DELTA LAKE™

Des Data Lakes fiables à grande échelle

+

François de Buttet

développeur - architecture data

@f2buttet
François de Buttet

Sommaire

Introduction

Introduction

Databricks a créé et porté ce projet jusqu'a le rendre open-source en avril 2019 sous license Apache v2, puis confié à la fondation Linux le 16 octobre 2019.

Cette présentation de **Delta Lake** date de Janvier 2020, la version étudiée est la 0.5.0 parue le 13 Décembre 2019.

Les ressources utilisées pour la rédaction proviennent essentiellement de **Databricks** (documentation, blog, webinar…), de conférences et de sites d'informations techniques.

Introduction

Les Data Lake ont souvent des problèmes de qualité des données, en raison d'un manque de contrôle sur les données ingérées.

Delta Lake propose une solution de gestion de ces données.

Comparatif des solutions de stockage

Data lake **VS** Base de données **VS** Data warehouse

Données et traitement:

Un data lake accepte tous types de données (structurées, semi-structurées ou non structurées). Lorsqu'on a besoin d'utiliser ces données, on leur donne une structure (schema-on-read).

Les autres solutions ont besoin que les données soient structurées avant l'enregistrement (schema-on-write).

Data lake **VS** Base de données **VS** Data warehouse

Coût de stockage:

Un data lake est conçu pour un stockage à faible coût, l'utilisation de logiciels libres et souvent gratuit y contribue, la base de données a des coûts plus flexibles selon les besoins. Le stockage dans un data warehouse peut être coûteux, surtout si le volume est important.

Data lake **VS** Base de données **VS** Data warehouse

Flexibilité:

Les données structurées dans un data warehouse ou une base de données rendent plus difficile la modification de structure pour tenir compte de l'évolution des processus métier. La reconfiguration des modèles de données est plus facile sur un data lake et permet de s'adapter aux exigences métier.

Data lake **VS** Base de données **VS** Data warehouse

Chaque solution a ses avantages, ses inconvénients, son public d'utilisateurs…

Les data lake sont destinés à être exploités par des utilisateurs sachant transformer les données pour les rendre exploitables.

Les data warehouse conviennent aux entreprises, rendant les données accessibles a plus d'utilisateurs.

Les bases de données sont optimisées pour le stockage mais pas pour l'analyse.

Qu’est ce que Delta Lake

Delta Lake est une c**ouche de stockage open source** qui apporte la fiabilité aux data lake, garantissant qu'ils contiennent uniquement des données de haute qualité pour les consommateurs.

Delta Lake permet des **transactions ACID**, une **gestion évolutive des métadonnées** et u**nifie le streaming et le traitement des données par lots**. Il s'exécute au-dessus d'un data lake existant et est entièrement compatible avec les APIs d'Apache Spark.

Les données sont **stockées au format libre Apache Parquet**, permettant d'être lues par n'importe quel lecteur compatible et de tirer parti des schémas de compression et de codage efficaces natifs de ce format.

Delta Lake permet de **gérer l’évolution des schémas** à mesure que les exigences métier le demandent. Il permet d'apporter des modifications à un schéma de table qui peut être appliqué automatiquement.

Le journal des transactions (log records) de Delta Lake enregistre des détails sur chaque modification apportée aux données, fournissant un **historique complet** pour la conformité, l'audit et la reproduction.

Delta Lake va au-delà de l'architecture Lambda avec un streaming et un batch véritablement unifiés utilisant le même moteur, les mêmes API et le même code. **Une table dans Delta Lake est à la fois une table de batch, ainsi qu'une source et un récepteur de streaming**.

Le **versioning de données** permet d'accéder et de revenir aux versions antérieures pour auditer les modifications de données, annuler les mauvaises mises à jour ou reproduire des expériences.

Delta Lake fournit des **API de manipulation de données** pour fusionner, mettre à jour et supprimer des ensembles de données. Cela permet de se conformer facilement au RGPD et de simplifier la capture des données modifiées.

Les tables de données existantes peuvent être converties à partir de n’importe quel format dans lequel elles sont actuellement stockées.

devenu open source en 2019

utilise le format libre *parquet*

permet les transactions ACID

gestion de versions des données

historique des actions sur les tables

gestion des schemas de données

traitements streaming et batch unifiés

les métadonnées comme les données sont traitées de manière distribué

Architecture

Problèmes rencontrés avec les Data Lake

Les entreprises collectent beaucoup de données brutes et non structurées auprès de diverses sources. Ces données stagnent dans des Data Lake.

Comment s’assurer de la qualité des données ?

