Catalogue de données et lineage de bout en bout

Déploiement d'une plateforme de catalogage open source, alimentée depuis quatre familles de sources hétérogènes, avec des connecteurs développés pour celles que l'outil ne couvre pas nativement.

Résultat : analyse d'impact amont et aval disponible sur toute la chaîne, de la table source jusqu'au rapport, avec lineage colonne à colonne sur le périmètre relationnel.

Fiche de méthode associée : `../../docs/09-data-catalog-lineage.md`

Architecture

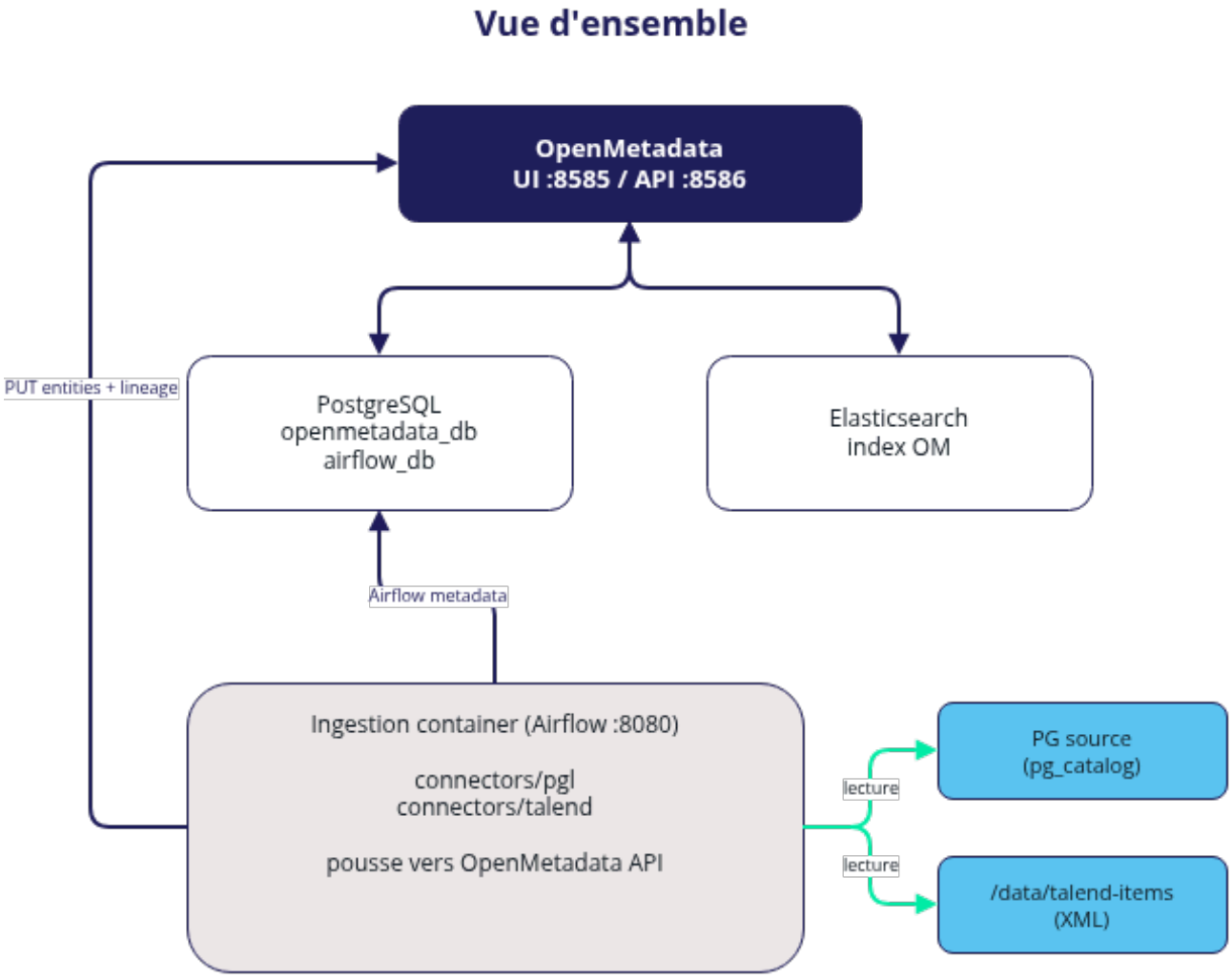


Figure 1: vue d'ensemble

```
                CATALOGUE (serveur : interface web + API REST)
                          │                │                │
          ┌───────────────┘                │                └──────────────┐
          ▼                                ▼                               ▼
base de métadonnées              moteur de recherche              ingestion + ordonnanceur
                                                                            │
                                                  ┌─────────────────────────┤
                                                  ▼                         ▼
                                  connecteurs natifs   connecteurs      enrichissement
                                  (déclaratifs)        développés       (propriétés perso)
```

