# Модуль 3. Управление потоком операций (xFlow).

## В этом модуле

Конвейеры состоят из операций, ходом выполнения которых 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 последовательности, заданной в DAG, создает поток (flow) действий и появляющихся артефактов. Существуют различные инструменты для работы с таким потоком операций, с условным названием xFlow. Два таких инструмента, AirFlow и MLFlow мы рассмотрим в этом модуле.

## Содержание юнитов

|  |
| --- |
| 1. Установка AirFlow |
| 1. Примеры скриптов AirFlow |
| 1. Пример использования AirFlow |
| 1. Установка MLFlow |
| 1. Пример использования MLFLow |

# Модуль 2. Юнит 1. Установка AirFlow.

## Введение

В этом юните вы научитесь устанавливать AirFlow.

## Содержание

Сначала необходимо установить все необходимые пакеты, которые потребуются для начала работы с Airflow:

**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**

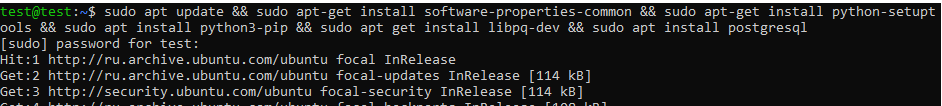


Рисунок 1 - Установка пакетов

Дальше необходимо указать с помощью команды «export» eуказать директорию в которой будут храниться конфигурационные файлы, а также «Dags» которые будет запускать AirFlow

**export AIRFLOW\_HOME=~/airflow**

После того как домашняя директория была указана можно приступить к установке самого Airflow.

**sudo pip3 install apache-airflow**

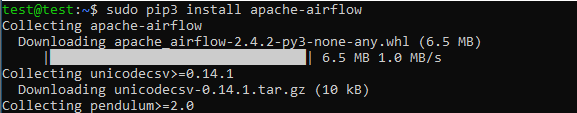


Рисунок 3 - Установка AirFlow

Дальше необходимо создать пользователя, задать пользователю пароль. Далее создать базу данных куда будут записываться различные данные и 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;**

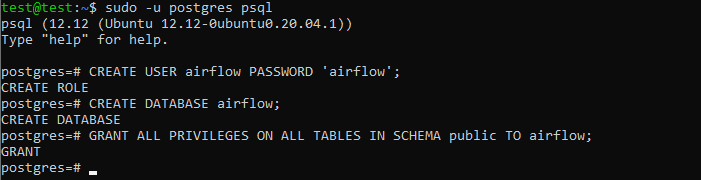


Рисунок 4 - Создание базы данных

Следующим шагом необходимо создать конфиги внутри домашней директории которую указывали для конфигурационных данных AirFlow.

**airflow db init**

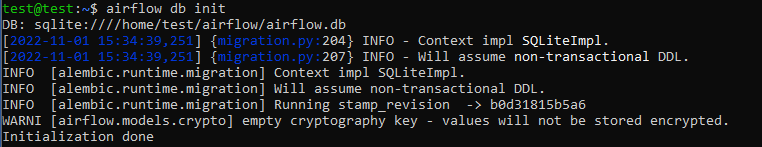


Рисунок 5 - Инициализация данных

После появится директория в указанном ранее месте. Перейдя в созданный каталог и посмотрев его содержимое можно увидеть конфигурационные данные.

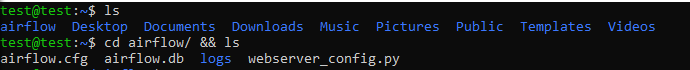


Рисунок 6 - Каталог с конфигурационными файлами

На следующем этапе в файле airflow.cfg необходимо изменить переменную «sql\_alchemy\_conn». Это необходимо чтобы связать AirFlow с базой данных Postgresql:

Измененная строка:

**sql\_alchemy\_conn = postgresql+psycopg2://airflow:password@localhost/airflow**

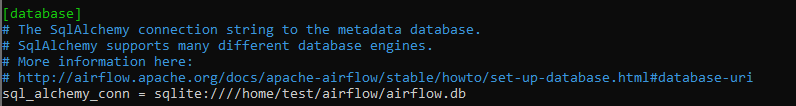


Рисунок 7 - Строка для изменения

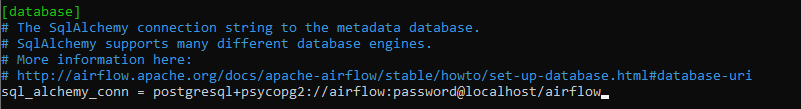


Рисунок 8 - Строка после изменения

Чтобы запустить работу с базой данных необходимо заново инициализировать базу данных.

**airflow db init**

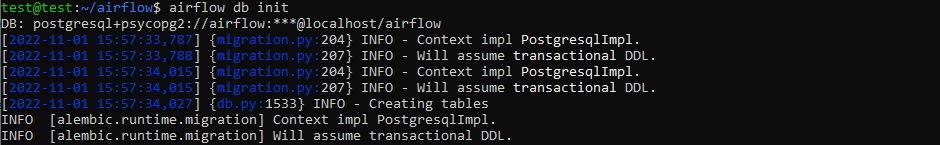


Рисунок 9 - Инициализация базы данных

После подключения базы данных необходимо создать пользователя для работы с AirFlow на сайте.

**airflow users create --username Admin --firstname admin --lastname test --role Admin --email** [**airflow@airflow.com**](mailto:airflow@airflow.com)



Рисунок 10 - Создание пользователя

В конце создания пользователя ему задается пароль.

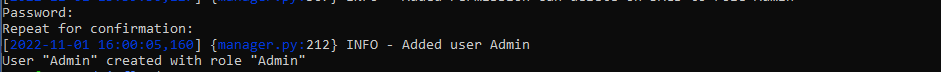


Рисунок 11 - Установка пароля пользователя

Все установки завершены. Для запуска AirFlow необходимо запустить веб-сервер, а также помимо веб-сервера необходимо запустить schedule, который будет обрабатывать все запущенные скрипты внутри AirFlow.

**airflow webserver -p 8080**



Рисунок 12 - Запуск Веб-сервера

**airflow scheduler**

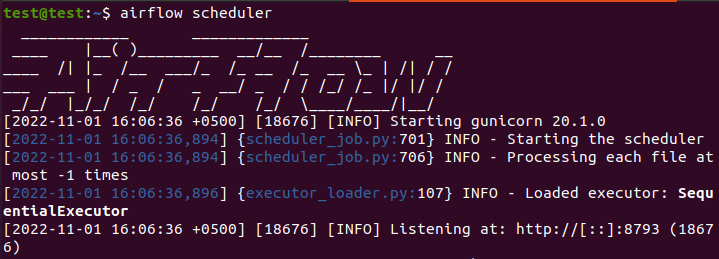


Рисунок 13 - Запуск scheduler

## Тест

1. ? (0.25)
   1. **a**
   2. b
   3. **c**
   4. **d**
2. ? (0.25)
   1. **a**
   2. b
   3. c
   4. d
3. ? (0.25)
   1. a
   2. **b**
   3. c
   4. d
4. ? (0.25)
   1. a
   2. **b**
   3. c
   4. d

## Итоги/выводы

В этом юните вы научились устанавливать AirFlow

# Модуль 3. Юнит 2. Примеры скриптов AirFlow.

## Введение

В этом юните мы рассмотрим некоторые примеры скриптов AirFlow.