Problèmes rencontrés avec les Data Lake

Traditionnellement, on pourrait utiliser une architecture Lambda. Mais cela présente des problèmes propres à sa complexité, ainsi que sa tendance à provoquer la perte ou la corruption de données.

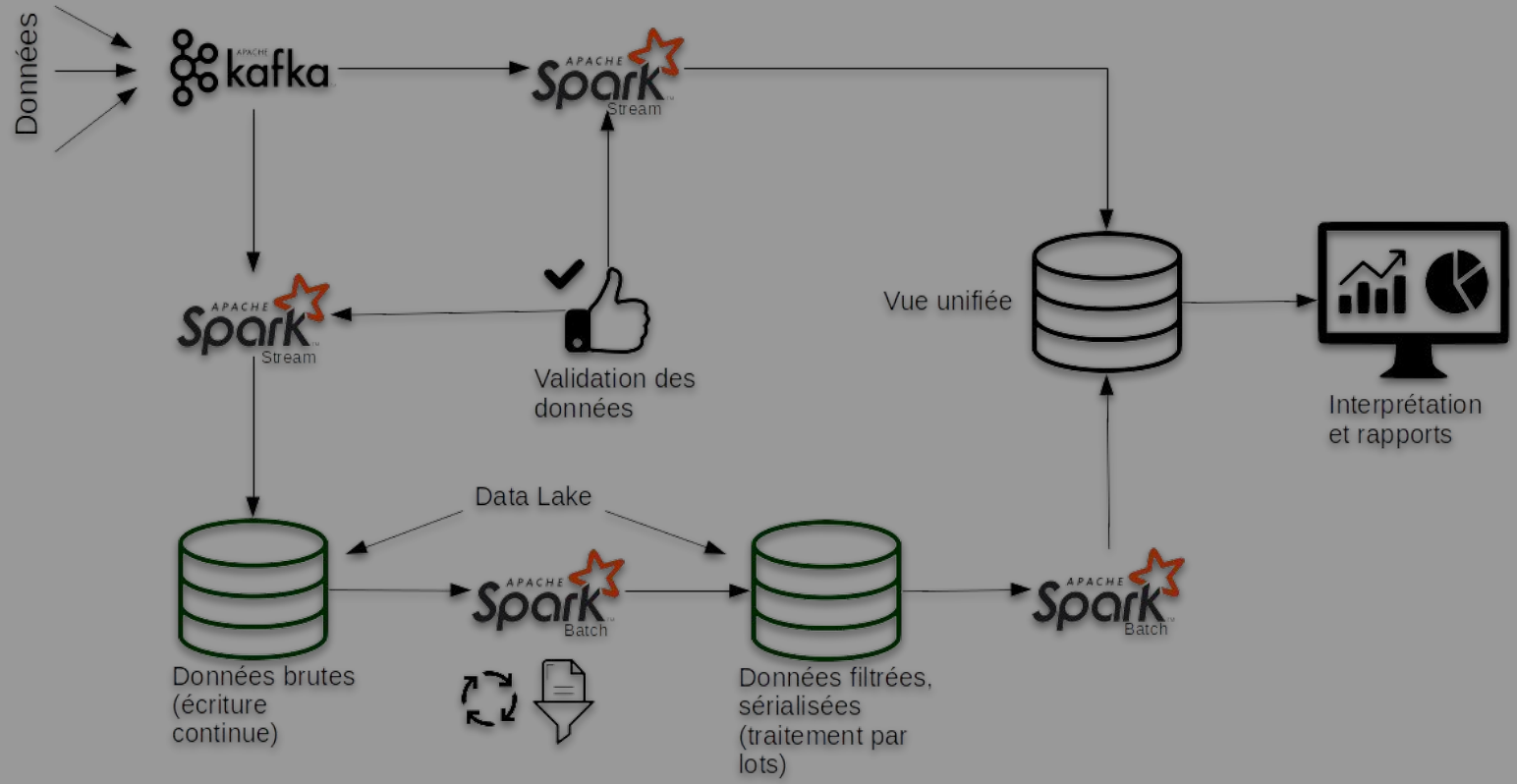
Proposition

Dans cette partie, nous étudierons:

