Airflow 3 - Explications Techniques Détaillées

Slide 5 : Changements Majeurs

Ce document fournit des explications techniques approfondies pour chacun des six changements majeurs présentés dans la slide 5 de la présentation sur la migration Airflow 3.

■ 1. Architecture Auth Manager

Vue d'ensemble du changement

Airflow 3 introduit une refonte complète du système d'authentification et d'autorisation. Le système Flask-AppBuilder (FAB) qui gérait l'authentification dans Airflow 2.x est complètement supprimé et remplacé par une nouvelle architecture modulaire appelée Auth Manager.

Problèmes avec Flask-AppBuilder (Airflow 2.x)

• **Couplage fort** : FAB était étroitement couplé à l'application web Airflow, rendant difficile la personnalisation ou le remplacement du système d'authentification.

• **Limitations OIDC** : Le support OIDC dans FAB était basique et nécessitait des workarounds pour des scénarios d'entreprise complexes.

• **Complexité de maintenance** : FAB apportait beaucoup de fonctionnalités non nécessaires pour Airflow, augmentant la surface d'attaque de sécurité.

Nouvelle architecture Auth Manager

L'Auth Manager est une interface abstraite permettant de plugger différents systèmes d'authentification. Au lieu d'un système monolithique, Airflow 3 définit un contrat (interface) que toute implémentation d'Auth Manager doit respecter.

Composants clés :

• **BaseAuthManager** : Classe de base abstraite définissant les méthodes que tout Auth Manager doit implémenter (is_logged_in, get_user, check_authorization, etc.).

• **FabAuthManager** : Implémentation par défaut maintenant une compatibilité avec FAB pour faciliter la migration.

• **Custom Auth Managers** : Possibilité de créer des implémentations personnalisées pour des besoins spécifiques (comme notre Keycloak Auth Manager).

Support OIDC natif amélioré

Le nouveau système offre un meilleur support pour les protocoles modernes d'authentification :

• **JWT Token Management** : Gestion native des tokens JWT avec validation, refresh automatique, et gestion de l'expiration.

• **Claims Mapping** : Mapping flexible des claims OIDC vers les rôles et permissions Airflow.

- **Multi-tenant** : Support amélioré pour les architectures multi-tenant.

RBAC amélioré

Le système de contrôle d'accès basé sur les rôles (RBAC) est plus granulaire et performant :

- **Permissions granulaires** : Nouvelles permissions au niveau des ressources (DAG, Connection, Variable, Pool) avec actions spécifiques (read, edit, delete, create).
- **Performance** : Mise en cache optimisée des permissions, réduisant les requêtes DB.
- **Audit trail** : Meilleur logging des actions d'autorisation pour la conformité.

Notre implémentation Keycloak

Pour DataProduct BNPP, nous avons développé un Custom Keycloak Auth Manager qui :

- Extrait et valide les tokens JWT de Keycloak
- Mappe les groupes Keycloak vers les rôles Airflow
- Gère le refresh automatique des tokens avant expiration
- Intègre avec notre infrastructure Zero Trust (Illumio)

■ 2. Amélioration des Performances du Scheduler

Rôle critique du Scheduler

Le Scheduler est le cœur d'Airflow. Il est responsable de :

- Parser les fichiers DAG pour détecter les changements
- Créer les DagRuns et TaskInstances selon les schedules
- Déterminer quelles tâches sont prêtes à s'exécuter
- Soumettre les tâches aux executors

Optimisation du Parsing Parallèle

Airflow 2.x - Problèmes :

- Le parsing des DAGs se faisait en multiprocessing avec des subprocesses Python
- Overhead important du fork/spawn de processus
- Pas de partage de mémoire entre processus
- Délai de démarrage élevé pour chaque subprocess

Airflow 3 - Améliorations :

- **Process pooling** : Réutilisation des processus de parsing entre les cycles, éliminant le coût de création/destruction répétée.
- **Parsing incrémental** : Seuls les DAGs modifiés sont re-parsés (détection via mtime du fichier).
- **Shared memory structures** : Utilisation de structures de données partagées pour réduire la duplication en mémoire.
- **Async I/O** : Opérations de lecture/écriture DB asynchrones pendant le parsing.

Impact sur les performances

- **Réduction de 40-60%** du temps de parsing pour les environnements avec 100+ DAGs
- **Diminution de 30%** de l'utilisation CPU du scheduler
- **Meilleure scalabilité** : Supporte efficacement 500+ DAGs sur une instance scheduler

Réduction de la latence

La latence entre le moment où une tâche devient eligible et son démarrage effectif a été réduite grâce à :

- **Optimisation des requêtes DB** : Réduction du nombre de requêtes et utilisation d'indexes plus efficaces.
- **Batching** : Les TaskInstances sont créées et mises à jour par batches plutôt qu'individuellement.
- **Lock reduction** : Moins de contentions sur les verrous DB grâce à une meilleure stratégie de locking.

