

APACHE AIRFLOW

КАРТАШОВ АНДРЕЙ

apk92@icloud.com

1.52

© 000 «Учебный центр «Коммерсант», 2023





Apache
Airflow

Введение в Apache Airflow

модуль 1



ШКОЛА БОЛЬШИХ ДАННЫХ

Программа курса

Введение в Airflow

Что такое Airflow?
Почему Airflow?
История создания
Аналоги и конкуренты
Airflow vs Oozie
“Киты” Airflow
Настройка образа в YandexCloud

Программа курса

Базовый Airflow

Верхнеуровневая архитектура
Компоненты: подробнее
Executors
LocalExecutor
Схема учебного стенда
DAG
DAG: параметры
Operators
Operators: виды
WEB UI: обзор

Пайплайн по созданию DAG
Dag: context
Operator: основные параметры
Composition
EmptyOperator
BashOperator
Написание первого DAG
TaskFlowApi
PythonOperator
[Практика № 1. Создание первого DAG](#)

Запуск DAG с ручной конфигурацией
[Практика № 2. Написание DAG](#)
с ручной передачей параметров
Variables
[Практика № 3. Применение Variables, default_args](#)
[Практика № 4. Применение Variables расширенное Connections](#)
Sensors
[Практика № 5. Применение fileSensor](#)
ExternalTaskSensor



ШКОЛА БОЛЬШИХ ДАННЫХ

Программа курса

Расширенный Airflow

Trigger Rules

Практика № 6. Использование fileSensor + triggerRules

Backfill & catchup

Templates

Macros

Yandex Managed Service for PostgreSQL

Демонстрация ETL процесса на временном DataProc(Spark) кластере в Yandex Cloud

Практика № 7. Использование PostgresOperator, оркестрация ETL процесса

Hooks

Практика № 8. Применение Hooks

TaskGroup

XCOM

Dynamic Tasks

XCOM vs Variable

Практика № 9. Финальная практика, включающая в себя все вышеизученное



ШКОЛА БОЛЬШИХ ДАННЫХ

Что такое Apache Airflow?

Оркестратор рабочих процессов: Airflow предоставляет среду для программного создания, планирования и управления рабочими процессами.

Написан на Python: Реализован на языке программирования Python, что делает его гибким и расширяемым инструментом.

Open-source с активным комьюнити: Airflow распространяется по лицензии open-source, что позволяет пользователям свободно использовать, модифицировать и распространять его. Комьюнити Airflow активно развивается и поддерживает разнообразие плагинов и расширений.

Apache Airflow - это мощный инструмент для оркестрации рабочих процессов, который позволяет автоматизировать создание, планирование и мониторинг задач.

Почему Airflow?

Масштабируемость
модульная архитектура,
очередь сообщений

Кастомизация
возможность настройки
собственных операторов

Инструментарий
WEB UI, CLI, REST API

Интеграция со множеством
сервисов и источников
данных

Apache [Spark, Hive,
Hadoop], MySQL, Postgres,
MongoDB, Redis e.t.c.

Мониторинг и алерting
поддерживается
множество интеграций с
различными сервисами
сбора и отправки логов и
метрик

Ролевой доступ
5 ролей по дефолту +
возможна интеграция с
Active Directory

Поддержка и
тестирование
Можно добавлять базовые
unit тесты, которые могут
проверять как пайплайны
в целом, так и отдельные
таски

История создания Airflow

- ❖ October 2014: начало разработки (Maxime Beauchemin, Airbnb)
- ❖ June 2015: анонсирован в Airbnb GitHub
- ❖ March 2016: вступление в Apache Software Foundation's Incubator program
- ❖ January 2019: проект верхнего уровня в Apache Software Foundation
- ❖ October 2020: версия 1.10.14
- ❖ December 2020: версия 2.0.0
- ❖ сейчас - 2.9.x

Сравнение с прямыми конкурентами



- + Python для DAG's
- + Лучший WEB UI и API
- + Продвинутые метрики
- + Возможность создания сложных workflow's
- + Connections to HDFS, HIVE и т.д.
- + Будущее + комьюнити



- Java or XML для DAG's
- ужасный WEB UI и Java API
- умирающий, маленькое комьюнити



ШКОЛА БОЛЬШИХ ДАННЫХ

Киты Airflow



**НЕ инструмент
разработки**



Написан на python

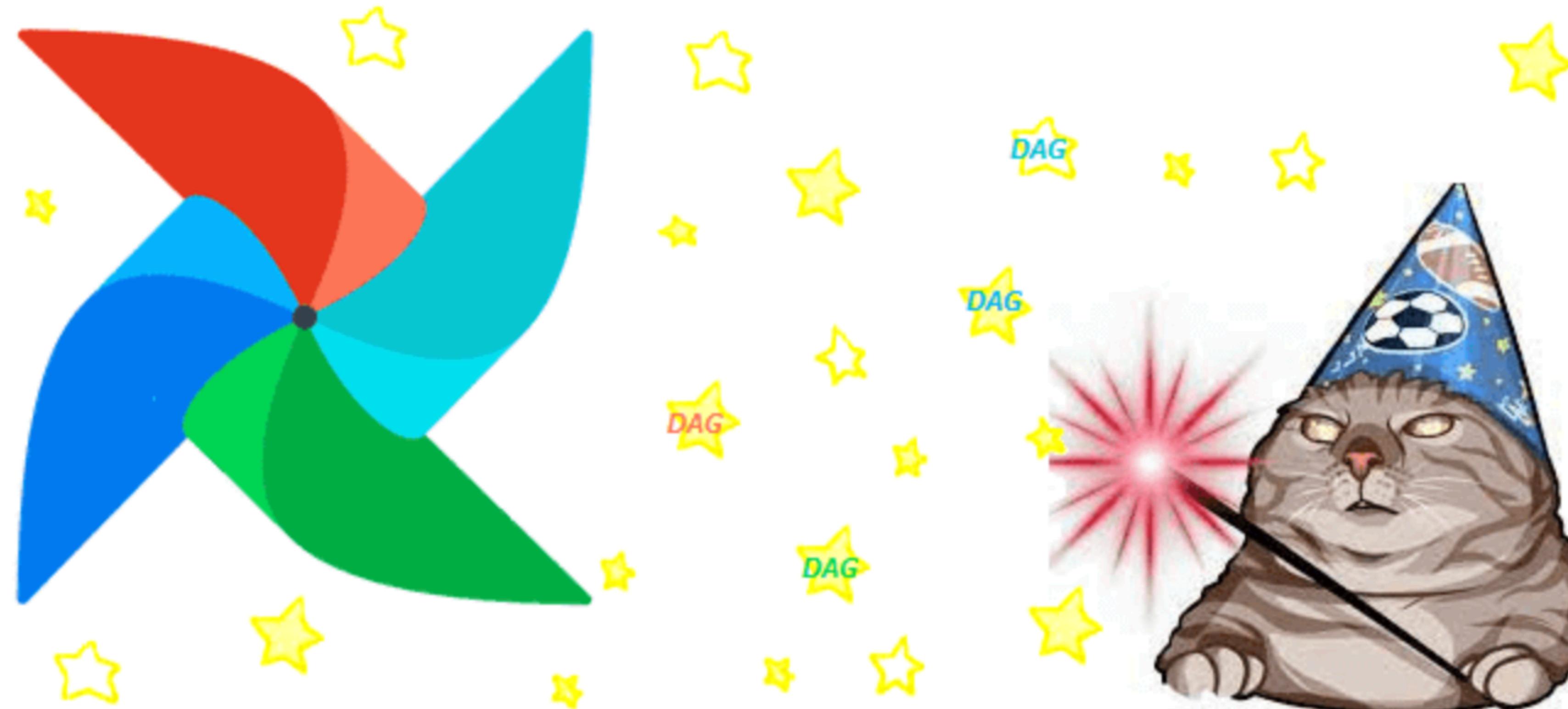


**НЕ ETL
инструмент**



ШКОЛА БОЛЬШИХ ДАННЫХ

Вопросы?



