



Курс Apache Airflow

Содержание лекции

1. Назначение Apache Airflow
2. Основные особенности
3. Передача данных между задачами
4. Что такое глобальные переменные в Apache Airflow и зачем они нужны
5. Как создавать глобальные переменные и использование их в коде
6. Основная идея рабочих процессов в управлении данными
7. Поток управления
8. Рабочие нагрузки
9. Передача данных между задачами
10. Jinja



Назначение Apache

Airflow – платформа для **создания, оркестрации, управления** расписанием и **мониторингом** Workflow-процессов загрузок данных.

Основные сущности рабочего процесса на Apache Airflow:

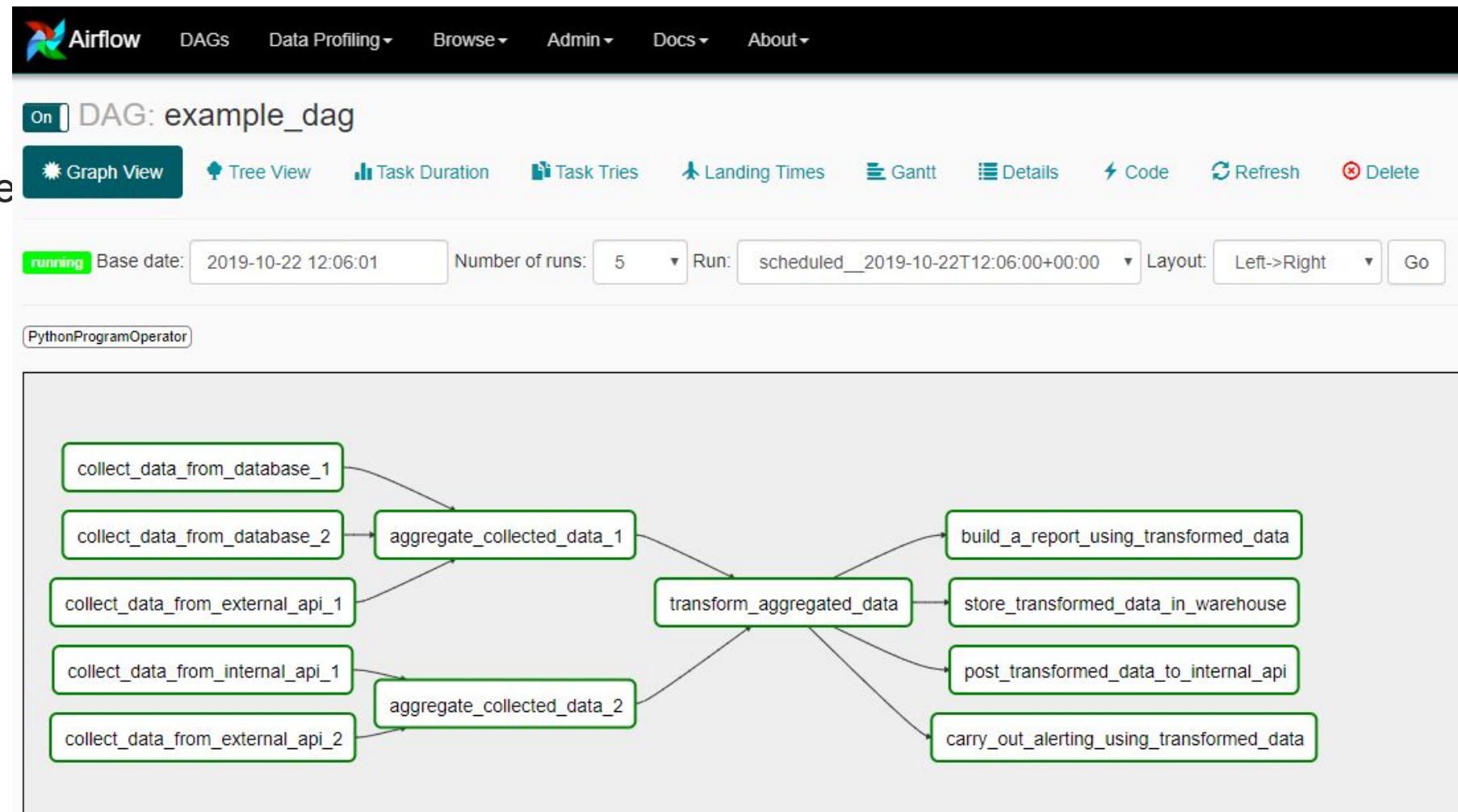
- Направленные ациклические графы (DAG)
- Планировщик (Scheduler)
- Операторы (Operators)
- Задачи (Tasks)



Основные особенности

Airflow — это платформа для программного создания, планирования и мониторинга рабочих процессов.