Meilleure gestion mémoire

- **Garbage collection optimisé** : Meilleur cycle de GC pour les objets temporaires

• **Memory pooling** : Réutilisation des allocations mémoire

• **Leak fixes** : Correction de plusieurs memory leaks présents dans Airflow 2.x

■ 3. Interface Utilisateur Moderne

Migration vers React

L'interface web d'Airflow a été complètement réécrite en utilisant React, remplaçant l'ancien système basé sur Flask templates avec JavaScript vanilla.

Architecture technique

- **Frontend SPA** : Single Page Application avec React 18
- **State Management** : Utilisation de React Query pour la gestion du cache et des requêtes API
- **Component Library** : Composants réutilisables avec Chakra UI
- **Build System** : Vite pour des temps de build rapides et HMR (Hot Module Replacement)

Améliorations de navigation

- **Routing côté client** : Navigation instantanée sans rechargement de page (React Router)
- **Breadcrumbs intelligents** : Navigation contextuelle basée sur la hiérarchie
- **Search global** : Recherche unifiée de DAGs, tasks, et runs avec auto-complétion
- **Favoris et historique** : Accès rapide aux DAGs fréquemment consultés

Performances accrues

- **Virtual scrolling** : Affichage de grandes listes (milliers de task instances) sans lag
- **Lazy loading** : Chargement progressif des données à la demande
- **Code splitting** : Téléchargement uniquement du code JavaScript nécessaire
- **Asset optimization** : Images et assets optimisés pour le web

Nouvelles visualisations des DAGs

- **Graph View amélioré** : Rendu SVG performant avec zoom/pan fluide et détection de collision
- **Gantt Chart interactif** : Visualisation temporelle avec filtres et drill-down
- **Calendar View** : Vue calendrier des runs avec indicateurs de statut
- **Task Duration heatmap** : Identification visuelle des tasks lentes

Responsive Design

L'interface s'adapte maintenant aux différentes tailles d'écran (desktop, tablette, mobile) grâce à un design responsive basé sur CSS Grid et Flexbox.

■ 4. API REST Enrichie

Nouveaux endpoints

Airflow 3 ajoute de nombreux nouveaux endpoints pour des opérations qui nécessitaient auparavant des requêtes SQL directes ou des workarounds :

- **Bulk operations** : /dags/~/dagRuns/bulk pour opérations par batch
- **Task logs streaming** : /dags/{dag_id}/dagRuns/{run_id}/taskInstances/{task_id}/logs/stream
- **Health checks** : /health endpoint détaillé avec statut des composants
- **Metrics** : /metrics pour Prometheus-compatible metrics
- **Import errors** : /importErrors pour déboguer les problèmes de parsing DAG

Documentation OpenAPI améliorée

- **OpenAPI 3.1** : Spécification complète et à jour
- **Exemples interactifs** : Swagger UI avec possibilité de tester les endpoints
- **Schémas détaillés** : Tous les modèles de données documentés avec exemples
- **Code generation** : Génération de clients SDK dans différents langages

Support GraphQL (Expérimental)

Airflow 3 introduit un support expérimental de GraphQL permettant aux clients de requêter précisément les données dont ils ont besoin :

- **Flexible queries** : Récupérer uniquement les champs nécessaires
- **Nested resources** : Obtenir DAG + runs + tasks en une seule requête
- **Subscriptions** : Mises à jour en temps réel via WebSockets

Pagination optimisée

L'API utilise maintenant une pagination cursor-based au lieu de offset-based :

- **Performance** : Pas de OFFSET coûteux en base de données
- **Consistency** : Les résultats restent cohérents même si des données sont ajoutées/supprimées
- **Scalability** : Supporte efficacement des millions de records

Versioning de l'API

L'API inclut maintenant un système de versioning explicite :

- **URL versioning** : /api/v1/, /api/v2/
- **Deprecation warnings** : Headers indiquant les endpoints obsolètes
- **Backward compatibility** : Support de multiples versions en parallèle

■ 5. Versioning des DAGs

Problème dans Airflow 2.x

Airflow 2.x ne conservait pas d'historique des versions des DAGs. Si un DAG était modifié, il était impossible de savoir exactement quelle version du code avait exécuté un DagRun passé. Cela posait plusieurs problèmes :

- **Debugging difficile** : Impossible de reproduire un bug si le DAG a changé
- **Audit limité** : Pas de traçabilité des changements
- **Rollback complexe** : Nécessitait des manipulations Git manuelles

Architecture du versioning dans Airflow 3

Airflow 3 introduit un système de versioning natif qui capture et stocke les versions des DAGs :

Mécanisme de capture

- **Hash du code DAG** : Chaque fois qu'un DAG est parsé, un hash SHA-256 du code est calculé
- **Stockage en DB** : Le hash et le code complet sont stockés dans une table 'dag_version'
- **Référence dans DagRun** : Chaque DagRun pointe vers la version de DAG qui l'a créé
- **Détection automatique** : Le système détecte automatiquement les changements de code

Structure de données

La table dag_version contient :