Contenu

Chemin	Rôle
`src/product/openmetadata-ingestion/connectors/`	les connecteurs développés : base de données, ETL, transferts de fichiers, plateforme BI
`src/product/openmetadata-ingestion/talend_lineage/`	extraction du lineage depuis un dépôt de jobs ETL
`src/product/openmetadata-ingestion/lineage/`	briques de lineage partagées
`src/product/mcp-proxy/`	proxy d'exposition du catalogue à un assistant
`src/config/ingestion/`	définitions déclaratives et tâches d'ordonnancement
`scripts/`	scripts d'audit, d'enrichissement et de purge
`deployment/`, `distribution/`, `compose.yaml`	déploiement conteneurisé
`docs/`	architecture, procédures, incidents, notes de patch
`skills/`	procédures méthodologiques du projet

Ce qui vaut d'être regardé

1. Les deux stratégies d'ingestion, choisies consciemment

Stratégie	Quand	Où
Connecteur natif déclaratif	la source est couverte officiellement	`src/config/ingestion/workflows/`
Connecteur développé	source non couverte : entrepôt spécifique, ETL propriétaire, transferts	`connectors/`

Le critère n'est pas idéologique : un connecteur officiel existe et fonctionne, on l'utilise ; sinon on écrit un envoi direct via l'API du catalogue.

2. Le socle partagé entre connecteurs (`connectors/_om_base/`)

Ce qui évite de réimplémenter quatre fois la même chose : résolution de noms qualifiés, normalisation des noms de service, création idempotente d'entités, gestion des conflits d'écriture.

Point d'attention traité : une écriture idempotente qui renvoie un conflit signifie « déjà présent », pas un échec. L'identifiant est extrait du corps de la réponse d'erreur plutôt que d'abandonner.

3. La résolution d'ambiguïté entre environnements

(`connectors/powerbi/`) Un même objet logique existe sur plusieurs instances (production, anonymisée, réplica). Le nom seul ne suffit pas.

Résolution en cascade : correspondance par hôte de connexion d'abord, puis comptage de correspondances, et retour d'une absence de valeur plutôt qu'un candidat arbitraire si rien ne tranche.

C'est le principe directeur du projet : mieux vaut aucune valeur qu'une valeur fausse présentée comme fiable.

4. Les notes de patch du connecteur amont

(`connectors/powerbi/PATCH_NOTES.md`) Le document le plus intéressant du projet. Douze patches sur un connecteur open source, chacun documenté avec son problème, sa cause racine, son correctif et son statut (appliqué, retiré, jamais implémenté).

Les entrées « retiré » et « jamais implémenté » ont autant de valeur que les autres : elles gardent la trace d'une réflexion et évitent de refaire le chemin.

Neuf de ces douze patches sont contribuables en amont, dont trois prioritaires. Analyse complète et démarche dans `../../docs/25-contribution-opensource.md`.

Exemples de correctifs génériques, utiles à toute la communauté :

Patch	Problème corrigé
Support d'une base répandue dans l'analyseur d'expressions	lacune de couverture, aucun lineage produit pour cette source

Suppression d'un repli de propriétaire par correspondance approximative	attribution silencieuse d'un mauvais propriétaire, incident réel sur environ 150 objets
Lineage des sources fichiers	capacité entièrement absente, cas d'usage très courant
Vérification du schéma après recherche plein texte	faux positif plausible retenu comme correspondance

5. Le réglage qui peut détruire un catalogue

`markDeleted*` laissé à faux dans les définitions d'ingestion.

Un connecteur qui marque supprimé tout ce qui n'apparaît pas dans son exécution courante efface le travail des autres dès que deux traitements partagent un périmètre. C'est le réglage le plus dangereux d'une configuration d'ingestion.

