Livre Blanc : Architecture de Streaming Lakehouse – La Convergence de Kafka, Apache Iceberg et Microsoft Fabric

Sommaire Exécutif

L'architecture des données d'entreprise traverse actuellement une mutation sans précédent, marquée par la convergence de deux paradigmes historiquement distincts : le traitement des flux de données en temps réel (*data streaming*) et l'analytique sur de grands volumes de données stockées (*data lakehouse*). Ce livre blanc examine en profondeur l'architecture de **Streaming Lakehouse**, une approche unifiée qui exploite la puissance de la plateforme Confluent (basée sur Apache Kafka) pour l'ingestion et le transport des données, la robustesse transactionnelle du format de table ouvert Apache Iceberg pour le stockage, et les capacités analytiques avancées de l'écosystème Microsoft Fabric.

Dans un contexte canadien marqué par des exigences réglementaires strictes — notamment la **Loi 25** au Québec et le projet de loi fédéral **C-27** — et par une adoption accélérée de l'infonuagique par les grandes institutions financières et de télécommunications (Banque Royale du Canada, Bell Canada, Banque Nationale), la maîtrise de cette architecture devient un impératif stratégique. Ce document détaille les mécanismes techniques, les stratégies de modélisation, les motifs d'ingestion et les protocoles de gouvernance nécessaires pour déployer un Streaming Lakehouse performant, sécuritaire et conforme.

Chapitre 1 : L'Évolution vers le Streaming Lakehouse

1.1 De l'Architecture Lambda au Streaming Lakehouse

Historiquement, les entreprises ont été contraintes de diviser leurs architectures de données en deux voies parallèles pour répondre à des besoins contradictoires : la vitesse et l'exhaustivité.

Les limites de l'Architecture Lambda

L'architecture Lambda, longtemps considérée comme la norme, proposait deux couches distinctes :

1. **La couche de vitesse (Speed Layer) :** Utilisant des technologies comme Apache Storm ou les premières versions de Kafka, elle traitait les données en temps réel pour fournir des indicateurs approximatifs ou urgents.
2. **La couche de lot (Batch Layer) :** Utilisant Hadoop HDFS ou des entrepôts de données traditionnels, elle retraitait l'intégralité des données historiques périodiquement (souvent la nuit) pour corriger les erreurs et fournir des analyses précises.

Cette dualité imposait une complexité opérationnelle majeure : la logique de traitement devait être codée deux fois (une fois pour le streaming, une fois pour le batch), entraînant des divergences de données et une lourdeur de maintenance.[1] De plus, la latence inhérente à la couche batch signifiait que les décideurs travaillaient souvent avec des données vieilles de 24 heures (J+1), inadaptées à la détection de fraude ou à la personnalisation client en temps réel.

L'avènement de l'Architecture Kappa

L'architecture Kappa a tenté de simplifier ce modèle en proposant de traiter toutes les données comme des flux. Dans ce modèle, le système de streaming devient la source unique de vérité. Cependant, les moteurs de streaming purs peinaient historiquement à gérer efficacement les requêtes analytiques complexes (OLAP) sur des années d'historique, en raison de limitations de stockage et d'indexation.

Le Streaming Lakehouse : La Synthèse

Le Streaming Lakehouse résout ces limitations en introduisant une couche de stockage transactionnelle (Apache Iceberg) alimentée en continu par le flux (Kafka). Cette architecture permet :

- **L'ingestion en temps réel :** Les données sont disponibles pour l'analyse quelques secondes après leur génération.
- **La correction transactionnelle :** Contrairement aux simples fichiers dans un lac de données, Iceberg garantit les propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité), permettant des mises à jour et des suppressions fiables.[2]
- **L'unification du stockage :** Un seul référentiel de données sert à la fois les applications opérationnelles temps réel et les rapports analytiques BI (Business Intelligence), éliminant les silos.[4]

1.2 Le Rôle de Confluent et Kafka dans l'Écosystème Moderne

Apache Kafka, et sa distribution entreprise Confluent, agit comme le système nerveux central de cette architecture. Il ne s'agit plus seulement d'un tuyau de transport, mais d'une plateforme de traitement et de gouvernance des données en mouvement.

