Iceberg Catalogs

Зачем нужны каталоги в Iceberg?

Каталоги в Iceberg необходимы для управления метаданными таблиц, контроля доступа и поддержания согласованности при работе с большими данными. Они определяют местоположение таблиц и метаданных, что помогает пользователям находить и использовать таблицы в распределенной среде.

Типы каталогов iceberg:

- Hadoop Catalog
- Hive Catalog
- AWS Glue Catalog
- Nessie Catalog
- REST Catalog
- JDBC Catalog

Hadoop Catalog

- **Преимущества:**

  Интегрируется с экосистемой Hadoop и HDFS, что делает его удобным для пользователей, уже использующих Hadoop для хранения данных.

- **Недостатки:**

  Не поддерживает многотабличные транзакции и ограничен возможностями Hadoop, что снижает гибкость для использования в других средах и с другими хранилищами данных.

- **Когда использовать:**

  Подходит для пользователей, которые уже используют Hadoop и HDFS, и хотят интегрировать Iceberg для управления метаданными без необходимости в сложных функциях Iceberg.

Hive Catalog

- **Преимущества:**

  Интегрируется с Apache Hive и совместим с инструментами и SQL-запросами Hive, что облегчает миграцию на Iceberg в существующих инфраструктурах Hive.

- **Недостатки:**

  Ограничен функциональностью Hive и не поддерживает функции Iceberg, такие как версионирование и многотабличные транзакции, что ограничивает его возможности. Также необходимо развернуть доп. Сервис.

- **Когда использовать:**

  Полезен для пользователей, которые уже работают с Hive и хотят интегрировать Iceberg без необходимости в дополнительных функциях, таких как многотабличные транзакции или версионирование данных.

AWS Glue Catalog

- **Преимущества:**

  Управляемая услуга, легко интегрируется с другими сервисами AWS, автоматически масштабируемая, поддерживает высокую доступность и отказоустойчивость.

- **Недостатки:**

  Привязанность к AWS, не поддерживает многотабличные транзакции, работает только в экосистеме AWS, что ограничивает возможность использования в мультиоблачной среде.

- **Когда использовать:**

  Для пользователей, которые уже работают в AWS и не планируют переходить на другие облачные платформы.

Nessie Catalog

- **Преимущества:**

  Поддержка версионирования, мульти-табличные транзакции, возможность работать в мультиоблачной среде, независимость от конкретного облака.

- **Недостатки:**

  Не все инструменты поддерживают Nessie, требуется управление собственной инфраструктурой (хотя есть хостируемая версия от Dremio).

- **Когда использовать:**

  Если нужны мульти-табличные транзакции и поддержка версионирования данных. Хорош для пользователей, которые хотят избегать зависимости от конкретных облачных провайдеров.

REST Catalog

- **Преимущества:**

  Простота развертывания, поддержка мульти-табличных транзакций, облачная независимость.

- **Недостатки:**

  Необходимость разрабатывать или использовать сторонний сервис для обработки REST-запросов, отсутствие публичной реализации бэкенда для поддержки REST на момент написания текста.

- **Когда использовать:**

  Для гибких, настраиваемых решений, которые могут интегрироваться с различными хранилищами данных и требующих мульти-табличных транзакций.

JDBC Catalog

- **Преимущества:**

  Простота интеграции с существующими JDBC-совместимыми базами данных, высокая доступность при использовании облачных сервисов (например, Amazon RDS), облачная независимость.

- **Недостатки:**

  Не поддерживает многотабличные транзакции, требует JDBC-драйвера для всех инструментов и движков.

- **Когда использовать:**

  Для пользователей, использующих базы данных, совместимые с JDBC, или тех, кто нуждается в высокой доступности и облачной независимости.

Преимущества использования каталогов Iceberg?

Управление метаданными: Каталоги управляют метаданными таблиц, что важно для работы с большими данными.

Согласованность и надежность: Обеспечивают согласованность при работе с несколькими читателями и писателями данных, поддерживают транзакции и атомарные операции.

Масштабируемость и гибкость: Обеспечивают масштабируемость и могут работать в различных облаках и инфраструктурах, позволяя пользователю легко переносить данные и таблицы между каталогами.