6. L'enrichissement par propriétés personnalisées (`scripts/`)

Le seul pont propre entre catalogue et supervision : un traitement périodique qui ajoute des propriétés sur des entités déjà cataloguées, sans dupliquer le lineage ni le monitoring.

Piège traité dans le code : une entité qui n'a jamais eu de propriété personnalisée n'a pas de sous-objet dédié côté API. Il faut détecter l'absence et basculer entre deux formes de modification, sinon l'appel échoue en erreur serveur.

7. Le proxy d'exposition à un assistant (`src/product/mcp-proxy/`)

Le serveur natif du catalogue est générique et verbeux. Le proxy encapsule la logique métier : déduplication des environnements, recherche multi-critères, élagage des réponses.

Deux règles apprises pour un outil destiné à un assistant :

- exclure les champs volumineux par défaut, les rendre optionnels : un outil de recherche qui retourne la liste exhaustive des colonnes sature la fenêtre de contexte en trois appels ;
- des outils spécialisés plutôt qu'un outil générique : « chercher un pipeline » est plus utile que « chercher n'importe quoi ».

8. La documentation d'incidents (`docs/`)

Procédures de reprise, analyses d'incidents, plans de dette. Ce qui rend le déploiement maintenable par quelqu'un d'autre.

Quelques pièges qui y sont consignés, tous vécus :

Piège	Conséquence
Redémarrer un conteneur ne recharge pas un fichier d'environnement modifié	configuration figée sans raison apparente, heures de diagnostic
Un retour à une version antérieure impose une réinitialisation de base	les migrations de schéma ne sont pas réversibles
Un jeu de données restauré peut écraser la configuration d'authentification	comparer environnement, base et API avant de diagnostiquer
Le rapprochement d'identité se fait sur le premier attribut configuré	un même utilisateur crée deux comptes selon son chemin de connexion
Un service qui ne retente jamais la découverte d'authentification	reste cassé après un incident réseau transitoire, même rétabli

Vérification après anonymisation

Contrôle	Résultat
Compilation des 131 fichiers Python	ok
Tests des connecteurs	289 passent, 24 échouent
Tests du proxy	28 passent
Schémas régénérés depuis les sources anonymisées	8 images
Motifs sensibles résiduels	aucun

Sur les 24 échecs : ils sont strictement identiques à ceux du dépôt d'origine avant anonymisation (liste comparée fichier à fichier). Ils viennent de l'absence du SDK du catalogue dans l'environnement de test, pas du code. Zéro régression introduite.

C'est le point qui a demandé le plus de soin sur ce projet : plusieurs allers-retours ont été nécessaires pour que les substitutions restent cohérentes avec ce que les tests vérifient.

Ce qui a été retiré

Retiré	Volume	Raison
`data/`	12 Go	dumps, exports, dépôts de jobs clonés : données de production
`deployment/seed/`	1,8 Go	dump de la base de métadonnées
`transcripts/`, `comptes-rendus/`	-	transcriptions et comptes rendus de réunions internes
`graphify-out/`	4,6 Mo	graphe de connaissance généré
notebook d'analyse	-	sorties de données possibles
logo	-	nominatif
`.claude/`, `context.xml`	-	états locaux et export pour assistant
historique de contrôle de version	-	contient ce qui a été supprimé

Note sur l'anonymisation

Ce projet a demandé une anonymisation plus fine que les autres, parce que les tests comparent entre eux des identifiants dérivés de noms d'hôtes.

Élément	Traitement
Hôtes	28 hôtes distincts remplacés par 28 adresses de documentation distinctes, pour préserver la sémantique des tests qui vérifient qu'ils diffèrent
Formes dérivées	les identifiants slugifiés construits à partir d'un hôte ont été substitués de façon cohérente
Bases et schémas	noms génériques et parlants : `warehouse_prod`, `tickets_tra`, `billing_bi`, `assets_src`
Variantes de casse et séparateurs	couvertes, car le code normalise ces formes avant comparaison
Identifiants techniques	mot neutre sans chevrons, les marqueurs cassant la normalisation
Personnes, projets, partenaires	`<USER>`, `<COLLEAGUE>`, `<PROJECT_A>`, `<PARTNER_A>`

La leçon : une anonymisation qui collapse plusieurs valeurs distinctes en un seul marqueur casse tout code qui les compare. La cohérence de la substitution est une propriété fonctionnelle, pas seulement une question de confidentialité.