# Модуль 3.1 Управление потоком операций (xFlow)

## Введение

В проектах машинного обучения важную роль играют конвейеры. Они позволяют автоматизировать процесс создания и проверки конечного продукта — так же, как конвейеры в промышленных производствах, например, автомобилей.

Конвейеры машинного обучения состоят из различных операций. Ходом выполнения этих операций необходимо управлять:

* контролировать последовательность выполнения;
* осуществлять мониторинг;
* своевременно устранять ошибки;
* версионировать сами конвейеры и результаты их работы.

Операции в конвейере объединяются с использованием направленного ациклического графа (DAG, Directed Acyclic Graph). Последовательное и параллельное выполнение операций в порядке, заданном в DAG, создает поток (flow) действий и появляющихся артефактов. Для работы с таким потоком операций используют различные инструменты под общим условным названием xFlow. Два из них — AirFlow и MLFlow — мы рассмотрим далее. Предварительно нужно будет установить эти инструменты — инструкции вы также найдете в модуле.

СОДЕРЖАНИЕ ЮНИТОВ

|  |
| --- |
| **КАК МЫ БУДЕМ ДВИГАТЬСЯ ПО МОДУЛЮ** |
| 1. Установка AirFlow |
| 2. Примеры скриптов AirFlow |
| 3. Пример работы с AirFlow |
| 4. Установка MLFlow |
| 5. Пример использования MLFlow |

# Модуль 3.1 Юнит 1. Установка AirFlow

Прежде всего необходимо установить AirFlow. Этим мы и займемся в первом уроке.

|  |  |
| --- | --- |
| 1 | Установим все необходимые пакеты, которые потребуются для работы.  Для этого введем следующую команду:  **sudo apt-get update -y && sudo apt-get install software-properties-common -y && sudo apt-get install python-setuptools -y && sudo apt install python3-pip -y && sudo apt install libpq-dev -y && sudo apt install postgresql -y && sudo pip install psycopg2**  Рисунок 1. Установка пакетов |
| 2 | С помощью команды **export** указываем домашнюю директорию:  **export AIRFLOW\_HOME=~/airflow**  В ней будут храниться конфигурационные файлы, а также скрипты, которые будет запускать AirFlow. |
| 3 | Устанавливаем Airflow с помощью команды:  **sudo pip3 install apache-airflow**  Рисунок 2. Установка AirFlow |
| 4 | Вводим несколько команд, чтобы сделать следующее:   * создать пользователя; * задать пароль для пользователя; * создать базу данных; * выдать права пользователю.   **sudo -u postgres psql**  **postgres=# CREATE USER airflow PASSWORD password';**  **postgres=# CREATE DATABASE airflow;**  **postgres=# GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO airflow;**  Рисунок 3. Создание базы данных |
| 5 | Создаем конфиги внутри домашней директории. Запускаем инициализацию базы данных:  **airflow db init**  Рисунок 4. Инициализация данных  После выполнения команды в указанном месте появится директория. Заглянув в созданный каталог, увидим, что в нем содержатся конфигурационные данные.  Рисунок 5. Каталог с конфигурационными файлами |
| 6 | Связываем AirFlow с базой данных PostgreSQL. Для этого в файле airflow.cfg меняем значение переменной **sql\_alchemy\_conn**.  Строка в измененном виде должна выглядеть так:  **sql\_alchemy\_conn = postgresql+psycopg2://airflow:password@localhost/airflow**  Рисунок 6. Строка для изменения  Рисунок 7. Строка после изменения |
| 7 | Заново инициализируем базу данных, чтобы запустить работу с ней:  **airflow db init**  Рисунок 8. Инициализация базы данных |
| 8 | После подключения базы данных создаем пользователя для работы с AirFlow на сайте:  **airflow users create --username Admin --firstname admin --lastname test --role Admin --email airflow@airflow.com**  Рисунок 9. Создание пользователя  Также задаем для него пароль.  Рисунок 10. Установка пароля пользователя |

Установка завершена!

Можем приступать к работе с AirFlow.

Для этого необходимо запустить:

* веб-сервер:

**airflow webserver -p 8080**



Рисунок 11. Запуск веб-сервера

* scheduler, который будет обрабатывать все запущенные скрипты внутри AirFlow:

**airflow scheduler**

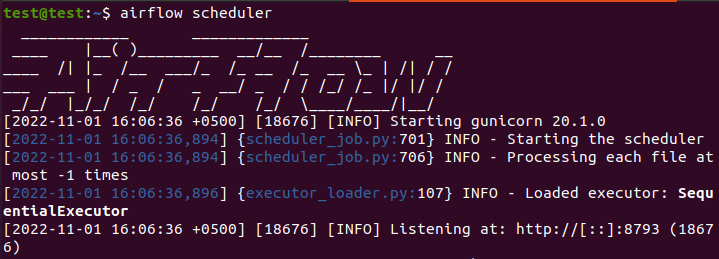


Рисунок 12. Запуск scheduler

В этом юните вы установили и запустили AirFlow. Далее разберем, как его использовать.

Тест

1. Что такое DAG? (0.25)

**[x] Направленный граф без циклов**

[] Directed Automatic Graph

**[x] Directed Acyclic Graph**

[] Матрица выполнения операций

1. Какая база данных используется для организации работы AirFlow? (0.25)

**(x) postgresql**

() mysql

() memsql

() sqlite

1. Какая команда инициализирует базу данных для AirFlow? (0.25)

() database airflow start

**(x) airflow db init**

() sudo systemctl db init

() db airflow start

1. Какой номер порта используется для запуска web-сервиса с пользовательским интерфейсом Apache? (0.25)

() 8888

**(x) 8080**

() 8000

() 8765

# Модуль 3. Юнит 2. Примеры скриптов AirFlow

Airflow — удобный и гибкий инструмент. Эти преимущества обеспечиваются тем, что логика его работы удобно описывается скриптовым языком Python.

В этом юните мы рассмотрим некоторые примеры скриптов в AirFlow.

### Исполнение кода в AirFlow

Перед вами пример скрипта (DAG):

from airflow import DAG

import airflow

from datetime import datetime

from airflow.operators.python import PythonOperator

def test123():

print("Hello world!")

args = {

'owner': 'dimon',

'start\_date':datetime(2018, 11, 1),

'provide\_context':True

}

with DAG('Hello-world\_example', description='Hello-world example', schedule\_interval='\*/1 \* \* \* \*', catchup=False, default\_args=args) as dag:

task\_1 = PythonOperator(task\_id="task\_1", python\_callable=test123)

Разберем его подробнее.

Создаем функцию **test123**, которая будет выводить текст «Hello world!». Далее указываем аргументы, которые позднее будут переданы функции DAG.

В переменную **task\_1** записываем результат функции PythonOperator, который будет использовать функцию **task123**.

Если скрипт сконфигурирован верно, он появится в домашней папке, указанной при установке AirFlow, — на вкладке **DAGs**.

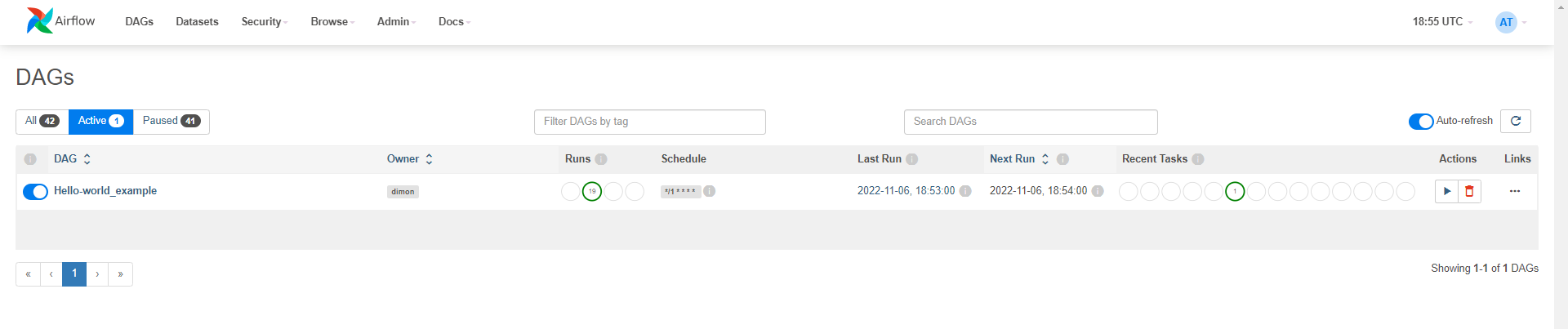


Рисунок Активные DAG

Если открыть конкретный DAG, можно увидеть прогресс выполнения при каждом запуске скрипта.