- Airflow используется для создания рабочих процессов в виде направленных ациклических графов (DAG) задач.
- Планировщик Airflow выполняет задачи на множестве рабочих процессов, следуя указанным зависимостям.
- Богатые утилиты командной строки упрощают выполнение сложных операций с DAGs.
- Богатый пользовательский интерфейс позволяет легко визуализировать конвейеры, работающие в производственной среде, отслеживать ход выполнения и при необходимости устранять неполадки.



Передача данных между задачами

XComs («кросс-коммуникации»)

система, в которой вы можете задавать задачи для передачи и извлечения небольших фрагментов метаданных (наверно надо написать что только для маленьких данных)

Загрузка и скачивание больших файлов из общей системы хранения (либо запущенной вами, либо части общедоступного облака)



Airflow отправляет задачи для выполнения на рабочих процессах по мере освобождения места (пула ресурсов), поэтому нет гарантии, что все задачи в вашем DAGe будут выполняться на одном Worker или на одной и той же машине.

ЧТО ТАКОЕ ПЕРЕМЕННЫЕ В PARASCH AIRFLOW И ЗАЧЕМ ОНИ НУЖНЫ

Возможность передачи/приема информации через XCom имеется у каждого экземпляра задачи. XCom предназначен для взаимодействия внутри одного DAG'а, в то время как переменные являются глобальными, предназначены для общей конфигурации и существуют только во время выпол



КАК СОЗДАВАТЬ ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ И ИСПОЛЬЗОВАНИЕ ПЕРЕМЕННЫХ В КОДЕ

- через ключ set:

```
airflow variables set my_key "1"
```

- экспортировать эту переменную в файл

```
airflow variables export vars.json
```

- достать переменную через Variable:

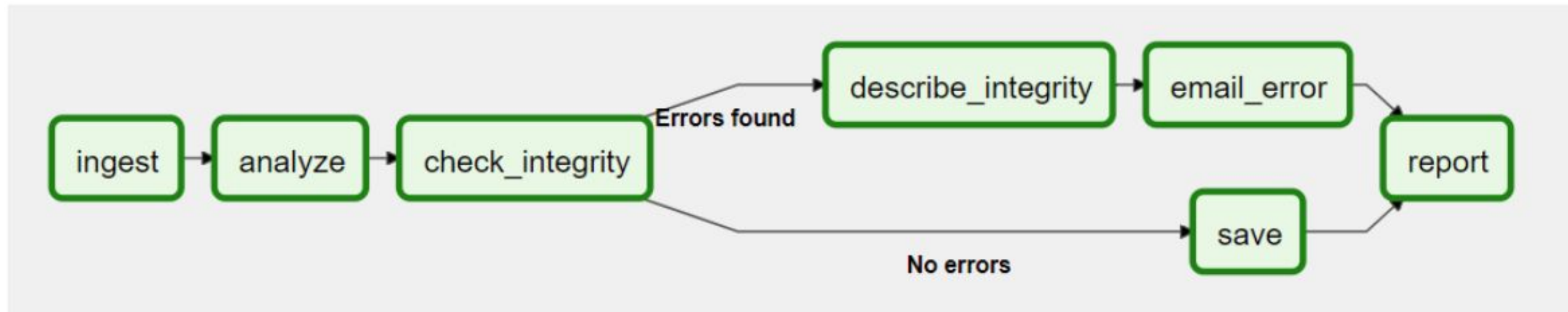
```
from airflow.models import Variable def result(ti): my_var = Variable.get("my_key")
```

- через шаблоны переменные также просто достаются:

```
ba = BashOperator( task_id="ba", bash_command="echo my_key = {{ var.value.my_key }}" )
```

Основная идея рабочих процессов в управлении

данными Airflow — это платформа, которая позволяет создавать и запускать рабочие процессы. Рабочий процесс представлен как DAG (направленный ациклический граф) и содержит отдельные части работы, называемые задачами, организованные с учетом зависимостей и потоков данных.



DAG определяет зависимости между задачами и порядок их выполнения и выполнения повторных попыток

Сами задачи описывают, что нужно делать, будь то получение данных, запуск анализа, запуск других систем, скриптов ML, проверки качества данных, e-mail рассылка и т. д.

