# Airflow

# Проверка связи

## Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включён звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти

## Поставьте в чат:

\+ если меня видно и слышно

\- если нет

# Вспоминаем прошлое занятие

**Вопрос:** к какому компоненту DWH относится Airflow?

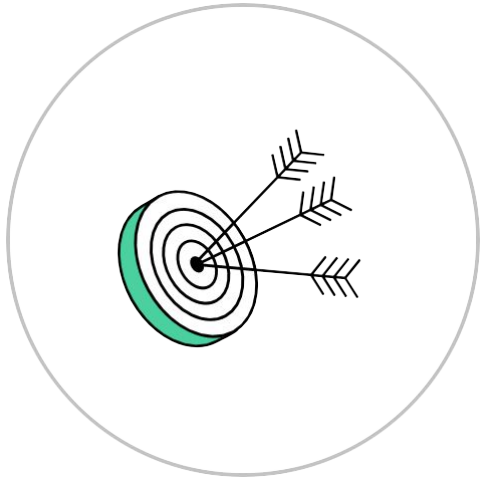
# Вспоминаем прошлое занятие

**Вопрос:** к какому компоненту DWH относится Airflow?

**Ответ:** Airflow — ETL-инструмент

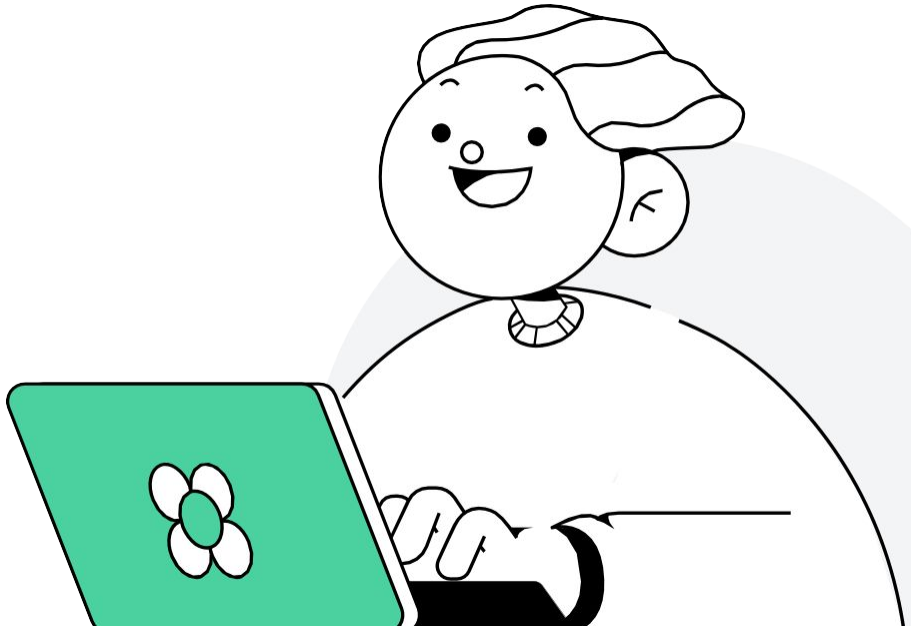
# Цели занятия

- Особенности архитектуры Airflow
- Познакомиться с синтаксисом Airflow
- Попрактиковаться в реализации ETL-процессов



# План занятия

1. Архитектура Airflow
2. Синтаксис Airflow
3. Практика



*Нажми на нужный раздел для перехода

# Архитектура Airflow

1

# Airflow

**Airflow** — инструмент разработки, координации и управления ETL-процессами.

1 2 3

## Инструмент open-source

Бесплатный инструмент с открытым исходным кодом

## Python-based

Для описания процессов использует язык программирования Python

## Может использоваться как планировщик ETL-процессов

Автоматически запускает закодированный процесс по расписанию

4

## Позволяет осуществлять мониторинг ETL-процессов

Есть пользовательский интерфейс с результатами выполнения процесса и его задач: логи, истории запусков