ШКОЛА БОЛЬШИХ ДАННЫХ

Managed Service for Apache Airflow

<https://yandex.cloud/ru/docs/managed-airflow/operations/>

Пошаговые инструкции для Managed Service for Apache Airflow™

Статья создана  Yandex Cloud

Обновлена 28 марта 2024 г.

- [Создание кластера](#)
- [Изменение кластера](#)
- [Работа с интерфейсами Apache Airflow™](#)
- [Загрузка DAG-файлов в кластер](#)
- [Удаление кластера](#)



ШКОЛА БОЛЬШИХ ДАННЫХ

Managed Service for Apache Airflow

<https://yandex.cloud/ru/docs/managed-airflow/operations/cluster-create>

Создание кластера Apache Airflow™

Статья создана  Yandex Cloud Обновлена 14 июня 2024 г.

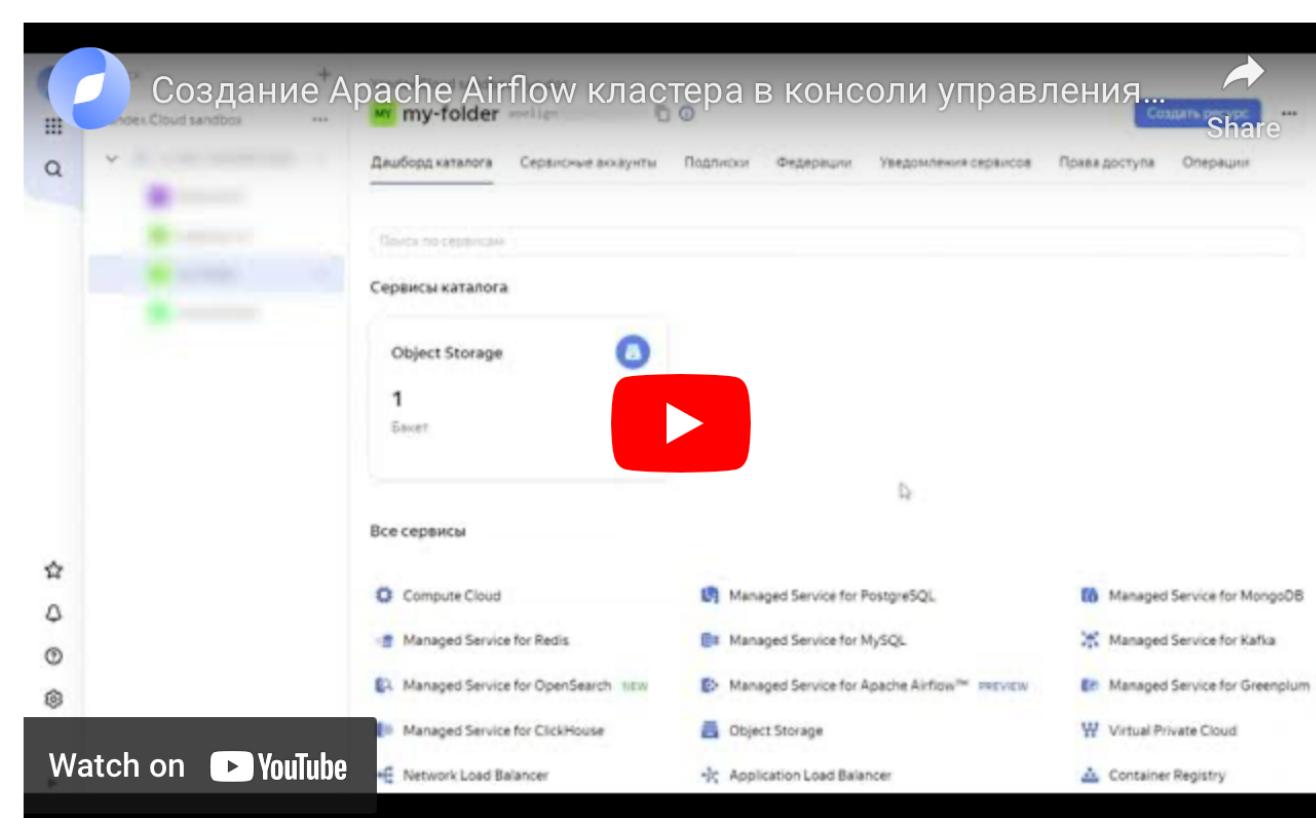
Каждый [кластер Managed Service for Apache Airflow™](#) состоит из набора компонентов Apache Airflow™, каждый из которых может быть представлен в нескольких экземплярах. Экземпляры могут находиться в разных зонах доступности.

Перед созданием кластера

1. В каталоге, в котором планируется создать кластер, [создайте сервисный аккаунт](#) с ролью `managed-airflow.integrationProvider`.
2. [Создайте статический ключ доступа](#) для сервисного аккаунта.
3. [Создайте бакет Yandex Object Storage](#), в котором будут храниться [DAG-файлы](#).

Создайте кластер

Консоль управления



1. В консоли управления  выберите каталог, в котором нужно создать кластер.

1. В [консоли управления](#)  выберите каталог, в котором нужно создать кластер.

2. Выберите сервис **Managed Service for Apache Airflow™**.

3. Нажмите кнопку **Создать кластер**.

4. В блоке **Базовые параметры**:

4.1. Введите имя кластера. Имя должно быть уникальным в рамках каталога.

4.2. (Опционально) Введите описание кластера.

4.3. (Опционально) Создайте [метки](#):

4.3.1. Нажмите кнопку **Добавить метку**.

4.3.2. Введите метку в формате `ключ: значение`.

4.3.3. Нажмите **Enter**.

5. В блоке **Настройки доступа**:

- Задайте пароль пользователя-администратора. Пароль должен иметь длину не менее 8 символов и содержать как минимум:

- одну заглавную букву;
- одну строчную букву;
- одну цифру;
- один специальный символ.

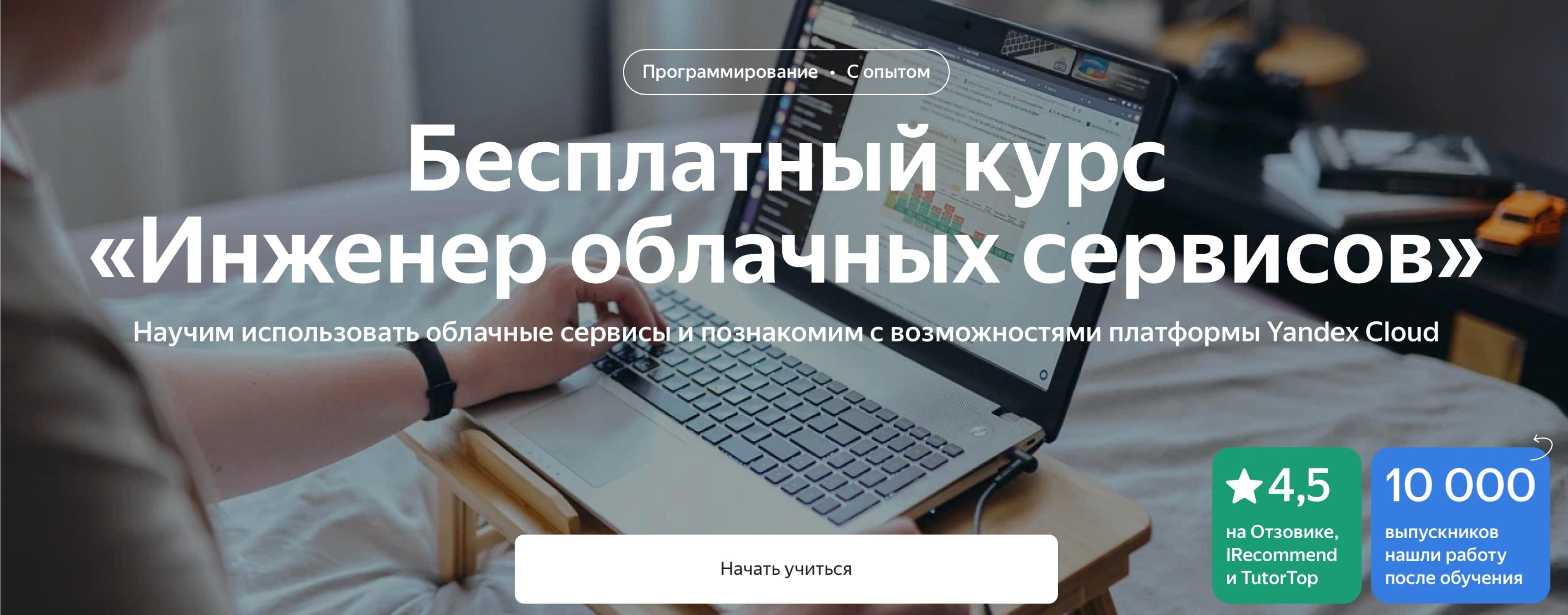
Примечание

Сохраните пароль локально или запомните его. Сервис не показывает пароли после создания.