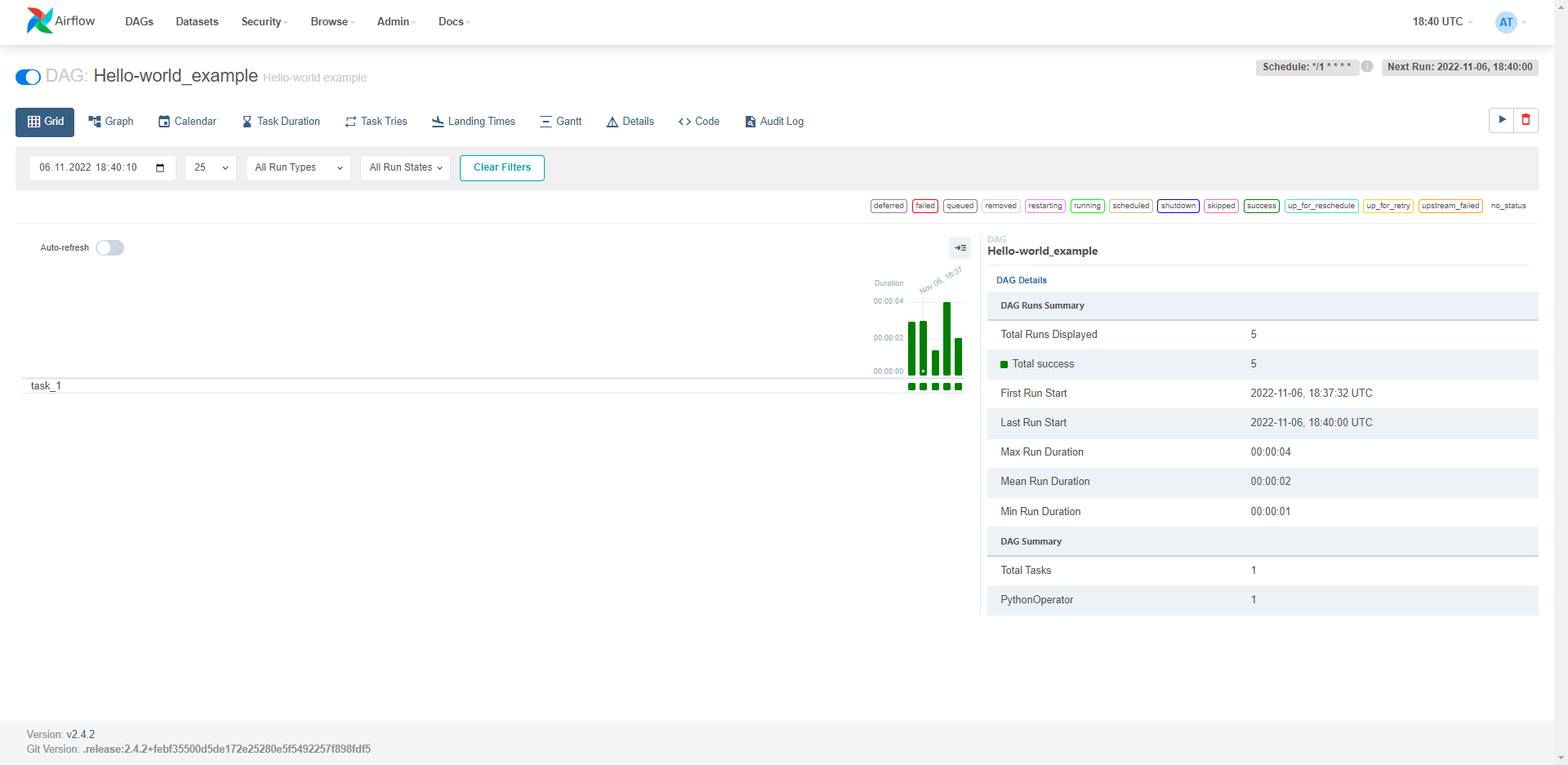


Рисунок . Статус выполнения DAG

Разберем, какая еще информация о DAG доступна в этом разделе.

* Во вкладке **Code** можно посмотреть код исполняемого DAG.

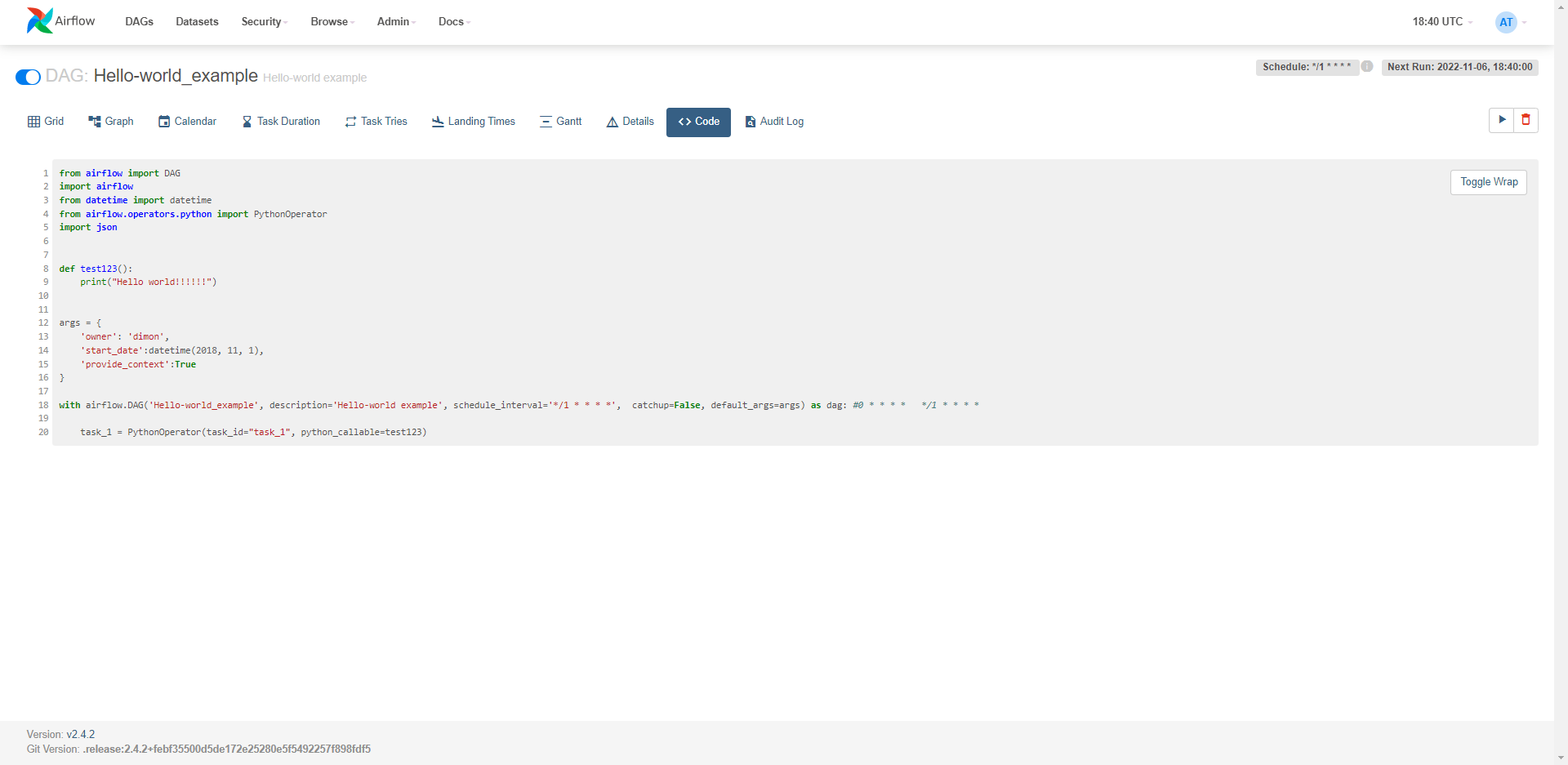


Рисунок . Код DAG

* На вкладке **Log** можно посмотреть состояние программы, например, при выводе текста во время ее выполнения.