Pour des institutions comme la **Banque Royale du Canada (RBC)**, l'adoption de Kafka a été le catalyseur d'une transformation majeure. En passant d'une infrastructure basée sur des mainframes monolithiques à une architecture événementielle, RBC a pu "libérer" les données de ses systèmes cœurs.[5] Au lieu d'interroger le mainframe pour chaque consultation de solde (ce qui est coûteux et peu scalable), les événements de transaction sont publiés dans Kafka. Ces flux alimentent ensuite des microservices et des lacs de données en temps réel, réduisant le temps de détection des anomalies de plusieurs semaines à quelques secondes.[5]

Cette capacité à découpler les systèmes producteurs (Legacy, IoT, Apps) des consommateurs analytiques est fondamentale. Elle permet à des organisations comme **Shopify** de traiter des milliards d'événements par jour, en utilisant Kafka comme tampon résilient avant de déverser les données dans des formats analytiques pour des requêtes via Trino.[6]

Chapitre 2 : Anatomie Technique d'Apache Iceberg

Pour comprendre pourquoi Apache Iceberg est le pivot du Streaming Lakehouse, il est essentiel de disséquer son fonctionnement interne. Iceberg n'est pas un service de stockage, mais une spécification de format de table ouverte qui apporte les fonctionnalités d'une base de données SQL aux fichiers stockés dans un lac de données (S3, ADLS, GCS).

2.1 Hiérarchie des Fichiers et Gestion des Métadonnées

L'architecture d'Iceberg repose sur une structure arborescente de métadonnées qui isole les lecteurs des écrivains, permettant une concurrence optimiste sans verrouillage global coûteux.

2.1.1 Le Fichier de Métadonnées (metadata.json)

Au sommet de la hiérarchie se trouve le fichier de métadonnées de la table. Ce fichier JSON contient :

- Le schéma actuel de la table et son historique d'évolution.
- La spécification de partitionnement actuelle.
- La liste des instantanés (*snapshots*), avec une référence vers l'instantané courant.[3]

Chaque opération d'écriture (INSERT, UPDATE, DELETE) produit un *nouveau* fichier de métadonnées. Le catalogue de la table (par exemple, Hive Metastore, AWS Glue ou un catalogue REST) ne fait que pointer vers ce dernier fichier JSON. Ce mécanisme de "pointeur atomique" est la clé de l'isolation ACID : une requête lit toujours un instantané immuable et cohérent, même si une nouvelle écriture est en cours.[2]

2.1.2 La Liste des Manifestes (Manifest List)

Chaque instantané pointe vers un fichier "Liste de Manifestes" (format Avro). Ce fichier agit comme un index grossier. Il contient des métadonnées agrégées pour chaque fichier manifeste qu'il référence, notamment les plages de valeurs des clés de partition (Min/Max).

- **Optimisation :** Lors d'une requête, le moteur peut lire la liste des manifestes et ignorer immédiatement des groupes entiers de fichiers si les prédicats de la requête (ex: date >= '2024-01-01') ne correspondent pas aux plages de partitions stockées dans la liste. Cela réduit considérablement les appels d'E/S vers le stockage objet.[3]

2.1.3 Les Fichiers Manifestes (Manifest Files)

Les manifestes (également en Avro) listent les fichiers de données réels (Parquet, ORC, Avro). Mais contrairement à la liste des fichiers d'un système de fichiers classique, le manifeste Iceberg contient des statistiques détaillées pour chaque fichier de données :

- Compte de lignes.
- Limites inférieures et supérieures pour chaque colonne (pas seulement les colonnes de partition).
- Compte des valeurs nulles.