5

## Поддерживается сообществом разработчиков

Русское коммьюнити в Telegram

6

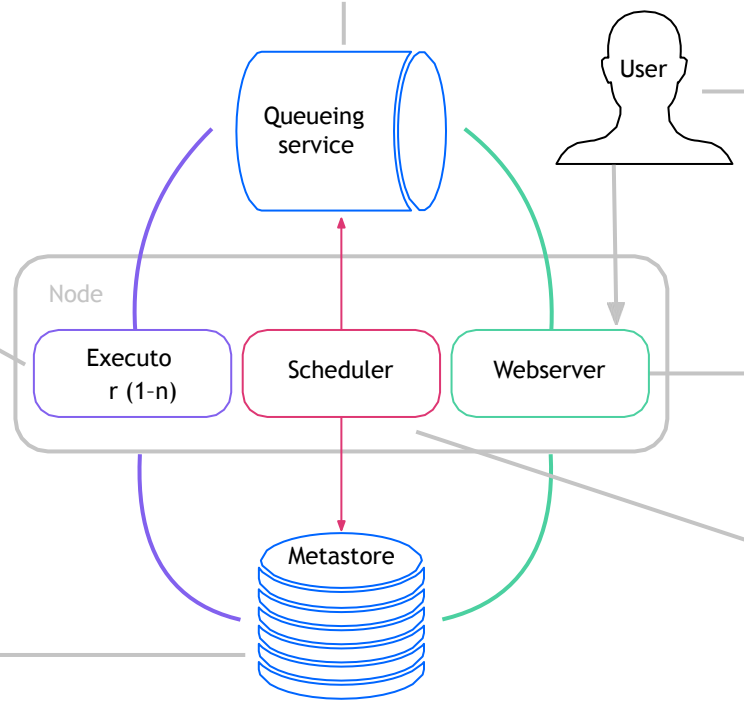
## Инструмент, расширяемый кастомными разработками

На базе Python можно подключать различные совместимые библиотеки, плагины для оптимизации процесса разработки

# Airflow. Архитектура

**Queueing service**

Отвечает за очередь выполнения процессов




**Пользователь**

Через WebUI может запустить процесс

**Worker (executor)**

Сервис, где фактически работает ETL-процесс: выполняет задачу, поставленную в очередь планировщиком Scheduler

**Webserver**

Отображает информацию о ETL-процессах и их логах

**База метаданных**

PostgreSQL, MySQL, SQLite, вся информация о процессах: название, логи, связи, настройки

**Scheduler**

Запускает ETL-процессы по расписанию: находит задачи, которые должны запуститься по расписанию, ставит их в очередь

Чтобы запустить Airflow, нужно хотя бы по одному компоненту, но количество worker может быть и больше

**Scheduler** — это ключевой компонент Apache Airflow, отвечающий за планирование и выполнение задач в DAG. Он управляет временем запуска задач и обеспечивает их выполнение в соответствии с заданным расписанием.

- **Планирование**
  - Scheduler анализирует DAG и определяет, какие задачи могут быть выполнены на основе их зависимостей и состояния.
  - Он использует параметры `schedule_interval` и `start_date` для определения времени запуска задач.
- **Обработка состояний**
  - отслеживает статусы выполнения задач (например, `queued`, `running`, `success`, `failed`) и обновляет их в метаданных базы данных.
- **Управление очередью задач**

# Узкие места планировщика

**Нагрузка на планировщик**:

При большом количестве DAG или задач Scheduler может стать узким местом, что приведет к задержкам в планировании.

**Решение**: Увеличение ресурсов планировщика или добавление дополнительных рабочих узлов может помочь справиться с нагрузкой 1

**Конфликты параллелизма**:

Если настройки параллелизма (например, `dag_concurrency` или `max_active_runs_per_dag`) слишком низкие, это может привести к тому, что задачи будут оставаться в состоянии ожидания.

**Решение**: Настройка параметров параллелизма для увеличения числа одновременно выполняемых задач

**Проблемы с зависимостями**:

Неправильные зависимости между задачами могут привести к тому, что Scheduler не сможет определить, какие задачи можно запускать.

**Решение**: Внимательное проектирование DAG и проверка зависимостей перед запуском.

**Задержки из-за внешних сервисов**:

Если задачи зависят от внешних сервисов или данных, задержки в этих системах могут замедлить выполнение DAG.

**Решение**: Оптимизация взаимодействия с внешними системами и использование кэширования данных

# Executors в Apache Airflow

## SequentialExecutor

*Это стандартный исполнитель, который выполняет задачи последовательно, одну за другой.*

**Плюсы**:

Простота настройки и использования.

Идеален для разработки и тестирования, так как не требует сложной инфраструктуры.

**Минусы**:

Не подходит для производственных сред из-за отсутствия параллелизма.

Ограничивает скорость выполнения задач, что может быть критично при большом объеме данных.

**Когда использовать**: Рекомендуется использовать для разработки и тестирования DAG, когда требуется простота и отсутствие нагрузки на систему.

# LocalExecutor

*Позволяет выполнять задачи параллельно на одном узле, используя многопоточность.*

**Плюсы**:

Поддерживает параллельное выполнение задач, что увеличивает производительность по сравнению с SequentialExecutor.