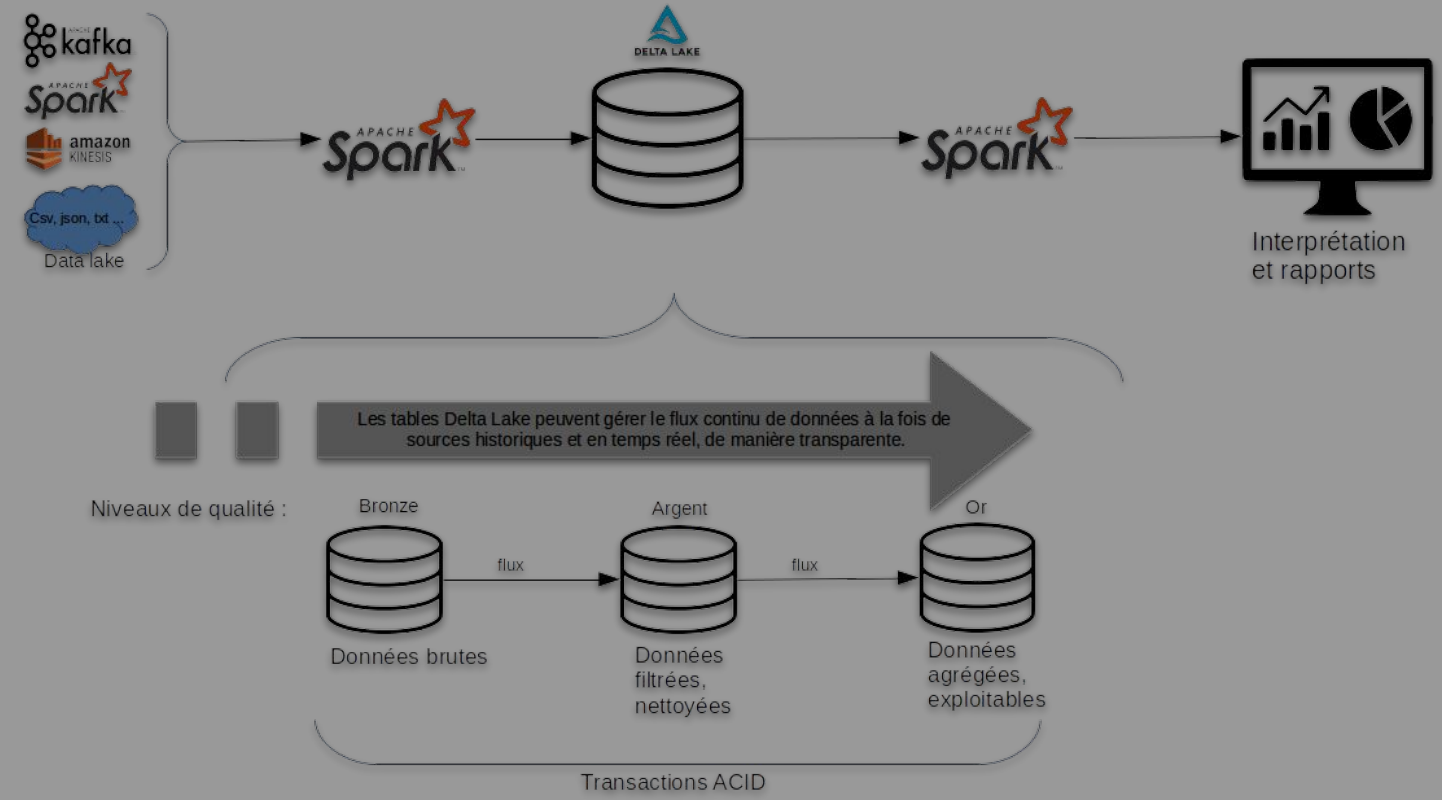
- Une architecture Lambda classique
- Une architecture utilisant **Delta Lake**

Nous observerons les avantages qu’apporte cette solution.

Architecture Lambda



Architecture Delta



Observations

L’utilisation d’Apache Spark permet la gestion simplifiée des traitements temps réel et batch qui sont unifiés.

Le flux unique de données supprime le processus de validation.

Les niveaux de qualité sont des tables Delta et font parti du flux, ce peut être des sources de données et/ou des récepteurs.

L’atomicité, garantit que les opérations effectuées s’ont exécutées entièrement ou ne s’ont pas exécutées du tout.

Observations

Delta Lake traite l’isolement entre les transactions s'exécutant simultanément en appliquant une règle d'exclusion mutuelle, puis en essayant de résoudre tout conflit de manière optimiste.

Delta Lake utilise la validation de schéma lors de l'écriture, ce qui signifie que toutes les nouvelles écritures dans une table sont vérifiées pour la compatibilité avec le schéma de la table cible au moment de l'écriture. Si le schéma n'est pas compatible, il annule complètement la transaction (aucune donnée n'est écrite) et lève une exception pour informer l'utilisateur de la non-concordance.

Démo

Configuration

Requiert Apache Spark version 2.4.2 ou supérieur.