- Выберите [созданный ранее](#) сервисный аккаунт с ролью `managed-airflow.integrationProvider`.



Курс по облачным сервисам



Программирование · С опытом

Бесплатный курс «Инженер облачных сервисов»

Научим использовать облачные сервисы и познакомим с возможностями платформы Yandex Cloud

Начать учиться

★ 4,5
на Отзовике,
IRecommend
и TutorTop

10 000
выпускников
нашли работу
после обучения



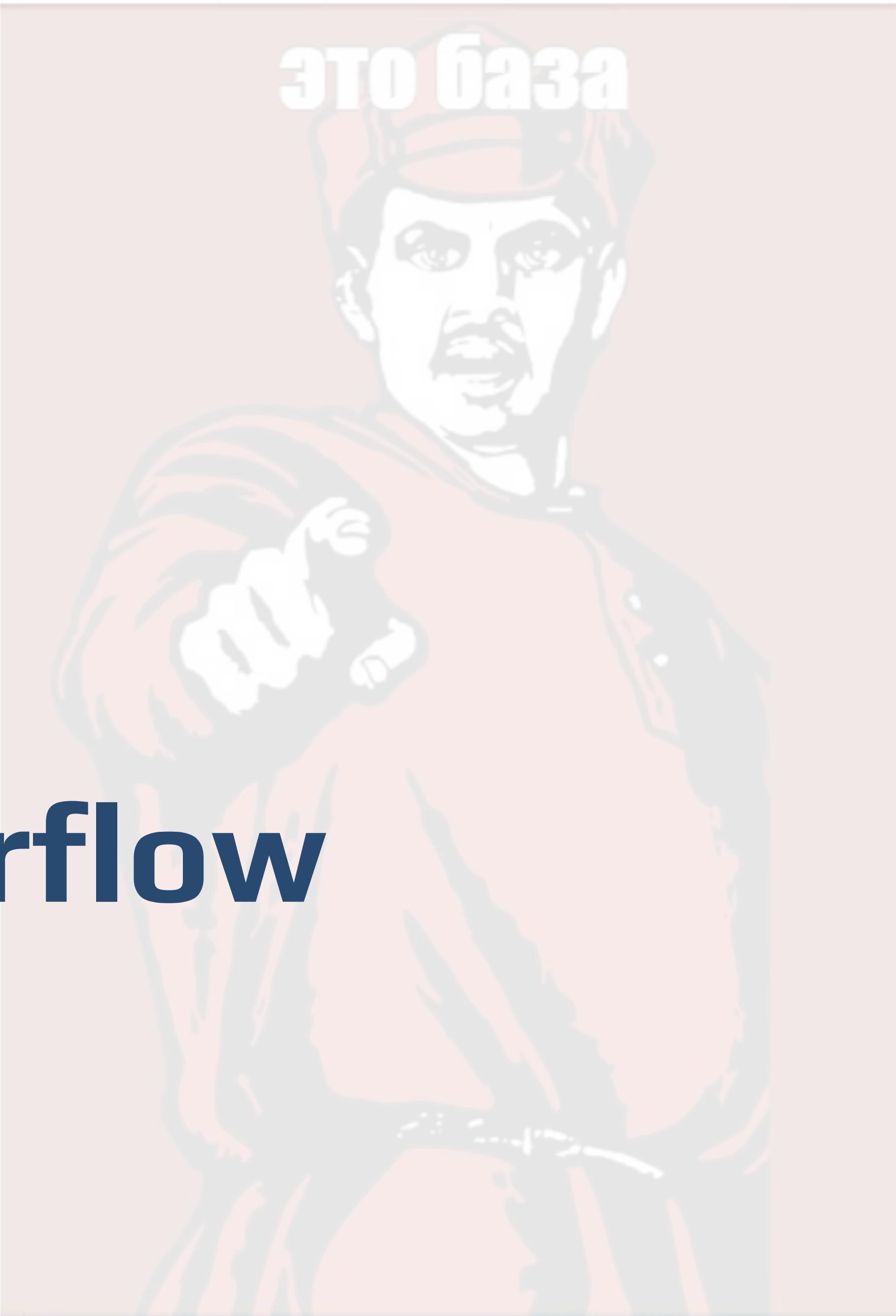
ШКОЛА БОЛЬШИХ ДАННЫХ



Apache
Airflow

Базовый Airflow

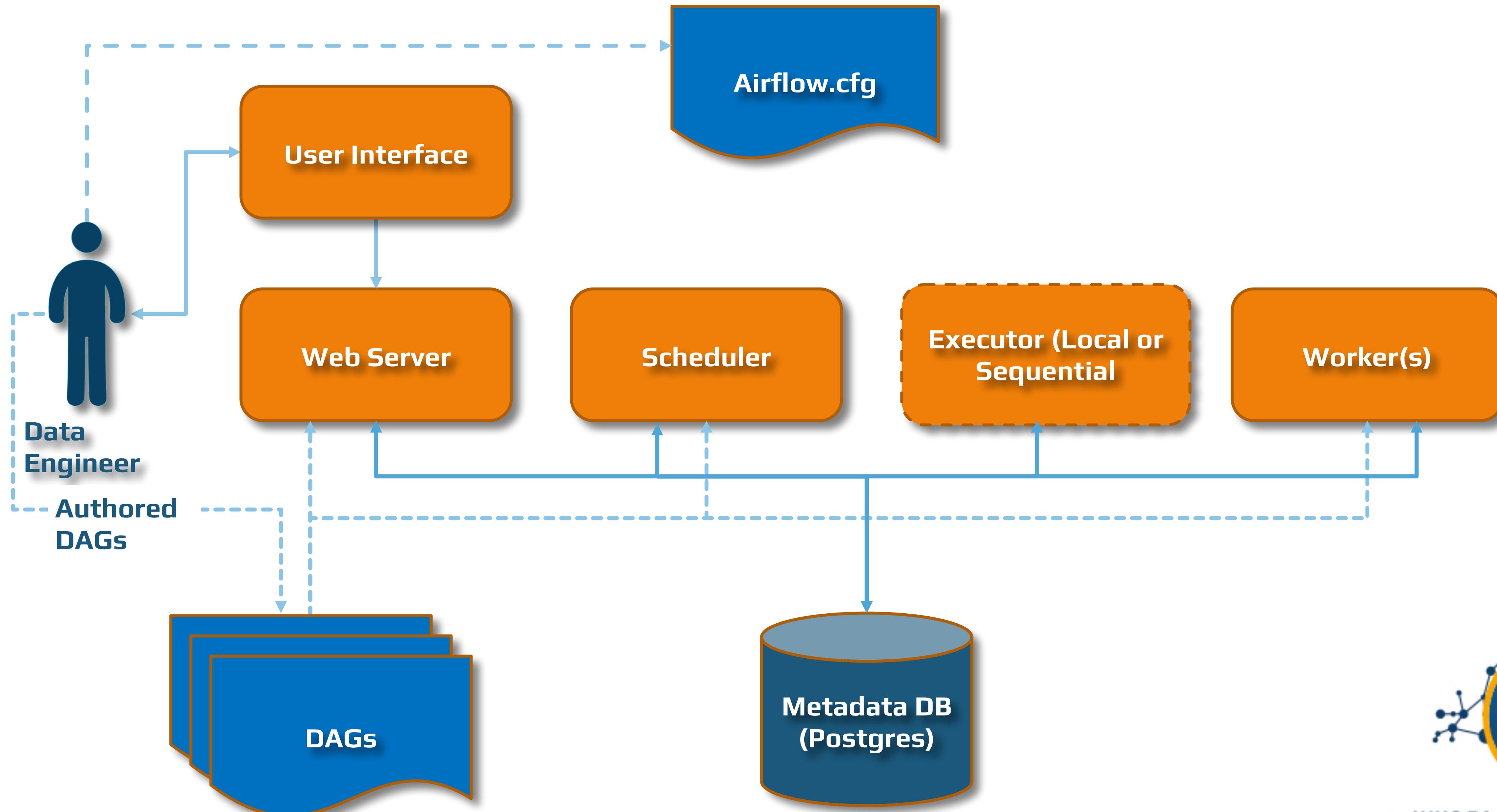
модуль 2



ШКОЛА БОЛЬШИХ ДАННЫХ

Компоненты

Верхнеуровневая архитектура



Компоненты: подробнее

scheduler

Отвечает за планирование и запуск задач в соответствии с заданным расписанием.

Работает непрерывно, периодически проверяя расписание задач. Когда время выполнения задачи наступает, планировщик отправляет задачи на исполнителя (executor) для выполнения.

worker

Рабочий процесс или контейнер, который фактически выполняет задачи, полученные от исполнителя (executor). Это может быть физический сервер, виртуальная машина или контейнер Docker, где запускаются задачи и операторы, определенные в дагах (DAGs).

webserver

Web интерфейс

executor

Отправляет задачи на рабочие процессы или контейнеры (worker). Обрабатывает задачи в соответствии с определенными правилами и настройками, такими как количество одновременно выполняемых задач, приоритеты и другие параметры. В Airflow доступны различные исполнители, такие как SequentialExecutor, LocalExecutor, CeleryExecutor и другие, каждый из которых имеет свои особенности и применение в зависимости от требований к масштабируемости и надежности.

metadata db

База данных, которая хранит метаданные о задачах, DAGs (Directed Acyclic Graphs - направленные ациклические графы), операторах и других объектах, управляемых Apache Airflow. Эти метаданные включают в себя информацию о состоянии выполнения задач, их зависимостях, параметрах запуска и других важных аспектах, необходимых для правильного выполнения и мониторинга рабочих процессов.



Executor