## Содержание

### Пример исполнения кода в AirFlow

Код 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)

Пишем функцию, которая будет выводить текст «Hello world!!!!!!», назовем функцию «test123». После нам необходимо указать аргументы, которые позже будут переданы функции DAG.

В переменную task\_1 присвоим результат функции PythonOperator, который будет использовать ранее написанную функцию «task123».

При создании скрипта в указанной папке при установке AirFlow при верно сконфигурированном скрипте он появится на вкладке «DAGs».

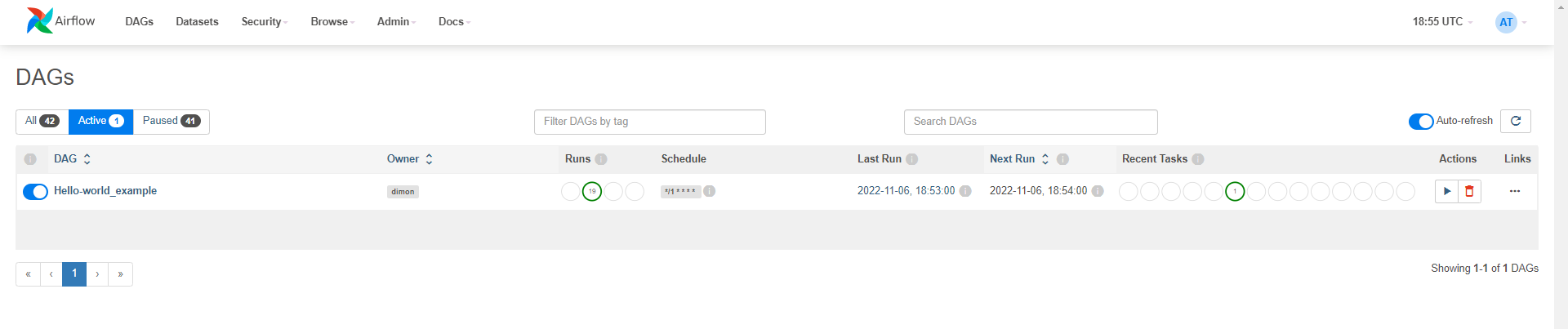


Рисунок - Активные Dag

Если перейти в dag, то можно увидеть прогресс выполнения при каждом запуске скрипта.

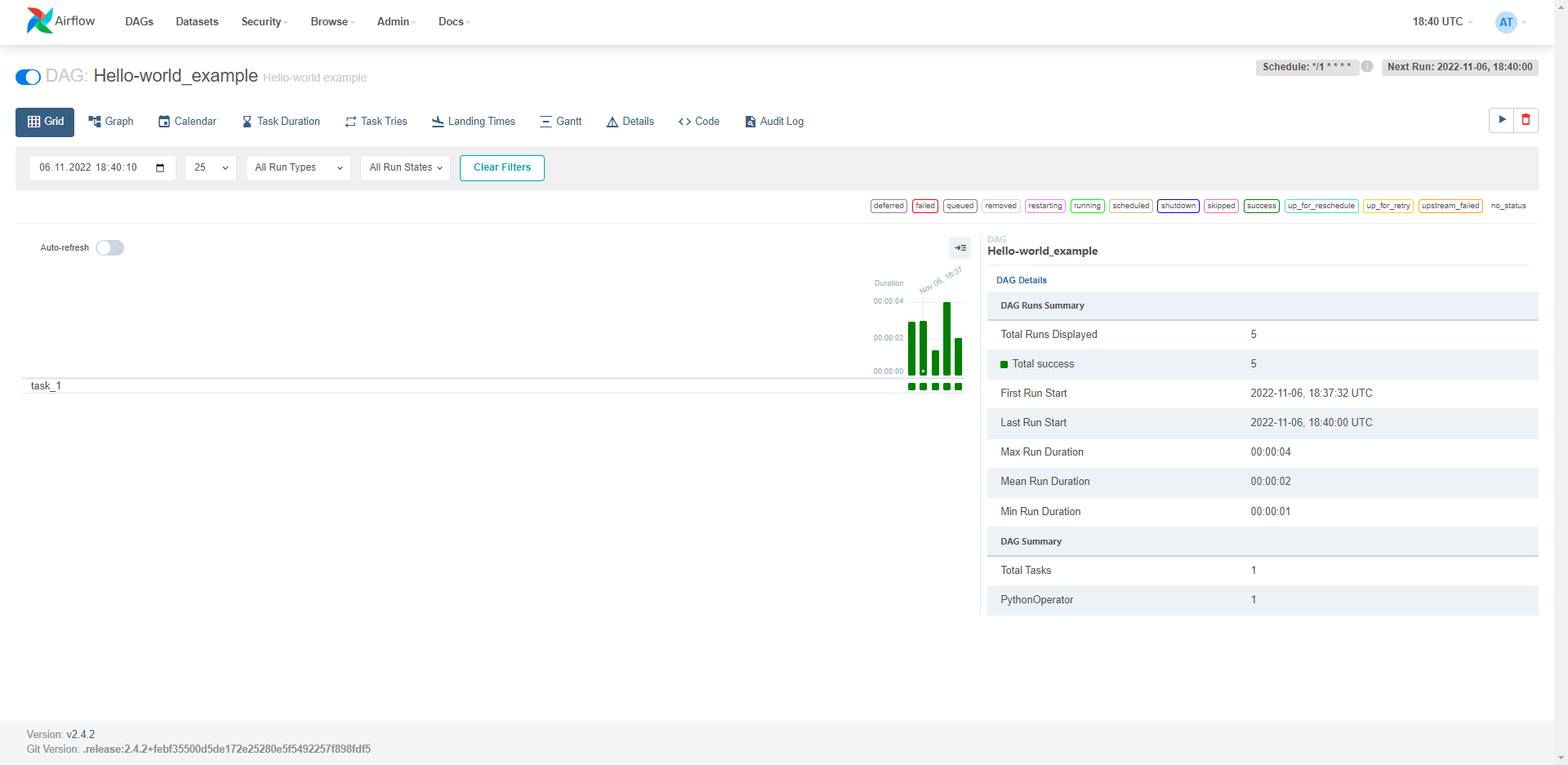


Рисунок - Статус выполнения Dag

Так же можно перейти на вкладку «Code», чтобы просмотреть код исполняемого dag.

