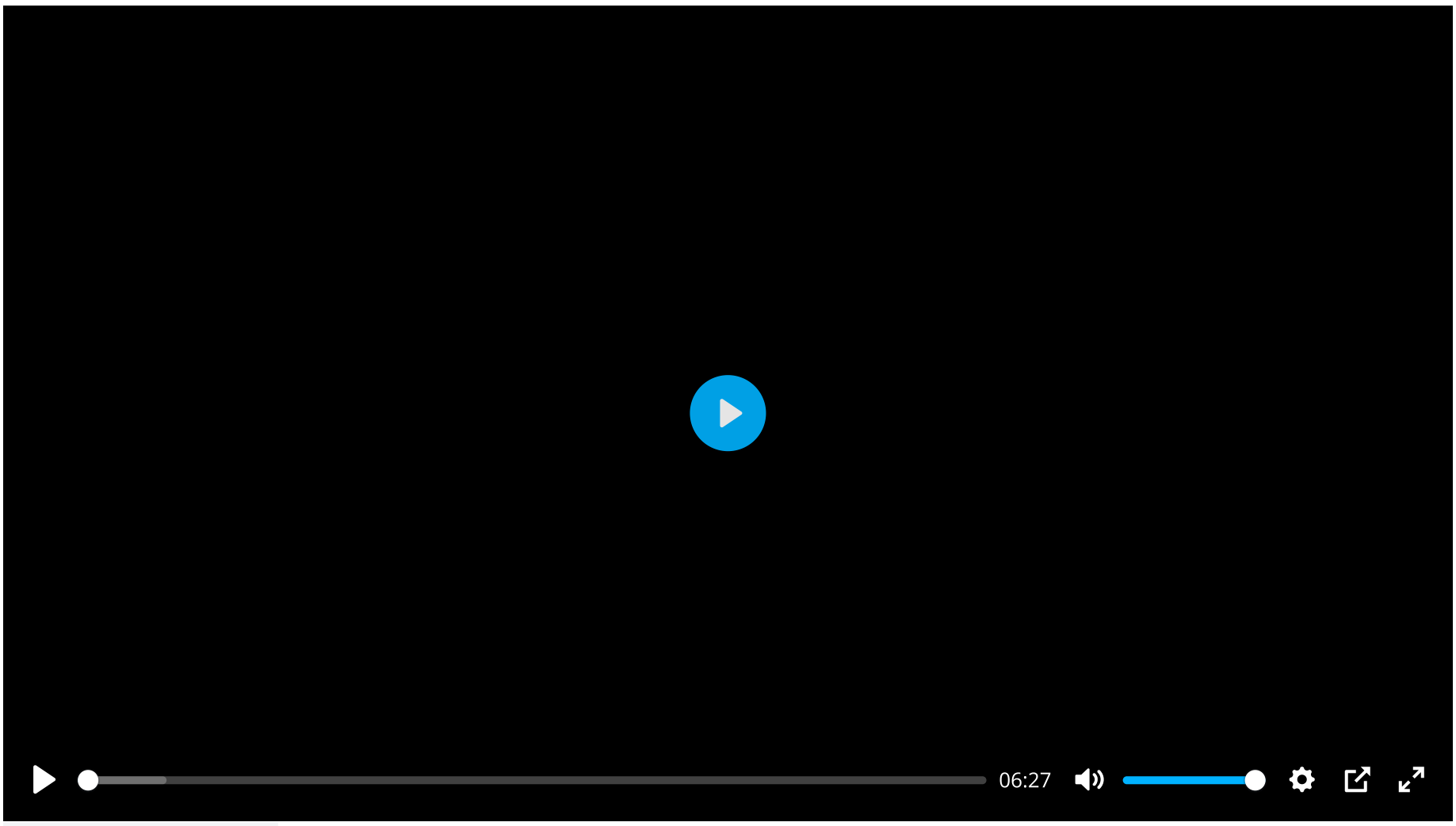
8.4 Работа с компонентами Airflow



Презентация к уроку 8.4.4

Текстовая расшифровка видео:

КОМПОНЕНТЫ AIRFLOW. ПЛАНИРОВЩИК (SCHEDULER)

План:

- Планировщик (Scheduler);
- Пример выполнения DAG по времени;
- Backfilling (catch up).

Планировщик (Scheduler)

Планировщик (Scheduler) – это компонент, который отвечает за задания, расписания, выполнение тасков.