Миграция каталогов:

Общие замечания:

- Миграция каталогов Iceberg — это сравнительно легкий процесс, так как данные хранятся в data lake, и миграция касается только метаданных (путей к метафайлам). Однако для успешной миграции необходим правильный план для обработки операций записи и использования инструментов.
- Миграция уменьшает риски зависимости от конкретного поставщика и помогает избежать блокировки системы каталогов.

Когда нужно мигрировать каталоги:

- Если вы хотите сменить каталог для использования в продакшн-среде, например, для перехода с Hadoop на более подходящий каталог.
- Если нужно воспользоваться дополнительными возможностями, которые отсутствуют в текущем каталоге.
- При смене расположения среды, например, при миграции из on-premise в облако (например, с Hive Metastore на AWS Glue).

Методы миграции:

Использование инструмента миграции Iceberg Catalog Migration CLI:

- Это инструмент командной строки, позволяющий переносить таблицы между каталогами без копирования данных.
- Поддерживаются такие каталоги как AWS Glue, Nessie, Dremio Arctic, Hadoop, Hive, REST, JDBC.
- Два основных режима: migrate (перемещает таблицы в новый каталог) и register (регистрирует таблицы в новом каталоге без удаления из исходного).
- Процесс миграции не создает копии данных, а лишь переносит метаданные и историю таблиц.

Использование движка, например Apache Spark:

- В Spark можно использовать процедуры register_table() и snapshot() для регистрации или создания легких копий таблиц в новом каталоге.
- Процедура register_table() создаёт копию таблицы, используя исходные файлы данных. Изменения в целевом каталоге не влияют на источник, если только данные не удаляются.
- Процедура snapshot() работает аналогично, но изменения затрагивают только целевой каталог и не влияют на

Процедуры миграции в Spark

register_table():

- Создает легкую копию таблицы в целевом каталоге, сохраняя историю и метаданные.
- Изменения в целевом каталоге не затрагивают данные в исходном каталоге, если только не происходит физическое удаление файлов данных.
- Используется для тестирования миграции, если не нужно вносить изменения в таблицу после миграции.

snapshot():

- Создает копию таблицы с изменением местоположения файлов данных в новом каталоге.
- Это позволяет использовать миграцию для переноса данных в новый каталог с изменением пути данных.
- История таблицы сохраняется, и операции, такие как time travel, будут работать и в новом каталоге.

Процедуры миграции в Spark

Аргументы для процедуры register_table():

Название аргумента	Требуется?	Тип	Описание
table	Да	String	Название целевой таблицы, которую нужно зарегистрировать
metadata_file	Да	String	Файл метаданных, который будет зарегистрирован как текущий файл метаданных для новой целевой таблицы

Вывод для процедуры register_table():

Название вывода	Тип	Описание
current_snapshot_id	Long	Текущий идентификатор снимка новой зарегистрированной таблицы Iceberg
total_records_count	Long	Общее количество записей в новой зарегистрированной таблице Iceberg
total_data_files_count	Long	Общее количество файлов данных в новой зарегистрированной таблице Iceberg

Аргументы для процедуры snapshot():

Название аргумента	Требуется?	Тип	Описание
source_table	Да	String	Имя таблицы, для которой будет создан снимок
table	Да	String	Имя новой таблицы Iceberg, которая будет создана
location	Да	String	Местоположение новой таблицы (по умолчанию делегируется каталогу)
properties	Нет	map<string, string>	Свойства, которые будут добавлены в новую таблицу

Аргументы для процедуры snapshot():

Название вывода	Тип	Описание
imported_files_count	Long	Количество файлов, добавленных в новую таблицу

Примеры команд для миграции Spark

register_table():

```
CALL target_catalog.system.register_table(
    'target_catalog.db1.table1',
    '/path/to/source_catalog_warehouse/db1/table1/metadata/xxx.json'
)
```

Эта команда позволяет зарегистрировать таблицу из одного каталога в другом. Важно, что таблица останется в исходном каталоге, и данные будут доступны в новом каталоге без удаления из старого.

snapshot():

```
CALL target_catalog.system.snapshot(
    'source_catalog.db1.table1',
    'target_catalog.db1.table1'
)
```

Эта команда создает снимок таблицы из исходного каталога и регистрирует его в целевом каталоге. Снимок включает все данные и метаданные, но изменения в целевом каталоге не влияют на исходный каталог.