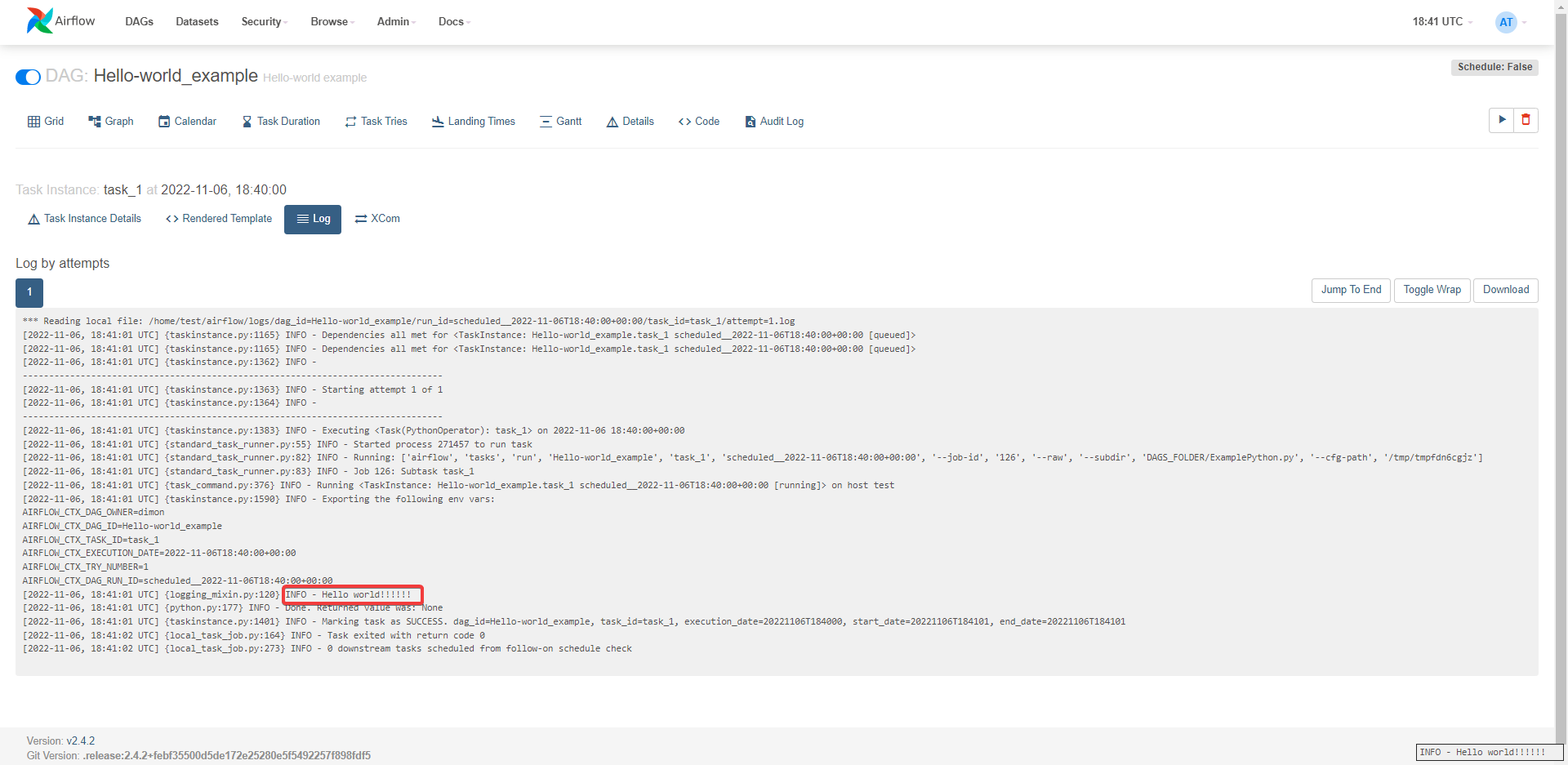


Рисунок Логирование выполения DAG и вывод print

* На вкладке **Graph** можно увидеть последовательность выполнения программы.

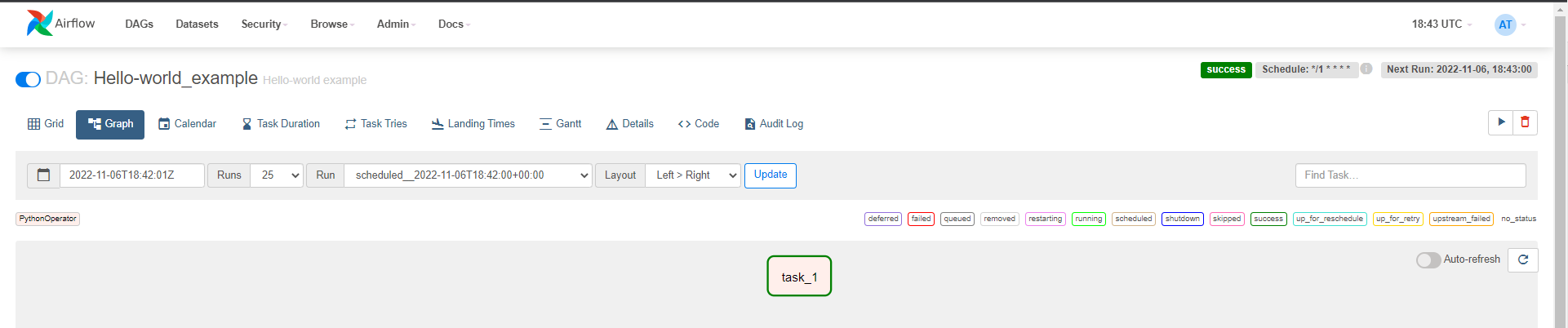


Рисунок График последовательности выполнения задач

### Использования bash-скриптов

from airflow import DAG

import airflow

from datetime import datetime

from airflow.operators.python import PythonOperator

from airflow.operators.bash import BashOperator

import json

args = {

    'owner': 'dimon',

    'start\_date':datetime(2018, 11, 1),

    'provide\_context':True

}

with airflow.DAG('LS\_Bash\_example', description='Hello-world example', schedule\_interval='\*/1 \* \* \* \*',  catchup=False, default\_args=args) as dag: #0 \* \* \* \*   \*/1 \* \* \* \*

    task\_1 = BashOperator(task\_id="task\_1", bash\_command="ls -la ~/airflow/dags")

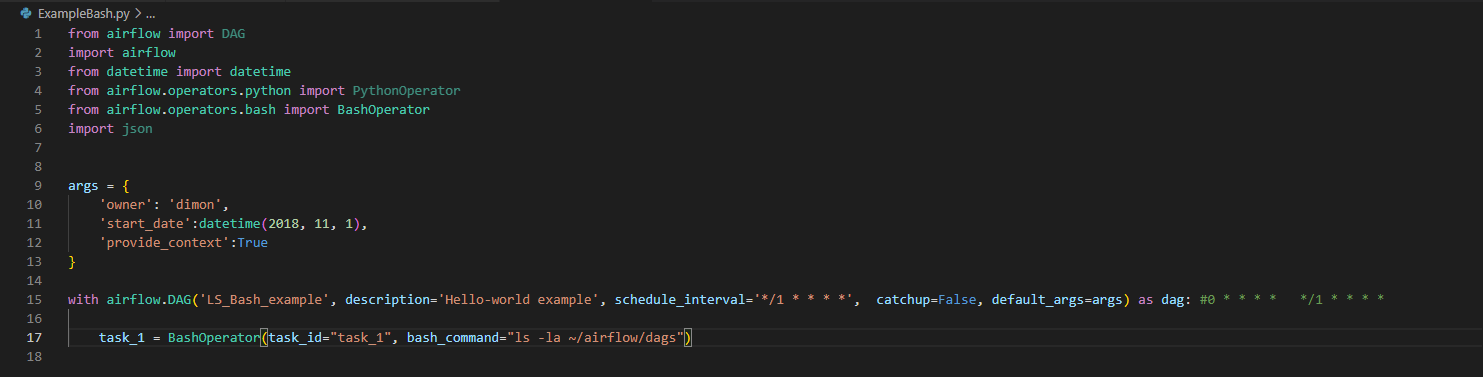


Рисунок Использование bash-скрипта

Результат выполнения Bash-скрипта выводится в log.