Cette granularité permet un élagage (*pruning*) au niveau du fichier, même pour des colonnes non partitionnées. Par exemple, si une requête filtre sur user_id, Iceberg peut sauter les fichiers Parquet où la plage de user_id ne contient pas la valeur recherchée, offrant des performances proches de celles d'une base de données indexée.[8]

2.2 Stratégies d'Écriture : Copy-on-Write vs Merge-on-Read

Dans un contexte de streaming, la latence d'écriture est critique. Iceberg offre deux modes de gestion des mises à jour et des suppressions, définis dans la spécification v2.[2]

2.2.1 Copy-on-Write (CoW)

Dans ce mode, lorsqu'une ligne doit être mise à jour ou supprimée, le fichier de données entier contenant cette ligne est lu, modifié en mémoire, et réécrit comme un nouveau fichier. L'ancien fichier est marqué comme obsolète dans le nouvel instantané.

- **Performance :** Les lectures sont maximales (pas de fusion à faire), mais les écritures sont lentes et coûteuses en E/S (amplification d'écriture).
- **Usage :** Adapté aux tables de dimension peu modifiées ou aux ingestions par lots massifs, mais généralement inadapté au streaming haute fréquence.[10]

2.2.2 Merge-on-Read (MoR)

Le mode MoR est conçu pour le streaming. Au lieu de réécrire les fichiers, Iceberg écrit les changements dans des fichiers séparés :

- **Data Files :** Pour les nouvelles insertions.
- **Delete Files :** Fichiers spéciaux (Position Delete ou Equality Delete) qui listent les lignes à supprimer ou mettre à jour.[9]
- **Mécanisme :** À la lecture, le moteur de requête fusionne à la volée les fichiers de données de base avec les fichiers de suppression pour présenter une vue à jour.
- **Avantage Streaming :** La latence d'écriture est très faible, permettant des commits fréquents (ex: chaque minute). Cependant, la performance de lecture se dégrade si les fichiers de suppression s'accumulent, nécessitant une compaction régulière (voir Chapitre 5).[10]

Caractéristique	Copy-on-Write (CoW)	Merge-on-Read (MoR)
Latence d'écriture	Élevée (Réécriture complète)	Faible (Append only)
Latence de lecture	Faible (Données prêtes)	Variable (Coût de fusion)
Amplification d'écriture	Très forte	Faible
Complexité maintenance	Faible	Élevée (Compaction requise)
Cas d'usage optimal	Batch, Lecture intensive	Streaming, CDC, Écriture intensive

---

Chapitre 3 : Design et Stratégie de Modélisation

La réussite d'un Streaming Lakehouse ne repose pas uniquement sur l'infrastructure, mais sur une modélisation intelligente des données adaptée aux spécificités d'Iceberg et de Kafka.

3.1 Gestion de l'Évolution de Schéma

L'un des cauchemars des Data Lakes traditionnels est la fragilité des pipelines face aux changements de schéma. Iceberg introduit une robustesse native grâce à l'utilisation d'identifiants de colonnes uniques (*Column IDs*) plutôt que de se baser sur le nom ou la position des colonnes.[14]

Intégration avec Kafka Schema Registry

Dans une architecture Confluent, les schémas sont gérés centralement par le Schema Registry (supportant Avro, Protobuf, JSON Schema). Le Streaming Lakehouse doit synchroniser ces évolutions :

1. **Ajout de colonne :** Iceberg gère cela comme une opération de métadonnées instantanée. Les anciens fichiers de données sont lus comme ayant null pour cette colonne.
2. **Renommage :** Grâce aux IDs, renommer une colonne dans le Schema Registry et propager ce changement vers Iceberg ne casse pas les requêtes historiques. La colonne garde le même ID interne.[14]
3. **Typage Complexe :** Iceberg supporte nativement les structures imbriquées (Maps, Lists, Structs) provenant de formats comme Avro. Cela est crucial pour ingérer des documents complexes (ex: logs télémétriques de Bell Canada) sans les aplatir, préservant ainsi la sémantique de la donnée source.[4]

