

[Презентация к уроку 8.5.1](#)

Текстовая расшифровка видео:

## НАСТРОЙКИ AIRFLOW. ФАЙЛ КОНФИГУРАЦИИ

**План:**

- Дефолтный конфиг (Default config);
- Таймзоны;
- Pools;
- Переменные в Airflow. Задание переменных;
- Считывание кода DAG;
- Источники переменных;
- Разбор примера.

### Дефолтный конфиг (Default config)

Файл конфигурации Airflow помогает настраивать инстанс. Эти настройки очень гибкие. Airflow отличается от многих оркестраторов тем, что в нем можно настраивать практически все.

Мы можем скопировать файл конфигурации, **туда не будут вноситься изменения:**

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

С документацией по параметрам вы можете ознакомиться, перейдя по ссылке:

<https://airflow.apache.org/docs/apache-airflow/stable/configurations-ref.html>.



▶ Запустить стенд

⊗ Дедлайн 04 августа, 23:59 Мск



## Таймзоны

Один из важных параметров – Таймзона. По умолчанию **Таймзона=utc**. Все задачи будут выполняться на три часа позднее, если вы находитесь в utc+3.

Вы можете задавать таймзону либо в файле конфигурации, либо в DAG (в параметре **start\_date=pendulum.datetime(конкретная\_таймзона)**):

```
default_timezone = utc

import pendulum

...
start_date = pendulum.datetime(2023, 1, 1, tz = "UTC")
```

## Pools

```
aggregate_db_message_job=BashOperator(
    task_id='task_1',
    pool='non_default_pool',
    bash_command="echo 1",
    dag=dag,
)
```

Механизм Pools позволяет ограничивать одновременно выполняющиеся процессы – задачи.

По умолчанию Pools выставлен на **128**, то есть на машине может выполняться 128 одновременных процессов задач. Если 128 много для сервера, можно в каждом DAG выставить отдельный параметр того, чему должен быть равен pool. Например, это можно сделать для очень тяжелой задачи, искусственно ограничивая количество Pool'ов, которая она может занять. Так, задача будет выполняться дольше по времени, но в то же время не будет занимать все мощности сервера.

## Переменные в Airflow. Задание переменных

Переменные в Airflow можно задавать:

- Через **WebUI**;
- Через **CLI** командой **airflow variables set**;
- В коде **DAG** с помощью импорта **Variable**.

```
from airflow.models import Variable
my_var=Variable.set("my_key", "my_value")
```

## Считывание кода DAG

В **mine\_file\_process\_interval** лежит значение «30», то есть код всех DAG парсится каждые тридцать секунд. Каждые тридцать секунд выполняется тот код, который находится вне DAGs. Не следует писать код вне задач! Это будет замедлять программу.

## Источники переменных

**Последовательность наблюдения Airflow за переменными:**

- Airflow variables.get;
- Backend Secret (Kubernetes, сохранение в облаке);
- AIRFLOW\_ENV;
- Metastore (Если заданы переменные в файле конфигурации, они могут перетираться теми, которые указаны в

## Разбор примера

Для начала посмотрим в файл конфигурации. Мы скопировали его, чтобы было проще смотреть через папо:

```
File Edit View Search Terminal Help
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$ nano airflow.cfg
```

Здесь находятся несколько разделов «core», находится то, что относится к WebUI и т.д. Например, директория «DAGs folder» – директория, где мы ищем DAGs:

```
File Edit View Search Terminal Help
GNU nano 4.8 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

# This defines the maximum number of task instances that can run concurrently per scheduler in
# Airflow, regardless of the worker count. Generally this value, multiplied by the number of
# schedulers in your cluster, is the maximum number of task instances with the running
# state in the metadata database.
parallelism = 32

# The maximum number of task instances allowed to run concurrently in each DAG. To calculate
# the number of tasks that is running concurrently for a DAG, add up the number of running
# tasks for all DAG runs of the DAG. This is configurable at the DAG level with "max_active_tasks",
# which is defaulted as "max_active_tasks_per_dag".

[ Read 1375 lines ]
Get Help  Write Out  Where Is  Cut Text  Justify  Cur Pos  Undo  Mark Text
Exit      Read File  Replace  Paste Text  To Spell  Go To Line  Redo  Copy Text
```

Это раздел «core», где параметр – параллелизм (это то, сколько может выполняться параллельно задач).

