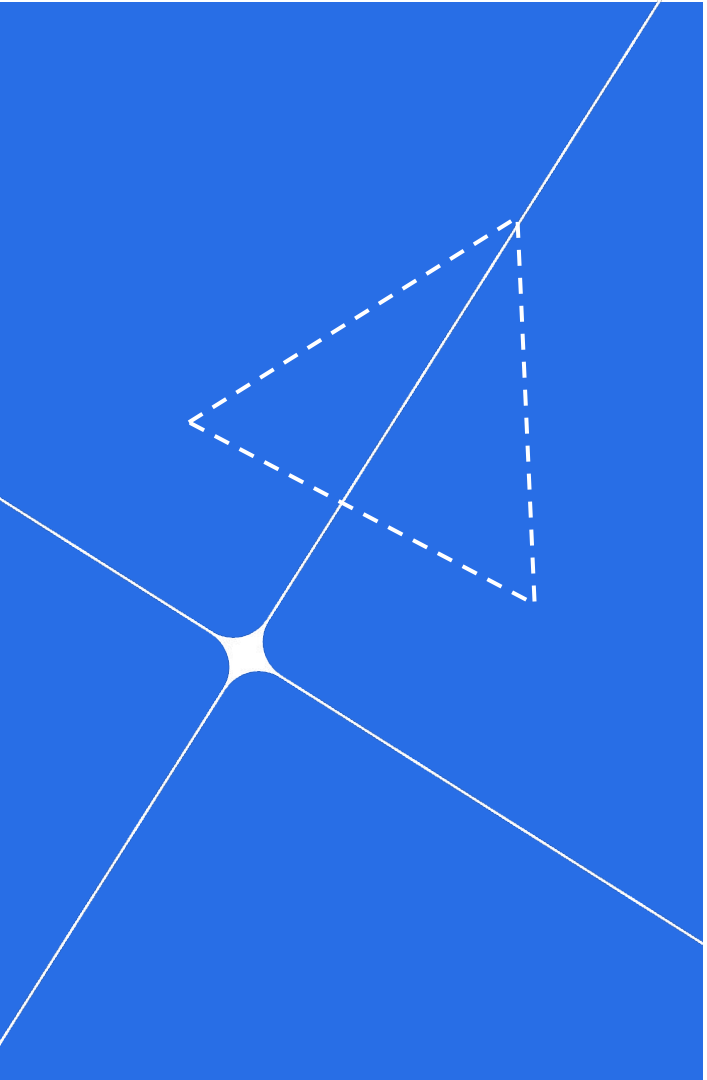
- Clarification essentielle
  - Le drift n’est pas une action
  - Le drift est un signal
- Risque naïf
  - Retrain à chaque fluctuation
  - Sur-automatisation
  - Instabilité du système
- Principe MLOps
  - Observer d’abord
  - Décider ensuite

Logique de décision basée sur le drift

- Cas nominal
  - Drift faible
  - Variations attendues
  - => Aucune action
- Cas suspect
  - Drift significatif
  - Signal statistique fort
  - => Investigation ou réentraînement
- Idée clé
  - Le drift déclenche une décision, pas une pipeline aveugle

Point de décision formel

- Position dans le système
  - Entre monitoring et training
  - Barrière de contrôle
- Rôle
  - Éviter les réentraînements inutiles
  - Centraliser la logique métier
  - Rendre le système gouvernable
- Concept
  - Le réentraînement est une conséquence contrôlée
  - Pas un effet de bord implicite