Не требует внешних брокеров сообщений, что упрощает настройку.

Подходит для небольших и средних производственных развертываний.

**Минусы**:

Ограничен ресурсами одного узла; не масштабируется на несколько машин.

При высокой нагрузке может стать узким местом, если не оптимизирован.

**Когда использовать**: Рекомендуется для небольших производственных сред или в ситуациях, когда необходима параллельная обработка задач на одном сервере.

# CeleryExecutor

*Использует распределенные рабочие узлы для выполнения задач. Задачи помещаются в очередь и обрабатываются рабочими процессами, которые могут находиться на разных машинах.*

**Плюсы**:

Высокая масштабируемость; можно добавлять новые рабочие узлы по мере необходимости.

Подходит для больших производственных развертываний с высокой нагрузкой.

Обеспечивает отказоустойчивость; если один рабочий узел выходит из строя, другие продолжают работу.

**Минусы**:

Сложность настройки и управления; требует настройки брокера сообщений (например, RabbitMQ или Redis).

Может потребовать дополнительных ресурсов для управления очередями задач.

**Когда использовать**: Рекомендуется для крупных производственных сред с высоким уровнем параллелизма и необходимостью масштабирования.

# KubernetesExecutor

*Позволяет запускать задачи в контейнерах Kubernetes. Каждая задача выполняется в отдельном поде Kubernetes.*

**Плюсы**:

Полная изоляция задач благодаря контейнеризации; минимизирует конфликты между задачами.

Высокая масштабируемость; можно динамически добавлять ресурсы в зависимости от нагрузки.

Упрощает управление зависимостями и окружением для задач.

**Минусы**:

Сложность настройки Kubernetes и интеграции с Airflow.

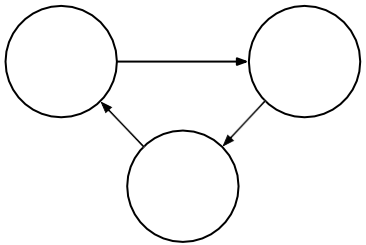
Может потребовать значительных ресурсов на начальном этапе развертывания.

**Когда использовать**: Рекомендуется для облачных решений или организаций, уже использующих Kubernetes для управления контейнерами.

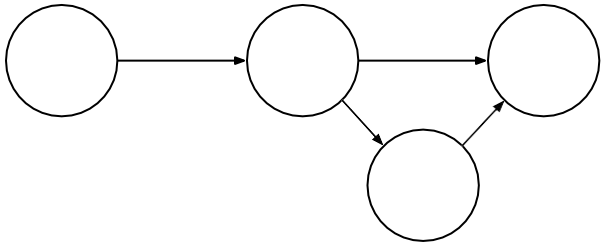
# Airflow. Сущности

**DAG** — ориентированный ациклический граф.

- **D**irect — для любой задачи есть понятие направления
- **A**cyclic — отсутствие петель
- **G**raph — всё связано, есть структура, очевидна связь «родитель — потомок»



Не DAG

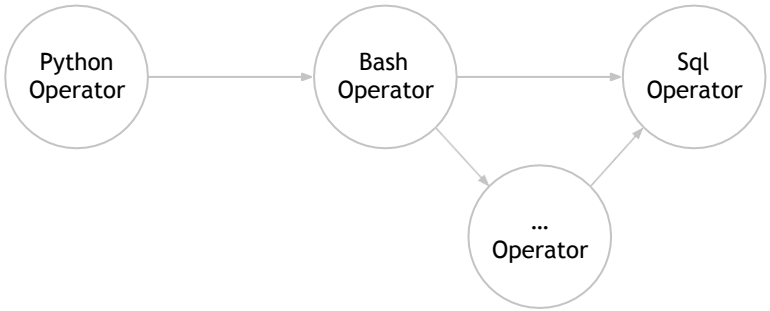


DAG

# Airflow. Сущности

Operator — объект, имеющий определённый набор параметров, в котором происходит выполнение определённой последовательности действий
(по сути, класс Python).

- **SqlOperator** — запускает SQL
- **BashOperator** — запускает командную строку в shell
- **PythonOperator** — запускает функцию




DAG

# PythonOperator

Выполняет функцию Python, переданную в параметре `python_callable`.

**Плюсы**:

Позволяет использовать произвольный код на Python.

Легко интегрируется с другими библиотеками Python.

**Минусы**:

Зависит от окружения Python; могут возникнуть проблемы с совместимостью библиотек.