Les systèmes de stockage ne fournissent pas nécessairement toutes les garanties d’atomicité et de durabilité, les opérations transactionnelles passent généralement par l'API *LogStore* au lieu d'accéder directement au système de stockage. **Delta Lake** dispose de l'implémentation *LogStore* intégrée pour *HDFS* depuis 0.1.0 et pour les services de stockage *Amazon S3* et *Azure* depuis 0.2.0. (https://docs.delta.io/latest/delta-storage.html)

Lancement du shell Spark

Pour cette démo, nous utiliserons Pyspark en ajoutant le package **Delta Lake**:

```
$ pyspark --packages io.delta:delta-core_2.11:0.5.0
```

Création d'une table Delta

Définir le chemin vers un répertoire qui sera initialisé comme une table Delta:

local:

```
>>> myPath = "/tmp/delta_table"
```

HDFS:

```
>>> myPath = "hdfs://<your-hdfs>/<path>/<to>/delta_table"
```

S3:

```
>>> myPath = "s3a://<your-s3-bucket>/<path>/<to>/delta_table"
```

Création d'une table Delta

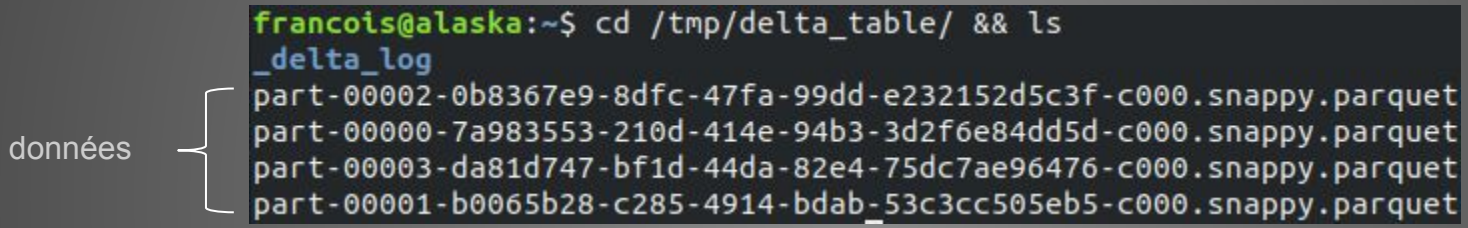
Simplement enregistrer un DataFrame au format **delta**:

```
>>> data = spark.range(0, 5)

>>> data.write.format("delta").save(myPath)
```

Création d'une table Delta

Le dossier sur le disque:



Lors de l'écriture, un partitionnement est possible (partitionBy) ce qui crée des sous répertoires dans ce dossier ou les données sont rangées.

Création d'une table Delta

Le sous-dossier _delta_log contient les logs de transaction

```
francois@alaska:/tmp/delta_table$ cd _delta_log/ && ls
00000000000000000000.json
```

chaque version de la table est représentée par un fichier json

Création d'une table Delta

```
$ cat 00000000000000000000.json
```

mode par défaut

pas de partitionnement lors de l'enregistrement de la donnée

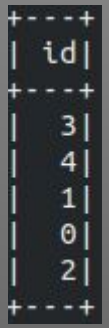

```
{"commitInfo":{"timestamp":1580285289902,"operation":"WRITE","operationParameters":{"mode":"ErrorIfExists","partitionBy":"[]"},"isBlindAppend":true}}
{"protocol":{"minReaderVersion":1,"minWriterVersion":2}}
{"metaData":{"id":"832aabe6-7cbf-4a6d-9860-b95787d96e2e","format":
    {"provider":"parquet","options":{}},"schemaString":"{\"type\":\"struct\",\"fields\":[{\"name\":\"id\",\"type\":\"long\",\"nullable\":true,\"metadata\":{}}]}",
        "partitionColumns":[],"configuration":{},"createdTime":1580285289819}
        }
{"add":{"path":"part-00000-7a983553-210d-414e-94b3-3d2f6e84dd5d-c000.snappy.parquet","partitionValues":{},"size":429,"modificationTime":1580285289000,"dataChange":true}}
{"add":{"path":"part-00001-b0065b28-c285-4914-bdab-53c3cc505eb5-c000.snappy.parquet","partitionValues":{},"size":429,"modificationTime":1580285289000,"dataChange":true}}
{"add":{"path":"part-00002-0b8367e9-8dfc-47fa-99dd-e232152d5c3f-c000.snappy.parquet","partitionValues":{},"size":429,"modificationTime":1580285289000,"dataChange":true}}
{"add":{"path":"part-00003-da81d747-bf1d-44da-82e4-75dc7ae96476-c000.snappy.parquet","partitionValues":{},"size":437,"modificationTime":1580285289000,"dataChange":true}}
```