Automatisation avec Prefect

Pourquoi la planification est indispensable

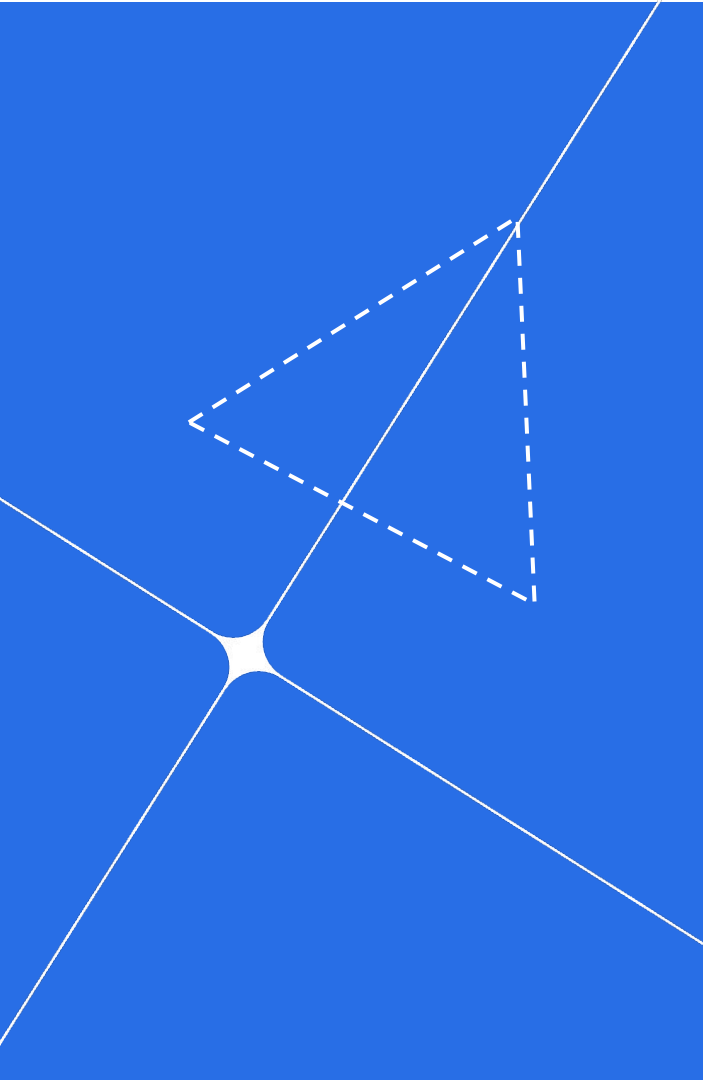
- Réalité opérationnelle
  - Le drift apparaît progressivement
  - Pas forcément visible en temps réel
  - Aucun événement brutal
- Limite humaine
  - Les dashboards ne sont pas consultés en continu
  - L’attention décroît avec le temps
- Conclusion
  - L’observation doit être automatisée
  - La vigilance humaine ne suffit pas

Prefect comme plan de contrôle

- Rôle de Prefect
  - Orchestration des pipelines
  - Planification temporelle
  - Exécution fiable et traçable
- Dans ce contexte
  - Lancer régulièrement des analyses de drift
  - Centraliser les décisions
  - Observer l’exécution (logs, états)
- Idée clé
  - Prefect devient le chef d’orchestre du système ML

Pipeline conceptuel de détection de drift

- Étapes
  1. Charger les snapshots temporels
  2. Calculer les métriques de drift
  3. Journaliser les résultats
  4. Appliquer la logique de décision
- Caractéristiques
  - Déterministe
  - Rejouable
  - Observable



Mettre tout ensemble

Boucle complète d'observabilité

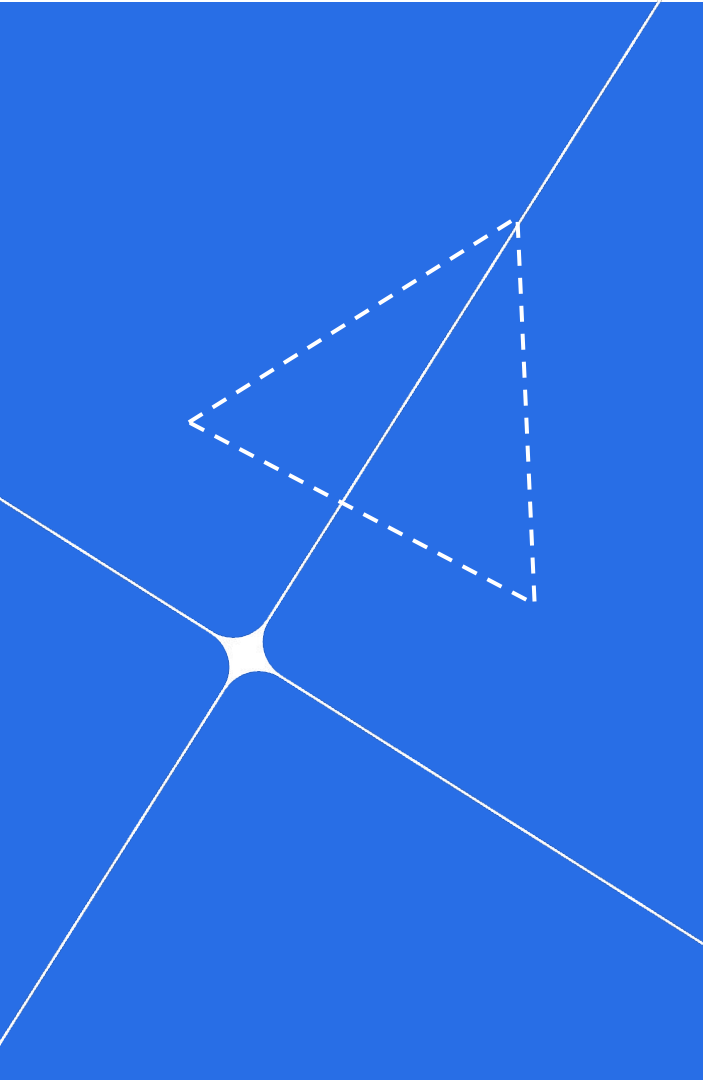
- Chaîne opérationnelle
  - Métriques API en continu
  - Visualisation via dashboards
  - Détection de drift périodique
  - Prise de décision formelle
- Lecture du système
  - Le modèle n'est plus aveugle
  - Le système devient interprétable dans le temps