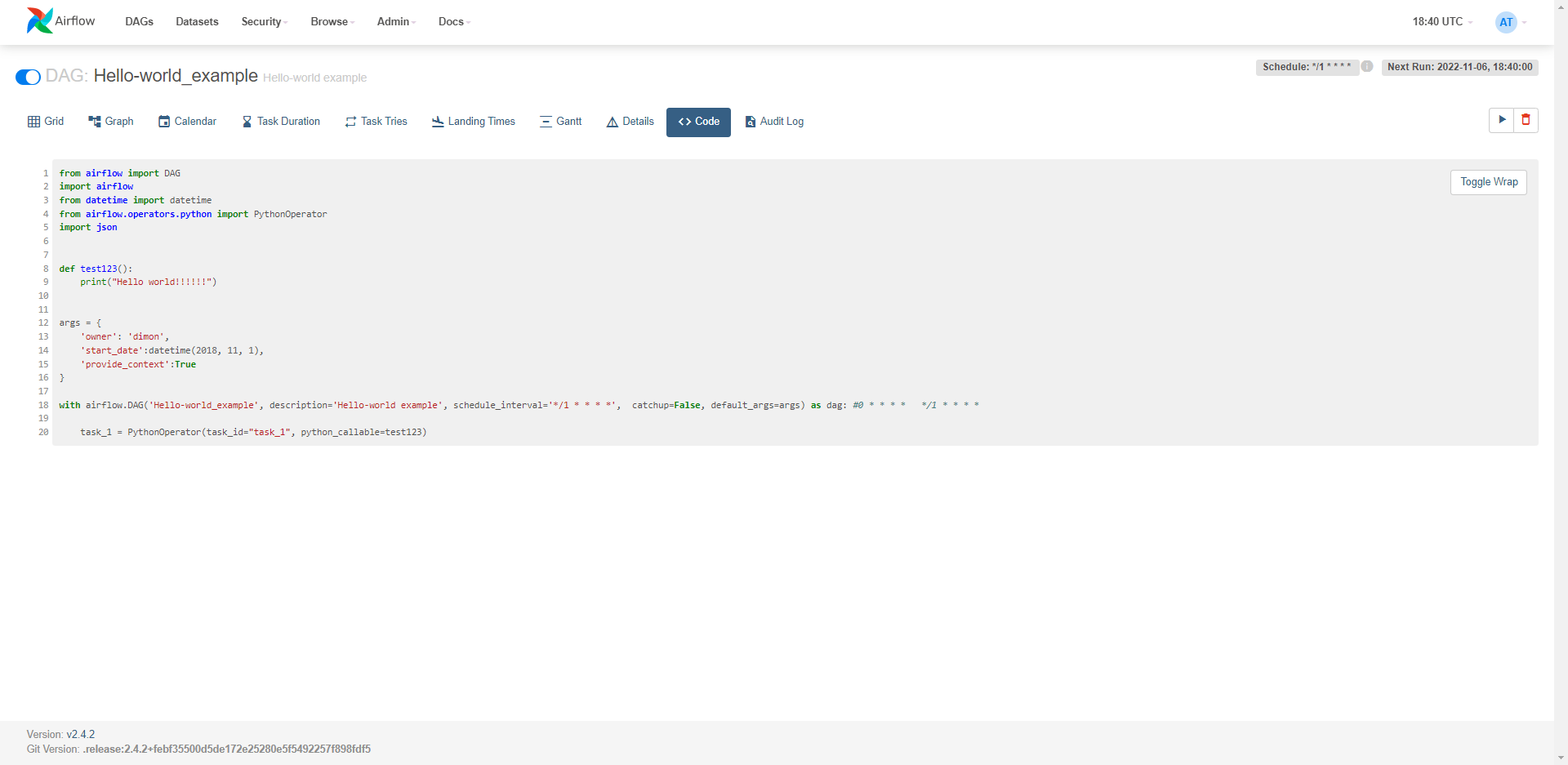


Рисунок - Код Dag'а

На вкладке «Log», можно просмотреть состояния программы, например, при выводе текста во время выполнения программы.

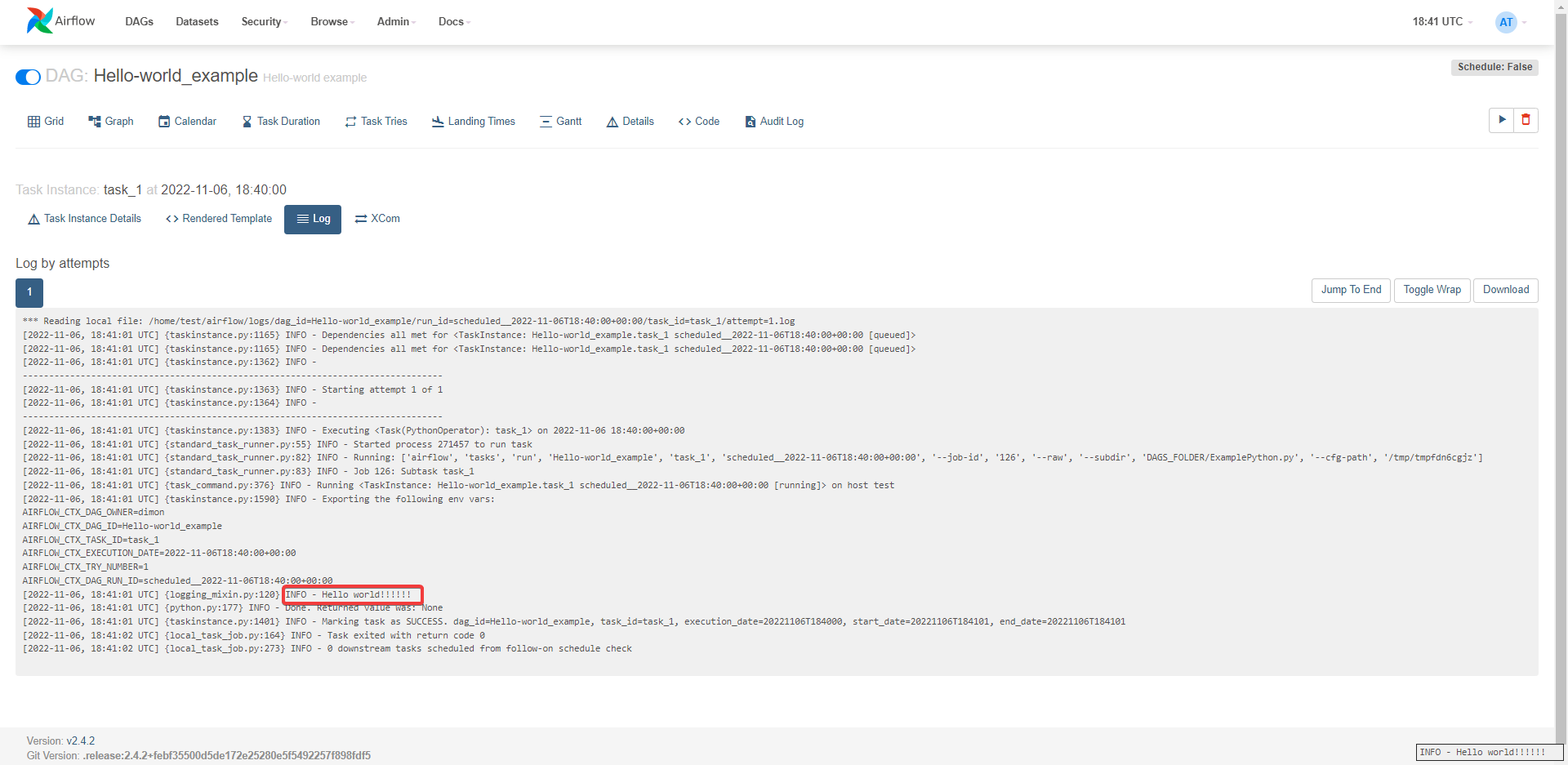


Рисунок - Логирование выполенния Dag и вывод print

На вкладке «graph», можно увидеть последовательность выполнения программы.