Активных задач по одному DAG – 16 (по умолчанию).

Активных инстансов DAG Run для одного DAG – 16 (по умолчанию).

Параметр «examples=true» означает, что мы подгружаем примеры, которые у нас отображаются.

Здесь находится таймзона, отображающаяся в WebUI:

```
File Edit View Search Terminal Help
GNU nano 4.8 airflow.cfg
default_queue = default

# Is allowed to pass additional/unused arguments (args, kwargs) to the BaseOperator operator.
# If set to False, an exception will be thrown, otherwise only the console message will be displayed.
allow_illegal_arguments = False

[hive]
# Default mapreduce queue for HiveOperator tasks
default_hive_mapred_queue =

# Template for mapred_job_name in HiveOperator, supports the following named parameters
# hostname, dag_id, task_id, execution_date
# mapred_job_name_template =

[webserver]
# The base url of your website as airflow cannot guess what domain or
# cname you are using. This is used in automated emails that
# airflow sends to point links to the right web server
base_url = http://localhost:8080

# Default timezone to display all dates in the UI, can be UTC, system, or
# any IANA timezone string (e.g. Europe/Amsterdam). If left empty the
# default value of core/default timezone will be used
# Example: default_ui_timezone = America/New_York
default_ui_timezone = UTC

# The ip specified when starting the web server
web_server_host = 0.0.0.0

# The port on which to run the web server
web_server_port = 8080

# Paths to the SSL certificate and key for the web server. When both are
# provided SSL will be enabled. This does not change the web server port.
web_server_ssl_cert =

# Paths to the SSL certificate and key for the web server. When both are
# provided SSL will be enabled. This does not change the web server port.
web_server_ssl_key =

# The type of backend used to store web session data, can be 'database' or 'securecookie'
```

«**Timezone=UTC**» – не совсем та таймзона, отвечающая за выполнение задач. Это та таймзона, которая отображается, когда вы смотрите в браузере. Это разные таймзоны!

Также здесь указываем url веб-сервера. Если вы хотите открыть его удаленно, то вместо localhost 8080 указывайте другой host и порт.

**Default timezone** – это таймзона по выполнению всех DAGs. Если вы хотите ее исправить, делать это нужно здесь:

```
File Edit View Search Terminal Help
GNU nano 4.8 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

# This defines the maximum number of task instances that can run concurrently per scheduler in
# Airflow, regardless of the worker count. Generally this value, multiplied by the number of
# schedulers in your cluster, is the maximum number of task instances with the running
# state in the metadata database.
parallelism = 32

# The maximum number of task instances allowed to run concurrently in each DAG. To calculate
# the number of tasks that is running concurrently for a DAG, add up the number of running
# tasks for all DAG runs of the DAG. This is configurable at the DAG level with `max_active_tasks`,
# which is defaulted as `max_active_tasks_per_dag`.
```

Таймзону можно указывать в разном формате. Вместо UTC можно указывать города (в интернете можно почитать о ключевых).

В дополнительных материалах будут указаны ссылки.

Проверяя параметр Pool, видим 128:

```
# Task Slot counts for "default_pool". This setting would not have any effect in an existing
# deployment where the "default_pool" is already created. For existing deployments, users can
# change the number of slots using Webserver, API or the CLI
default_pool_task_slot_count = 128
```