**Когда использовать:** Когда необходимо выполнить произвольный код на Python в рамках DAG.

# BashOperator

Выполняет команды Bash или скрипты.

**Плюсы:**

Универсальность; можно запускать любые команды, доступные в командной строке.

**Минусы:**

Зависит от среды выполнения и конфигурации системы.

**Когда использовать:**

Для выполнения системных команд или скриптов на Bash.

# SQL Оператор в Apache Airflow

SQL оператор в Apache Airflow предназначен для выполнения SQL-запросов в различных базах данных. Он предоставляет более универсальный подход по сравнению с конкретными операторами, такими как `PostgresOperator`, и может использоваться с несколькими типами баз данных.

## Принципы работы SQL оператора

- **Интерфейс**: SQL оператор позволяет выполнять произвольные SQL-запросы, предоставляя пользователю возможность взаимодействовать с базами данных через стандартные SQL-инструкции.

- **Подключение**: Для работы с базой данных используется механизм соединений Airflow, который позволяет управлять учетными данными и параметрами подключения.

# SQL Оператор в Apache Airflow

## Плюсы:

- Универсальность: Может использоваться для выполнения запросов в различных базах данных, что делает его более гибким по сравнению с конкретными операторами.
- Простота использования: Позволяет выполнять сложные SQL-запросы без необходимости создания дополнительных операторов.
- Поддержка ORM: Можно интегрировать с ORM (например, SQLAlchemy) для более удобной работы с данными.

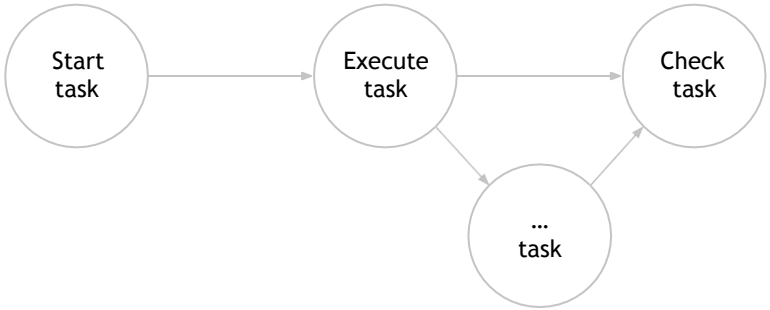
## Минусы:

- Зависимость от базы данных: Необходимость настройки соединения для каждой базы данных может усложнить конфигурацию.
- Отсутствие специфичных функций: Некоторые специфические функции или оптимизации для конкретных баз данных могут быть недоступны.
- Потенциальные проблемы производительности: Выполнение сложных запросов может привести к задержкам, особенно если они не оптимизированы.

# Airflow. Сущности

Task — экземпляр оператора (operator), для которого определены параметры.

- Базовая единица (узел) — DAG
- Выполняет действия, свойственные оператору для конкретного DAG
- Может быть связан с другими задачами (tasks)



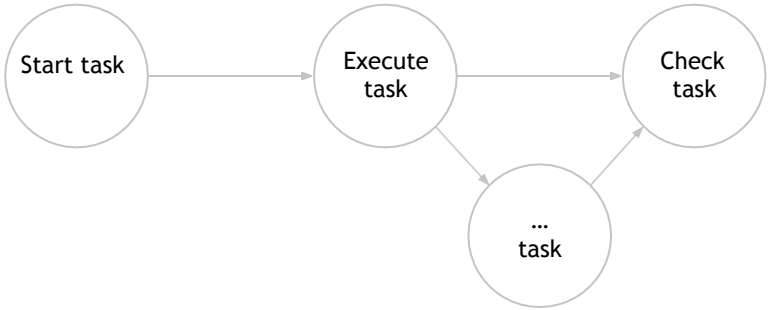

DAG

# Airflow. Сущности

**Зависимости** — связи между задачами (tasks).

- Направление
- Условие перехода

Определяется символом >> (set_upstream) или << (set_downstream)




DAG

# Ваши вопросы?

# Синтаксис Airflow

2

# Создание DAG

```
default_args = { 'owner':
    'IvanovIvan',
    'depends_on_past': True,
    'start_date': start_dt, 'email':
    ['mymail@mail.ru',],
    'email_on_failure': True,
    'email_on_retry': False,
    'retries': 3,
    'max_active_runs': 1
}
with DAG('my_first_dag', default_args=default_args,
schedule_interval="@hourly", catchup=False) as dag:
```

1. Задайте аргументы DAG
2. Сформируйте конструкцию DAG
3. Назовите DAG
4. Присвойте аргументы DAG
5. Расписание DAG
6. Запуск DAG от текущего момента