3.2 Partitionnement Caché (Hidden Partitioning)

Le partitionnement est essentiel pour la performance, mais il a longtemps été une source d'erreurs (ex: utilisateurs oubliant de filtrer sur la colonne de partition). Iceberg résout cela avec le *Hidden Partitioning*.[15]

- **Concept :** Au lieu de créer une colonne physique day_of_event, on définit une transformation de partition sur la colonne event_timestamp (ex: day(event_timestamp)).
- **Avantage :** Les utilisateurs écrivent leurs requêtes naturellement : WHERE event_timestamp > '2024-01-01'. Iceberg comprend automatiquement qu'il doit utiliser la partition dérivée pour élaguer les données.
- **Transformations Supportées :** year, month, day, hour (pour le streaming haute vélocité), et bucket (pour distribuer uniformément les données volumineuses et éviter les hotspots).[15]

Pour des volumes de données comme ceux gérés par Shopify (plusieurs pétaoctets), une stratégie combinant partitionnement temporel (day ou hour) et *bucketing* sur une clé à haute cardinalité (ex: shop_id) est souvent nécessaire pour optimiser à la fois les écritures et les lectures.[7]

3.3 Choix du Catalogue de Métadonnées

Le catalogue est le composant qui résout le nom de la table vers l'emplacement du fichier metadata.json.

3.3.1 Iceberg REST Catalog

Devenant rapidement le standard de l'industrie, le protocole REST permet une interopérabilité maximale. Il découple le client du stockage backend des métadonnées. Des services comme **Apache Polaris** (récemment open-sourcé par Snowflake) implémentent cette interface, offrant une gouvernance centralisée compatible avec plusieurs moteurs (Spark, Flink, Trino).[18]

3.3.2 AWS Glue Data Catalog

Pour les entreprises canadiennes hébergeant leurs données dans la région AWS ca-central-1, Glue est une option naturelle. Il offre une intégration sans couture avec IAM et Lake Formation. Cependant, il faut noter que Glue peut introduire une latence lors des opérations de commit intensives comparé à un catalogue dédié transactionnel.[18]

3.3.3 Project Nessie

Nessie apporte les concepts de "Git pour les données" (branches, commits, tags, merges). C'est une option puissante pour les pipelines de streaming CI/CD.

- **Cas d'usage :** Tester une nouvelle logique d'ingestion Flink sur une branche dev isolée, valider les données, puis fusionner atomiquement vers la branche main visible par les utilisateurs BI. Cela évite d'exposer des données potentiellement corrompues en production.[18]

Chapitre 4 : Architectures et Patterns d'Ingestion

L'alimentation du lac Iceberg depuis Kafka peut se faire selon plusieurs niveaux de complexité et de contrôle.

4.1 Pattern 1 : Confluent Tableflow (L'Approche Zero-ETL)

Tableflow, intégré à Confluent Cloud, vise à simplifier radicalement l'ingestion en traitant les topics Kafka comme des tables pré-existantes.[4]

- **Fonctionnement :** Tableflow matérialise automatiquement les schémas et les données des topics Kafka en fichiers Parquet/Iceberg dans le stockage objet du client (BYOS - Bring Your Own Storage).
- **Avantages :** Élimine le besoin de développer et maintenir des connecteurs complexes. La gestion des "petits fichiers" et la compaction sont gérées par la plateforme.[22]
- **Limitations :** Moins flexible pour les transformations complexes à la volée (ex: jointures de flux avant écriture). Idéal pour une réplication fidèle "Topic-to-Table".
- **Coût/Performance :** Tableflow optimise les coûts en effectuant des écritures groupées, ce qui peut introduire une latence légèrement supérieure à une solution Flink customisée, mais pour un coût total de possession (TCO) et une complexité opérationnelle drastiquement réduits.[23]

4.2 Pattern 2 : Ingestion via Apache Flink SQL

Pour les scénarios nécessitant des transformations complexes (ETL temps réel), enrichissement, ou masquage de données sensibles (PII) avant persistance, Flink est la solution de référence.[24]