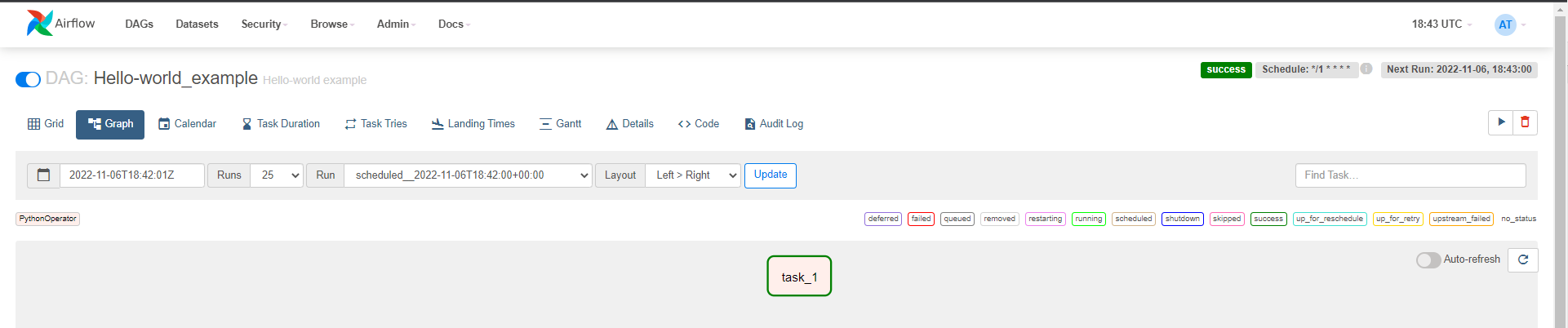


Рисунок - График последовательности выполнения задач

### Пример 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")

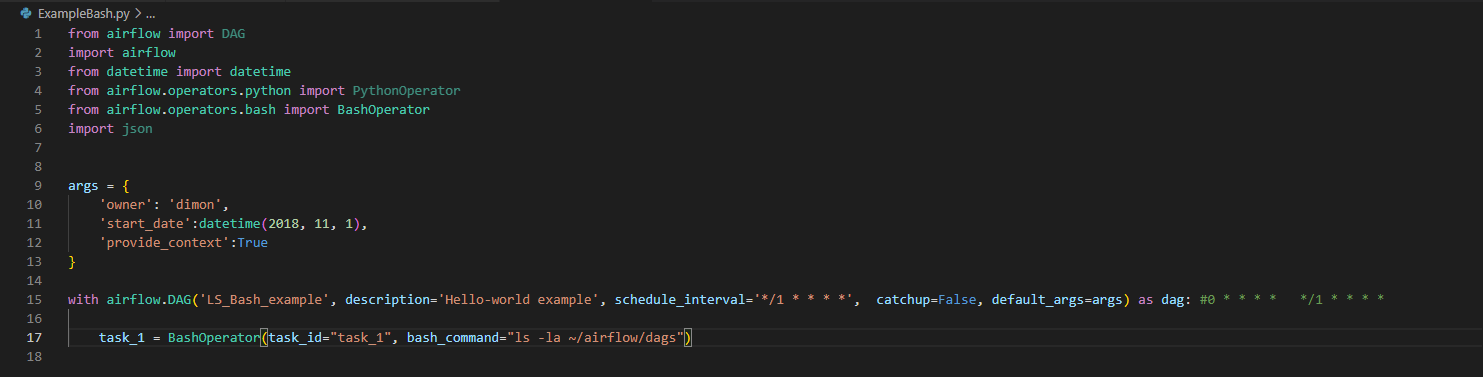


Рисунок – Использование Bash скрипта

Вывод результата выполнения Bash скрипта производится в log.