Lire les données

La lecture des données de la table Delta se fait en spécifiant le chemin d'accès aux fichiers:

```
>>> df = spark.read.format("delta").load(myPath)
```

```
>>> df.show()
```



Update

Les options disponibles pour mettre à jour une table Delta sont celles du DataFrameWriter(overwrite, append, ignore, error, errorifexists):

```
>>> data = spark.range(5, 10)

>>> data.write.format("delta").mode("overwrite").save(myPath)

>>> data.show()
```

Update

Suite à la mise à jour un nouveau fichier json a été créé

```
$ cat 00000000000000000001.json
```


```
{"commitInfo":{"timestamp":1580288250220,"operation":"WRITE","operationParameters":{"mode":"Overwrite","partitionBy":"[]"},"readVersion":0,"isBlindAppend":false}}
{"add":{"path":"part-00000-e0b6bb1b-76b0-4023-9601-7f1ca7925d7b-c000.snappy.parquet","partitionValues":{},"size":429,"modificationTime":1580288249000,"dataChange":true}}
{"add":{"path":"part-00001-b35f0afc-6233-44df-8db9-060026e7851e-c000.snappy.parquet","partitionValues":{},"size":429,"modificationTime":1580288249000,"dataChange":true}}
{"add":{"path":"part-00002-de6f11a9-3660-433a-8f78-eccaec985508-c000.snappy.parquet","partitionValues":{},"size":429,"modificationTime":1580288249000,"dataChange":true}}
{"add":{"path":"part-00003-e8b710bb-1583-4579-bc9d-9af6bfef236a-c000.snappy.parquet","partitionValues":{},"size":437,"modificationTime":1580288249000,"dataChange":true}}
{"remove":{"path":"part-00003-da81d747-bf1d-44da-82e4-75dc7ae96476-c000.snappy.parquet","deletionTimestamp":1580288250219,"dataChange":true}}
{"remove":{"path":"part-00001-b0065b28-c285-4914-bdab-53c3cc505eb5-c000.snappy.parquet","deletionTimestamp":1580288250220,"dataChange":true}}
{"remove":{"path":"part-00000-7a983553-210d-414e-94b3-3d2f6e84dd5d-c000.snappy.parquet","deletionTimestamp":1580288250220,"dataChange":true}}
{"remove":{"path":"part-00002-0b8367e9-8dfc-47fa-99dd-e232152d5c3f-c000.snappy.parquet","deletionTimestamp":1580288250220,"dataChange":true}}
```


il permettra de garder l'historique des actions sur la table

Update

Les données sont toujours présente malgré le mode ‘overwrite’

```
francois@alaska:/tmp/delta_table$ ls
_delta_log
part-00000-7a983553-210d-414e-94b3-3d2f6e84dd5d-c000.snappy.parquet
part-00000-e0b6bb1b-76b0-4023-9601-7f1ca7925d7b-c000.snappy.parquet
part-00001-b0065b28-c285-4914-bdab-53c3cc505eb5-c000.snappy.parquet
part-00001-b35f0afc-6233-44df-8db9-060026e7851e-c000.snappy.parquet
part-00002-0b8367e9-8dfc-47fa-99dd-e232152d5c3f-c000.snappy.parquet
part-00002-de6f11a9-3660-433a-8f78-eccaec985508-c000.snappy.parquet
part-00003-da81d747-bf1d-44da-82e4-75dc7ae96476-c000.snappy.parquet
part-00003-e8b710bb-1583-4579-bc9d-9af6bfef236a-c000.snappy.parquet
```

History

Delta Lake permet de récupérer des informations sur les opérations, l'utilisateur, l'horodatage, etc. pour chaque écriture dans une table Delta en exécutant la commande ‘history()’. Les opérations sont retournées dans l'ordre chronologique inverse. Par défaut, l'historique des tables est conservé pendant 30 jours

History

Création d’un objet DeltaTable

```
>>> from delta.tables import *

>>> deltaTable = DeltaTable.forPath(spark, myPath)
```