Отправляет задачи на worker.

Обрабатывает задачи в соответствии с определенными правилами и настройками, такими как количество одновременно выполняемых задач, приоритеты и т. д.

SequentialExecutor

(по умолчанию): Задачи выполняются последовательно в одном процессе. Этот режим прост в настройке и подходит для разработки и тестирования, но не рекомендуется для продакшн-среды.

✓ не требует отдельных настроек

- ограниченный параллелизм
- слабая отказоустойчивость
- последовательное выполнение задач на одном worker
- не масштабируется

LocalExecutor

Задачи выполняются параллельно в рамках локальной среды. Каждая задача запускается в отдельном процессе. Этот режим хорошо подходит для небольших проектов с ограниченным количеством задач

✓ простой и удобный в использовании

- ограничения, связанные с ресурсами локальной машины

CeleryExecutor

Задачи выполняются с использованием Celery, распределенной системы обработки задач. В этом режиме задачи могут быть выполнены на разных рабочих узлах. Этот режим позволяет обрабатывать большие объемы задач и масштабироваться горизонтально.

✓ распределенное выполнение
✓ масштабируемость
✓ отказоустойчивость

- сложность настройки
- зависимость от внешних сервисов
- доп. накладные расходы

KubernetesExecutor

Задачи выполняются в контейнерах Kubernetes. Этот режим позволяет запускать задачи в кластере Kubernetes, что обеспечивает масштабируемость и изоляцию задач.

✓ распределенное выполнение
✓ масштабируемость
✓ отказоустойчивость

- сложность настройки
- зависимость от внешних сервисов
- доп. накладные расходы



ШКОЛА БОЛЬШИХ ДАННЫХ

LocalExecutor

dagConcurrency

определяет максимальное количество дагов, которые могут быть выполнены параллельно.
Значение по умолчанию равно 16.

taskConcurrency

определяет максимальное количество задач, которые могут быть выполнены параллельно в пределах одного DAG Run. Значение по умолчанию равно 64.

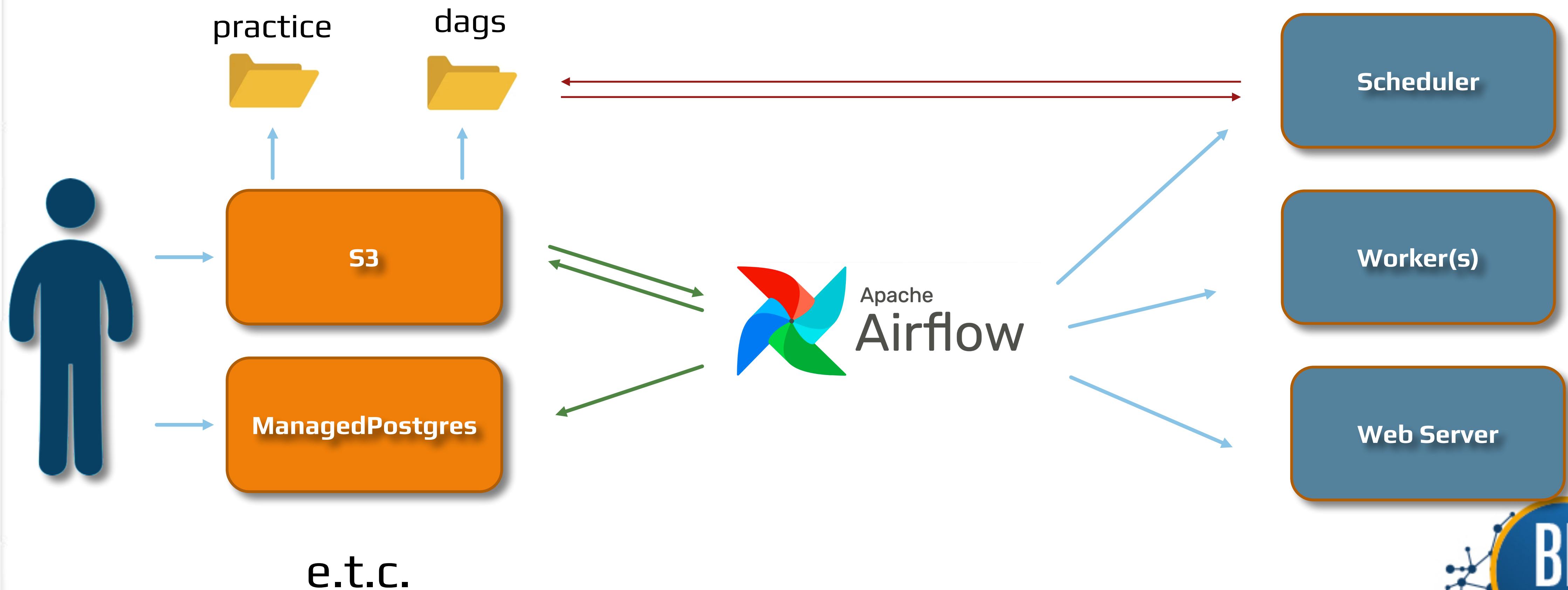
maxActiveRunsPerDag

определяет максимальное количество активных запусков для каждого DAG. Значение по умолчанию равно 16.