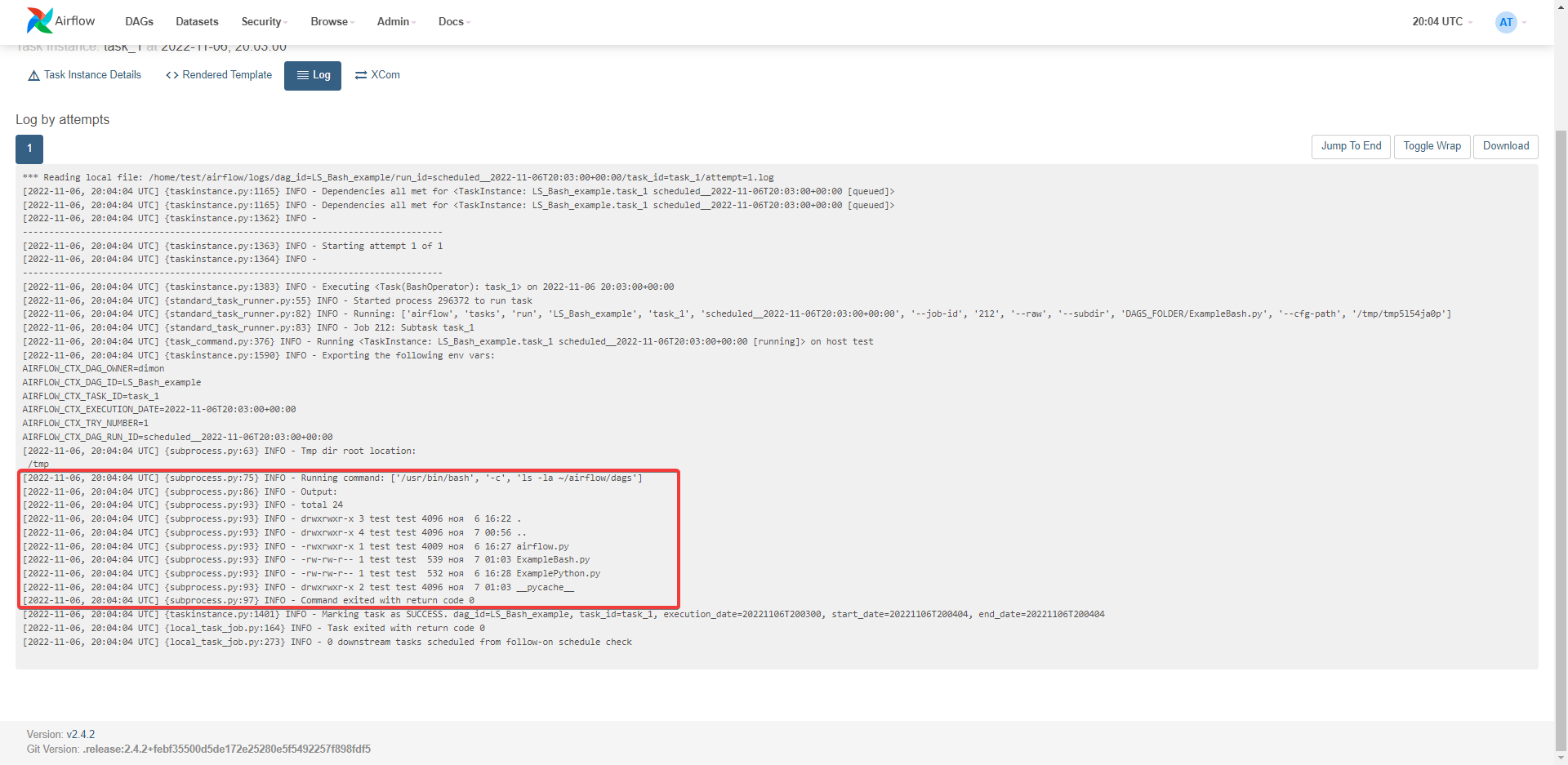


Рисунок - Log при использовании bash скрипта

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

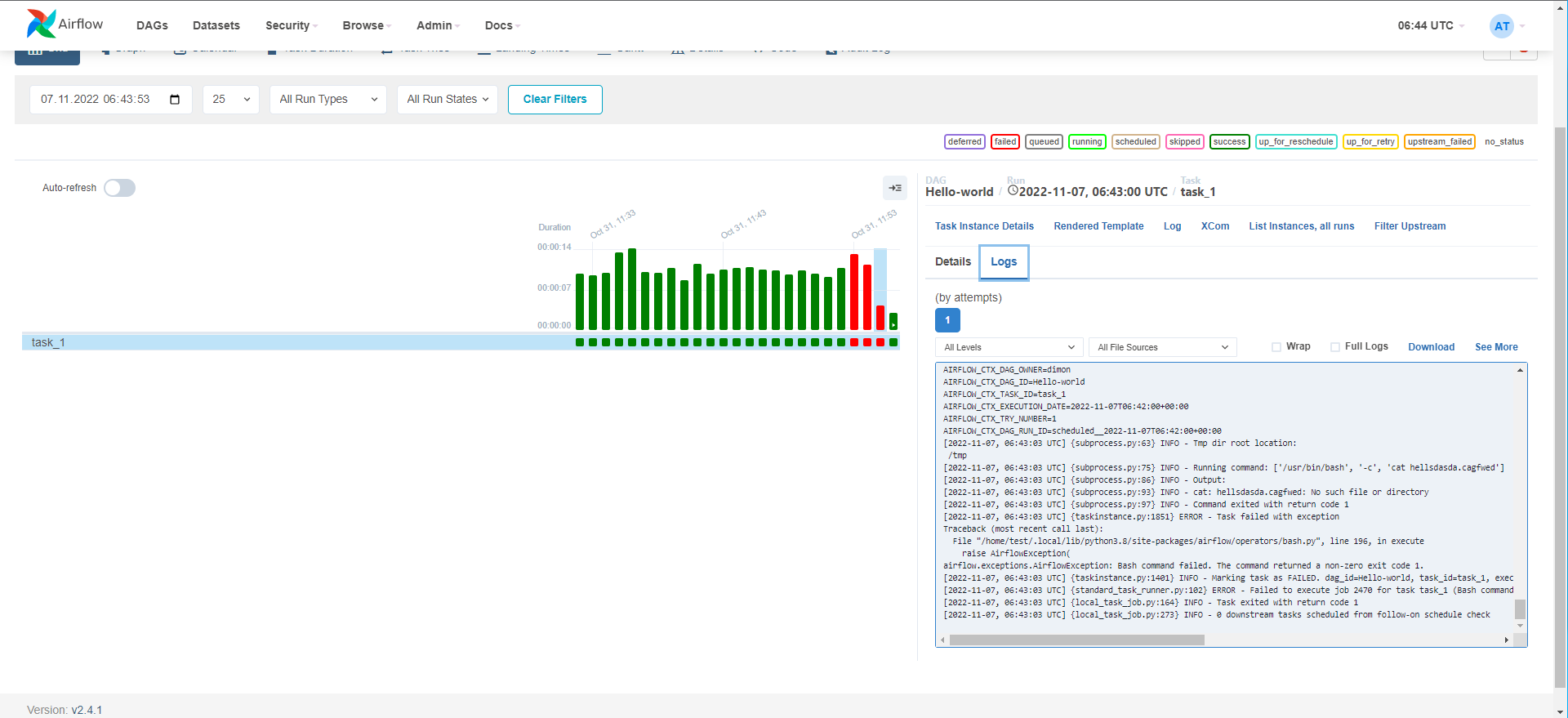


Рисунок - Ошибка выполнения bash скрипта

## Тест

1. ? (0.25)
   1. a
   2. **b**
   3. c
   4. d
2. ? (0.25)
   1. **a**
   2. b
   3. c
   4. d
3. ? (0.25)
   1. **a**
   2. b
   3. c
   4. d
4. ? (0.25)
   1. a
   2. **b**
   3. c
   4. d

## Итоги/выводы

В этом юните вы изучили некоторые виды скриптов AirFlow.

# Модуль 3. Юнит 3. Пример работы с AirFlow.

## Введение

В этом юните вы рассмотрите пример работы с AirFlow.

## Содержание

### ML Regression Models

Код скрипта:

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

* ti.xcom\_push – Используется для загрузки данных на платформу AirFlow.
* ti.xcom\_pull – Используется для выгрузки данных, например из предыдущего шага (При использовании ti.xcom\_push).