**Вопрос:** можно ли иметь несколько подключений к базе данных?

**Ответ:** если параметр – «true», то можно.

Здесь указывается максимальное количество подключений к базе данных:

```
File Edit View Search Terminal Help
GNU nano 4.8 airflow.cfg
# SQLAlchemy supports many different database engines.
# More information here:
# http://airflow.apache.org/docs/apache-airflow/stable/howto/set-up-database.html#database-uri
sql_alchemy_conn = sqlite:///opt/airflow/airflow.db

# Extra engine specific keyword args passed to SQLAlchemy's create_engine, as a JSON-encoded value
# Example: sql_alchemy_engine_args = {"arg1": True}
# sql_alchemy_engine_args =

# The encoding for the databases
sql_engine_encoding = utf-8

# Collation for `dag_id`, `task_id`, `key`, `external_executor_id` columns
# in case they have different encoding.
# By default this collation is the same as the database collation, however for `mysql` and `mariadb`
# the default is `utf8mb3_bin` so that the index sizes of our index keys will not exceed
# the maximum size of allowed index when collation is set to `utf8mb4` variant
# (see https://github.com/apache/airflow/pull/17603#issuecomment-901121618).
# sql_engine_collation_for_ids =

# If SQLAlchemy should pool database connections.
sql_alchemy_pool_enabled = True

# The SQLAlchemy pool size is the maximum number of database connections
# in the pool. 0 indicates no limit.
sql_alchemy_pool_size = 5

# The maximum overflow size of the pool.
# When the number of checked-out connections reaches the size set in pool_size,
# additional connections will be returned up to this limit.
# When those additional connections are returned to the pool, they are disconnected and discarded.
# It follows then that the total number of simultaneous connections the pool will allow
# is pool_size + max overflow,
# and the total number of "sleeping" connections the pool will allow is pool_size.
# max overflow can be set to `1` to indicate no overflow limit;
# no limit will be placed on the total number of concurrent connections. Defaults to `10`.
sql_alchemy_max_overflow = 10

# The SQLAlchemy pool recycle is the number of seconds a connection
# can be idle in the pool before it is invalidated. This config does
# not apply to sqlite. If the number of DB connections is ever exceeded,
# Search [pool]:
^G Get Help      M-C Case Sens  M-B Backwards  ^P Older
^C Cancel        M-R Regexp      ^R Replace      ^N Newer
```

Попробуем убрать загрузку примеров.

Чтобы примеры не загружались, можно поставить **load\_example=false**:

```
File Edit View Search Terminal Help
GNU nano 4.8 airflow.cfg
# Are DAGs paused by default at creation
dags_are_paused_at_creation = True

# The maximum number of active DAG runs per DAG. The scheduler will not create more DAG runs
# if it reaches the limit. This is configurable at the DAG level with `max_active_runs`,
# which is defaulted as `max_active_runs_per_dag`.
max_active_runs_per_dag = 16

# The name of the method used in order to start Python processes via the multiprocessing module.
# This corresponds directly with the options available in the Python docs:
# https://docs.python.org/3/library/multiprocessing.html#multiprocessing.set_start_method.
# Must be one of the values returned by:
# https://docs.python.org/3/library/multiprocessing.html#multiprocessing.get_all_start_methods.
# Example: mp_start_method = fork
# mp_start_method =

# Whether to load the DAG examples that ship with Airflow. It's good to
# get started, but you probably want to set this to `False` in a production
# environment
load_examples = True

# Path to the folder containing Airflow plugins
plugins_folder = /opt/airflow/plugins

# Should tasks be executed via forking of the parent process ("False",
# the speedier option) or by spawning a new python process ("True" slow,
# but means plugin changes picked up by tasks straight away)
execute_tasks_new_python_interpreter = False

# Secret key to save connection passwords in the db
fernet_key =

# Whether to disable pickling dags
donot_pickle = True

# How long before timing out a python file import
dagbag_import_timeout = 30.0

# Should a traceback be shown in the UI for dagbag import errors,
# instead of just the exception message
Search Wrapped
^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos      M-U Undo
^X Exit          ^R Read File    ^\ Replace      ^U Paste Text  ^T To Spell     ^_ Go To Line    M-E Redo
```

Мы посмотрели файл конфигурации:

```
File Edit View Search Terminal Help
asya@asya-Aspire-XC-886:~/airflow_edu$ ls
airflow.cfg  config  logs  plugins  venv
asya@asya-Aspire-XC-886:~/airflow_edu$ nano airflow.cfg
asya@asya-Aspire-XC-886:~/airflow_edu$ sudo nano airflow.cfg
[sudo] password for asya:
asya@asya-Aspire-XC-886:~/airflow_edu$
```

Посмотрим переменные, а именно, каким образом их можно задавать. Поскольку ранее мы их не задавали, для примера укажем ключ-значение. Желательно добавить описание по той же причине, что и connections, поскольку все variables хранятся в одном месте:

Add Variable

Key \*

Val

Description

Some variable должен сохраниться. Смотрим в Airflow (в контейнере «Scheduler»). Здесь можно обращаться к variables:

```
File Edit View Search Terminal Help
u-airflow-webserver-1
6c240cfd3515 postgres:13 "docker-entrypoint.s..." 52 minutes ago Up 52 minutes (healthy) 5432/tcp
u-postgres-1
1d5bcd08a06 redis:latest "docker-entrypoint.s..." 52 minutes ago Up 52 minutes (healthy) 6379/tcp
u-redis-1
asya@asya-Aspire-XC-886:~/airflow_edu$ docker exec -it airflow_edu-airflow-scheduler-1 /bin/bash
airflow@6533003cc733:/opt/airflow$ airflow -h
Usage: airflow [-h] GROUP_OR_COMMAND ...

Positional Arguments:
  GROUP_OR_COMMAND

Groups:
  celery          Celery components
  config          View configuration
  connections     Manage connections
  dags           Manage DAGs
  db             Database operations
  jobs          Manage jobs
  kubernetes    Tools to help run the KubernetesExecutor
  pools        Manage pools
  providers     Display providers
  roles        Manage roles
  tasks       Manage tasks
  users       Manage users
  variables   Manage variables

Commands:
  cheat-sheet      Display cheat sheet
  dag-processor   Start a standalone Dag Processor instance
  info            Show information about current Airflow and environment
  kerberos       Start a kerberos ticket renewer
  plugins        Dump information about loaded plugins
  rotate-fernet-key
                 Rotate encrypted connection credentials and variables
  scheduler      Start a scheduler instance
  standalone     Run an all-in-one copy of Airflow
  sync-perm     Update permissions for existing roles and optionally DAGs
  triggerer     Start a triggerer instance
  version       Show the version
  webserver     Start a Airflow webserver instance

Optional Arguments:
  -h, --help show this help message and exit
airflow@6533003cc733:/opt/airflow$
```

Можно сделать лист всех переменных при помощи **variables -h**:

```
Optional Arguments:
  -h, --help show this help message and exit
airflow@6533003cc733:/opt/airflow$ airflow variables -h
Usage: airflow variables [-h] COMMAND ...

Manage variables

Positional Arguments:
  COMMAND
  delete   Delete variable
  export   Export all variables
  get      Get variable
  import   Import variables
  list     List variables
  set      Set variable

Optional Arguments:
  -h, --help show this help message and exit
airflow@6533003cc733:/opt/airflow$
```

Можно установить некую новую переменную при помощи **new\_war 4321**:

```
Optional Arguments:
-h, --help show this help message and exit
airflow@6533003cc733:/opt/airflow$ airflow variables set new_var 4321
[2023-08-28T13:25:53.830+0000] {crypto.py:83} WARNING - empty cryptography key - values will not be stored encrypted.
Variable new_var created
airflow@6533003cc733:/opt/airflow$ airflow variables
Usage: airflow variables [-h] COMMAND ...

Manage variables

Positional Arguments:
COMMAND
delete      Delete variable
export      Export all variables
get         Get variable
import      Import variables
list       List variables
set        Set variable

Optional Arguments:
-h, --help show this help message and exit

airflow variables command error: the following arguments are required: COMMAND, see help above.
airflow@6533003cc733:/opt/airflow$ airflow variables list
```