# Создание DAG. Аргументы

| Аргумент | Описание |
|---|---|
| owner | Владелец DAG |
| depends_on_past | Запустить следующий экземпляр DAG будет можно только после удачного выполнения предыдущего экземпляра DAG |
| start_date | Первая дата, когда будет выполнен ваш DAG. Этот параметр обязателен, чтобы поставить DAG в расписание Airflow |
| email | Электронная почта |
| email_on_failure | Стоит ли отправлять на электронную почту сообщение при ошибочном выполнении DAG |
| email_on_retry | Стоит ли отправлять на электронную почту сообщение при повторном выполнении DAG |
| retries | Возможное количество повторных попыток выполнения DAG |
| max_active_runs | Количество одновременно запущенных экземпляров DAG |

**Выражение cron — строка, состоящая из шести или семи подвыражений (полей), описывающих отдельные детали расписания**

# Создание DAG. Расписание

Расписание DAG задают в виде cron-выражения параметром schedule_interval:

| Значение | Описание | Cron-выражение |
|---|---|---|
| None | Расписание не задано. Возможен только ручной запуск | 0 0 1 1 * |
| @once | Возможен только один запуск по расписанию | 0 0 1 1 * |
| @hourly | Запуск раз в час в 00 минут | 0 * * * * |
| @daily | Запуск раз в день в 00 часов | 0 0 * * * |
| @weekly | Запуск раз в неделю в воскресенье в 00 часов | 0 0 * * 0 |
| @monthly | Запуск раз в месяц в первый день месяца в 00 часов | 0 0 1 * * |
| @yearly | Запуск раз в год 1 января в 00 часов | 0 0 1 1 * |
| 0 15 10 * * | Запуск раз в день в 10:15 | 0 15 10 * * |
| 0 0,55 14 * * | Запуск каждый день в 14:00 и в 14:55 | 0 0,55 14 * * |

Пример:

| 0 | 15 | 10 | * | ? | 6L | 2002-2005 |
|---|---|---|---|---|---|---|
| Секунда | Минута | Час | День месяца | Месяц | День недели | Год |

→ Запуск каждую пятницу* каждый месяц в 10:15 в 2002, 2003, 2004 и 2005 году

*расчёт дней недели начинается с воскресенья

# Время в Apache Airflow

## Основные концепции времени в Airflow

- start_date: Дата и время, с которых начинается выполнение DAG. Задача будет запущена после наступления этого времени, учитывая заданный интервал.
- schedule_interval: Интервал, по которому DAG будет запускаться. Может быть задан в виде объекта `timedelta`, cron-выражения или пользовательского класса расписания.
- execution_date: Время, соответствующее запуску DAG. Это время, когда DAG должен был бы обработать данные за указанный период.

## Способы задания расписания

- **timedelta:**
  - Используется для задания интервалов времени (например, каждые 5 минут).
  - Пример: `schedule_interval=timedelta(minutes=5)` .
- **cron:**
  - Позволяет задавать более сложные расписания с использованием cron-выражений.
  - Пример: `schedule_interval='0 12 * * *'` (ежедневно в 12:00).
- **Пользовательские классы расписания**:
  - Позволяют создавать собственные логики расписания, например, для специфических временных интервалов или условий.

# Время в Apache Airflow

## Макросы

Apache Airflow поддерживает макросы, которые позволяют динамически подставлять значения в задачи. Они могут использоваться для работы с датами и временем.

- Примеры макросов:
  - `{{ ds }}`: Дата выполнения в формате YYYY-MM-DD.
  - `{{ prev_ds }}`: Дата предыдущего выполнения.
  - `{{ next_ds }}`: Дата следующего выполнения.

https://airflow.apache.org/docs/apache-airflow/1.10.12/macros-ref.html

# Время в Apache Airflow

## Ошибки при работе со временем

- Неправильная настройка `start_date`:
  - Ошибка: Установка `start_date` в будущее время может привести к тому, что DAG не будет запущен.
  - Решение: Убедитесь, что `start_date` установлен в прошлое время.
- Ошибки в `schedule_interval`:
  - Ошибка: Неправильное использование cron-выражений или `timedelta` может привести к неожиданному поведению.
  - Решение: Тщательно проверяйте синтаксис и тестируйте расписания на простых примерах.
- Игнорирование `execution_date`:
  - Ошибка: Непонимание того, как работает `execution_date`, может привести к путанице в данных.
  - Решение: Осознайте, что `execution_date` соответствует времени, когда DAG должен был бы обработать данные за указанный период.

