Сценарий работы сенсоров в AirFlow

Реализовать зависимости между несколькими дагами Airflow (cross-dag dependecies) можно несколькими способами:

1) Сенсор ExternalTaskSensor

2) Сенсор SqlSensor

3) Асинхронный сенсор (smart sensors) - ExternalTaskSensorAsync

4) Служба "умных" сенсоров (smart sensors) - SmartSensorOperator (поддерживается до версии Airflow 2.4)

5) TriggerDagRunOperator

6) Data-aware scheduling (доступно с версии Airflow 2.4)

7) Airflow REST API

Сенсоры - это тип Airflow-операторов, которые с определенными интервалами проверяют выполнение какого-то условия, и если условие выполняется, сенсор помечается как успешный, и workflow переходит к следующим таскам в даге. Все сенсоры имеют следующие основные атрибуты:

- mode - режим выполнения:

- poke - в этом режиме сенсор занимает рабочий слот воркера в течение всего времени работы. Этот режим оптимальный, когда предполагается работа сенсора в течение короткого промежутка времени, ;
- reschedule - в этом режиме сенсор занимает слот воркера только в моменты запуска проверок, а между проверками освобождает слот. Этот режим оптимальный, если предполагается длительная работа сенсора, т.к. он менее ресурсно-затратный, но не гибкий, т.к. он может возобновлять свою работу только через определенные интервалы времени, а не при наступлении событий (триггеров);
- poke_interval - интервал (в секундах) между запусками проверок;
- timeout - максимальный интервал времени, в течение которого работает сенсор. Если условие для сенсора не выполнилось за этот интервал, таска падает с ошибкой (дефолтное значение = 7 дней);
- soft_fail - таска помечается как skipped, если условие для сенсора не выполнилось за период таймаута.

Для реализации cross-dag-dependencies чаще всего используется тип сенсора **ExternalTaskSensor**. Сенсор этого типа ожидает выполнение какого-либо таска/набора тасков в другом (внешнем) даге, либо всего дага целиком на определенную логическую дату execution_date этого внешнего дага (по дефолту на дату его последнего выполнения).

```
child_task1 = ExternalTaskSensor(
        task_id="child_task1",
        external_dag_id=parent_dag.dag_id,
        external_task_id=parent_task.task_id,
        timeout=600, allowed_states=['success'],
        failed_states=['failed', 'skipped'],
        mode="reschedule",
)
```

Для каждой ветки дага можно определить отдельный сенсор, который будет ожидать выполнения каких-либо тасков в разных дагах. Так же, для одного таска можно определить несколько сенсоров, и таким образом реализовать зависимость от нескольких тасков/дагов. Если дополнительно требуется, чтобы при сбросе "родительского" дага так же сбрасывался и "дочерний" даг, в дочернем даге небходимо дополнительно определить таск ExternalTaskMarker и при сбросе родительского дага указывать опцию Recursive

```
parent_task = ExternalTaskMarker(
        task_id="parent_task",
        external_dag_id="example_external_task_marker_child",
        external_task_id="child_task1",
)
```

Ссылка на описание ExternalTaskSensor - https://registry.astronomer.io/providers/apache-airflow/modules/externaltasksensor

SqlSensor - еще один тип сенсора, который можно использовать для реализации cross-dag dependecies, если в качестве триггера для сенсора использовать событие записи в таблицы логов строки с информацией об успешном/неуспешном завершении зависимых дагов
Ссылка на описание сенсора: https://registry.astronomer.io/providers/apache-airflow/modules/sqlsensor

Асинхронные сенсоры - сенсоры, которые используют функционал библиотеки asyncio для того, чтобы не держать слоты воркера во время ожидания завершения какого-то внешнего процесса. Вместо этого процесс опроса внешнего выгружается в виде триггера в triggerer (специальный сервис Airflow, аналогичный шедулеру или воркеру), и слоты освобождаются для других задач. Преимущество асинхронных сенсоров по сравнению с обычными заключается в том, что они привязаны не к определенным периодам времени, а к триггерам. Есть только 2 встроенных класса deferred-сенсоров в Airflow - TimeSensorAsync и DateTimeSensorAsync, для всех остальных типов сенсоров необходимо создавать собственные кастомные классы deferrable-операторов и триггеров, либо использовать готовый функционал из внешних библиотеки. Ссылка на описание процесса создания классов - https://airflow.apache.org/docs/apache-airflow/stable/concepts/deferring.html#smart-sensors. Асинхронный сенсор для cross-dag-dependencies - ExternalTaskSensorAsync реализован в библиотеке astronomer-providers

Умные сенсоры (smart sensors) - технология применения смарт-сенсоров является аналогичной deferrable operators, но при этом считается устаревшей. Ссылка на описание сервиса: https://airflow.apache.org/docs/apache-airflow/2.3.3/concepts/smart-sensors.html#smart-sensors

TriggerDagRunOperator - встроенный Airflow-оператор, при помощи которого можно из одного дага "триггерить" запуск другого дага (дагов) в любом точке этого дага. Метод больше подходит для кейсов, когда один даг имеет несколько зависимых от него дагов, которые он должен "триггерить"
Ссылка на описание оператора: https://docs.astronomer.io/learn/cross-dag-dependencies#triggerdagrunoperator

Data-aware scheduling - Зависимости между дагами загрузки могут быть определены через встроенный механизм датасетов. В качестве датасета может быть использована любая строка в формате URI-идентификатора, записанная с применением ASCII-символов. Например:

```
mydbms_dataset = Dataset(‘mydbms’)]
```

Таск, который должен обновлять датасет(ы) (producer task), определяется при помощи атрибута outlets:

```
outlets=[Dataset(‘mydbms/mytable’), Dataset(’mydbms/mytable/mycolumn’)]
```