Пример команды для миграции CLI-Tool

Команда для миграции таблицы из каталога AWS Glue в каталог Nessie:

```
java -jar iceberg-catalog-migrator-cli-0.2.0.jar migrate \
  --source-catalog-type GLUE \
  --source-catalog-properties warehouse=s3a://bucket/gluecatalog/,io-impl=org.apache.iceberg.aws.s3.S3FileIO \
  --source-catalog-hadoop-conf fs.s3a.secret.key=$AWS_SECRET_ACCESS_KEY,fs.s3a.access.key=$AWS_ACCESS_KEY_ID \
  --target-catalog-type NESSIE \
  --target-catalog-properties uri=http://localhost:19120/api/v1,ref=main,warehouse=s3a://bucket/nessie/,io-
impl=org.apache.iceberg.aws.s3.S3FileIO \
  --identifiers db1.nominees
```

- Когда эта команда будет выполнена, она мигрирует таблицу db1.nominees (указанную флагом --identifiers) из каталога AWS Glue (указанного через флаг --source-catalog-type) для учетной записи AWS, соответствующей AWS-ключам (указанным через --source-catalog-hadoop-conf), в каталог Nessie (указанный через флаг --target-catalog-type), работающий на локальном хосте на порту 19120 (указано в uri внутри --target-catalog-properties).

Рекомендации по миграции

- Во время миграции не рекомендуется выполнять операции записи в исходный каталог, так как это может привести к несоответствиям данных в целевом каталоге.

- Для успешной миграции лучше использовать поэтапный подход (например, с помощью регулярных выражений для выбора таблиц по namespaces или именам), чтобы минимизировать риск ошибок.

- Хорошей практикой является использование промежуточного уровня для проверки правильности конфигурации перед окончательной миграцией.

  Эти методы и рекомендации помогут эффективно управлять каталогами Iceberg, обеспечивая миграцию данных и метаданных с минимальными рисками потери данных или нарушения целостности.

Заключение и рекомендации

- Выбор подходящего каталога зависит от используемой инфраструктуры и требований к функциональности.
- Каталоги Iceberg обеспечивают эффективное управление метаданными в распределенных системах, поддерживают высокую производительность и масштабируемость.
- Миграция между каталогами позволяет гибко адаптироваться к изменяющимся требованиям, без необходимости переноса данных.

Iceberg Catalogs

Зачем нужны каталоги в Iceberg?

Каталоги в Iceberg необходимы для управления метаданными таблиц, контроля доступа и поддержания согласованности при работе с большими данными. Они определяют местоположение таблиц и метаданных, что помогает пользователям находить и использовать таблицы в распределенной среде.

Типы каталогов iceberg:

- Hadoop Catalog
- Hive Catalog
- AWS Glue Catalog
- Nessie Catalog
- REST Catalog
- JDBC Catalog

Hadoop Catalog

- **Преимущества:**

  Интегрируется с экосистемой Hadoop и HDFS, что делает его удобным для пользователей, уже использующих Hadoop для хранения данных.

- **Недостатки:**

  Не поддерживает многотабличные транзакции и ограничен возможностями Hadoop, что снижает гибкость для использования в других средах и с другими хранилищами данных.

- **Когда использовать:**

  Подходит для пользователей, которые уже используют Hadoop и HDFS, и хотят интегрировать Iceberg для управления метаданными без необходимости в сложных функциях Iceberg.

Hive Catalog

- **Преимущества:**

  Интегрируется с Apache Hive и совместим с инструментами и SQL-запросами Hive, что облегчает миграцию на Iceberg в существующих инфраструктурах Hive.

- **Недостатки:**

  Ограничен функциональностью Hive и не поддерживает функции Iceberg, такие как версионирование и многотабличные транзакции, что ограничивает его возможности. Также необходимо развернуть доп. Сервис.

- **Когда использовать:**

  Полезен для пользователей, которые уже работают с Hive и хотят интегрировать Iceberg без необходимости в дополнительных функциях, таких как многотабличные транзакции или версионирование данных.

AWS Glue Catalog

- **Преимущества:**

  Управляемая услуга, легко интегрируется с другими сервисами AWS, автоматически масштабируемая, поддерживает высокую доступность и отказоустойчивость.

- **Недостатки:**

  Привязанность к AWS, не поддерживает многотабличные транзакции, работает только в экосистеме AWS, что ограничивает возможность использования в мультиоблачной среде.

- **Когда использовать:**

  Для пользователей, которые уже работают в AWS и не планируют переходить на другие облачные платформы.