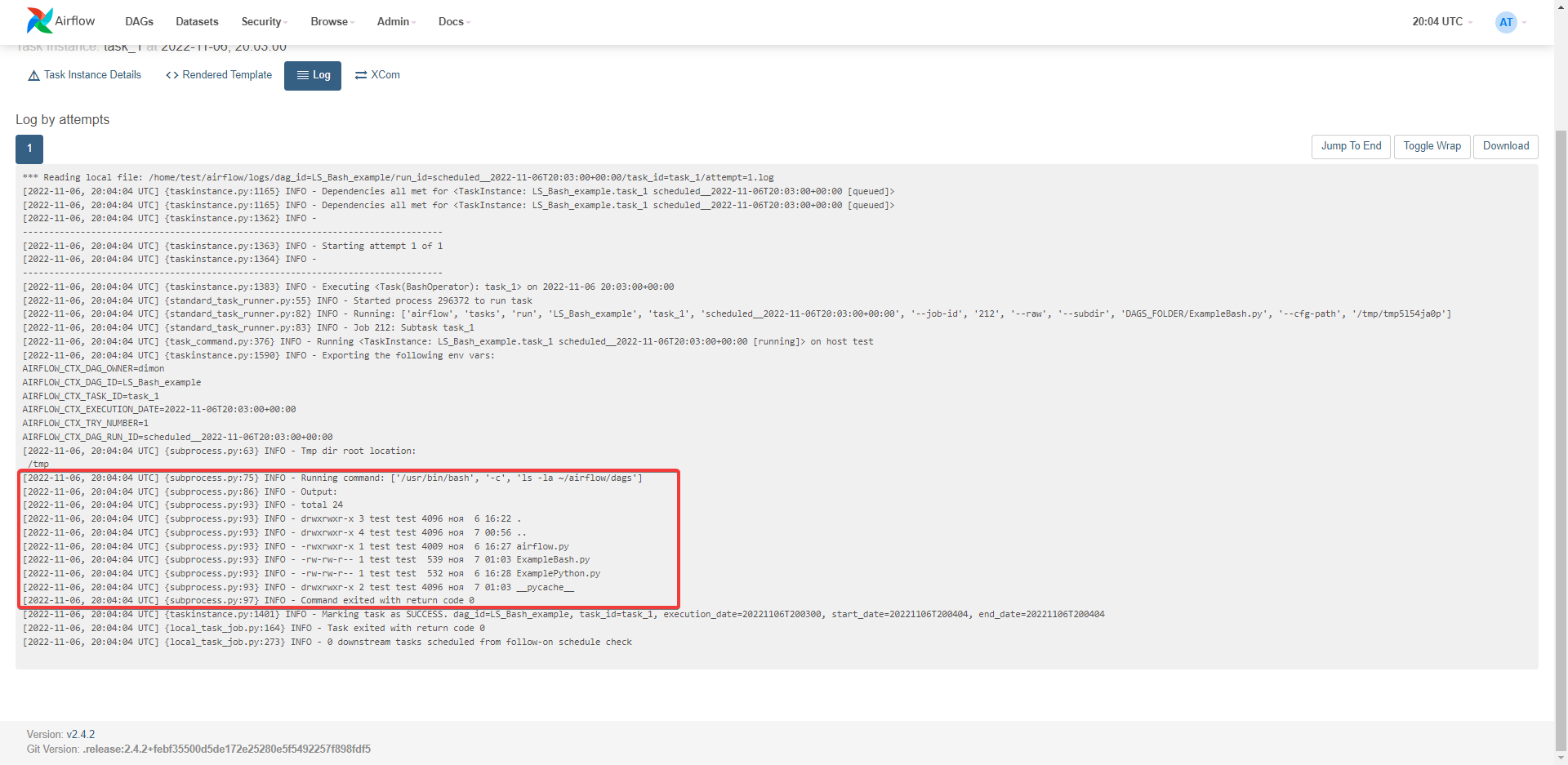


Рисунок Log при использовании bash-скрипта

Если команда завершилась из-за ошибки (например, файл не найден), то в графическом интерфейсе маркер, обозначающий выполнение заданного шага, будет красного цвета.

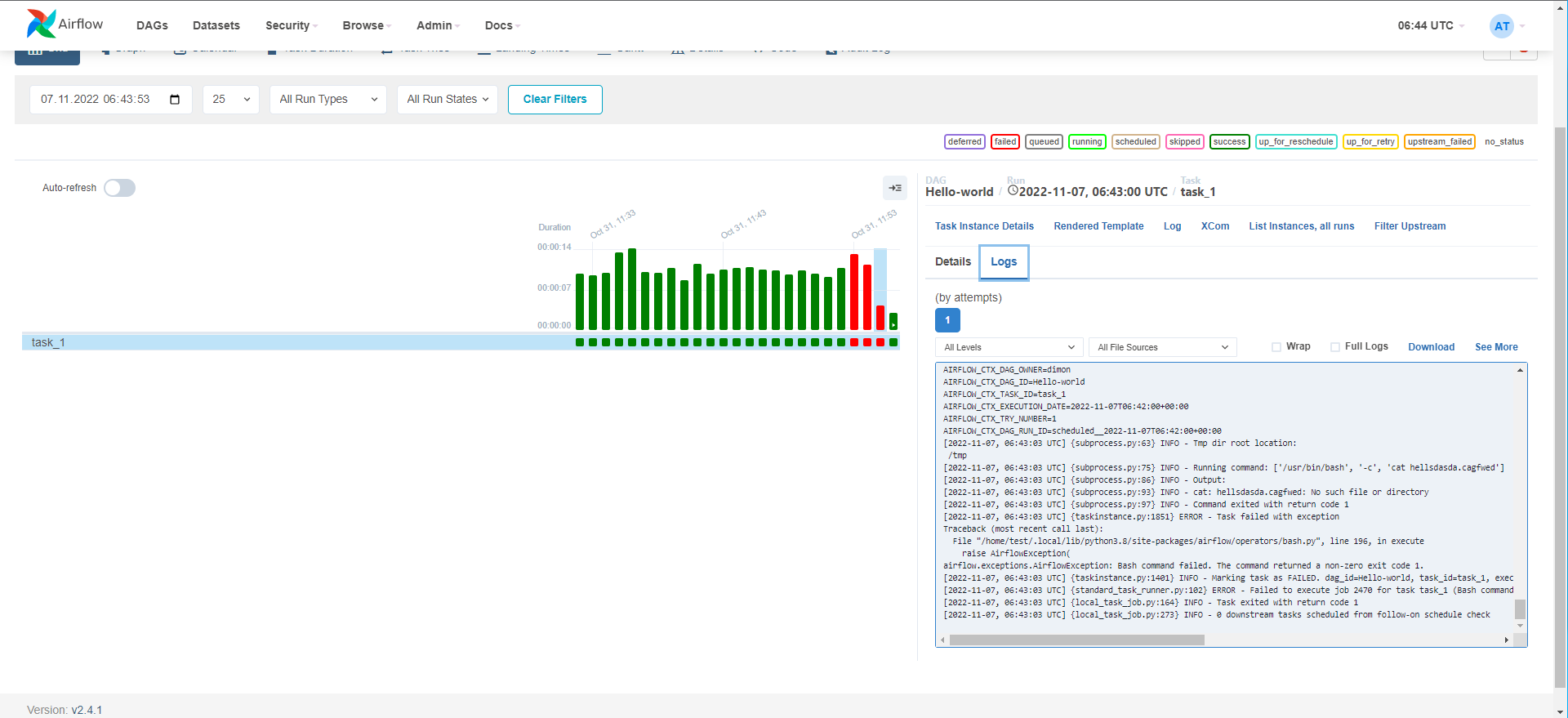


Рисунок Ошибка выполнения bash-скрипта

В этом юните мы рассмотрели, как выполняются некоторые скрипты в AirFlow. Далее мы подробнее разберем, как работать в этом инструменте.