ШКОЛА БОЛЬШИХ ДАННЫХ

Обучение: как все устроено



ШКОЛА БОЛЬШИХ ДАННЫХ

DAG

A **DAG (Directed Acyclic Graph)** is the core concept of Airflow, collecting Tasks together, organized with dependencies and relationships to say how they should run.

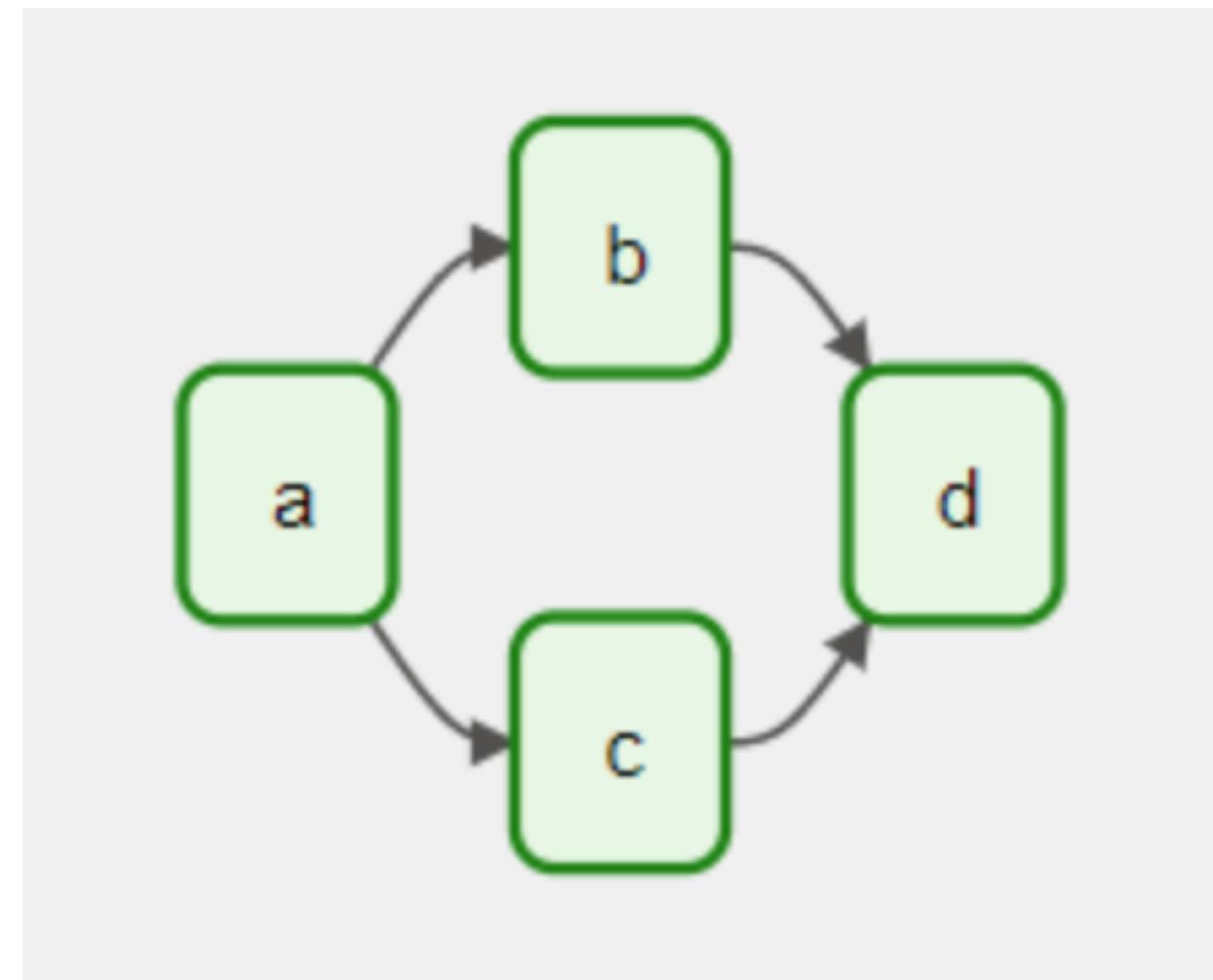
DAG

Граф, который представляет собой направленный ациклический граф задач (операторов), которые нужно выполнить в определенном порядке.

DAG определяет зависимости между задачами и определяет порядок их выполнения.

Состоит из узлов (операторов)

Basic example DAG



ШКОЛА БОЛЬШИХ ДАННЫХ

DAG: основные параметры

dag_id

Уникальный идентификатор

schedule_interval

Интервал запуска DAG

Значение по дефолту - `@daily`

start_date

Дата начала выполнения

description

Описание DAG

catchup

определяет, должен ли DAG
"догонять" пропущенные запуски,
если он был выключен на некоторое время.
Значение по умолчанию - `True`

retries

Кол-во попыток рестарта.
Значение по умолчанию - `3`

retry_delay

Время между рестартами.
Значение по умолчанию - `300` сек

DAG: еще про параметры

Параметров у DAG достаточно много, которые можно найти в официальной документации:
<https://airflow.apache.org/docs/>

default_args

словарь с аргументами по умолчанию для операторов в DAG.

```
default_args = {
    'owner': 'anpkartashov',
    'provide_context': True,
    'email': ['anpkartashov@beeline.ru'],
    'email_on_failure': True,
    'start_date': datetime( year: 2023, month: 8, day: 1, tzinfo=local_tz),
    'retry_delay': timedelta(minutes=5),
    'retries': 2
}
```



ШКОЛА БОЛЬШИХ ДАННЫХ

Operator

While **DAGs** describe how to run a workflow, **operators** determine what actually gets done

Operator

Абстрактный класс или функция, которая представляет собой конкретное действие или задачу, которую нужно выполнить в рамках DAG.

Операторы могут выполнять различные действия, такие как выполнение SQL-запроса, запуск скрипта Python, отправка электронной почты и другие операции.

Operator

Основные типы и виды операторов

Назначение

Operator - базовый класс
Sensor - ожидает событие

«Язык»

PythonOperator
BashOperator
EmailOperator
SimpleHttpOperator
PostgresOperator
...



ШКОЛА БОЛЬШИХ ДАННЫХ

Обзор WEB UI

Возможности:

- ❖ Просмотр состояния и истории DAGов
- ❖ Статистика выполнения DAGов
- ❖ Логи
- ❖ Рендер переменных
- ❖ Просмотр исходного кода DAGa
- ❖ Различное конфигурирование
(переменные, коннекты...)
- ❖ Старт - стоп DAGов
- ❖ Визуальное представление DAG
- ❖ И многое другое

Создание DAG (pipeline)



DAG (link)

```

from airflow.models.dag import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from datetime import datetime

# Функция для задания PythonOperator
def print_name(name):
    print(f"My name is {name}")

# Определение параметров DAG
default_args = {
    'start_date': datetime(2023, 1, 15),
    'description': "Привет, мир",
}

# Инициализация DAG
with DAG(
    dag_id="first_dag",
    default_args=default_args,
    schedule_interval="@once",
    tags=["lect1", "first_dag", "apk"]
) as dag:
    # Задачи DAG
    start = EmptyOperator(task_id="start")

    with_bash = BashOperator(
        task_id="bash_task",
        bash_command="echo Hello, world",
    )

    pwd_cmd = BashOperator(
        task_id="pwd_cmd",
        bash_command='pwd'
    )

    with_python = PythonOperator(
        task_id="python_task",
        python_callable=print_name,
        op_args=["Andrei"]
    )

    stop = EmptyOperator(task_id="stop")

    # Определение последовательности выполнения задач
    start >> [with_bash, with_python] >> stop

```

1. Импорты

2. Функция, команда, запрос

3. Создание объекта DAG

4. Создание операторов

5. Композиция



DAG: Context (**kwargs) (link)

В **Apache Airflow**, контекст (context) представляет собой словарь, который содержит информацию о текущем выполнении задачи в **DAG**. Этот словарь передается в функцию DAG-оператора в качестве аргумента.