History

Affichage

```
>>> fullHistoryDF = deltaTable.history() # historique complet
```

```
>>> lastOperationDF = deltaTable.history(1) # dernière opération
```

```
>>> fullHistoryDF.show(truncate=False)
+-------+-------------------+------+--------+---------+-----------------------------------------+----+--------+---------+-----------+--------------+-------------+
|version|timestamp          |userId|userName|operation|operationParameters                      |job |notebook|clusterId|readVersion|isolationLevel|isBlindAppend|
+-------+-------------------+------+--------+---------+-----------------------------------------+----+--------+---------+-----------+--------------+-------------+
|1      |2020-01-29 09:57:30|null  |null    |WRITE    |[mode -> Overwrite, partitionBy -> []]   |null|null    |null     |0          |null          |false        |
|0      |2020-01-29 09:08:09|null  |null    |WRITE    |[mode -> ErrorIfExists, partitionBy -> []]|null|null    |null     |null       |null          |true         |
+-------+-------------------+------+--------+---------+-----------------------------------------+----+--------+---------+-----------+--------------+-------------+
```

Upsert (merge)

```
>>> newData = spark.range(0, 20)

>>> from pyspark.sql.functions import *

>>> deltaTable.alias("oldData")\
    .merge(
    newData.alias("newData"),
    "oldData.id = newData.id")\
    .whenMatchedUpdate(set={"id": col("newData.id")})\
    .whenNotMatchedInsert(values={"id": col("newData.id")})\
    .execute()

>>> deltaTable.toDF().show()
```

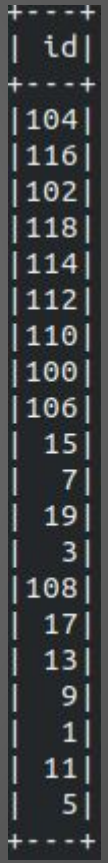
```
+---+
| id|
+---+
| 15|
|  8|
|  7|
| 12|
|  4|
| 19|
|  3|
|  6|
| 17|
|  2|
| 16|
| 13|
|  9|
| 14|
|  1|
|  0|
| 18|
| 10|
| 11|
|  5|
+---+
```

Update

Ajoute 100 aux nombres pairs

```
>>> deltaTable.update(
      condition=expr("id % 2 == 0"),
      set={"id": expr("id + 100")})

>>> deltaTable.toDF().show()
```



```
+---+
| id|
+---+
|104|
|116|
|102|
|118|
|114|
|112|
|110|
|100|
|106|
| 15|
|  7|
| 19|
|  3|
|108|
| 17|
| 13|
|  9|
|  1|
| 11|
|  5|
+---+
```

Delete

Suppression des nombres pairs

```
>>> deltaTable.delete(condition=expr("id % 2 == 0"))
>>> deltaTable.toDF().show()
```

```
+---+
| id|
+---+
| 15|
|  7|
| 19|
|  3|
| 17|
| 13|
|  9|
|  1|
| 11|
|  5|
+---+
```

Versioning

Lire une ancienne version

```
>>> deltaTable.history().show(truncate=False)
+-------+-------------------+------+--------+---------+--------------------------------------------------------------------------------------------+----+--------+---------+-----------+--------------+-------------+
|version|timestamp          |userId|userName|operation|operationParameters                                                                         |job |notebook|clusterId|readVersion|isolationLevel|isBlindAppend|
+-------+-------------------+------+--------+---------+--------------------------------------------------------------------------------------------+----+--------+---------+-----------+--------------+-------------+
|4      |2020-01-30 17:25:12|null  |null    |DELETE   |[predicate -> ["((`id` % CAST(2 AS BIGINT)) = CAST(0 AS BIGINT))"]]                         |null|null    |null     |3          |null          |false        |
|3      |2020-01-30 17:20:30|null  |null    |UPDATE   |[predicate -> ((id#1004L % cast(2 as bigint)) = cast(0 as bigint))]                         |null|null    |null     |2          |null          |false        |
|2      |2020-01-30 17:11:38|null  |null    |MERGE    |[predicate -> (oldData.`id` = newData.`id`)]                                                |null|null    |null     |1          |null          |false        |
|1      |2020-01-29 09:57:30|null  |null    |WRITE    |[mode -> Overwrite, partitionBy -> []]                                                      |null|null    |null     |0          |null          |false        |
|0      |2020-01-29 09:08:09|null  |null    |WRITE    |[mode -> ErrorIfExists, partitionBy -> []]                                                  |null|null    |null     |null       |null          |true         |
+-------+-------------------+------+--------+---------+--------------------------------------------------------------------------------------------+----+--------+---------+-----------+--------------+-------------+
```

```
>>> df = spark.read.format("delta").option("versionAsOf", 0).load(myPath)
```

```
>>> df.show()
```

```
+---+
| id|
+---+
|  3|
|  4|
|  2|
|  1|
|  0|
+---+
```