- dag_id : Identifiant du DAG
- version_number : Numéro de version incrémental
- version_hash : Hash SHA-256 du code
- dag_code : Code Python complet du DAG
- created_at : Timestamp de création
- created_by : Utilisateur ou système ayant créé la version

Historique des changements

L'interface web affiche maintenant :

- **Timeline des versions** : Liste chronologique de toutes les versions
- **Diff view** : Comparaison visuelle entre deux versions (colored diff)
- **Blame/attribution** : Qui a fait quel changement et quand
- **Impact analysis** : Quels DagRuns utilisent quelle version

Rollback simplifié

Le rollback devient une opération simple via l'UI ou l'API :

- **UI one-click** : Bouton 'Restore to this version' dans l'interface

- **API endpoint** : POST /dags/{dag_id}/versions/{version_number}/restore
- **Safe rollback** : Validation avant restauration (pas de runs en cours)
- **Rollback history** : Les rollbacks sont eux-mêmes versionnés

Traceabilité améliorée

Le versioning améliore considérablement l'audit et la conformité :

- **Compliance** : Preuve de quel code a traité quelles données
- **Reproduction** : Capacité de re-exécuter un DagRun avec sa version exacte
- **Change tracking** : Audit trail complet des modifications de pipeline

Intégration Git (Optionnelle)

Bien que le versioning soit natif dans Airflow, il peut s'intégrer avec Git :

- **Git commit tracking** : Liaison entre version Airflow et commit Git
- **Annotations** : Possibilité d'ajouter des messages de commit aux versions
- **CI/CD integration** : Les pipelines CI/CD peuvent taguer les versions

■ 6. Support de Python 3.12

Évolution de la compatibilité Python

Airflow 3 marque une étape importante dans la modernisation du support Python :

- **Airflow 2.x** : Python 3.7, 3.8, 3.9, 3.10, 3.11
- **Airflow 3.x** : Python 3.9, 3.10, 3.11, 3.12 (3.13 en preview)
- **Minimum requis** : Python 3.9
- **Recommandé** : Python 3.12 pour nouvelles installations

Améliorations de performance Python 3.12

Python 3.12 apporte des gains de performance significatifs dont Airflow bénéficie directement :

- **15-25% plus rapide** : Amélioration générale des performances vs Python 3.11
- **PEP 709** : Comprehension inlining (list/dict comprehensions plus rapides)
- **Improved GC** : Garbage collector plus efficace
- **Faster startup** : Temps de démarrage Python réduit de 10-15%

Nouvelles fonctionnalités du langage

Python 3.12 introduit des features que les DAGs peuvent maintenant utiliser :

- **PEP 695 - Type Parameter Syntax** : Syntaxe simplifiée pour les génériques

```
# Old (Python < 3.12) from typing import TypeVar, Generic T = TypeVar('T') class Stack(Generic[T]): ... # New (Python 3.12+) class Stack[T]: ...
```

- **PEP 698 - Override decorator** : Validation des overrides de méthodes

```
from typing import override class CustomOperator(BaseOperator): @override def execute(self, context): # Vérifié à runtime ...
```

- **PEP 701 - f-strings améliorés** : f-strings plus puissants avec quotes imbriquées

Sécurité améliorée

- **CVE fixes** : Python 3.12 corrige plusieurs vulnérabilités présentes dans 3.8/3.9
- **hashlib hardening** : Algorithmes de hash plus sécurisés par défaut
- **SSL/TLS updates** : Support des derniers protocoles TLS

Dépendances modernisées

Avec Python 3.12, Airflow peut utiliser des versions plus récentes de ses dépendances :

- **SQLAlchemy 2.0** : Meilleure performance et API modernisée
- **Pydantic 2.x** : Validation de données beaucoup plus rapide
- **FastAPI récent** : Si utilisé pour des custom endpoints
- **NumPy/Pandas modernes** : Meilleures performances pour le traitement de données

Impact sur nos DAGs

Pour DataProduct BNPP, la migration vers Python 3.12 offre :

- **Performance** : DAGs avec beaucoup de logique Python s'exécutent plus rapidement
- **Maintenance** : Code plus moderne et plus facile à maintenir
- **Libraries** : Accès aux dernières versions des bibliothèques Python
- **Long-term support** : Python 3.12 supporté jusqu'en octobre 2028

Migration recommandée

Bien qu'Airflow 3 supporte Python 3.9+, nous recommandons Python 3.12 pour :

- Nouvelles instances Airflow
- Environnements où la performance est critique
- Projets planifiés pour 2+ ans

Conclusion

Ces six changements majeurs dans Airflow 3 représentent une évolution significative de la plateforme. Chaque amélioration a été conçue pour résoudre des limitations réelles d'Airflow 2.x et préparer la plateforme pour les besoins futurs en matière d'orchestration de données. Pour DataProduct BNPP, ces améliorations se traduisent par une infrastructure plus performante, plus sécurisée, et plus maintenable. Notre investissement dans la migration sera rapidement rentabilisé par les gains en efficacité opérationnelle et en expérience développeur.