# Модуль 3. Юнит 3. Пример работы с AirFlow

В этом юните вы рассмотрите пример работы с AirFlow.

Перед вами скрипт, реализующий модель линейной регрессии с разным количеством регрессоров, которое зависит от параметра:

from airflow import task, DAG

from datetime import datetime, timedelta

import airflow

from airflow.operators.python import PythonOperator

import json

def StepOne(ti):

from random import randint

import numpy as np

from sklearn.model\_selection import train\_test\_split

#ti = kwargs["ti"]

xs = np.linspace(0, 10, 50)

ys = xs\*\*2 + np.random.random(50) \* 10

ti.xcom\_push("xs", xs.tolist())

ti.xcom\_push("ys", ys.tolist())

def StepTwo(ti):

from random import randint

import numpy as np

from sklearn.model\_selection import train\_test\_split

xs = np.array(ti.xcom\_pull(task\_ids="task\_1", key="xs"))

# print(f"xcom pull {xs}")

# print(f"type is {type(xs)}")

xs.astype(int)

t = 1

if t == 0:

xs1 = np.c\_[xs]

if t == 1:

xs1 = np.c\_[xs, pow(xs,2)]

if t == 2:

xs1 = np.c\_[xs, pow(xs,2), pow(xs,3)]

if t == 3:

xs1 = np.c\_[xs, pow(xs,2), pow(xs,3), pow(xs,4)]

if t == 4:

xs1 = np.c\_[xs, pow(xs,2), pow(xs,3), pow(xs,4), pow(xs,5)]

print(xs1)

ti.xcom\_push("xs1", xs1.tolist())

def StepThree(ti):

from random import randint

import numpy as np

from sklearn.model\_selection import train\_test\_split

xs1 = np.array(ti.xcom\_pull(task\_ids="task\_2", key="xs1"))

ys = np.array(ti.xcom\_pull(task\_ids="task\_1", key="ys"))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(xs1, ys, test\_size=0.33, random\_state=42)

ti.xcom\_push("X\_train", X\_train.tolist())

ti.xcom\_push("X\_test", X\_test.tolist())

ti.xcom\_push("y\_train", y\_train.tolist())

ti.xcom\_push("y\_test", y\_test.tolist())

def StepFour(ti):

from random import randint

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

import jsonpickle

import json

#from keras.models import model\_from\_json

X\_train = np.array(ti.xcom\_pull(task\_ids="task\_3", key="X\_train"))

y\_train = np.array(ti.xcom\_pull(task\_ids="task\_3", key="y\_train"))

model = LinearRegression()

model.fit(X\_train, y\_train)

test = jsonpickle.encode(model)

ti.xcom\_push("model", test)

def StepFive(ti):

from random import randint

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

import jsonpickle

model = jsonpickle.decode(ti.xcom\_pull(task\_ids="task\_4", key="model"))

X\_test = np.array(ti.xcom\_pull(task\_ids="task\_3", key="X\_test"))

y\_test = np.array(ti.xcom\_pull(task\_ids="task\_3", key="y\_test"))

score = model.score(X\_test, y\_test)

ti.xcom\_push("Score", score)

args = {

'owner': 'dimon',

'start\_date':datetime(2018, 11, 1),

'provide\_context':True

}

with DAG('Hello-world', description='Hello-world', schedule\_interval='\*/1 \* \* \* \*', catchup=False, default\_args=args) as dag: #0 \* \* \* \* \*/1 \* \* \* \*

task\_1 = PythonOperator(task\_id="task\_1", python\_callable=StepOne)

task\_2 = PythonOperator(task\_id="task\_2", python\_callable=StepTwo)

task\_3 = PythonOperator(task\_id="task\_3", python\_callable=StepThree)

task\_4 = PythonOperator(task\_id="task\_4", python\_callable=StepFour)

task\_5 = PythonOperator(task\_id="task\_5", python\_callable=StepFive)

task\_1 >> task\_2 >> task\_3 >> task\_4 >> task\_5

Обратите внимание на функции **ti.xcom\_push** и **ti.xcom\_pull** — они используются для обмена данными с AirFlow.

* **ti.xcom\_push** позволяет загружать данные на платформу AirFlow.
* **ti.xcom\_pull** позволяет выгружать данные с платформы (при использовании **ti.xcom\_push**).

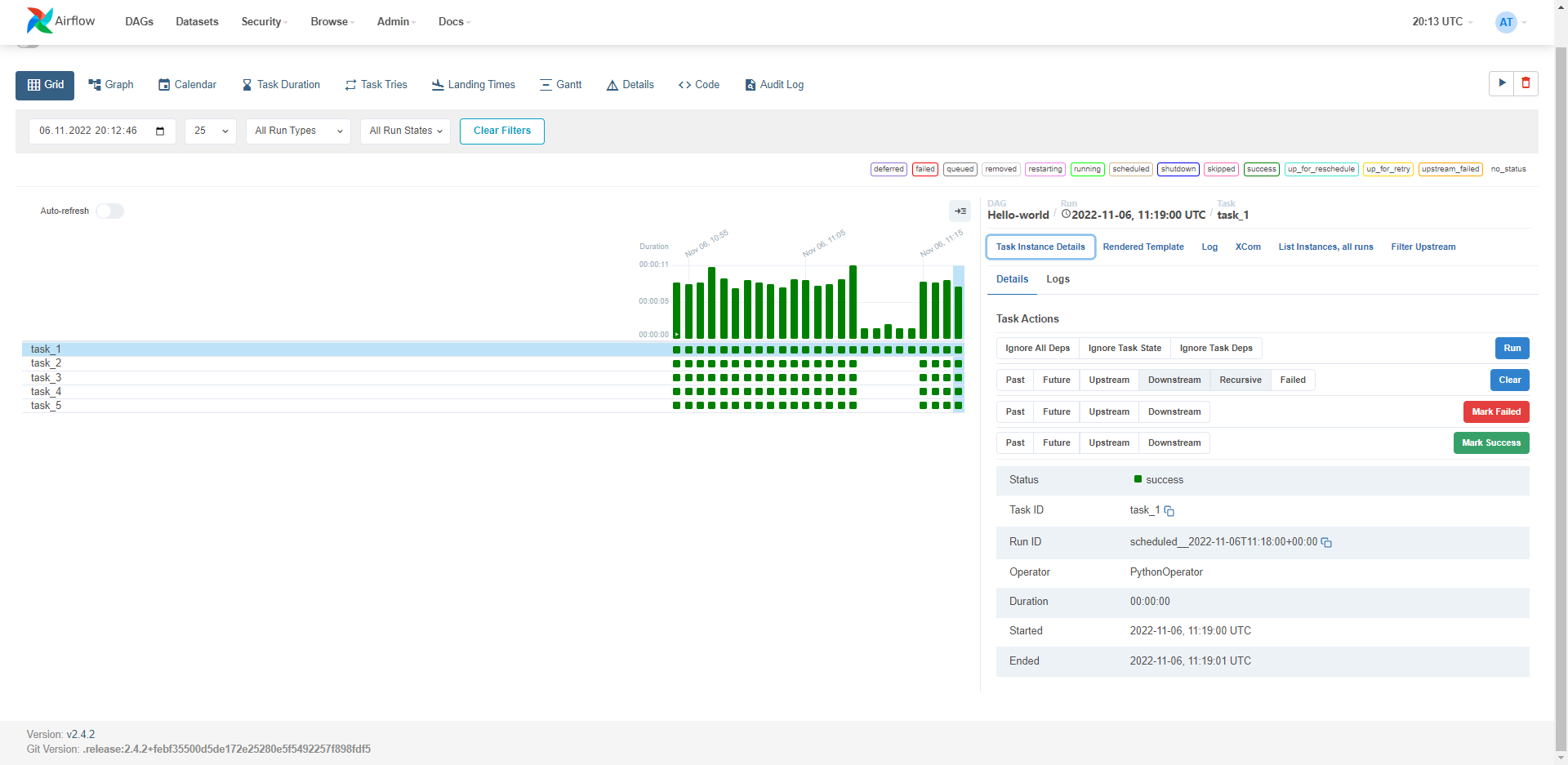


Рисунок Прогресс выполнения шагов скрипта

Если открыть вкладку **Graph**, можно заметить, что данный DAG имеет пять шагов и они выполняются последовательно друг за другом.