Exemple de Pipeline Flink SQL pour Iceberg :

SQL

```
-- Définition de la source Kafka
CREATE TABLE kafka_transactions (
  transaction_id STRING,
  amount DOUBLE,
  user_id STRING,
  ts TIMESTAMP(3),
  WATERMARK FOR ts AS ts - INTERVAL '5' SECOND
) WITH (
  'connector' = 'kafka',
  'topic' = 'transactions',
  'properties.bootstrap.servers' = 'pkc-xyz.ca-central-1.aws.confluent.cloud:9092',
  'format' = 'avro'
);

-- Définition de la table Iceberg (Sink)
CREATE TABLE iceberg_catalog.finance.transactions_clean (
  transaction_id STRING,
  amount DOUBLE,
  user_hash STRING, -- Donnée anonymisée
  transaction_time TIMESTAMP(3),
  PRIMARY KEY (transaction_id) NOT ENFORCED
) WITH (
  'connector' = 'iceberg',
  'write.upsert.enabled' = 'true' -- Active le mode Upsert (Merge-on-Read)
);
```

```
-- Job d'ingestion et transformation continue
INSERT INTO iceberg_catalog.finance.transactions_clean
SELECT
  transaction_id,
  amount,
  MD5(user_id) as user_hash, -- Masquage à la volée
  ts
FROM kafka_transactions;
```

Ce pattern permet d'implémenter des logiques de **Change Data Capture (CDC)** complexes, en gérant les mises à jour et les suppressions (Upserts) directement dans le flux avant l'écriture dans Iceberg.[26]

4.3 Pattern 3 : Kafka Connect Iceberg Sink

L'approche classique via Kafka Connect offre une granularité de configuration extrême, nécessaire pour certains environnements réglementés ou *on-premise*.[28]

Configuration Critique : Exactly-Once Semantics

Pour garantir qu'aucun doublon n'est généré (critique pour les données bancaires de la **Banque Nationale du Canada** ou **RBC**), le connecteur doit être configuré avec précision [30] :

- iceberg.control.commit.interval-ms : Définit la fréquence des commits (ex: 300000 ms pour 5 min). Un intervalle plus court réduit la latence mais crée plus de petits fichiers.
- exactly.once.source.support : Doit être activé au niveau du worker Connect.
- **Topic de Contrôle :** Le connecteur utilise un topic Kafka spécial (iceberg-control) pour coordonner les transactions entre les tâches distribuées. Cela assure que si une tâche échoue, le commit global est annulé ou retenté sans duplication de données.[29]

Chapitre 5 : Opérations, Maintenance et Conformité Réglementaire

Un Streaming Lakehouse en production nécessite une maintenance active pour contrer l'entropie naturelle (fragmentation des fichiers) et répondre aux exigences légales.

5.1 Stratégies de Compaction et le Problème des Petits Fichiers

L'ingestion en continu génère par définition de nombreux petits fichiers, ce qui dégrade les performances de lecture (latence d'ouverture des fichiers, overhead métadonnées). La compaction est le processus de réécriture de ces fichiers pour optimiser le stockage.[10]

Comparatif des Stratégies de Compaction

Stratégie	Description Technique	Coût CPU/IO	Impact Lecture	Fréquence Recommandée
Bin-Packing	Regroupe les petits fichiers en fichiers plus gros (target size ~128MB) sans trier les données.	Faible	Moyen (Réduit l'overhead métadonnées)	Continue (Trigger automatique)
Sort	Réécrit les fichiers en triant les données selon les clés de partition ou colonnes fréquentes.	Élevé	Élevé (Permet le *Min/Max pruning*)	Quotidienne / Hebdomadaire
Z-Order	Trie multidimensionnel (ex: region + date) préservant la localité spatiale des données.	Très Élevé	Très Élevé (Optimise les requêtes multi-prédicats)	Ponctuelle / Pour tables critiques

Source : Analyse des meilleures pratiques opérationnelles Iceberg.[33]