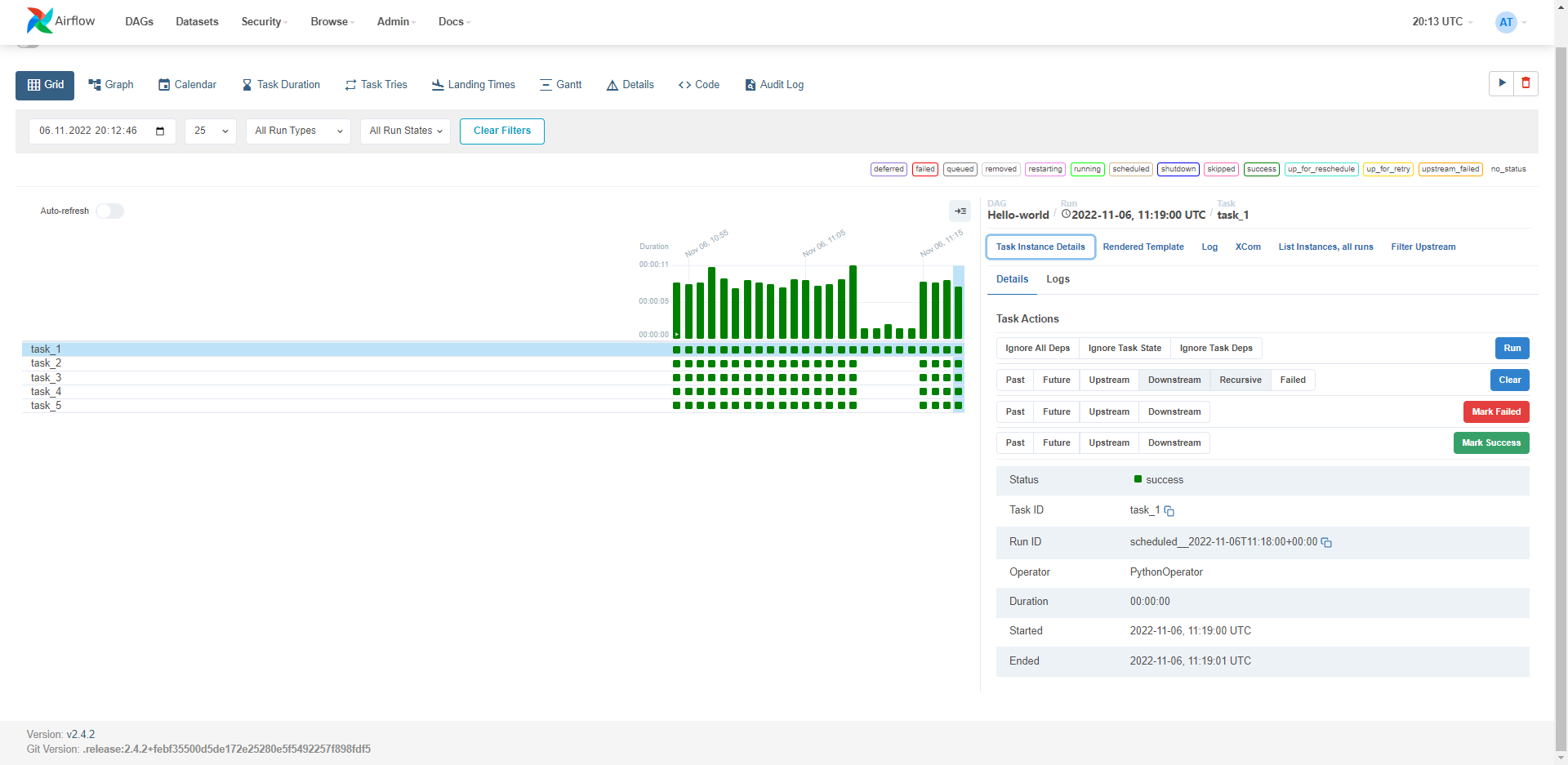


Рисунок - Прогресс выполнения шагов скрипта

Перейдя на вкладку «graph», можно заметить, что данный dag имеет 5 шагов и данные шаги выполняются последовательно друг за другом.

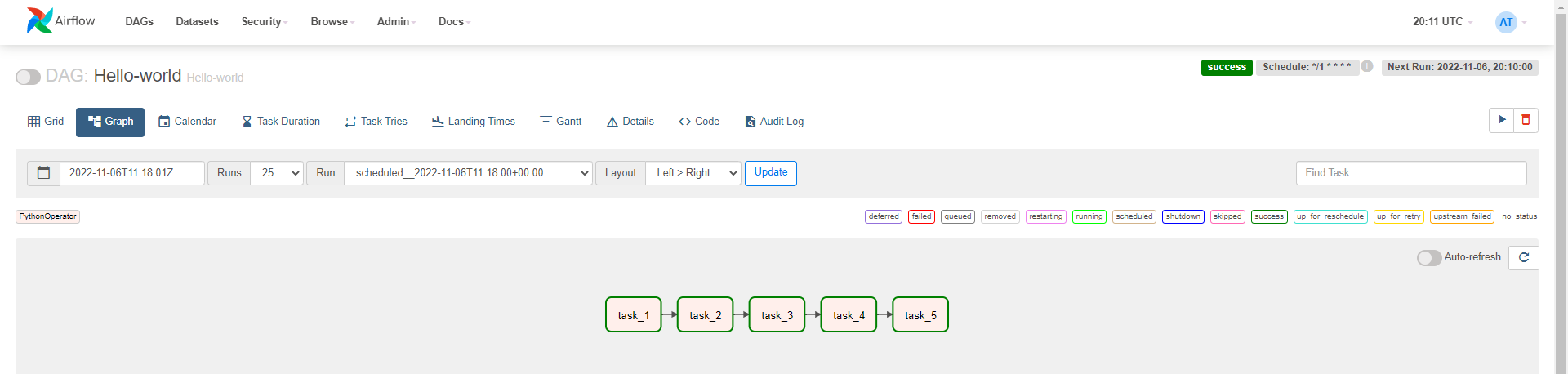


Рисунок - Последовательность выполнения шагов в скрипте

Пример сохранения данных в AirFlow. Данные хранятся в JSON формате.

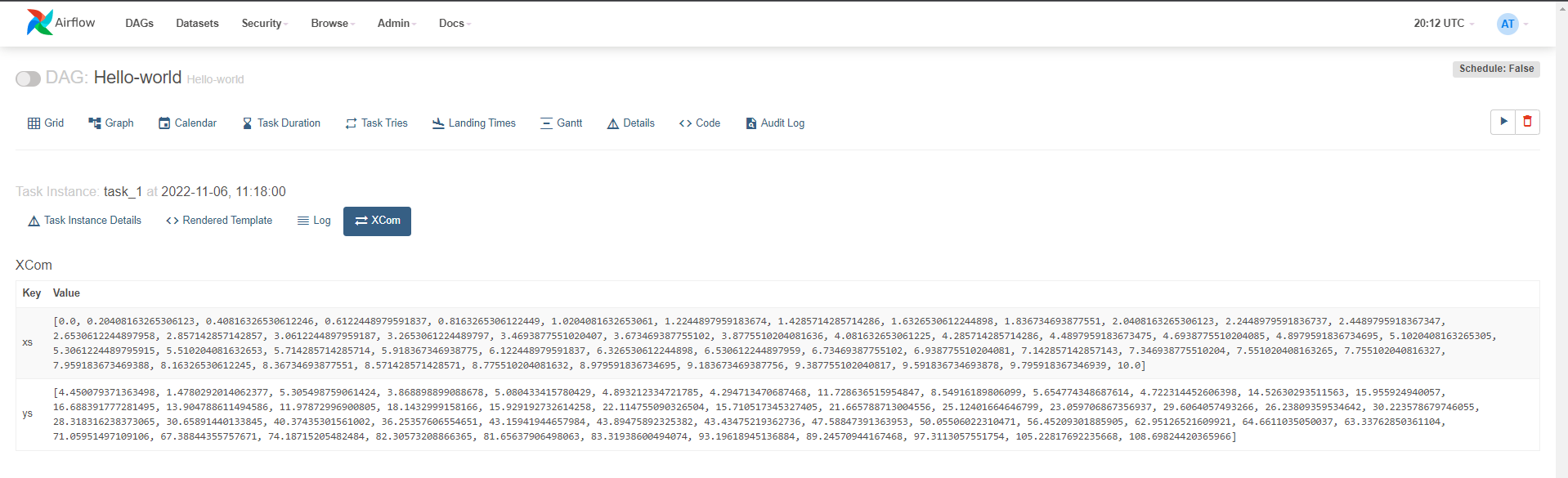


Рисунок - Сохраненные данные

Чтобы задать последовательное исполнение шагов используется «>>» оператор.

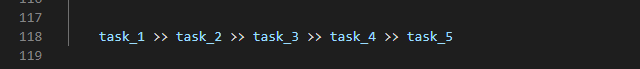


Рисунок - Указание последовательности выполнения шагов в скрипте

## Тест

1. ? (0.25)
   1. **a**
   2. b
   3. c
   4. d
2. ? (0.25)
   1. **a**
   2. b
   3. **c**
   4. d
3. ? (0.25)
   1. a
   2. **b**
   3. c
   4. d
4. ? (0.25)
   1. a
   2. **b**
   3. c
   4. d

## Итоги/выводы

В этом юните вы рассмотрели пример работы со скриптами AirFlow.

# Модуль 3. Юнит 4. Установка MLFlow.

## Введение

В этом юните вы рассмотрите установку MLFlow.

## Содержание

Сначала нужно создать директорию для конфигурационных файлов.

**mkdir mlflow**

Далее 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

После установки всех пакетов, переходим к установке пакета MlFlow.