Поток управления

- DAGs предназначены для многократного запуска, и несколько их запусков могут выполняться параллельно.
- DAGs параметризуются, всегда включая интервал, для которого они «выполняются» (интервал данных), но также и с другими необязательными параметрами.
- Задачи имеют зависимости друг от друга. Вы увидите это в DAG либо с помощью операторов >> и <<:

```
first_task >> [second_task, third_task]
fourth_task << third_task
```

- Или с помощью методов `set_upstream` и `set_downstream`:

```
first_task.set_downstream([second_task, third_task])
fourth_task.set_upstream(third_task)
```

- Эти зависимости составляют «ребра» графа и то, как Airflow определяет, в каком порядке выполнять задачи. По умолчанию задача будет ждать, пока все ее вышестоящие задачи не будут выполнены успешно, прежде чем она запустится, но это может быть настроено также с помощью таких функций, как `Branching`, `LatestOnly` и `Trigger Rules`.

Рабочие нагрузки

DAГ

BaseOperator Airflow

Operators (Операторы)

предопределенные задачи, которые можно быстро объединить для создания большинства частей ваших DAGs

Sensors (Сенсоры)

особый подкласс операторов, полностью ожидающих возникновения внешнего события

TaskFlow (поток задач)

который представляет собой пользовательскую функцию на основе Python, упакованную как Task

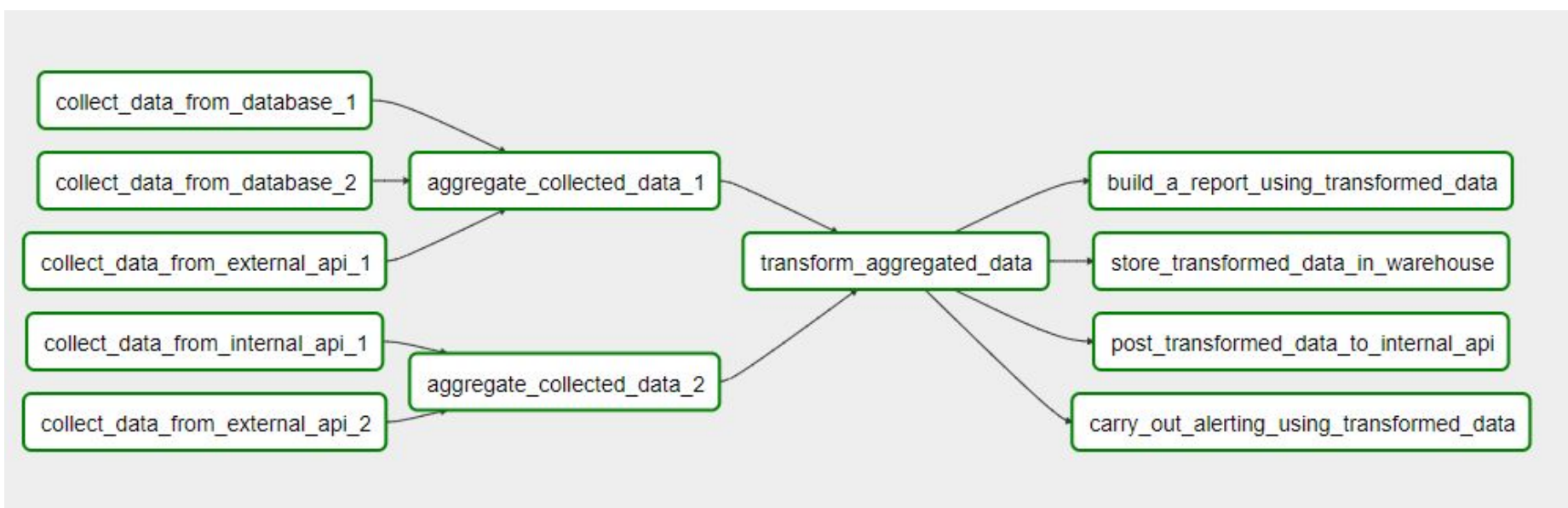
Внутренне все они на самом деле являются подклассами BaseOperator Airflow, а концепции Task и Operator несколько взаимозаменяемы, но полезно думать о них как об отдельных концепциях — по сути, Operators (Операторы) и Sensors (Сенсоры) являются шаблонами, и когда вы вызываете один из них в файле DAG-а, вы делаете задачу.