Контекст содержит различные полезные данные, такие как:

- `execution_date`: дата и время запуска задачи
- `task_instance`: экземпляр задачи, который предоставляет доступ к различным атрибутам и методам задачи
- `dag_run`: экземпляр DAG-запуска, который предоставляет доступ к информации о текущем запуске DAG

Например, использовав в операторе данную ф-ию, в логах можно будет увидеть контекст выполнения задачи:

```
def print_context_f(**kwargs):
    pprint(kwargs)
```

Operator

Обязательные параметры для любого оператора

Название параметра	Описание	Формат
task_id	Уникальное название задачи	<code>string</code>
В зависимости от типа - функция, команда, запрос	Инструкция для выполнения	В зависимости от оператора
dag	Связь с объектом dag	В виде переменной, в которой объявлен dag

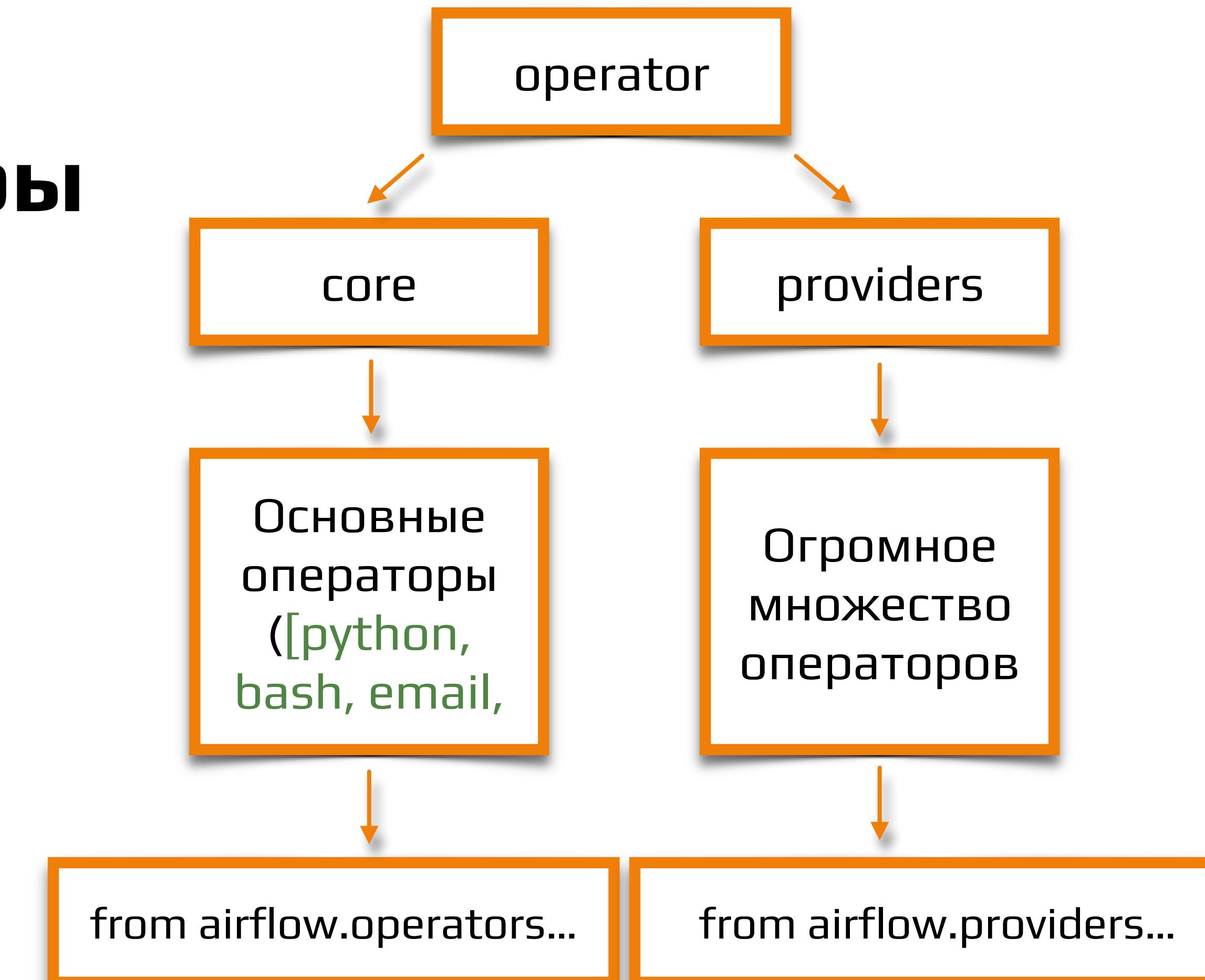


ШКОЛА БОЛЬШИХ ДАННЫХ

Operator

Основные параметры

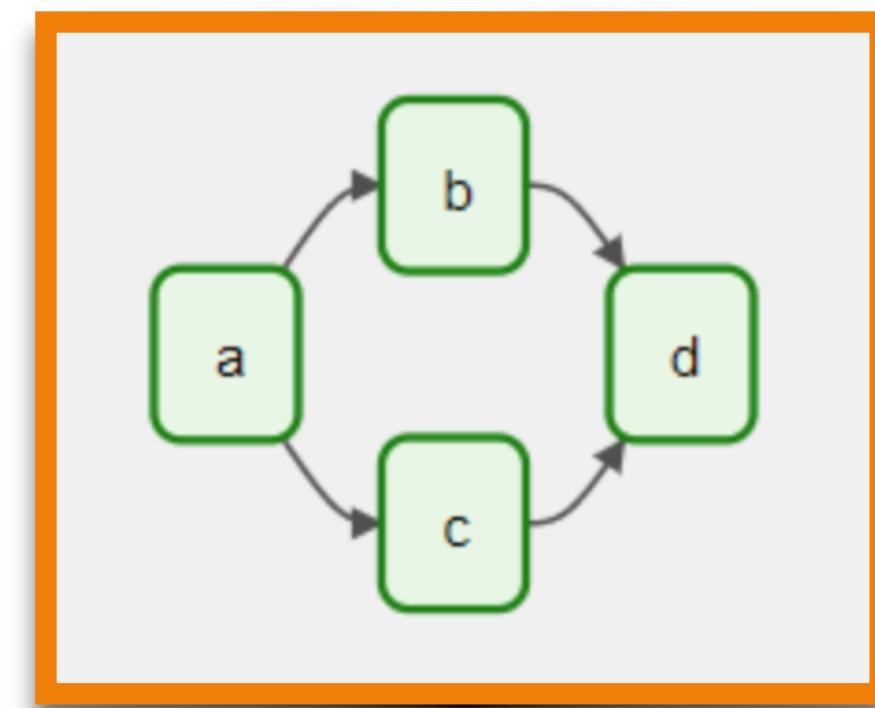
- ✓ `id, description`
- ✓ `dag`
- ✓ `trigger_rule`
- ✓ `action`
 - `PythonOperator (some_func)`
 - `BashOperator (script or file)`
 - `HiveOperator (hql)`
 - `[Postgres, MySQL]Operator (sql)`
 - ...
- ✓ ...