Versioning

Lire une ancienne version

```
>>> deltaTable.history().show(truncate=False)
+-------+-------------------+------+--------+---------+--------------------------------------------------------------------------------------------+----+--------+---------+-----------+--------------+-------------+
|version|timestamp          |userId|userName|operation|operationParameters                                                                         |job |notebook|clusterId|readVersion|isolationLevel|isBlindAppend|
+-------+-------------------+------+--------+---------+--------------------------------------------------------------------------------------------+----+--------+---------+-----------+--------------+-------------+
|4      |2020-01-30 17:25:12|null  |null    |DELETE   |[predicate -> ["((`id` % CAST(2 AS BIGINT)) = CAST(0 AS BIGINT))"]]                         |null|null    |null     |3          |null          |false        |
|3      |2020-01-30 17:20:30|null  |null    |UPDATE   |[predicate -> ((id#1004L % cast(2 as bigint)) = cast(0 as bigint))]                         |null|null    |null     |2          |null          |false        |
|2      |2020-01-30 17:11:38|null  |null    |MERGE    |[predicate -> (oldData.`id` = newData.`id`)]                                                |null|null    |null     |1          |null          |false        |
|1      |2020-01-29 09:57:30|null  |null    |WRITE    |[mode -> Overwrite, partitionBy -> []]                                                      |null|null    |null     |0          |null          |false        |
|0      |2020-01-29 09:08:09|null  |null    |WRITE    |[mode -> ErrorIfExists, partitionBy -> []]                                                  |null|null    |null     |null       |null          |true         |
+-------+-------------------+------+--------+---------+--------------------------------------------------------------------------------------------+----+--------+---------+-----------+--------------+-------------+
```

```
>>> df = spark.read.format("delta").option("timestampAsOf", “2020-01-29 09:57:30”).load(myPath)
```

```
>>> df.show()
```

```
+---+
| id|
+---+
|  8|
|  9|
|  7|
|  5|
|  6|
+---+
```

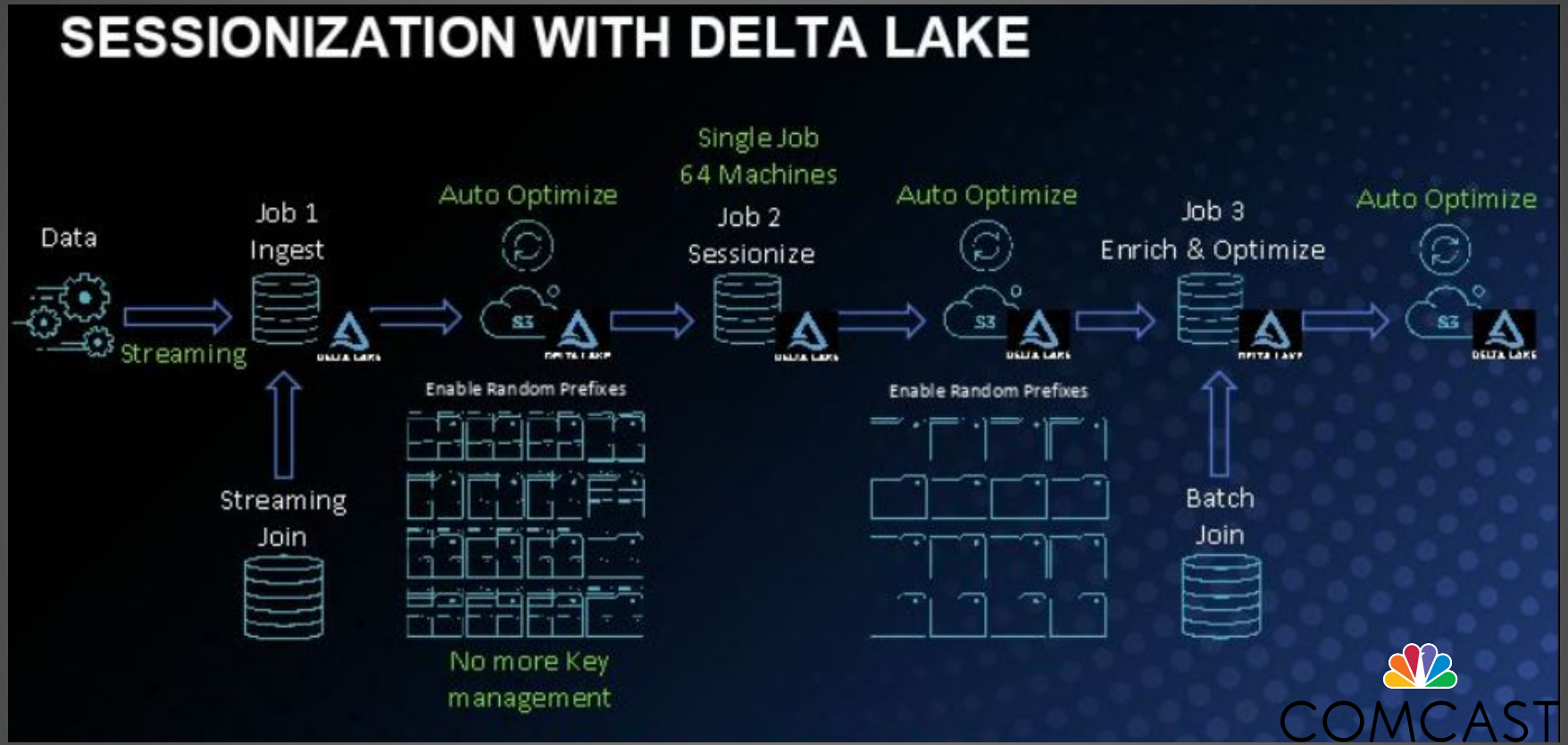
Cas d’utilisation, performances obtenues