Передача данных между задачами

По мере того, как вы создаете свои DAG, они становятся очень сложными и большими, поэтому Airflow предоставляет несколько механизмов для того, чтобы сделать это более устойчивым и удобным для восприятия:

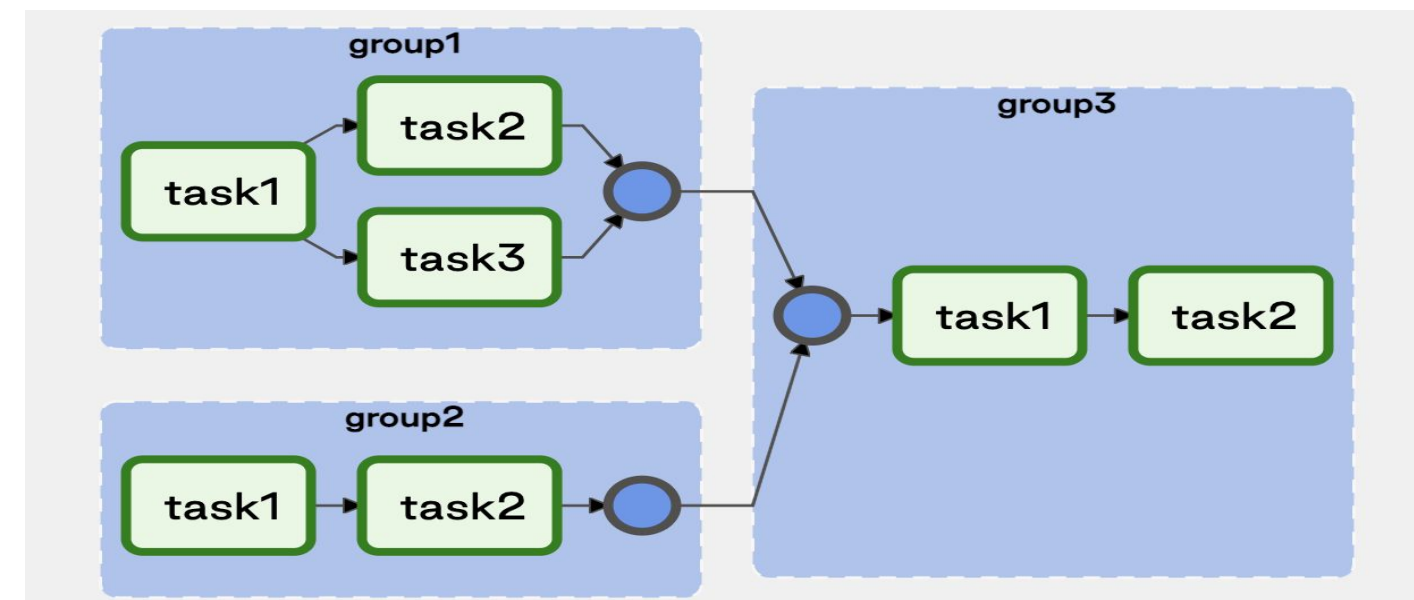
SubDAG

они позволяют вам создавать «повторно используемые» DAGs, которые вы можете встраивать в другие



TaskGroups

позволяют визуально группировать задачи в пользовательский интерфейс



Существуют также функции, позволяющие легко предварительно настроить доступ к центральному ресурсу, такому как хранилище данных, в форме Connections & Hooks, а также для ограничения параллелизма через Pools (пулы)

Jinja

Воздушный поток в полной мере использует мощь Jinja Templating, которая может быть мощным инструментом для работы с макросами.

Например, предположим, что вы хотите использовать BashOperator для передачи даты выполнения в качестве переменной среды в сценарий Bash.

```
# The execution date as YYYY-MM-DD
date = "{{ ds }}"
t = BashOperator(
    task_id='test_env',
    bash_command='/tmp/test.sh ',
    dag=dag,
    env={'EXECUTION_DATE': date})
```

Здесь `{{ds}}` является макросом, и поскольку параметр `env` в `BashOperator` формируется с использованием Jinja, дата выполнения будет предоставлена в виде переменной среды с именем `EXECUTION_DATE` в сценарии Bash.

Вы можете использовать шаблоны Jinja с каждым параметром, помеченным как «шаблонный» в документе. Замена шаблона происходит до вызова оператором функции `pre_execute`.

Спасибо за внимание!