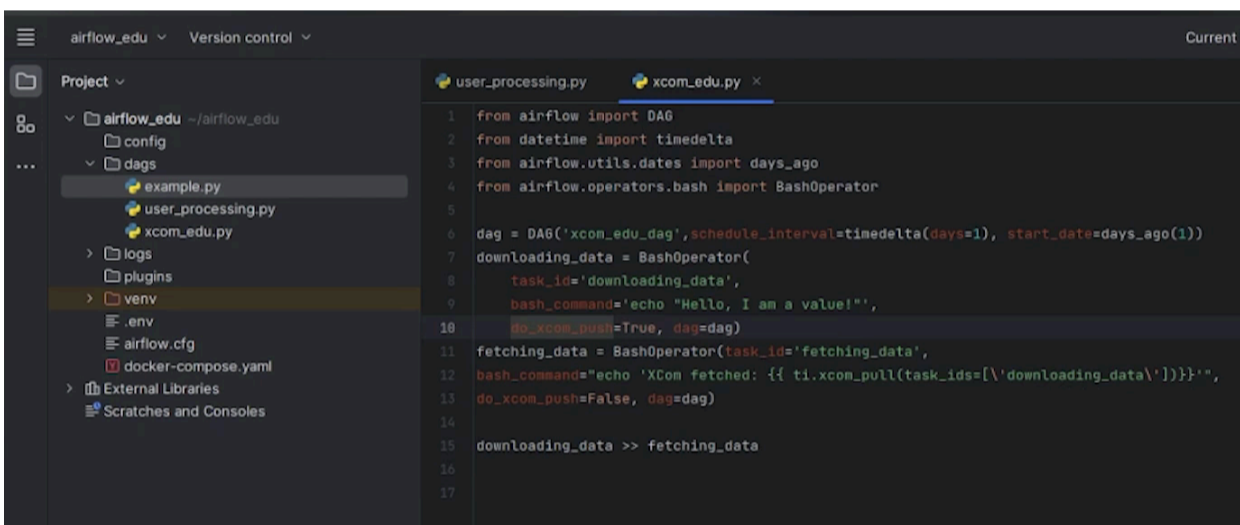
Мы видим два ключа: **some\_variable** и **new\_var**:

```
airflow variables command error: the following arguments are required: COMMAND, see help above.
airflow@6533003cc733:/opt/airflow$ airflow variables list
key
=====
some_variable
new_var
```

К ним можно обращаться из кода или получить здесь значение через get:

```
airflow@6533003cc733:/opt/airflow$ airflow variables get new_var
4321
airflow@6533003cc733:/opt/airflow$
```

Попробуем обратиться через код. Когда мы делаем `variable.get`, то получаем значения этой переменной. Таким образом выглядит получение переменной:



**Variable.set** – это установка.

Через `set` можно установить значение, а через `get` поставить получателя.

Видим ту самую переменную «`min_file_process_interval`», указывающую интервал, с которого считывается значение:

```
File Edit View Search Terminal Help
COMMAND      Delete variable
export      Export all variables
get         Get variable
import      Import variables
list       List variables
set        Set variable

Optional Arguments:
-h, --help show this help message and exit
airflow@6533003cc733:/opt/airflow$ airflow variables set new_var 4321
[2023-08-28T13:25:53.830+0000] {crypto.py:83} WARNING - empty cryptography key - values will not be stored encrypted.
Variable new_var created
airflow@6533003cc733:/opt/airflow$ airflow variables
Usage: airflow variables [-h] COMMAND ...

Manage variables

Positional Arguments:
COMMAND
delete      Delete variable
export      Export all variables
get         Get variable
import      Import variables
list       List variables
set        Set variable

Optional Arguments:
-h, --help show this help message and exit

airflow variables command error: the following arguments are required: COMMAND, see help above.
airflow@6533003cc733:/opt/airflow$ airflow variables list
key
=====
some_variable
new_var

airflow@6533003cc733:/opt/airflow$ airflow variables get new_var
4321
airflow@6533003cc733:/opt/airflow$ ls
airflow.cfg  config  logs  plugins  webserver  config.py
airflow@6533003cc733:/opt/airflow$ grep process_interval airflow.cfg
# `min_file_process_interval` number of seconds. Updates to DAGs are reflected after
min_file_process_interval = 30
airflow@6533003cc733:/opt/airflow$
```