Используя контейнер с scheduler

```
sudo docker exec -it airflow_airflow-scheduler_1  /bin/bash
airflow@7f115549f071:/opt/airflow$ ls
airflow.cfg  …
```

Чтобы редактировать можно скопировать себе

```
sudo docker cp airflow_airflow-scheduler_1:/opt/airflow/airflow.cfg .
```

Чтобы посмотреть настройки Scheduler’a, мы можем скопировать конфигурацию локально, но все изменения, которые мы внесем в файл, копироваться не будут. Сам по себе Scheduler находится в контейнере с суффиксом «Airflow-Scheduler», там также находится процесс Airflow. Именно через этот процесс можно делать, например, отладку.

Важные параметры для Scheduler:

Schedule_interval – это кроновское выражение, которое показывает с какой периодичностью нужно запускать DAG. Если вы его не указываете, то в данном случае DAG запускается один раз во время start_date или вручную. Данный интервал можно указывать через @daily, @ weekly и т.д.

Start_date и End_date – это timestamp/datetime-значение. End_date указывать необязательно, это время к которому DAG завершает свое выполнение. Это время, после которого DAG перестает запускаться. Start_date нужно указывать обязательно. Это время, с которого запускается DAG.

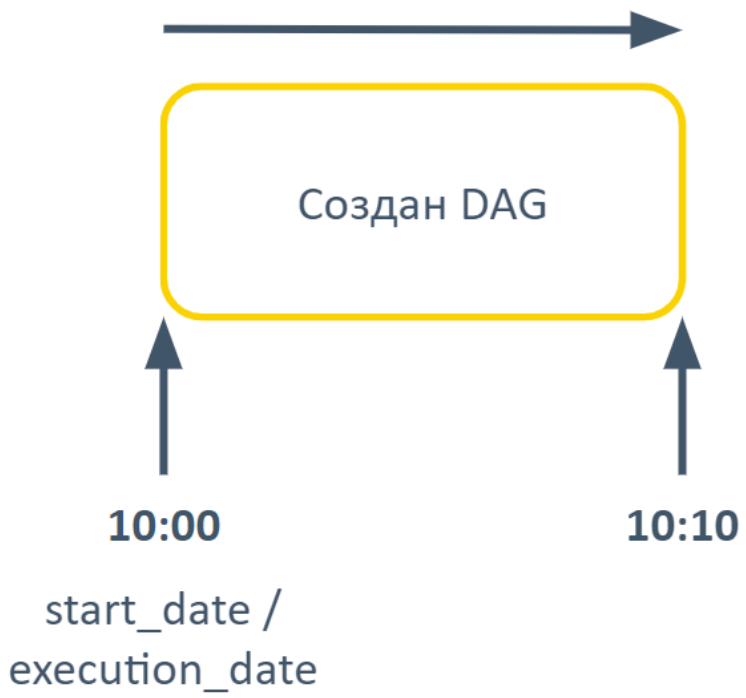
Execution_date – это время фактического выполнения, когда генерируется объект DAG Run, который соответствует запуску DAG, и таски в нем.

Пример выполнения DAG по времени

Рассмотрим вопросы:

- Каким образом работает DAG?
- Каким образом взаимодействуют между собой эти величины?

По указанному start_date или по последнему execution date отмеривается начало DAG. Он генерируется Scheduler'ом, задается на выполнение, но не выполняется в executor'е или в воркерах, а ожидает интервал выполнения. В данном примере он десять минут:



Спустя десять минут у нас происходит первый DAG Run-запуск:

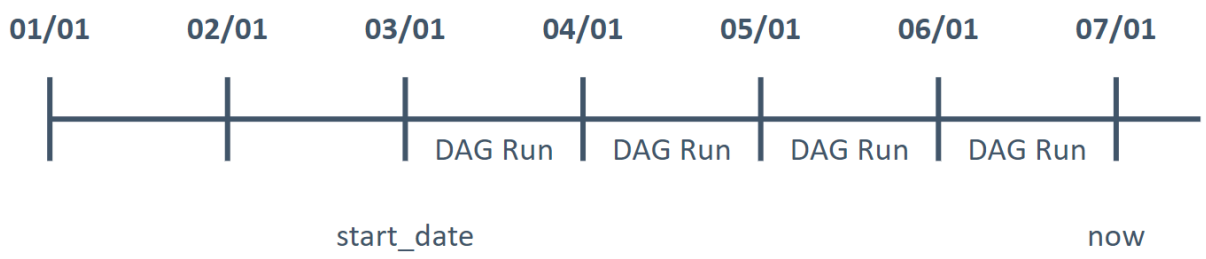
Scheduler ожидает еще десять минут интервала, после чего отправляет на запуск еще один DAG Run:

Если мы задаем интервал scheduling'a, то ожидаем, что за этот интервал должны быть выполнены все соответствующие задачи, связанные с этим DAG. AirFlow это учитывает и ждет интервал либо со start_date, либо с последнего execution date для того, чтобы успели выполниться все процессы, после чего начинает выполнение с этого времени. Если нас это не устраивает, мы можем нажать в Web-UI треугольник, чтобы DAG выполнился сразу.

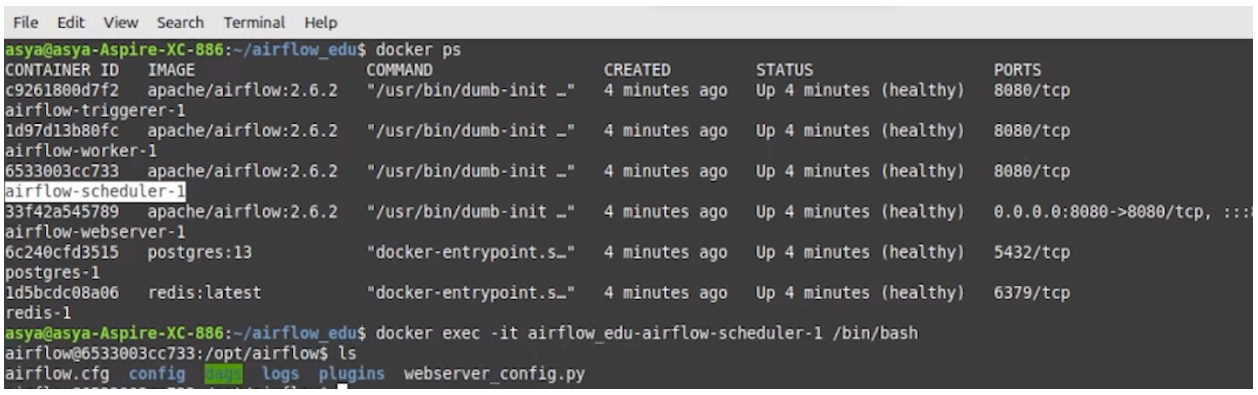
Backfilling (catch up)

Ранее мы говорили о том, что параметр «Catch up» нужно ставить в false, однако если поставить в true, то он может выполнить хоть и полезную функцию, но требующую большое количество ресурсов компьютера. Если нет необходимости, лучше ставить в false.

Если Catch up установлен в true, а start date был раньше сегодняшнего дня или между start date и нынешним моментом больше периодов, чем интервал, то за все промежутки времени запускаются DAG Run. DAG Run запускается с момента start date с заданным scheduling-интервалом. Это удобно, если нужно выполнить задачу за предыдущие несколько месяцев:



Зайдем в контейнер со Scheduler'ом. Посмотрим ID и создадим команду «**docker exec -it**». Команда, которая должна выполниться – «bin/bash»:



Здесь есть файл «config.py», попробуем его посмотреть, выведя первые тридцать строчек:

```
airflow.cfg  config  dags  logs  plugins  webserver_config.py
airflow@6533003cc733:/opt/airflow$ head -n 30 airflow.cfg
[core]
# The folder where your airflow pipelines live, most likely a
# subfolder in a code repository. This path must be absolute.
dags_folder = /opt/airflow/dags

# Hostname by providing a path to a callable, which will resolve the hostname.
# The format is "package.function".
#
# For example, default value "airflow.utils.net.getfqdn" means that result from patched
# version of socket.getfqdn() - see https://github.com/python/cpython/issues/49254.
#
# No argument should be required in the function specified.
# If using IP address as hostname is preferred, use value ``airflow.utils.net.get_host_ip_address``
hostname_callable = airflow.utils.net.getfqdn

# A callable to check if a python file has airflow dags defined or not
# with argument as: `(file_path: str, zip_file: zipfile.ZipFile | None = None)`
# return True if it has dags otherwise False
# If this is not provided, Airflow uses its own heuristic rules.
might_contain_dag_callable = airflow.utils.file.might_contain_dag_via_default_heuristic

# Default timezone in case supplied date times are naive
# can be utc (default), system, or any IANA timezone string (e.g. Europe/Amsterdam)
default_timezone = utc

# The executor class that airflow should use. Choices include
# ``SequentialExecutor``, ``LocalExecutor``, ``CeleryExecutor``, ``DaskExecutor``,
# ``KubernetesExecutor``, ``CeleryKubernetesExecutor`` or the
# full import path to the class when using a custom executor.
executor = SequentialExecutor
airflow@6533003cc733:/opt/airflow$
```