Pour une entreprise comme **Bell Canada** gérant des logs massifs, une stratégie de *Bin-Packing* continue combinée à un *Z-Order* hebdomadaire sur les colonnes de temps et d'identifiant d'appareil (device_id) offre le meilleur compromis coût/performance.[36]

5.2 Conformité Loi 25 (Québec) et Droit à l'Oubli

La **Loi 25** au Québec impose des règles strictes sur la conservation et la suppression des renseignements personnels, incluant le droit à l'effacement ("droit à l'oubli").[37] Dans un lac de données immuable avec historique (Time Travel), supprimer une ligne via DELETE ne suffit pas, car la donnée reste physiquement présente dans les anciens instantanés.

Procédure Technique de Conformité (Workflow)

Pour garantir la conformité légale, les ingénieurs de données doivent implémenter une procédure en trois étapes [39] :

1. Suppression Logique (Logical Delete) :

Exécuter la requête DELETE FROM clients WHERE user_id = 'XYZ'. Cela crée un nouveau snapshot où l'utilisateur n'est plus visible, mais les fichiers de données originaux existent encore.

2. Expiration des Instantanés (Expire Snapshots) :

Il faut supprimer les références aux instantanés historiques contenant la donnée.

SQL

```
-- Exemple de procédure Spark/Trino
CALL catalog.system.expire_snapshots(
 table => 'db.clients',
 older_than => TIMESTAMP '2024-06-01 00:00:00',
 retain_last => 1
);
```

Cette commande invalide les anciens instantanés. Cependant, les fichiers Parquet orphelins peuvent encore subsister sur le disque.[42]

3. Nettoyage Physique (Remove Orphan Files) :

C'est l'étape finale qui supprime physiquement les fichiers du stockage objet (S3/ADLS) pour rendre la donnée irrécupérable.

SQL

```
CALL catalog.system.remove_orphan_files(
 table => 'db.clients',
 older_than => TIMESTAMP '2024-06-01 00:00:00'
);
```

Seule l'exécution complète de ce cycle garantit la conformité avec les exigences de destruction de la Loi 25 et du RGPD.[43]

Chapitre 6 : L'Intégration avec Microsoft Fabric et Power BI

Pour de nombreuses entreprises canadiennes, l'objectif final du Streaming Lakehouse est de rendre les données accessibles aux utilisateurs métier via Power BI. L'intégration de Microsoft Fabric avec Iceberg (via OneLake) transforme cette capacité.

6.1 OneLake Shortcuts et Virtualisation

Microsoft Fabric introduit le concept de **Shortcuts** (Raccourcis) dans OneLake. Cela permet de monter des tables Iceberg externes (stockées sur S3 ou ADLS générées par Confluent/Snowflake) comme si elles étaient natives à Fabric, sans copie de données.[45]

- **Virtualisation Bidirectionnelle :** Fabric utilise une couche de traduction de métadonnées (basée sur le projet open-source Apache XTable) qui mappe dynamiquement les métadonnées Iceberg vers le format Delta Lake (format natif de Fabric). Cela rend les tables Iceberg lisibles par les moteurs Spark de Fabric et le SQL Endpoint sans duplication.[47]

- **Contrainte de Région :** Une limitation technique actuelle importante est que les Shortcuts OneLake doivent souvent résider dans la même région infonuagique que la source pour des raisons de performance et de coûts d'egress. Pour une banque canadienne, cela signifie que le stockage S3 (Iceberg) et le tenant Fabric doivent tous deux être dans Canada Central.[47]

6.2 Power BI Direct Lake : Latence et Performance

Le mode **Direct Lake** de Power BI est une rupture technologique. Au lieu d'importer les données (copie en mémoire, latence de rafraîchissement) ou d'utiliser DirectQuery (lent, traduction SQL à chaque clic), Direct Lake permet au moteur VertiPaq de Power BI de lire directement les fichiers Parquet du lac.[49]

- **Impact :** Cela permet de visualiser des volumes massifs de données (Pétaoctets) avec des performances interactives proches du mode Import.
- **Latence de Synchronisation :** Bien que qualifié de "temps réel", il existe une latence de synchronisation. Lorsque Kafka met à jour la table Iceberg, OneLake doit rafraîchir ses métadonnées virtuelles Delta avant que Power BI ne voie les nouvelles données. Cette latence peut varier de quelques secondes à quelques minutes selon la configuration de cache.[50] C'est un point critique à valider pour des tableaux de bord opérationnels nécessitant une fraîcheur sub-seconde.