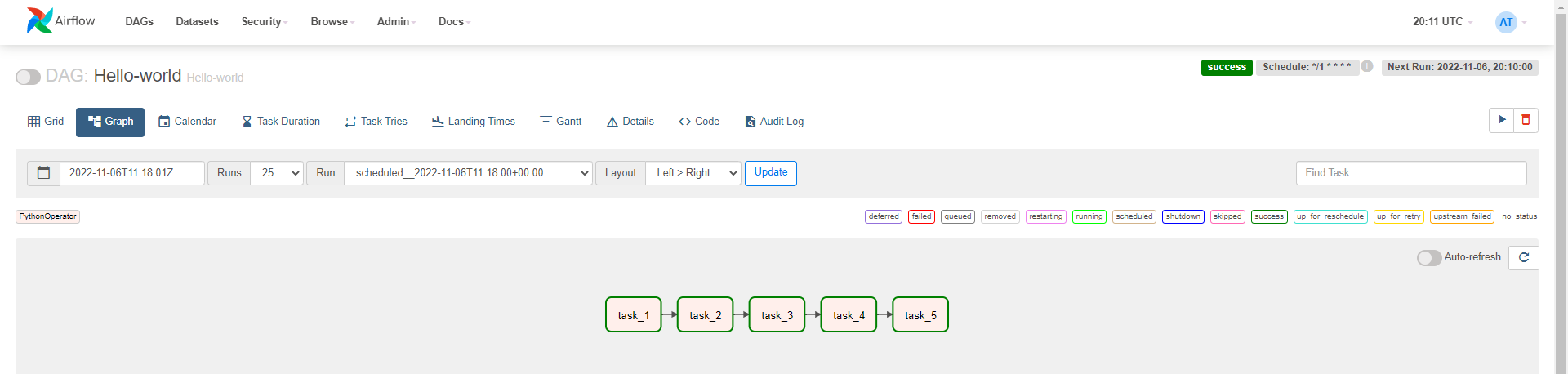


Рисунок Последовательность выполнения шагов в скрипте

На скриншоте ниже представлен пример сохранения данных в AirFlow. Обратите внимание: данные хранятся в JSON-формате.

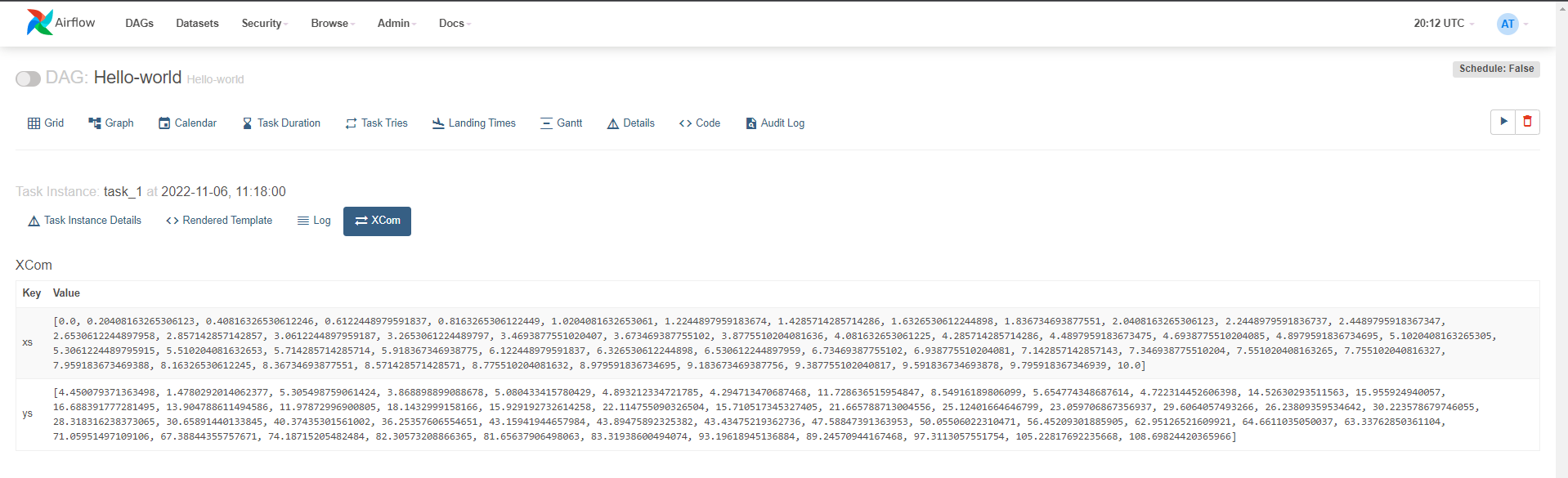


Рисунок . Сохраненные данные

Чтобы задать последовательное исполнение шагов, необходимо использовать оператор **>>**.

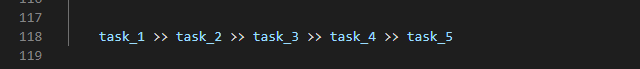


Рисунок .Указание последовательности выполнения шагов в скрипте

Итак, мы рассмотрели пример работы со скриптами AirFlow. В следующем юните мы начнем работать с MLFlow.

## Тест

1. Какие классы загружаются из Python-библиотеки airflow для описания работы графа операций? (0.25)

**[x] DAG**

[] job

**[x] task**

[] graph

1. Какие функции используются для обмена данными с AirFlow? (0.25)

**[x] xcom\_pull()**

[] xcom\_get()

**[x] xcom\_push()**

[] xcom\_send()

1. Какой оператор нужно использовать, чтобы определить последовательность выполнения операций в DAG AirFlow? (0.25)

() +

(x) **>>**

() ->

() =>

1. В каком формате хранятся данные AirFlow? (0.25)

() xml

(x) **json**

() doc

() txt

# Модуль 3. Юнит 4. Установка MLFlow

С этого юнита мы переходим к рассмотрению MLFlow. Для работы с инструментом его необходимо также предварительно установить.

|  |  |
| --- | --- |
| 1 | Создаем директорию для конфигурационных файлов.  **mkdir mlflow** |
| 2 | Устанавливаем все необходимые пакеты.  **sudo apt-get install sqlite3 libsqlite3-dev -y**  **sudo apt install python3-pip -y && sudo pip install pysqlite3**  **sudo apt install libpq-dev -y && sudo apt install gcc -y**  **sudo pip install psycopg2**  Рисунок Установка пакетов |
| 3 | Устанавливаем пакет MlFlow.  **sudo pip install mlflow**  Рисунок 2. Установка MlFlow |
| 4 | Устанавливаем набор пакетов postgresql для работы с базами данных.  **sudo apt-get install postgresql postgresql-contrib postgresql-server-dev-all**  Рисунок 3. Установка PostgreSQL |
| 5 | Создаем базу данных и пользователя, выдаем ему привилегированный доступ к базе данных.  **sudo -u postgres psql**  **CREATE DATABASE mlflow;**  **CREATE USER mlflow WITH ENCRYPTED PASSWORD 'mlflow';**  **GRANT ALL PRIVILEGES ON DATABASE mlflow TO mlflow;**  Рисунок 4. Настройка базы данных |

Установка и настройка завершены!