**sudo pip install mlflow**

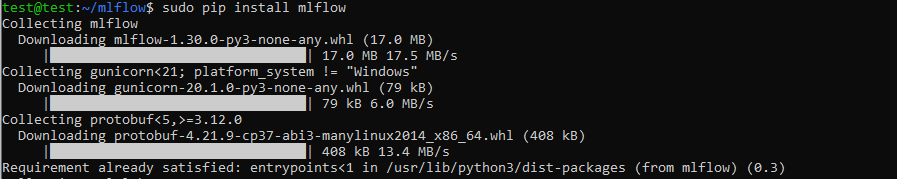
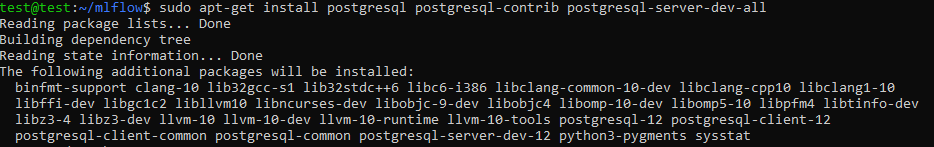


Рисунок 2 - Установка MlFlow

После установки MlFlow необходимо установить набор пакетов postgresql для работы с базами данных.

**sudo apt-get install postgresql postgresql-contrib postgresql-server-dev-all**



Далее переходим к настройки базы данных. Создание базы данных, пользователя, а также устанавливаем привилегии созданному пользователю.

**sudo -u postgres psql**

**CREATE DATABASE mlflow;**

**CREATE USER mlflow WITH ENCRYPTED PASSWORD 'mlflow';**

**GRANT ALL PRIVILEGES ON DATABASE mlflow TO mlflow;**

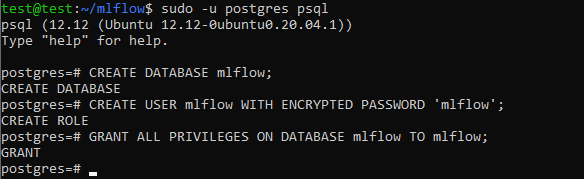


Рисунок - Настройка базы данных

После всех настроек можно запустить 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**

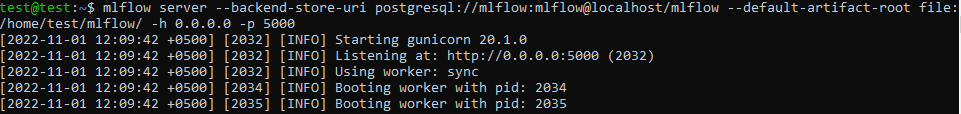


Рисунок - Запуск MlFlow

Осталось только перейти на страницу MlFlow

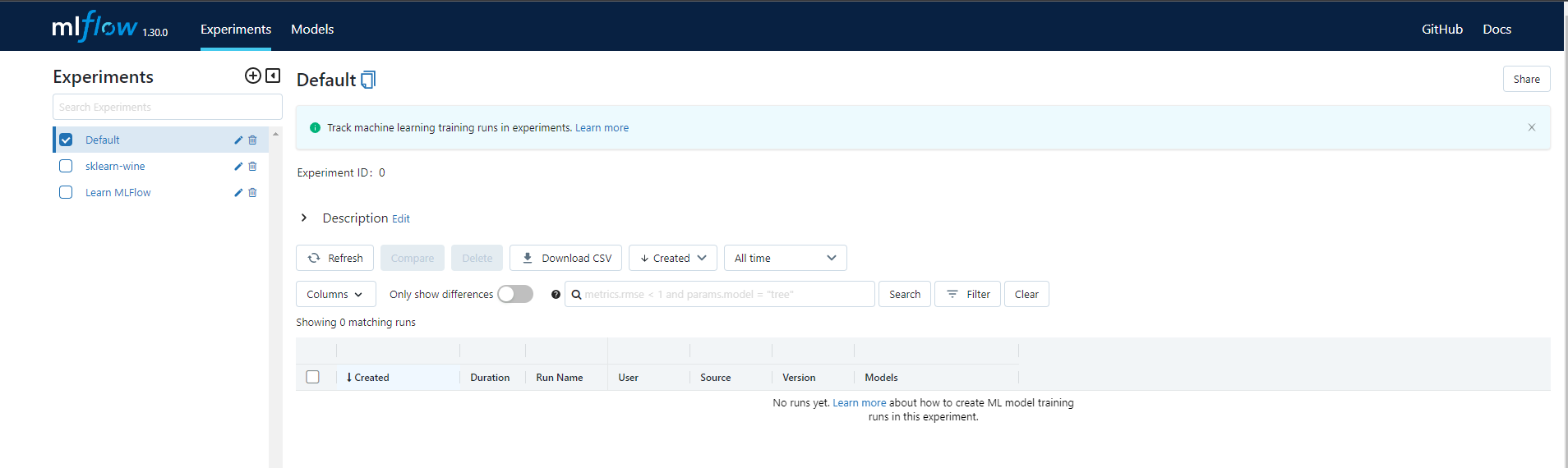


Рисунок - Сайт MlFlow (0.0.0.0:5000)

## Тест

1. ? (0.25)
   1. a,
   2. b,
   3. **c,**
   4. d
2. ? (0.25)
   1. a
   2. b
   3. c
   4. d
3. ? (0.25)
   1. a
   2. b
   3. c
   4. d
4. ? (0.25)
   1. a
   2. b
   3. c
   4. d

## Итоги/выводы

В этом юните вы научились устанавливать MLFlow.

# Модуль 3. Юнит 5. Пример использования MLFlow.

## Содержание

Код скрипта:

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 URL на котором будет производиться

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

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

Перейдя на страницу эксперимента, можно увидеть сколько раз запускался эксперимент, сохраненные метрики, а также увидеть гиперпараметры.