Ce que nous ne résolvons pas encore

- Limites assumées
  - Performance réelle du modèle
  - Mesure directe de la qualité prédictive
  - Compréhension causale des changements
- Pourquoi
  - Labels retardés
  - Environnements complexes
  - Décisions métier non observables directement
- Transition
  - Ces limites motivent la suite du cours

Messages clés à retenir

- Production ML
  - Se dégrade sans bruit
  - Ne casse pas proprement
  - Évolue avec le monde réel
- Observabilité
  - Non optionnelle
  - Condition de survie du système
- Drift
  - N’est pas un bug
  - N’est pas une erreur
  - Est un signal à interpréter



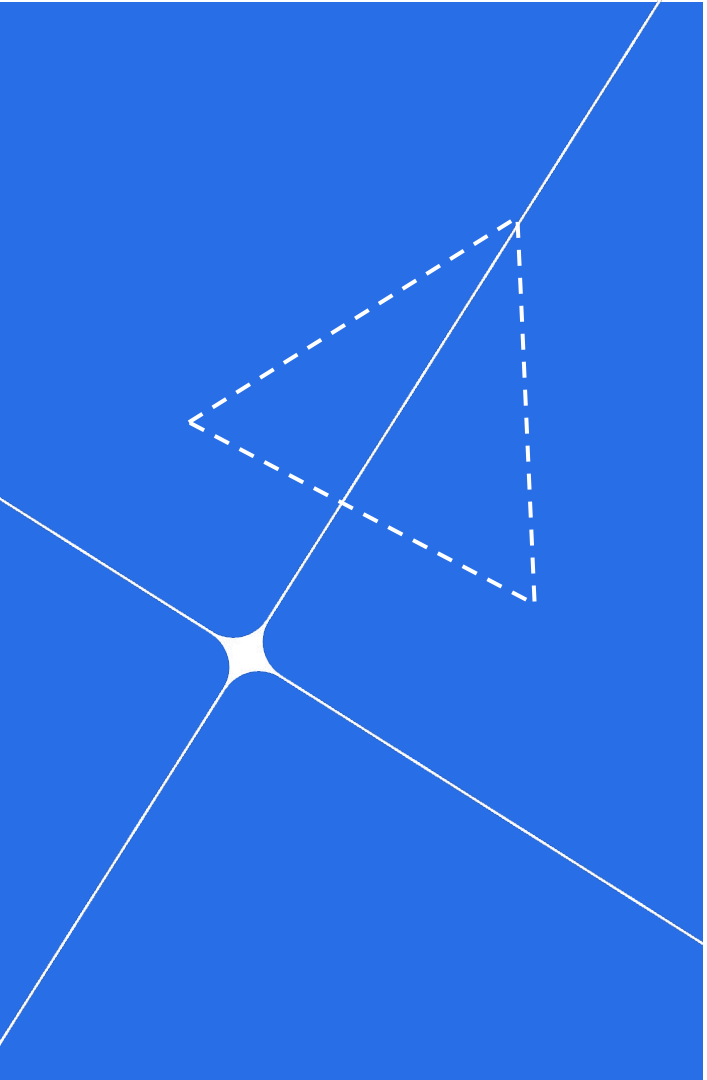
Lab 5

Ce que vous allez construire dans le TP

- Composants concrets
  - API instrumentée avec métriques Prometheus
  - Dashboards Grafana en temps réel
  - Pipeline de détection de drift automatisé
- Objectif
  - Rendre le système observable
  - Passer de l’intuition aux signaux mesurés

Ce que cela permet à long terme

- Automatisation sûre
  - Déclenchements contrôlés
  - Décisions traçables
- Réentraînement maîtrisé
  - Basé sur des signaux observés
  - Pas sur des impressions
- Préparation de la suite
  - CI/CD des modèles
  - Promotion automatique
  - Gouvernance (Lecture 6)



En route vers le TP