https://habr.com/ru/articles/682384/

# Создание Task (задачи)

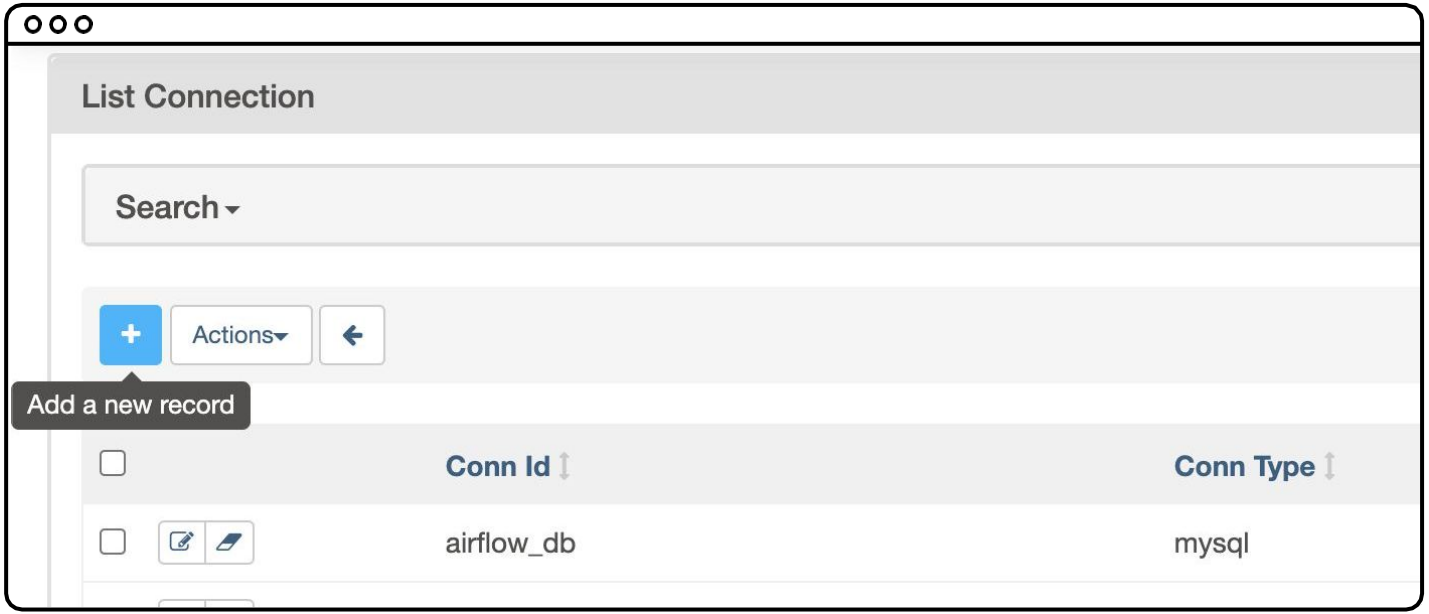
```
1 from airflow.operators.postgres_operator
  import PostgresOperator
2 load_to_db =
3 PostgresOperator(
  task_id='load_to_db',
  sql=base_oda_dml,
5 postgres_conn_id='conn_pg_airflow',
6 autocommit=True,
7 dag=dag
  )
```

1. Вызов оператора
2. Формирование задачи
3. Имя задачи
4. SQL-запрос
5. Название соединения
6. Выполнить коммит после запуска SQL
7. Присвоение задачи DAG