Chapitre 7 : Contexte Canadien et Études de Cas

L'adoption de ces technologies au Canada est fortement influencée par la souveraineté numérique et la modernisation des infrastructures héritées.

7.1 Étude de Cas : Banque Royale du Canada (RBC)

RBC a été pionnière dans l'adoption d'architectures événementielles pour moderniser ses systèmes bancaires.

- **Défi :** Dépendance aux mainframes coûteux (MIPS) et difficulté à innover sur des données cloisonnées.
- **Solution :** Utilisation de Kafka pour "découper le monolithe" (*Monolith Slicing*). Les transactions sont capturées en temps réel et diffusées vers des applications aval (détection de fraude, solde client, offres personnalisées) sans re-solliciter le mainframe.[5]
- **Résultat :** Réduction drastique des coûts MIPS et accélération de la détection de fraude de plusieurs semaines (batch) à quelques secondes (streaming). L'ajout d'Iceberg permet désormais d'historiser ces événements pour l'entraînement de modèles IA sur des données souveraines hébergées au Canada.

7.2 Étude de Cas : Bell Canada

Bell utilise ces technologies pour la supervision de ses réseaux et la cybersécurité.

- **Défi :** Volumes massifs de logs hétérogènes provenant de millions de routeurs, box, et antennes.

- **Solution :** Ingestion via Kafka et normalisation des logs. Le passage à une architecture Lakehouse permet de conserver ces logs sur de longues périodes (conformité légale) à faible coût sur stockage objet, tout en permettant des requêtes SQL rapides pour l'investigation d'incidents de sécurité par le SOC (Security Operations Center).[36]

7.3 Souveraineté des Données et Infrastructure Régionale

La conformité avec les directives fédérales (comme la "Stratégie infonuagique" du gouvernement du Canada) impose souvent la résidence des données au pays.

- **Comparaison Régionale :**
    - **AWS Canada (Central) vs US East (N. Virginia) :** Bien que la région canadienne puisse présenter des coûts légèrement supérieurs (souvent +10-15%) et un déploiement plus tardif de certains services de pointe, elle est mandataire pour les données PII (Personally Identifiable Information) bancaires et gouvernementales.[53]
    - **Latence :** Pour des utilisateurs basés à Toronto ou Montréal, l'utilisation de ca-central-1 offre une latence réseau inférieure (<10ms) comparée à la Virginie (~20-30ms), ce qui est bénéfique pour les applications interactives Power BI Direct Lake.[53]

Conclusion et Perspectives 2026

L'architecture de Streaming Lakehouse, unifiant Kafka, Iceberg et Fabric, représente l'état de l'art de la gestion de données en 2025. Elle permet aux entreprises de concilier l'agilité du temps réel avec la rigueur de l'analytique transactionnelle.

À l'horizon 2026, cette convergence va s'accélérer avec l'émergence du **"Diskless Kafka"** (où Kafka utilisera Iceberg/S3 comme stockage primaire, éliminant la duplication sur disques locaux) et la standardisation des catalogues via le protocole REST.[55] Pour les organisations canadiennes, investir aujourd'hui dans cette stack technologique n'est pas seulement un choix technique, mais une décision stratégique pour garantir l'innovation, la conformité et la compétitivité dans une économie numérique de plus en plus rapide.

Ce rapport a été rédigé avec l'objectif de fournir une expertise technique approfondie pour les architectes de données et décideurs TI, intégrant les spécificités de l'écosystème canadien.