Рассмотрим файл «docker compose». Здесь мы можем указывать переменные. Например, в PIP\_ADDITIONAL\_REQUIREMENTS можно добавить дополнительные библиотеки, которые тоже нужно установить:

```
File Edit View Search Tools Documents Help
docker-compose.yml x
# use this option ONLY for quick checks. installing requirements at container
# startup is done EVERY TIME the service is started.
# A better way is to build a custom image or extend the official image
# as described in https://airflow.apache.org/docs/docker-stack/build.html.
# Default: ''
# Feel free to modify this file to suit your needs.
---
version: '3.8'
x-airflow-common:
  &airflow-common
  # In order to add custom dependencies or upgrade provider packages you can use your extended image.
  # Comment the image line, place your Dockerfile in the directory where you placed the docker-compose.yml
  # and uncomment the "build" line below, Then run "docker-compose build" to build the images.
  image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.6.2}
  # build: .
  environment:
    &airflow-common-env
    AIRFLOW__CORE__EXECUTOR: CeleryExecutor
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow
    # For backward compatibility, with Airflow <2.3
    AIRFLOW__CORE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__RESULT_BACKEND: db+postgresql://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
    AIRFLOW__CORE__FERNET_KEY: ''
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'true'
    AIRFLOW__API__AUTH_BACKENDS: 'airflow.api.auth.backend.basic_auth,airflow.api.auth.backend.session'
    # yamllint disable rule:line-length
    # Use simple http server on scheduler for health checks
    # See https://airflow.apache.org/docs/apache-airflow/stable/administration-and-deployment/logging-monitoring/check-health.html#scheduler-health-check-server
    # yamllint enable rule:line-length
    AIRFLOW__SCHEDULER__ENABLE_HEALTH_CHECK: 'true'
    # WARNING: Use PIP_ADDITIONAL_REQUIREMENTS option ONLY for a quick checks
    # for other purpose (development, test and especially production usage) build/extend Airflow image.
    PIP_ADDITIONAL_REQUIREMENTS: ${PIP_ADDITIONAL_REQUIREMENTS:-}
  volumes:
    - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags
    - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs
    - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config
    - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins
  user: "${AIRFLOW_UID:-50000}:0"
  depends_on:
    &airflow-common-depends-on
    redis:
      condition: service_healthy
    postgres:
```

Делается это следующим образом:

```
File Edit View Search Tools Documents Help
*docker-compose.yml x
# use this option ONLY for quick checks. installing requirements at container
# startup is done EVERY TIME the service is started.
# A better way is to build a custom image or extend the official image
# as described in https://airflow.apache.org/docs/docker-stack/build.html.
# Default: ''
# Feel free to modify this file to suit your needs.
---
version: '3.8'
x-airflow-common:
  &airflow-common
  # In order to add custom dependencies or upgrade provider packages you can use your extended image.
  # Comment the image line, place your Dockerfile in the directory where you placed the docker-compose.yml
  # and uncomment the "build" line below, Then run "docker-compose build" to build the images.
  image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.6.2}
  # build: .
  environment:
    &airflow-common-env
    AIRFLOW__CORE__EXECUTOR: CeleryExecutor
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow
    # For backward compatibility, with Airflow <2.3
    AIRFLOW__CORE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__RESULT_BACKEND: db+postgresql://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
    AIRFLOW__CORE__FERNET_KEY: ''
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'true'
    AIRFLOW__API__AUTH_BACKENDS: 'airflow.api.auth.backend.basic_auth,airflow.api.auth.backend.session'
    # yamllint disable rule:line-length
    # Use simple http server on scheduler for health checks
    # See https://airflow.apache.org/docs/apache-airflow/stable/administration-and-deployment/logging-monitoring/check-health.html#scheduler-health-check-server
    # yamllint enable rule:line-length
    AIRFLOW__SCHEDULER__ENABLE_HEALTH_CHECK: 'true'
    # WARNING: Use PIP_ADDITIONAL_REQUIREMENTS option ONLY for a quick checks
    # for other purpose (development, test and especially production usage) build/extend Airflow image.
    # PIP_ADDITIONAL_REQUIREMENTS: ${PIP_ADDITIONAL_REQUIREMENTS:-}
    PIP_ADDITIONAL_REQUIREMENTS: ${PIP_ADDITIONAL_REQUIREMENTS:- apache-airflow-providers-apache-spark}
  volumes:
    - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags
    - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs
    - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config
    - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins
  user: "${AIRFLOW_UID:-50000}:0"
  depends_on:
    &airflow-common-depends-on
    redis:
      condition: service_healthy
    postgres:
```

Если мы перезапустим docker compose, то библиотека появится.

### На этом уроке мы:

- Научились вносить изменения в файл конфигурации;
- Посмотрели из чего состоит файл конфигурации;
- Каким образом можно менять настройки;
- Как ставить новые библиотеки;
- Как устанавливать переменные.

Как вам урок?



Изучил, далее >