Selon Ali Ghodsi, PDG de Databricks, pas moins de 4 000 entreprises utilisent le composant depuis son lancement en octobre 2017. « Delta Lake traite plus de 2 exaoctets de données par mois »

COMCAST

Comcast est un groupe de médias américain dont le siège est situé à Philadelphie, en Pennsylvanie. C’est le premier câblo-opérateur américain. Comcast a aussi des intérêts dans l'électronique grand public, dans des chaînes de télévision et le cinéma à la suite de l'achat de *NBCUniversal.*

Databricks présente dans un webinar son utilisation de **Delta Lake** et les performances obtenues.



Des pétaoctets de données … de 640 à 64 instances … de 84 à 3 jobs…
… latence réduite de moitié ...

APPLE

Apple est une entreprise multinationale américaine qui conçoit et commercialise des produits électroniques grand public, des ordinateurs personnels et des logiciels informatiques.

Ali Ghodsi (CEO de **Databricks**) propose lors du Keynote ‘Spark + Ai summit 2019’ le cas d’usage d’**Apple**. Démonstration de la rapidité d’exécution de Spark + Delta faite par **Michael Armbrust** (Principal Software Engineer chez **Databricks** et créateur de Spark SQL) appuyé par **Dominique Brezinski** (principal engineer dans l’équipe de direction de la sécurité des informations d'**Apple**) lors du ‘Spark + Ai summit 2018’.

Detect signal across user, application and network logs; Quickly analyze the blast radius with ad hoc queries; Respond quickly in an automated fashion; Scaling across petabytes of data and 100's of security analysts

- 100TB new data/day
- 300B events/day

Keynote Talk

BEFORE DELTA

- Took 20 engineers; 24 weeks to build
- Only able to analyze 2 week window of data

WITH DELTA

- Took 2 engineers; 2 weeks to build
- Analyze 2 years of batch with streaming data

Conclusion

Conclusion

Apache Spark + Delta Lake permet d’utiliser la puissance du calcul distribué pour la gestion des données

L’utilisation au dessus d’un data lake existant et la possibilité de convertir les données présentes

Une couche ACID sur HDFS

La qualité des données, l’historique des transactions, le versioning

Open source

Conclusion

A suivre:

Les apports d’Apache Spark 3.0 pour la couche transactionnelle

API déclarative permettant de générer des graphes orientés acycliques (DAG) correspondant aux flux de données des pipelines de Delta (début 2020)

Ressources

Ressources

- https://docs.delta.io/latest/
- https://github.com/delta-io/delta
- https://databricks.com/blog/2019/08/14/productionizing-machine-learning-with-delta-lake.html
- https://databricks.com/session/keynote-from-apple
- https://databricks.com/blog/2018/07/31/processing-petabytes-of-data-in-seconds-with-databricks-delta.html
- https://databricks.com/blog/2019/08/21/diving-into-delta-lake-unpacking-the-transaction-log.html
- https://www.lemagit.fr/actualites/252472421/Sans-surprise-le-projet-Delta-Lake-de-Databricks-va-rejoindre-la-fondation-Linux
- https://books.japila.pl/delta-lake-internals/delta/0.5.0/index.html (en cours d'écriture)
- https://www.oracle.com/fr/database/difference-data-warehouse-base-donnees.html
- https://www.youtube.com/watch?v=5l5pqDsvGEc&feature=emb_logo
- https://databricks.com/resources?_sf_s=delta%20lake&_sft_resource_type=webinars