В данном месте присутствует SequentialExecutor:

```
# The executor class that airflow should use. Choices include
# ``SequentialExecutor``, ``LocalExecutor``, ``CeleryExecutor``, ``DaskExecutor``,
# ``KubernetesExecutor``, ``CeleryKubernetesExecutor`` or the
# full import path to the class when using a custom executor.
executor = SequentialExecutor
airflow@6533003cc733:/opt/airflow$
```

На самом деле установлен другой, но в файле конфигурации прописано так. Позже мы объясним причину.

Можно скопировать данный файл, чтобы посмотреть в удобном редакторе.

Убедимся, что это тот же самый файл:

```
File  Edit  View  Search  Terminal  Help
exit
asya@asya-Aspire-XC-886:~/airflow_edu$ sudo docker cp airflow_airflow-scheduler_1:/opt/airflow/airflow.cfg .
[sudo] password for asya:
Error: No such container:path: airflow_airflow-scheduler_1:/opt/airflow/airflow.cfg
asya@asya-Aspire-XC-886:~/airflow_edu$ sudo docker cp airflow_edu-scheduler_1:/opt/airflow/airflow.cfg .
Error: No such container:path: airflow_edu-scheduler_1:/opt/airflow/airflow.cfg
asya@asya-Aspire-XC-886:~/airflow_edu$ sudo docker cp airflow_edu-airflow-scheduler_1:/opt/airflow/airflow.cfg .
Error: No such container:path: airflow_edu-airflow-scheduler_1:/opt/airflow/airflow.cfg
asya@asya-Aspire-XC-886:~/airflow_edu$ sudo docker cp airflow_edu-airflow-scheduler-1:/opt/airflow/airflow.cfg .
asya@asya-Aspire-XC-886:~/airflow_edu$ ls
airflow.cfg  config  dags  docker-compose.yaml  logs  plugins  venv
asya@asya-Aspire-XC-886:~/airflow_edu$ head -n 30 airflow.cfg
head: cannot open 'airflow.cfg' for reading: Permission denied
asya@asya-Aspire-XC-886:~/airflow_edu$ sudo head -n 30 airflow.cfg
[core]
# The folder where your airflow pipelines live, most likely a
# subfolder in a code repository. This path must be absolute.
dags_folder = /opt/airflow/dags

# Hostname by providing a path to a callable, which will resolve the hostname.
# The format is "package.function".
#
# For example, default value "airflow.utils.net.getfqdn" means that result from patched
# version of socket.getfqdn() - see https://github.com/python/cpython/issues/49254.
#
# No argument should be required in the function specified.
# If using IP address as hostname is preferred, use value ``airflow.utils.net.get_host_ip_address``
hostname_callable = airflow.utils.net.getfqdn

# A callable to check if a python file has airflow dags defined or not
# with argument as: `(file_path: str, zip_file: zipfile.ZipFile | None = None)`
# return True if it has dags otherwise False
# If this is not provided, Airflow uses its own heuristic rules.
might_contain_dag_callable = airflow.utils.file.might_contain_dag_via_default_heuristic

# Default timezone in case supplied date times are naive
# can be utc (default), system, or any IANA timezone string (e.g. Europe/Amsterdam)
default_timezone = utc

# The executor class that airflow should use. Choices include
# ``SequentialExecutor``, ``LocalExecutor``, ``CeleryExecutor``, ``DaskExecutor``,
# ``KubernetesExecutor``, ``CeleryKubernetesExecutor`` or the
# full import path to the class when using a custom executor.
executor = SequentialExecutor
asya@asya-Aspire-XC-886:~/airflow_edu$
```

На этом уроке мы познакомились с планировщиком Scheduler и узнали о функциях, которые он в себе несет. Также мы узнали, что такое Backfilling, почему устанавливали Catch up false, в какой последовательности выполняются таски в зависимости от интервала scheduling'а и установленного start date.