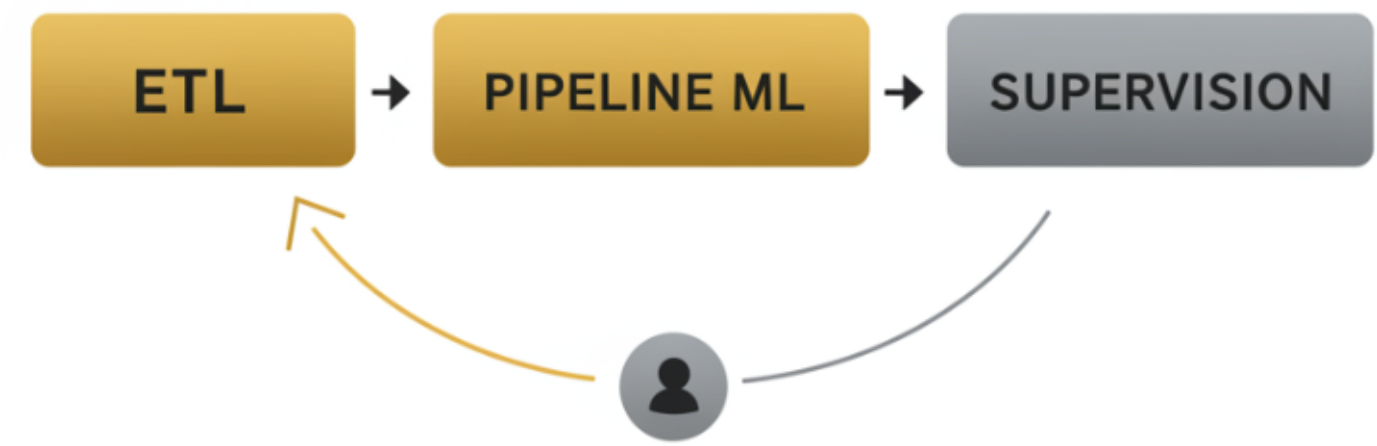
🏆 4. Exploitation

Ce chapitre explique comment la solution PariVision est exploité. Cela sert également de notice d'utilisation.



Pipeline ETL

La récupération et le traitement des données est ordonnées via Jenkins. Chaque requête permet de collecter un certain nombre documents pour alimenter un Lac de données afin d'avoir un contrôle granulaire sur la quantité de données ingéré par le modèle.

Finalement, est effectué la transformation et le stockage des données dans un entrepôt d'exploitation.

Pipeline ML

Toujours ordonnés via Jenkins, la Pipeline complète automatisé entraîne les actions suivantes :

1. Génère un rapport de surveillance de dérive des données
2. Entraîne un nouveau modèle sur les données fraîches et injecte toutes les métriques et artefacts sur un serveur de tracking MLFlow
3. Effectuer une comparaison entre tous les modèles entraînés afin de déterminer un modèle "champion", en ce focalisant sur les scores de performance
4. Déploie un service conteneurisé qui charge le modèle champion et le met à disposition pour effectuer des predictions via un service API
5. Déploie un service conteneurisé mettant à disposition une application pour l'utilisateur final

Supervision

Prometheus et Grafana

La supervision du projet PariVision est effectuée via Prometheus et Grafana.

Le service d'Alert Manager de Prometheus est configuré pour déclencher les alertes sur un salon SLACK dédié au projet.

Les détails de ces implémentations sont détaillés dans l'architecture ci-dessous :

```
monitoring
├── alertmanager
│   └── alertmanager.yml -> Configuration de Alter Manager (Slack)
├── evidently
│   └── main.py -> Génère un rapport de surveillance sur la dérive des données
├── grafana
│   └── provisioning -> La configuration de Grafana est entièrement provisionné
via GIT
│       ├── dashboards -> Les différents Dashbord (json) utile au projet
│       └── datasources -> Les sources de données (Prometheus)
└── prometheus
   ├── prometheus.yml -> Configuration de Prometheus (Maintenances appliqués via
Jenkins)
   └── rules -> Les règles de Prometheus
       ├── alerting.rules -> Règles d'alerte
       └── recording.rules -> Règles d'enregistrement des métriques
```

Slack

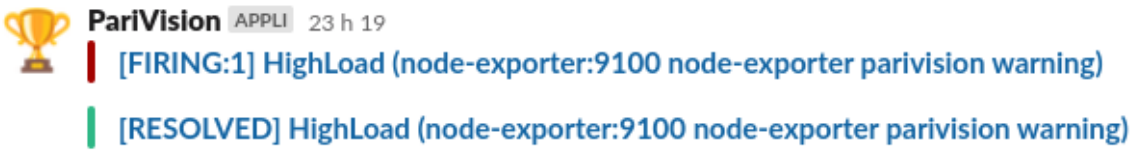
Une application sur Slack nous à été fournit par notre mentor, cette dernière est configuré avec AlertManager nous permet de recevoir les notifications d'alert de Prometheus.

Configuration de l'application : https://dst-dec.slack.com/marketplace/A0F7XDUAZ-webhooks-entrants

Appel Webhooks :

```
curl -X POST --data-urlencode "payload={\"channel\":
\"#dec24cmlops_paris_sportif\", \"username\": \"PariVision\", \"text\": \"TEST
ALERT PROMETHEUS\", \"icon_emoji\": \":trophy:\"}"
https://hooks.slack.com/services/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
```

Java



Evidently

La surveillance de la qualité des données utilisée pour l'entraînement est effectuée via Evidently afin de surveiller la dérive des données d'entrées du modèle.

PariVision-Exporter

Un exporter Prometheus sur-mesure pour le modèle de Machine Learning à été développé, afin de surveiller les métriques du modèle utilisés, comme le temps de prédiction, le nombre de prédiction effectués, etc …