Композиция

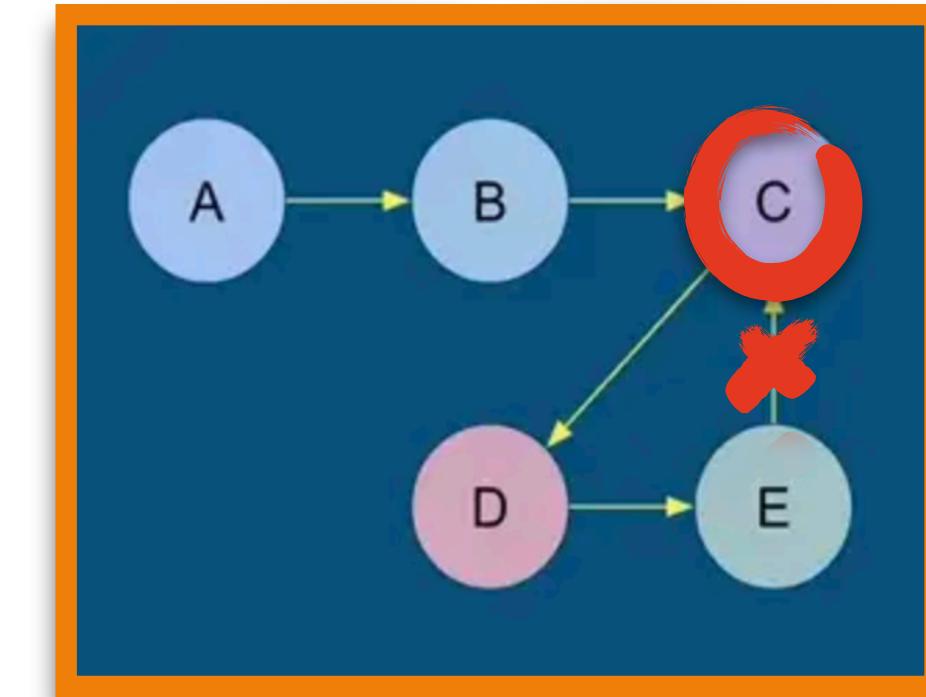
operator2 **следует** за operator1 (operator1 >> operator2)

Basic example DAG



a >> [b, c] >> d

! DAG Import Errors (1)



Broken DAG: [/opt/airflow/dags/broken_dag.py]

Cycle detected in DAG: broken_dag. Faulty task: c



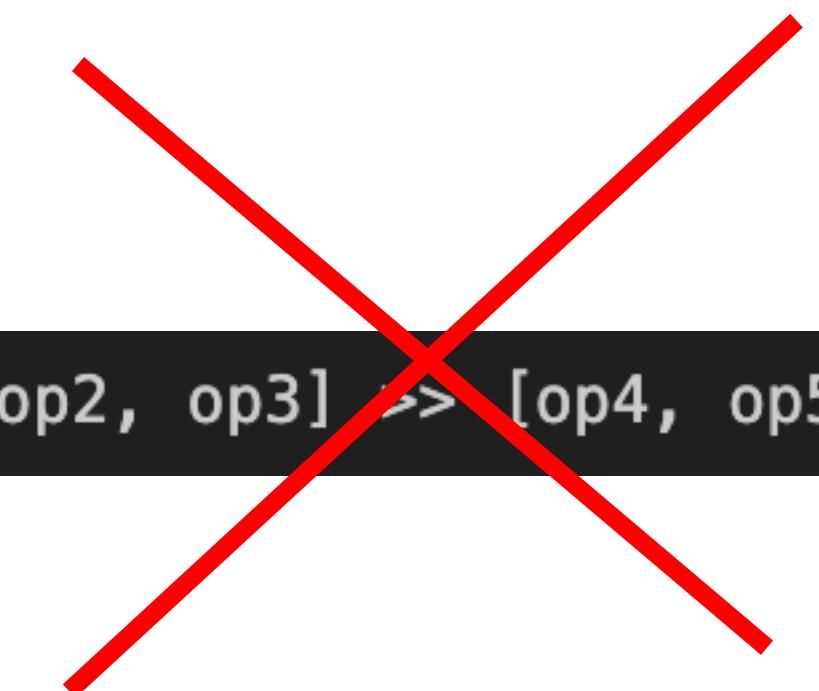
ШКОЛА БОЛЬШИХ ДАННЫХ

Композиция

❖ bitshift композиция (>><<)

```
op1 >> [op2, op3]
[op2, op3] >> [op4, op5]
op5 >> op6
```

~~op1 >> [op2, op3] >> [op4, op5] >> op6~~



❖ .set_[upstream, downstream]()

```
op2.set_upstream(op1)
op3.set_upstream(op1)

op4.set_upstream(op2)
op4.set_upstream(op3)

op5.set_upstream(op2)
op5.set_upstream(op3)

op6.set_upstream(op5)
```

EmptyOperator

EmptyOperator - это оператор, который не выполняет никаких действий и не выполняет никакой работы.

```
start = EmptyOperator(task_id="start")
```



start и stop здесь являются EmptyOperator, чтобы обозначить точку входа и выхода



BashOperator

Чтобы написать **BashOperator** нужно передать параметр **bash_command** типа String

```
with_bash = BashOperator(  
    task_id="bash_task",  
    bash_command="echo Hello, world! ps: with bash",  
    dag=dag  
)| Kartasov, 17.01.2023, 04:16 • Initial commit
```



ШКОЛА БОЛЬШИХ ДАННЫХ

Написание первого DAG:

```

from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.empty import EmptyOperator
from datetime import datetime

cmd = "echo Hello, world"

with DAG(
    dag_id: "first_dag",
    start_date=datetime( year: 2023, month: 1, day: 15),
    description="Привет, мир",
    schedule_interval=None,
    tags=["lect1"]
) as dag:

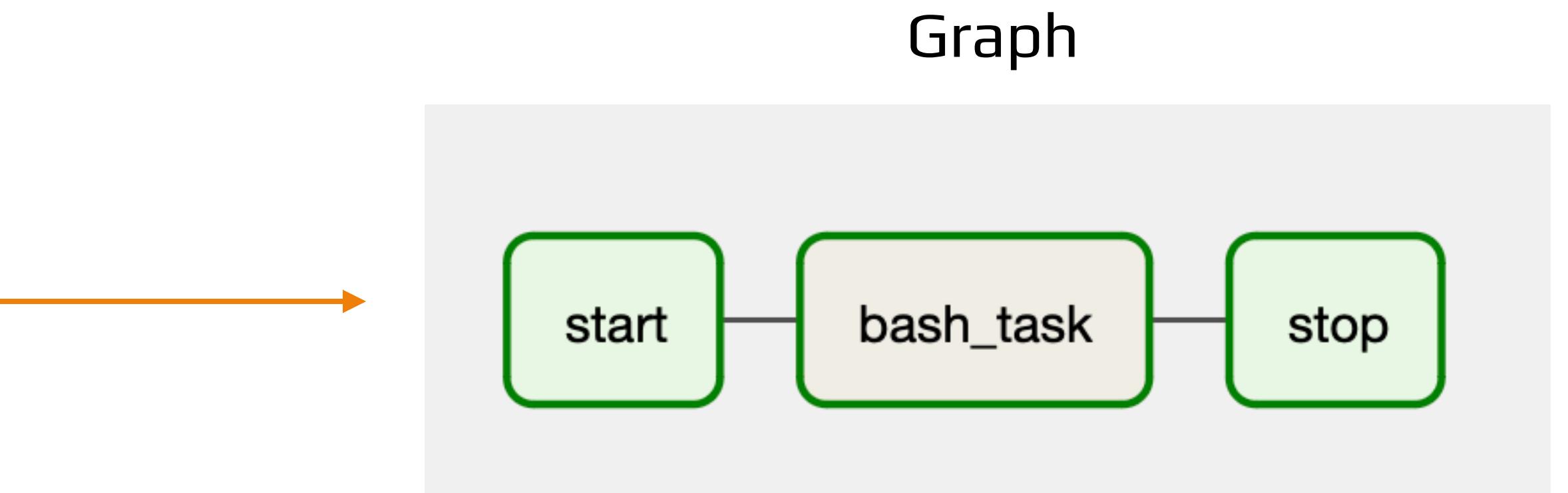
    start = EmptyOperator(task_id="start")

    with_bash = BashOperator(
        task_id="bash_task",
        bash_command=cmd,
        dag=dag
    )

    stop = EmptyOperator(task_id="stop")

    start >> with_bash >> stop

```



TaskFlowApi (link):

Альтернативный способ написания DAG'ов

Плюсы:

1. Более читаемый код (декораторы)
2. Более гибкое написание кода (возможно добавление проверок прямо в функцию - оператор)
3. Понятная передача аргументов в функцию

Минусы:

1. Совместимость с некоторыми провайдерами, которые ожидают классическую структуру
2. Сложнее в понимании уровней абстракций
3. Не популярный подход

```
from airflow.decorators import dag, task
@dag(
    dag_id="task_flow_api_dag",
    start_date=datetime(year=2023, month=1, day=15),
    description="Привет, мир и task flow api",
    schedule_interval="@once",
    tags=["lect1", "task_flow_api", "apk"]
)
def first_task_flow_api_dag():
    new *
    @task
    def print_hello_world():
        print("Hello, world!")

    new *
    def bash_command():
        import subprocess
        subprocess.run(['echo', 'Hello, world!'])

    new *
    @task
    def print_something_else(text):
        print(f"and something else {text}")

    # Инициализация задач с использованием Task Flow API
    op1 = print_hello_world()

    # Использование PythonOperator для вызова функции bash_command
    op2 = PythonOperator(
        task_id='python_bash_command',
        python_callable=bash_command
    )

    # Инициализация задачи с использованием Task Flow API
    op3 = print_something_else("something else")

    # Инициализация BashOperator
    op4 = BashOperator(
        task_id='bash_operator',
        bash_command='echo "Hello, world!"'
    )

    # Определение зависимостей задач
    op1 >> [op2, op3, op4]

    # Инициализация DAG
    dag = first_task_flow_api_dag()
```



ШКОЛА БОЛЬШИХ ДАННЫХ

PythonOperator

«Главный» оператор Apache Airflow

- ✓ **callable**

- Определяет обработку данных
- `return None` (как правило)

- ✓ **op_args:** positional params [`«some_param1»`,
`some_param2`]

- ✓ **op_kwargs:** dict params `{key: «value», key: «value»}`

PythonOperator (link)

Функция, которая определяет поведение задачи или саму задачу (как правило ничего не возвращает)

```
def print_hello_world(some_word):
    print(f"Hello, world! I'm {some_word}")
```



Варианты написания PythonOperator (слева аргументы передаются в списке, справа в словаре)

```
python_op = PythonOperator(
    task_id="python_op",
    python_callable=print_hello_world,
    op_args=["Andrei"]
)
```

```
python_op = PythonOperator(
    task_id="python_op",
    python_callable=print_hello_world,
    op_kwargs={"some_word": "Andrei"}
)
```



Краткое резюме



ШКОЛА БОЛЬШИХ ДАННЫХ

Практика. Введение.

Во всех написанных вами дагах обязательно наличие следующих тегов: **фамилия, {номер}_practice**

Пример: `kartashov, 01_practice`

Чтобы ваши даги появились в WebUI вам нужно скопировать их в каталог на S3:

`/dags/students/ваша_фамилия/ваш_даг.py`

Перед практическим занятием вы получите пройденные слайды презентации в pdf в чате тг
(напоминайте мне, если я забуду скинуть)

Сами задания находятся в каталоге: `/practice`

Рекомендация: так как мы не в школе, вы можете скопировать весь чужой код практики и вам ничего за это не будет, но лучше этого не делать и смотреть на пройденные примеры, а если непонятно спрашивать...



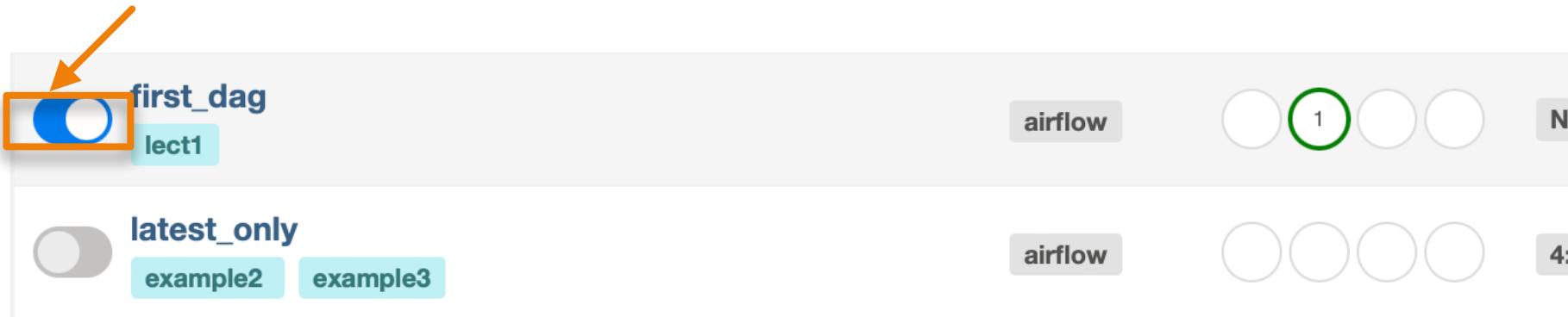
PythonOperator [«01_practice»]

Текст задания находится на:

`/practice_md/01_practice.md`

Запуск DAG + как запустить с параметрами

On/Off



В airflow есть возможность запуска с конфигурацией.

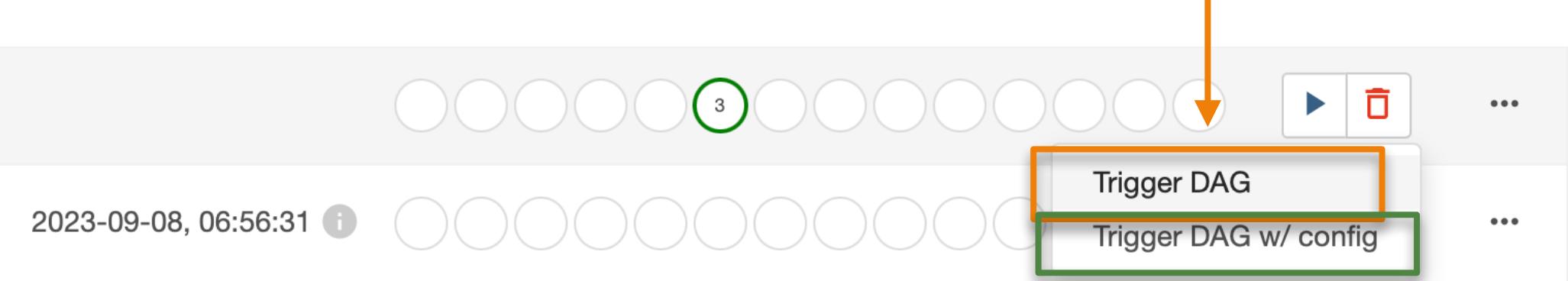
Предположим у вас есть несколько разных сред для выполнения вашего DAG (например **dev, stage, prod**), и вы хотите передавать соответствующую конфигурацию в зависимости от среды.

Вы можете создать несколько конфигураций и передавать их при запуске DAG.

`kwargs['dag_run'].conf` предоставляет доступ к словарю с параметрами выполнения, переданными при запуске DAG. Мы можем получить значение конкретного параметра выполнения, используя его ключ. В данном случае, `kwargs['dag_run'].conf['Foo']`

To access configuration in your DAG use `{{ dag_run.conf }}`. As `core.dag_run_conf_overrides_params` is set to `True`, so passing any configuration here will override task params which can be accessed via `{{ params }}`.

Запуск без параметров



Trigger DAG: practice_1

Logical date

2023-09-08 11:18:47+00

Run id (Optional)

Run ID

Configuration JSON (Optional, must be a dict object)

```
1 {
2   "Foo": "Fighters"
3 }
```



Запуск DAG + как запустить с параметрами

```
def my_task(**kwargs):
    dag_run = kwargs.get('dag_run')
    if dag_run:
        conf = dag_run.conf
        # Обработка параметров конфигурации
        print(f"Received config: {conf}")
    else:
        print("Start dag without conf")

default_args = {
    'owner': 'airflow',
    'start_date': days_ago(1),
}

with DAG(
    dag_id='my_dag',
    default_args=default_args,
    schedule_interval=None,
) as dag:
    task = PythonOperator(
        task_id='my_task',
        python_callable=my_task,
        provide_context=True,
    )
```



Operators: PythonOperator [«02_practice»]

Текст задания находится на:

/practice_md/02_practice.md