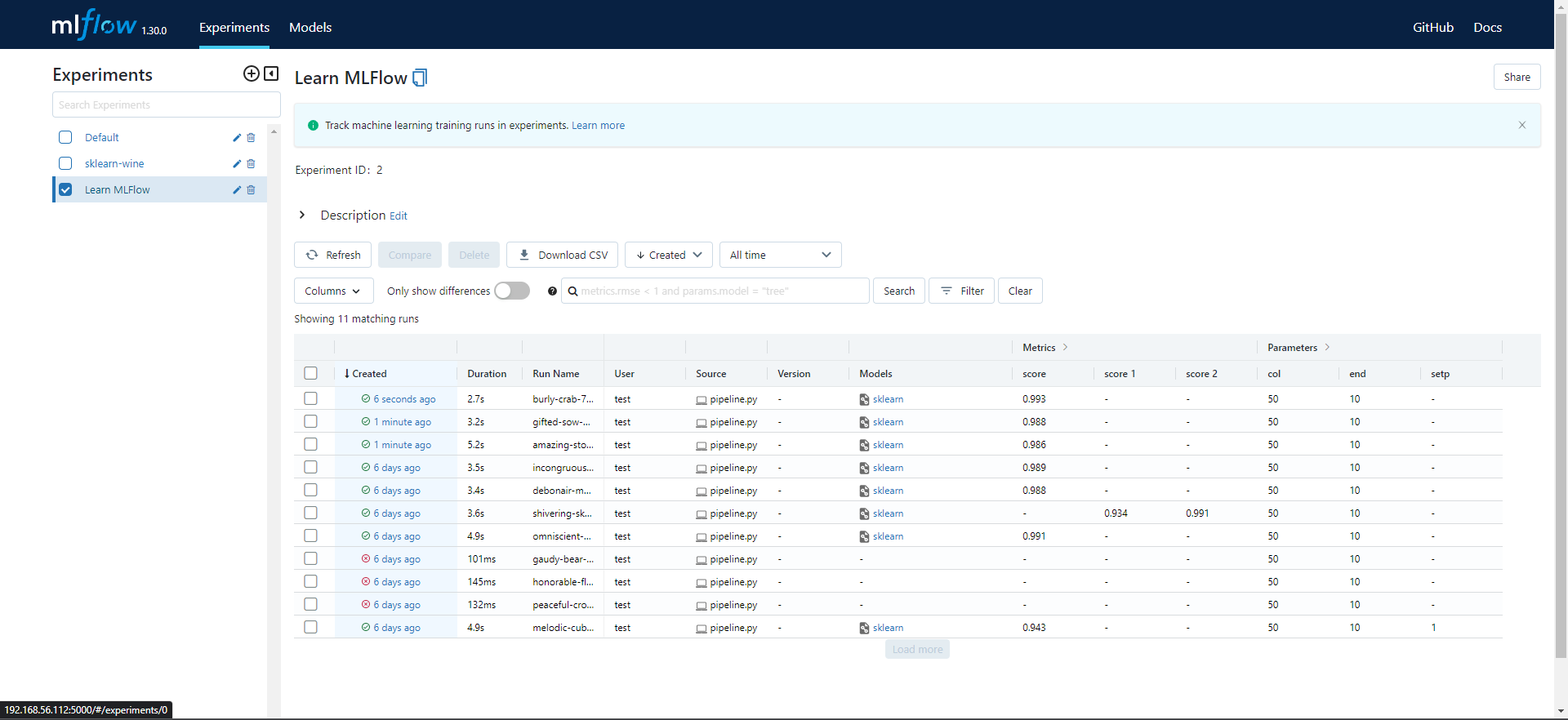


Рисунок 1 - Страница эксперимента

Если перейти в запускаемый эксперимент, видна более подробная информация о эксперименте, где можно посмотреть:

* Используемые гиперпараметры
* Полученные метрики
* Посмотреть информацию о модели

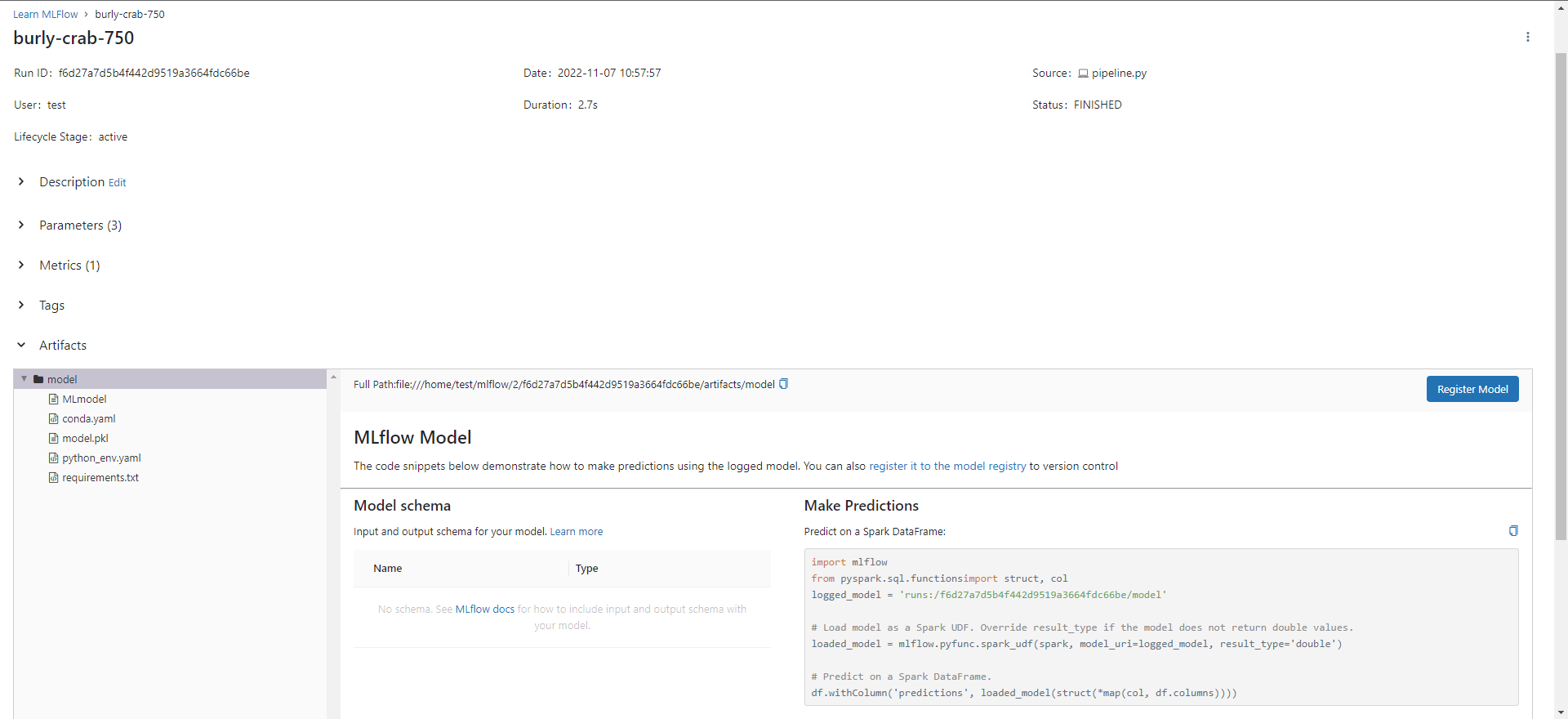


Рисунок 2 - Общая информация о проведенном эксперименте

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

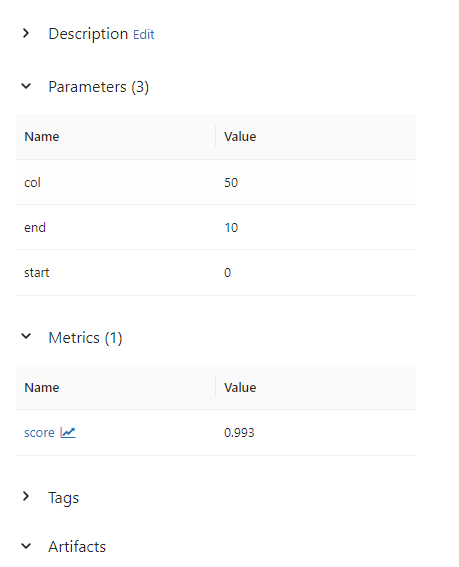


Рисунок 3 - Список гиперпараметров и метрик

Если был проведен эксперимент, где было получено несколько значений метрик для одного параметра, то выбрав из списка метрику, которая нас интересует, то можно увидеть график, где расположенные получаемые значения в ходе эксперимента.



Рисунок 4 - Отслеживание получаемой метрики при различных параметрах

## Тест

1. ? (0.25)
   1. **a**
   2. b
   3. c
   4. d
2. ? (0.25)
   1. **a**
   2. b
   3. **c**
   4. d
3. ? (0.25)
   1. a
   2. **b**
   3. c
   4. d
4. ? (0.25)
   1. a
   2. **b**
   3. c
   4. d

## Итоги/выводы

В этом юните вы рассмотрели пример работы со скриптами AirFlow.

# Список источников