Nessie Catalog

- **Преимущества:**

  Поддержка версионирования, мульти-табличные транзакции, возможность работать в мультиоблачной среде, независимость от конкретного облака.

- **Недостатки:**

  Не все инструменты поддерживают Nessie, требуется управление собственной инфраструктурой (хотя есть хостируемая версия от Dremio).

- **Когда использовать:**

  Если нужны мульти-табличные транзакции и поддержка версионирования данных. Хорош для пользователей, которые хотят избегать зависимости от конкретных облачных провайдеров.

REST Catalog

- **Преимущества:**

  Простота развертывания, поддержка мульти-табличных транзакций, облачная независимость.

- **Недостатки:**

  Необходимость разрабатывать или использовать сторонний сервис для обработки REST-запросов, отсутствие публичной реализации бэкенда для поддержки REST на момент написания текста.

- **Когда использовать:**

  Для гибких, настраиваемых решений, которые могут интегрироваться с различными хранилищами данных и требующих мульти-табличных транзакций.

JDBC Catalog

- **Преимущества:**

  Простота интеграции с существующими JDBC-совместимыми базами данных, высокая доступность при использовании облачных сервисов (например, Amazon RDS), облачная независимость.

- **Недостатки:**

  Не поддерживает многотабличные транзакции, требует JDBC-драйвера для всех инструментов и движков.

- **Когда использовать:**

  Для пользователей, использующих базы данных, совместимые с JDBC, или тех, кто нуждается в высокой доступности и облачной независимости.

Преимущества использования каталогов Iceberg?

Управление метаданными: Каталоги управляют метаданными таблиц, что важно для работы с большими данными.

Согласованность и надежность: Обеспечивают согласованность при работе с несколькими читателями и писателями данных, поддерживают транзакции и атомарные операции.

Масштабируемость и гибкость: Обеспечивают масштабируемость и могут работать в различных облаках и инфраструктурах, позволяя пользователю легко переносить данные и таблицы между каталогами.

Миграция каталогов:

Общие замечания:

- Миграция каталогов Iceberg — это сравнительно легкий процесс, так как данные хранятся в data lake, и миграция касается только метаданных (путей к метафайлам). Однако для успешной миграции необходим правильный план для обработки операций записи и использования инструментов.
- Миграция уменьшает риски зависимости от конкретного поставщика и помогает избежать блокировки системы каталогов.

Когда нужно мигрировать каталоги:

- Если вы хотите сменить каталог для использования в продакшн-среде, например, для перехода с Hadoop на более подходящий каталог.
- Если нужно воспользоваться дополнительными возможностями, которые отсутствуют в текущем каталоге.
- При смене расположения среды, например, при миграции из on-premise в облако (например, с Hive Metastore на AWS Glue).

Методы миграции:

Использование инструмента миграции Iceberg Catalog Migration CLI:

- Это инструмент командной строки, позволяющий переносить таблицы между каталогами без копирования данных.
- Поддерживаются такие каталоги как AWS Glue, Nessie, Dremio Arctic, Hadoop, Hive, REST, JDBC.
- Два основных режима: migrate (перемещает таблицы в новый каталог) и register (регистрирует таблицы в новом каталоге без удаления из исходного).
- Процесс миграции не создает копии данных, а лишь переносит метаданные и историю таблиц.

Использование движка, например Apache Spark:

- В Spark можно использовать процедуры register_table() и snapshot() для регистрации или создания легких копий таблиц в новом каталоге.
- Процедура register_table() создаёт копию таблицы, используя исходные файлы данных. Изменения в целевом каталоге не влияют на источник, если только данные не удаляются.
- Процедура snapshot() работает аналогично, но изменения затрагивают только целевой каталог и не влияют на

Процедуры миграции в Spark

register_table():

- Создает легкую копию таблицы в целевом каталоге, сохраняя историю и метаданные.
- Изменения в целевом каталоге не затрагивают данные в исходном каталоге, если только не происходит физическое удаление файлов данных.
- Используется для тестирования миграции, если не нужно вносить изменения в таблицу после миграции.

snapshot():

- Создает копию таблицы с изменением местоположения файлов данных в новом каталоге.
- Это позволяет использовать миграцию для переноса данных в новый каталог с изменением пути данных.
- История таблицы сохраняется, и операции, такие как time travel, будут работать и в новом каталоге.