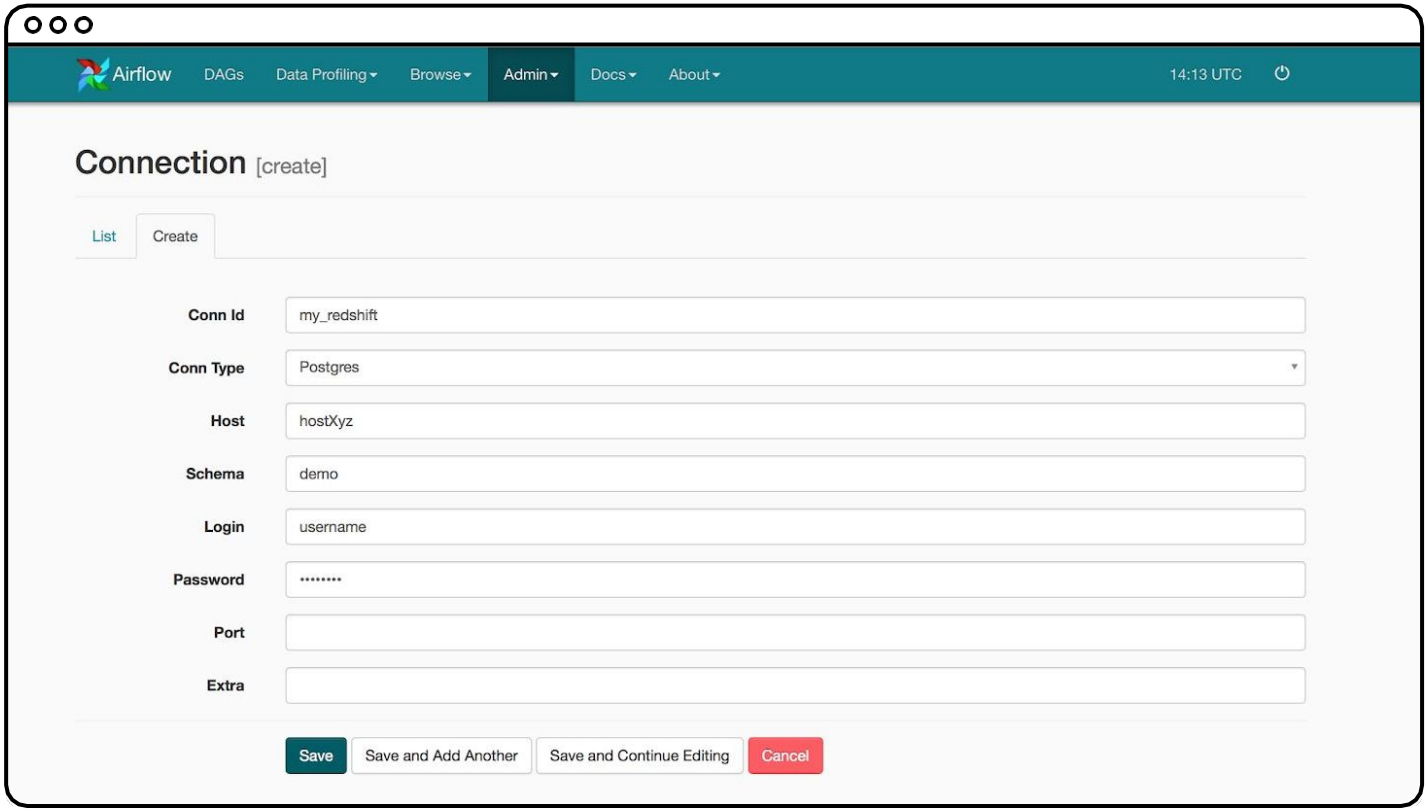
# Соединение

Используется для хранения учётных данных и другой информации, нужной для подключения к внешним сервисам, например, базам данных.

Настройки соединений Airflow через пользовательский интерфейс: Admin->Connections