даг, который "слушает" изменения датасетов (consumer dag), должен иметь параметр

```
    schedule=[
    Dataset('mydbms'),
    Dataset('mydbms/mytable’)
    Dataset('mydbms/mytable/mycolumn')
```

При этом обновлением каждого датасета считается событие, когда таск-producer завершается со статусом successful
Ссылка на описание способа - https://docs.astronomer.io/learn/airflow-datasets

Airflow REST API - триггерит запуск дага из другого дага при помощи POST-запроса к эндпойнту DAGRuns (https://airflow.apache.org/docs/apache-airflow/stable/stable-rest-api-ref.html#operation/post_dag_run)

В даге запрос реализуется при помощи оператора SimpleHttpOperator, который может быть использован аналогично оператору TriggerDagRunOperator. Используется в-основном в тех случаях, когда даги находятся в разных Airflow-окружениях.
Ссылка на описание способа - https://docs.astronomer.io/learn/cross-dag-dependencies#airflow-api

Что решили применять у нас в GP:

В результате обсуждения приняты следующие решения по реализации cross-dag dependecies:

будем использовать сенсоры типа **SqlSensor**, которые будут делать запросы к таблицам логов загрузки данных meta.detail_process_table_status и meta.preaggregate_process_table_status (при загрузке витрин и преагрегатов) и meta.oper_process_table_status (при загрузке таблиц dds-слоя)

- *что и куда пишем;*
  Будут задействованы служебные таблицы для записи логов загрузки таблиц;
- *с какой частотой сканируем служебную таблицу;*
  Таблицы будут проверяться каждые 15 минут, чтобы не перегружать сервер частыми запросами (значение атрибута poke_interval = 15*60);
- *как долго сенсор работает (как определять таймаут для каждого сенсора);*
  Величина таймаута будет определяться отдельно для каждого сенсора, и будет зависеть от максимальной продолжительности выполнения таска/тасков, загружающих данные в связанные таблицы.
  Варианты для определения таймаута, которые сейчас рассматриваются -

1) рассчитать максимальное время загрузки каждой таблицы на основе статистических данных, и прописать вручную это время как таймауты для сенсоров в загрузках зависимых от них таблиц (этот способ наименее гибкий из всех);
2) в конфиги dbt-моделей добавить дополнительный атрибут, который будет содержать максимальную (среднюю) продолжительность загрузки моделей, и рассчитывать таймаут для сенсора как максимальное значение этого атрибута из всех зависимых таблиц;
3) считать по дефолту максимальное время загрузки таблицы как размер интервала между ее загрузками (подходит только для таблиц ods-слоя);
4) синхронизировать расписания загрузки таблиц dds-слоя с расписаниями загрузок зависимых ods-таблиц;

- *что происходит если за определенное время (какое) не находим записи об обновлении данных;*
  Если за заданное время записи в логах об успешной загрузке всех зависимых таблиц так и не появились, сенсор помечается как Failed, и выполнение дага завершается с ошибкой (soft_fail = False)
- *как это должно выглядеть в конфигах;*
  в yml-конфиги для схем dbt-моделей предлагается добавить новый атрибут required_tables, который будет содержать список связанных таблиц. Если dbt-генератор дагов при сканировании конфиг-файла находит этот атрибут, в даге создается дополнительный оператор SqlSensor, которому на вход через аргумент params передается список таблиц, перечисленных в атрибуте required_tables
  (так же нужно рассмотреть возможность передачи списка зависимых таблиц через макросы/рефы dbt)
- *как будет выглядеть загрузка объекта, который должен ждать 2 и более дочерних объектов (много сенсоров или один сложный сенсор)*
  Предполагается, что зависимость от n дагов должна будет реализована через один сенсор, а не через n сенсоров. При этом sql-запрос для сенсора должен выглядеть примерно так (для загрузки витрины dm.dim_user):

```
--check_table_current_statuses.sql

with table_last_statuses  as  (

        SELECT table_name, first_value(status) over (partition by table_name order by created_dttm desc) as last_status

        FROM meta.detail_process_table_status

        where table_name in ('dds.h_user', 'dds.s_user')

)

select bool_and(last_status = 'SUCCESS') as all_table_success

from table_last_statuses

having not bool_or(last_status = 'START') ;
```

При каждом запуске сенсор будет проверять, таблице meta.detail_process_table_status нет записей с текущим статусом 'START'. Если такие записи есть (т.е., хоть одна зависимая таблица в данный момент загружается), сенсор "засыпает" до следующего запуска, если таких записей нет, сенсор проверяет первую возвращаемую запись на соответствие критериям успешного /неуспешного выполнения

```
def _success_criteria(record):

        return record.all_table_success

def _failure_criteria(record):

        return not record.all_table_success
```

```
#описание сенсора
waiting_for_child_tables = SqlSensor(
        task_id='waiting_for_tables',
        conn_id='greenplum',
        sql='check_table_current_statuses.sql',
        parameters={
                'table_list': #список таблиц из атрибута requires_tables конфиг-файла
        },
        success=_success_criteria,
        failure=_failure_criteria,
        fail_on_empty=False,
        poke_interval=15*60,
        mode='reschedule',
        timeout=60 * 5
)
```

- (Т.е. если все загрузки зависимых таблиц имеют статус 'SUCCESS', будут выполняться критерии для успешного завершения сенсора, если хоть одна загрузка не имеет такого статуса (т.е. имеет статус 'FAILED'), то выполняется критерий завершения сенсора с ошибкой)
- По расчету таймаутов для сенсоров пока выбрали следующее решение:
  В конфиги dbt-моделей вместе с required_tables добавляется атрибут sensor_timeout, в котором будет проставляться значение, рассчитанное вручную на основе статистических данных по времени выполнения загрузок связанных таблиц