Запускаем MlFlow.

**mlflow server --backend-store-uri postgresql://mlflow:mlflow@localhost/mlflow --default-artifact-root file:/home/<user>/mlruns -h 0.0.0.0 -p 5000**

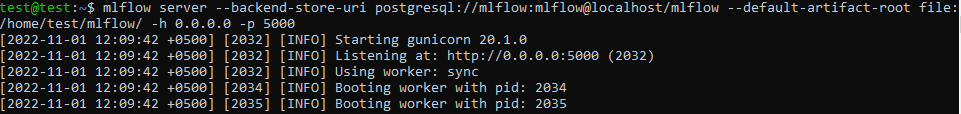


Рисунок 5. Запуск MlFlow

Открываем страницу MlFlow.

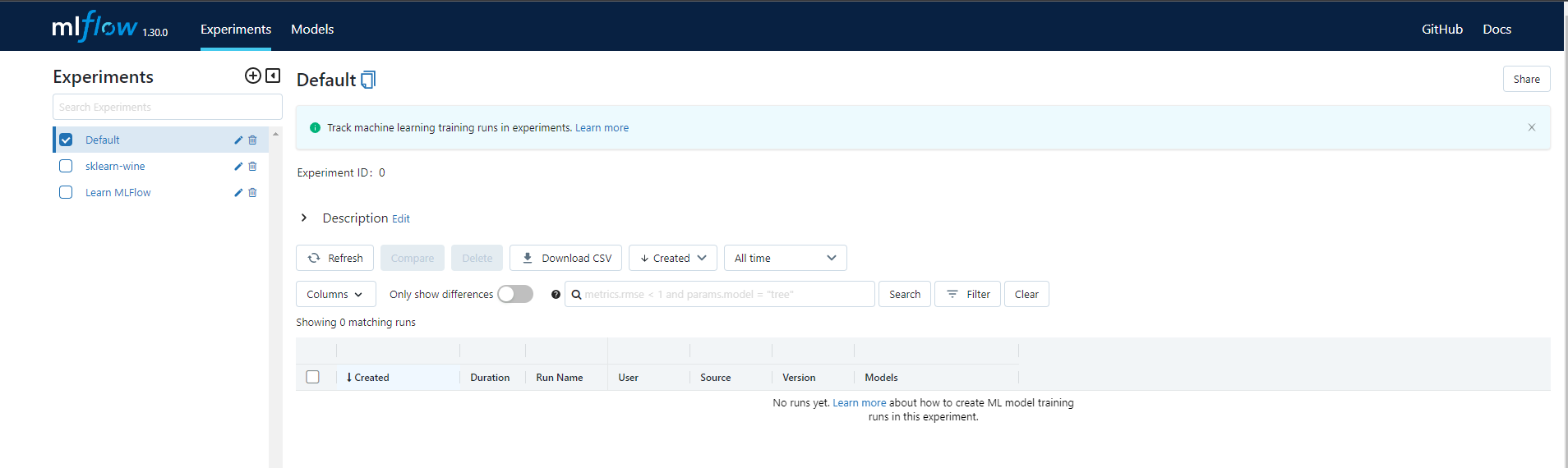


Рисунок 6. Веб-интерфейс MlFlow (0.0.0.0:5000)

В этом юните вы установили и запустили MLFlow. Далее мы разберем примеры использования этого инструмента.

## Тест

1. Какая команда создает базу данных? (0.25)

() START DATASET

**(x) CREATE DATASET**

() MAKE DATASET

() SET DATASET

1. Какая команда устанавливает MLFlow? (0.25)

() apt install mlflow

() mlflow.setup

() setup mlflow

**(x) sudo pip install mlflow**

1. Как называется утилита для работы с базой данных postgresql через консоль? (0.25)

() ps

**(x) psql**

() admin

() sudo

1. На каком порте по умолчанию запускается интерфейс MLFlow? (0.25)

**(x) 5000**

() 5050

() 5555

() 5001

# Модуль 3. Юнит 5. Пример использования MLFlow

В последнем юните мы посмотрим, как использовать MLFlow.

В качестве примера возьмем следующий скрипт:

import warnings

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import ElasticNet

from urllib.parse import urlparse

import mlflow

import mlflow.sklearn

if \_\_name\_\_ == "\_\_main\_\_":

mlflow.set\_tracking\_uri('http://0.0.0.0:5000') # Указываем местоположение сервера

experiment = mlflow.set\_experiment("Learn MLFlow") # Задаем имя проекта, в котором будет храниться различные эксперименты

print("mlflow tracking uri:", mlflow.tracking.get\_tracking\_uri())

print("experiment:", experiment)

warnings.filterwarnings("ignore")

with mlflow.start\_run(experiment\_id=experiment.experiment\_id):

from random import randint

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

start = 0

end = 10

col = 50

xs = np.linspace(start, end, col)

ys = xs\*\*2 + np.random.random(50) \* 10

mlflow.log\_param("start", start) # Логирование гиперпараметра start

mlflow.log\_param("end", end) # Логирование гиперпараметра end

mlflow.log\_param("col", col) # Логирование гиперпараметра col

for t in range(1, 6):

if t == 1:

xs1 = np.c\_[xs]

elif t == 2:

xs1 = np.c\_[xs, pow(xs,2)]

elif t == 3:

xs1 = np.c\_[xs, pow(xs,2), pow(xs,3)]

elif t == 4:

xs1 = np.c\_[xs, pow(xs,2), pow(xs,3), pow(xs,4)]

elif t == 5:

xs1 = np.c\_[xs, pow(xs,2), pow(xs,3), pow(xs,4), pow(xs,5)]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(xs1, ys, test\_size=0.33, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

score = model.score(X\_test, y\_test)

mlflow.log\_metric(f"score", score) # Логирование метрики

tracking\_url\_type\_store = urlparse(mlflow.get\_tracking\_uri()).scheme

mlflow.sklearn.log\_model(model, "model") # Логирование модели

На странице эксперимента можно увидеть количество запусков, сохраненные метрики и гиперпараметры.