Процедуры миграции в Spark

Аргументы для процедуры register_table():

Название аргумента	Требуется?	Тип	Описание
table	Да	String	Название целевой таблицы, которую нужно зарегистрировать
metadata_file	Да	String	Файл метаданных, который будет зарегистрирован как текущий файл метаданных для новой целевой таблицы

Вывод для процедуры register_table():

Название вывода	Тип	Описание
current_snapshot_id	Long	Текущий идентификатор снимка новой зарегистрированной таблицы Iceberg
total_records_count	Long	Общее количество записей в новой зарегистрированной таблице Iceberg
total_data_files_count	Long	Общее количество файлов данных в новой зарегистрированной таблице Iceberg

Аргументы для процедуры snapshot():

Название аргумента	Требуется?	Тип	Описание
source_table	Да	String	Имя таблицы, для которой будет создан снимок
table	Да	String	Имя новой таблицы Iceberg, которая будет создана
location	Да	String	Местоположение новой таблицы (по умолчанию делегируется каталогу)
properties	Нет	map<string, string>	Свойства, которые будут добавлены в новую таблицу

Аргументы для процедуры snapshot():

Название вывода	Тип	Описание
imported_files_count	Long	Количество файлов, добавленных в новую таблицу

Примеры команд для миграции Spark

register_table():

```
CALL target_catalog.system.register_table(
    'target_catalog.db1.table1',
    '/path/to/source_catalog_warehouse/db1/table1/metadata/xxx.json'
)
```

Эта команда позволяет зарегистрировать таблицу из одного каталога в другом. Важно, что таблица останется в исходном каталоге, и данные будут доступны в новом каталоге без удаления из старого.

snapshot():

```
CALL target_catalog.system.snapshot(
    'source_catalog.db1.table1',
    'target_catalog.db1.table1'
)
```

Эта команда создает снимок таблицы из исходного каталога и регистрирует его в целевом каталоге. Снимок включает все данные и метаданные, но изменения в целевом каталоге не влияют на исходный каталог.

Пример команды для миграции CLI-Tool

Команда для миграции таблицы из каталога AWS Glue в каталог Nessie:

```
java -jar iceberg-catalog-migrator-cli-0.2.0.jar migrate \
  --source-catalog-type GLUE \
  --source-catalog-properties warehouse=s3a://bucket/gluecatalog/,io-impl=org.apache.iceberg.aws.s3.S3FileIO \
  --source-catalog-hadoop-conf fs.s3a.secret.key=$AWS_SECRET_ACCESS_KEY,fs.s3a.access.key=$AWS_ACCESS_KEY_ID \
  --target-catalog-type NESSIE \
  --target-catalog-properties uri=http://localhost:19120/api/v1,ref=main,warehouse=s3a://bucket/nessie/,io-
impl=org.apache.iceberg.aws.s3.S3FileIO \
  --identifiers db1.nominees
```

- Когда эта команда будет выполнена, она мигрирует таблицу db1.nominees (указанную флагом --identifiers) из каталога AWS Glue (указанного через флаг --source-catalog-type) для учетной записи AWS, соответствующей AWS-ключам (указанным через --source-catalog-hadoop-conf), в каталог Nessie (указанный через флаг --target-catalog-type), работающий на локальном хосте на порту 19120 (указано в uri внутри --target-catalog-properties).

Рекомендации по миграции

- Во время миграции не рекомендуется выполнять операции записи в исходный каталог, так как это может привести к несоответствиям данных в целевом каталоге.
- Для успешной миграции лучше использовать поэтапный подход (например, с помощью регулярных выражений для выбора таблиц по namespaces или именам), чтобы минимизировать риск ошибок.
- Хорошей практикой является использование промежуточного уровня для проверки правильности конфигурации перед окончательной миграцией.

  Эти методы и рекомендации помогут эффективно управлять каталогами Iceberg, обеспечивая миграцию данных и метаданных с минимальными рисками потери данных или нарушения целостности.

Заключение и рекомендации

- Выбор подходящего каталога зависит от используемой инфраструктуры и требований к функциональности.
- Каталоги Iceberg обеспечивают эффективное управление метаданными в распределенных системах, поддерживают высокую производительность и масштабируемость.
- Миграция между каталогами позволяет гибко адаптироваться к изменяющимся требованиям, без необходимости переноса данных.