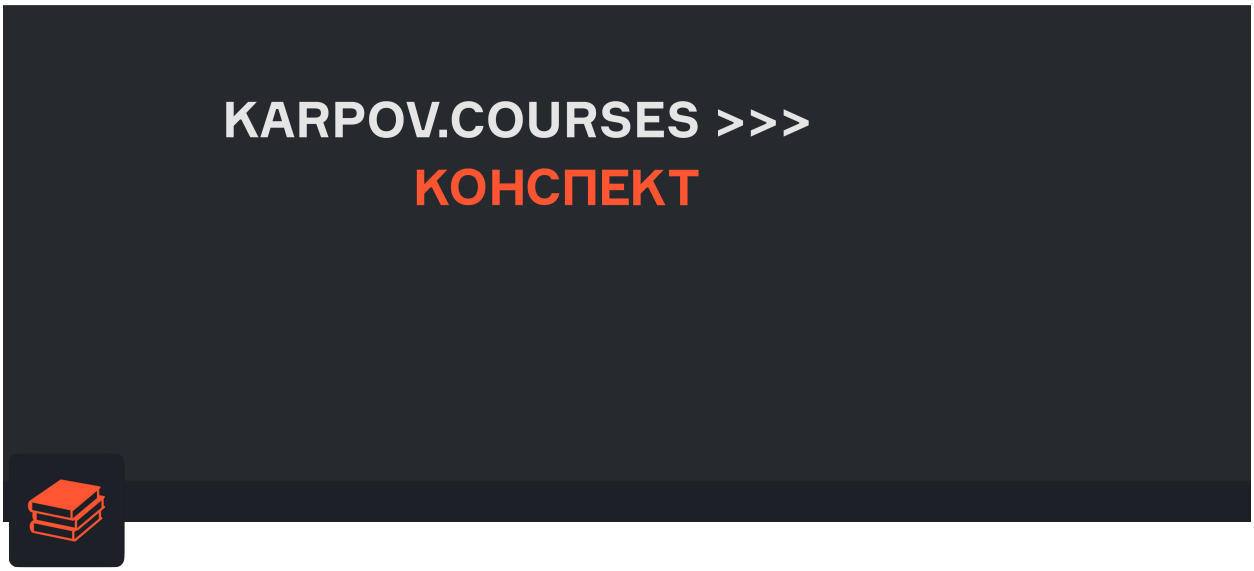

> Конспект > 6 урок > Установка и настройка Airflow

> Оглавление

> Разворачиваем Airflow локально

Для установки Airflow необходимо создать виртуальное окружение - назовем его Airflow и активируем. Разворачивать Airflow мы будем в Python 3.

Для начала необходимо установить локальную переменную `AIRFLOW_HOME` и направить её в папку, которая была вами создана. Обратите внимание какой версии у вас Airflow:

`python --version` - 3.7.6.

С этой версией мы пойдем в Github, и в репозитории Airflow вам будет доступен код установки. С помощью PyPI вы можете установить "прямо из коробки". Если у вас версия отличатся от 3.7., необходимо будет изменить `constraints` на необходимый вам вариант.

Выбираем нашу текущую версию (копируем), переходим, ждем, когда у нас загрузится и соберется Airflow, версия 2.2.

Далее нам нужно инициализировать базу данных. При инициализации создаются нужные таблицы, права и примеры соединений (connection). По умолчанию у нас встает SQLite.

После завершения процедуры инициализации вам необходимо, отвязанной от вашей текущей сессии, запустить web server на порту 8080 - `airflow webserver -p 8080 &` .

Поздравляю, вы подняли web server и теперь на `localhost:8080/` вас ждет окно с логином и паролем. Для этого вам необходимо завести своего пользователя, по умолчанию это будет admin:

```
airflow users create --username admin --firstname admin --lastname admin --role Admin email --admi.n
```

С появлением этого пользователя в базе, вы можете зайти на web server. После ввода пароля и его подтверждения, вы можете зайти и увидеть список DAGs с примерами, которые можно запустить. Но у нас не запущен scheduler - необходимо вернуться в терминал и запустить его? предварительно отвязав от сессии:

```
airflow scheduler &
```

После завершения этой операции, вы можете зайти в DAGs и посмотреть, что сейчас будет. Возможно у вас будут проблемы с некоторыми DAGs, это связано с обновлением Airflow. Так как у нас стоит SQLite, в проде он работать не будет - мы не можем работать с этим файлом двумя экземплярами и выполнять таски параллельно.

В версии Airflow 2.0, в командной строке наберите `airflow cheat-sheet`. Вам будут показаны:

- доступные команды;
- дополнительная информация о конфигурации;
- список параметров и их конкретные значения;

- соединения (connection) и доступные операции с ними;
- работа с DAGs: ставить на паузу, тестировать, включать задним числом и т.д.;
- работа с джобами и пулами;
- список ролей и провайдеров, таски, пользователи и переменные.