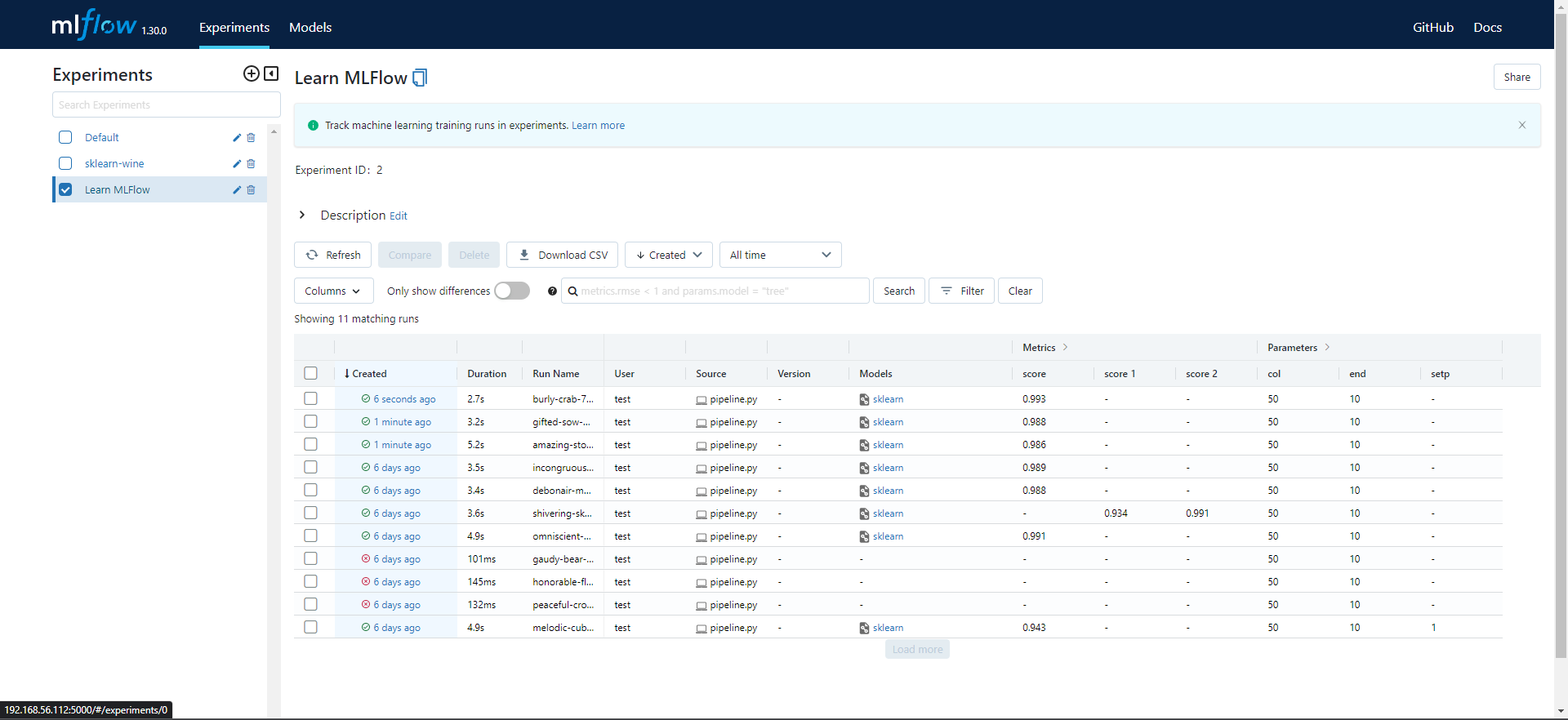


Рисунок 1. Страница эксперимента

Если перейти в запускаемый эксперимент, можно найти более детальные сведения о нем:

* используемые гиперпараметры,
* полученные метрики,
* информацию о модели.

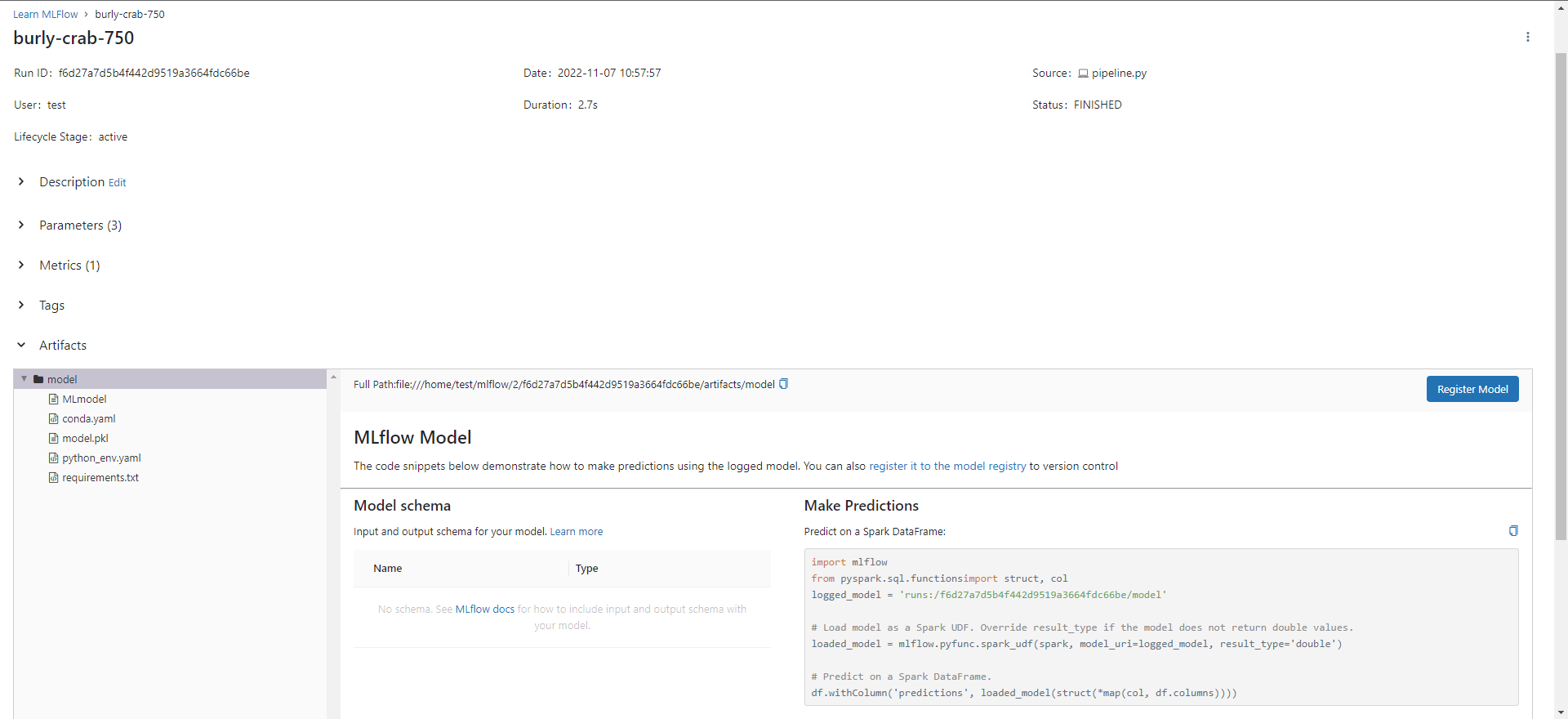


Рисунок 2. Общая информация о проведенном эксперименте

При раскрытии списков можно увидеть используемые и зафиксированные гиперпараметры, а также полученные и зафиксированные метрики.

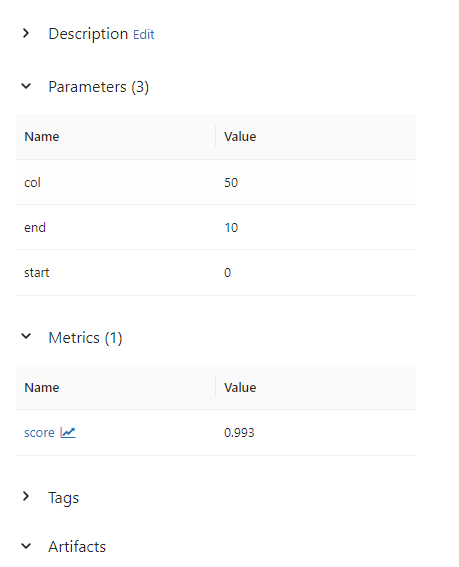


Рисунок 3. Список гиперпараметров и метрик

Для экспериментов, в ходе которых было получено несколько значений метрик для одного параметра, можно отслеживать положение отдельных значений. Для этого нужно выбрать из списка нужную метрику — на появившемся графике можно увидеть, как располагаются значения, получаемые в ходе эксперимента.



Рисунок 4. Отслеживание получаемой метрики при различных параметрах

В заключительном юните вы рассмотрели пример работы со скриптами MLFlow.

## Тест

1. Какой метод позволяет задать имя эксперимента? (0.25)

**(x) set\_experiment(“имя эксперимента”)**

() set\_name(“имя эксперимента”)

() set\_parameter(“имя эксперимента”)

() set\_env(“имя эксперимента”)

1. Какой метод запускает эксперимент? (0.25)

**(x) start\_run()**

() run\_experiment()

() run()

() start()

1. Какой метод сохраняет метрики? (0.25)

() save()

**(x) log\_metric()**

() save\_metric()

() metric()

1. Какой метод сохраняет модель? (0.25)

() save\_model()

**(x) log\_model()**

() log()

() model()

# Список источников

1. [Официальный сайт](https://airflow.apache.org/) Apache Airflow.
2. Статья [«Введение в Apache AirFlow»](https://khashtamov.com/ru/apache-airflow-introduction).
3. Статья [«Apache Airflow: делаем ETL проще»](https://habr.com/ru/post/512386).
4. Русскоязычное сообщество [Airflow](https://t.me/ruairflow) в Telegram.