# Соединение



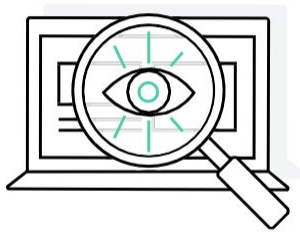

Укажите необходимые параметры для соединения

# Ваши вопросы?

# Перерыв

# Практика

# Демонстрация работы

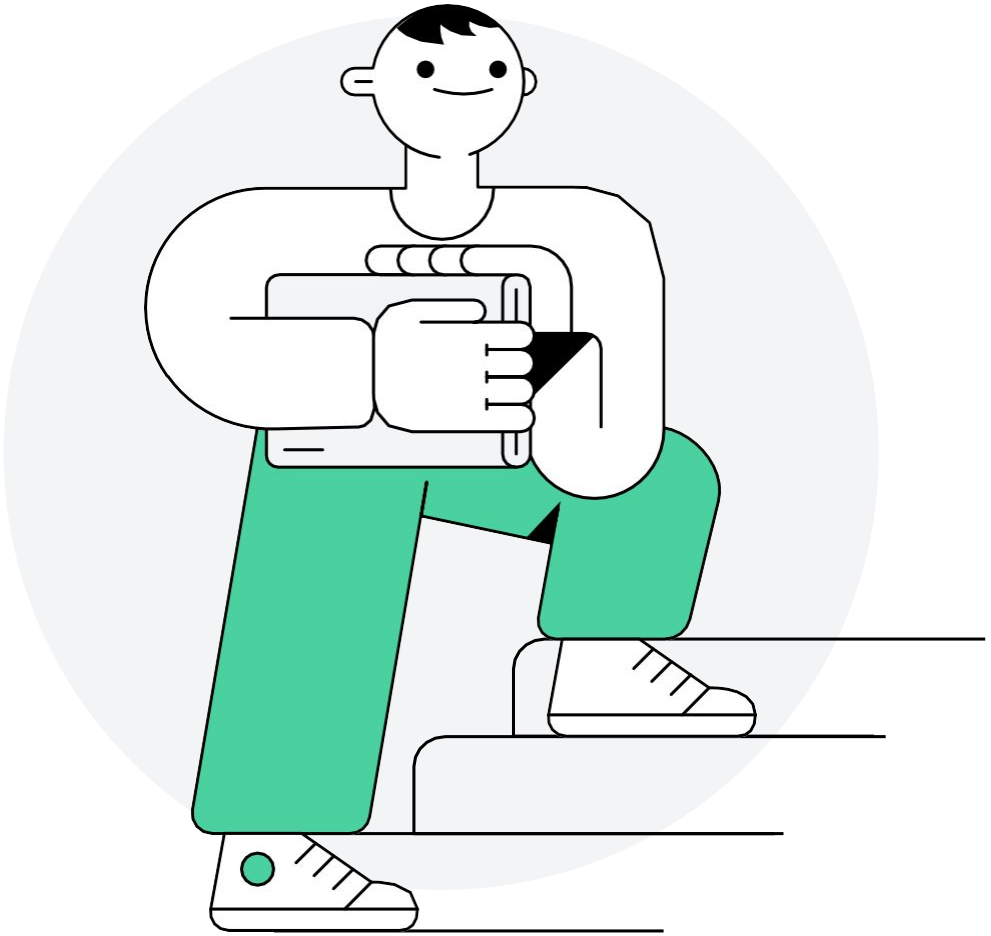


# Ваши вопросы?

# Итоги занятия

1. Вспомнили особенности архитектуры Airflow
2. Изучили синтаксис Airflow
3. Изучили Airflow на практике

# Домашнее задание

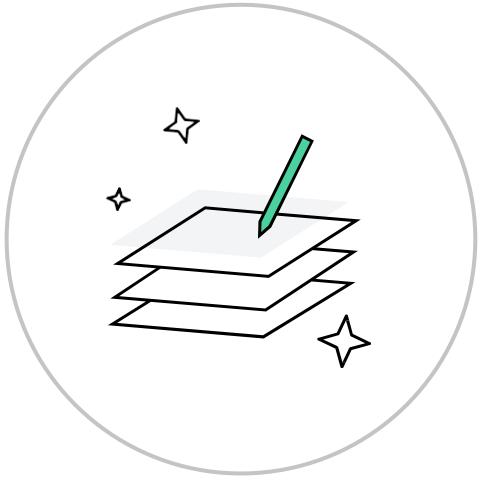


# Домашнее задание

**Описание:** контекст практической задачи №2 из лекции, но покупатель может сформировать возврат по заказу, указав причину. Данные о возвратах находятся в CRM- системе в таблице «Возвраты».

Отделу продаж нужно добавить показатель «количество возвратов» в отчёт о выручке.

**Задание:** нужно построить схему ETL-процесса на базе технических спецификаций Raw Data Layer, Core Data Layer и Data Mart Layer, сформированных в результате домашнего задания №2



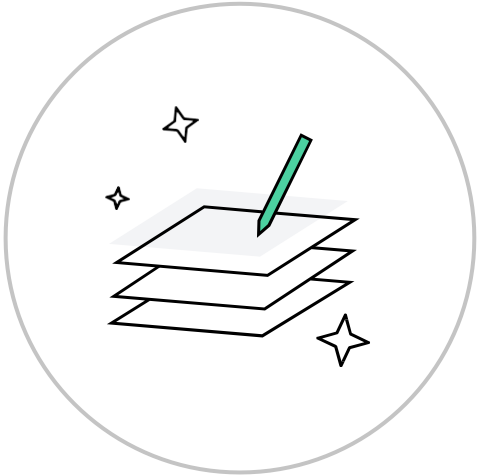
# Домашнее задание

**Цель:** научиться проектировать ETL-процесс на базе технических спецификаций Raw Data Layer, Core Data Layer и Data Mart Layer

**Инструменты:** любой инструмент для построения схемы

**Формат** выполнения: файл в формате pdf, png или jpeg

**Результат:** представлена схема ETL-процесса от Source Systems до Data Mart Layer с учётом нового объекта
«Возвраты»



# Ваши вопросы?

# Спасибо за внимание!

Соловьев Анатолий