Наберите следующие команды и ознакомьтесь с предоставленной информацией:

`airflow info`

`airflow version`

`airflow connections list`

`airflow pool list`

`airflow variables list` для последней команды необходимо добавить ключ и значение - `airflow variables set my_var value_of_my_var`

В Airflow, в списке переменных появилась переменная со значением `value_of_my_var`

Что еще хорошего можно сделать - посмотреть самое интересное - файл с конфигурацией `vim airflow.cfg`

Когда вы будете менять SQLite на PostgreSQL, обратите внимание в конфигурации есть ссылка на статью в стандартной документации как именно устанавливать Postgres.

> Настройки и конфигурация Airflow

Наберите `airflow config list` . Пройдемся по самым интересным, не вдаваясь в подробности:

`dags_folder` - папка, в которой мы ждем DAGs, чтобы они появились в Airflow;

`default_timezone` - по умолчанию;

`executor` - то, что мы сразу меняем сразу на local executor;

обязательно нужно, чтобы база, которой мы пользуемся поддерживала utf-8;

`parallelism` **-** количество воркеров, которые будут использованы при работе Airflow, в данном случае оно указывает, что мы можем запустить не более 32 тасок.

`max_active_tasks_per_dag` - количество активных тасок внутри одного DAGs, в нашем случае не более 16 тасок;

`max_active_runs_per_dag` - количество DAGs run'ов, указывающее на ограничение.

`xcom_backend` - поменять, написать собственный, чтобы xcom'ы передавались через наш бэкенд.

`lazy_load_plugins` - если его выставить в False, то тогда вам не нужно будет каждый раз нажимать кнопку обновить, если вы что-то поменяли в DAGs либо когда у вас появился новый плагин, оператор, хук или сенсор.

Дальше у нас идет блок с логированием, по умолчанию локальные логи пишем в папку **airflow/logs**.

Из интересных блоков обратите внимание на настройки **webserver** и **smart_sensor**.

В конце вам необходимо снять все процессы Airflow `pkill -f 'airflow'` и выйти `rm -r airflow` .

> Как вывести Airflow в прод

После того, как вы развернули Airflow локально, попробовали, сделали пару DAGs и поняли, что это вас устраивает, вы захотите вывести его в прод - сделать стабильным, масштабируемым и настроить под себя.

Что необходимо сделать:

- Заменяем SQLite на PostgreSQL;
- Заменяем SequentialExecutor;
- Одинаковое окружение на всех воркерах;
- Логирование (локальные логи, StatsD, ElasticSearch, S3, Azure, ...)
- Мониторинг процессов Airflow

> Airflow в докере

Чтобы развернуть Airflow в Docker, вам нужно с сайта Docker.com установить докер и Docker Compose.

В зависимости от операционной системы, вам потребуются разные установочные файлы. После установки рекомендуем зайти в Preferences и поднять количество памяти, выделяемое по умолчанию под Docker с 2 Гбайт до 4 Гбайт и более.

В официальной документации Airflow нас интересует `curl -LfO 'https://airflow.apache.org/docs/apache-airflow/2.2.1/docker-compose.yaml'`, чтобы скачать yaml файл.

Сначала описывается конфигурация Airflow - у нас внутри:

- **Celery Executor**
- **PostgreSQL**
- **Redis**
- **dags_are_paused_at_creation** - его необходимо будет включить
- **load_example** - мы добавляем примеры в наш Airflow
- **auth_backing** - авторизация для нашего REST API

В директории Airlow Docker обратите внимание на папки Dags, logs и plugins.

Вам необходимо ввести `docker-compose up airflow-init` и пару минут все будет загружаться. Если ввести docker ps, то в в соседнем окошке можно будет наблюдать процесс. После поднятия всех сервисов, в `localhost:8080` он будет смотреть не на локальный Airflow, а на Airflow в Docker.

Можно зайти на `localhost:5555` и посмотреть все текущие задачи и операции.

Давайте попробуем изменить конфигурацию нашего Docker'а. В `airflow_core_load_example` мы поставим `'false'`. Пару Dags необходимо будет добавить в локальный Airflow вместе с плагинами, добавлять необходимо в такую же папку, чтобы не пришлось менять в импортах ссылку. После этого необходимо еще раз поднять Docker - при успешном завершении у вас не будет Dag'ов с примерами, но появятся установленные Dags.

Обратите внимание - вам не будет хватать подключения, потому что вы его пока не завели.

Что еще можно сделать - например подключиться к command line interface, но это гораздо менее удобно и более затратно.

> REST API

Для версии Airflow 2.0 и выше добавилась крутая штука - REST API.

Это прикладной программный интерфейс (**API**), который использует HTTP-запросы для получения, извлечения, размещения и удаления данных. В официальной документации есть ссылка на стабильную сборку REST API с описанием функционала.

Можно найти конфигурацию, узнать в какой секции какой параметр включен, поработать с коннекторами и т.д.

Давайте попробуем через REST API вызвать список наших пулов (pool):

```
curl -X GET "http://localhost:8080/api/v1/pools"
```

для авторизации необходимо добавить пользователя `--user "airflow:airflow"` .

Мы видим название pool'a, ни одного занятого слота и 128 возможных слотов. Теперь необходимо почистить за собой: остановить докер, опустить и попросить почистить за собой, все что было создано - `docker-compose down --volumes  --rmi all`