**Лабораторная работ 3. Airflow and MLflow.**

**Грибанов Данил, 6233**

**Задачи**

1. **Пайплайн, который обучает любой классификатор из sklearn по заданному набору параметров.**
2. **Пайплайн, который выбирает лучшую модель из обученных и производит её хостинг.**

**Ход работы**

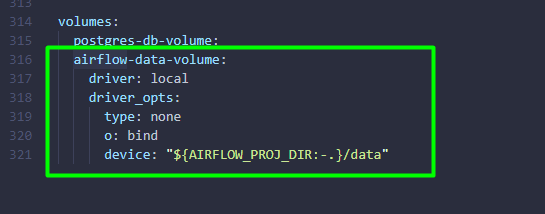
**Все файлы (dockerfile, docker-compose и прочее) находятся в папке req (так же dags и скрипты для запуска соотв. частей).**

**Важно отметить, что для запуска airflow, используется файл по пути ./req/airflow/docker-compose.yaml**

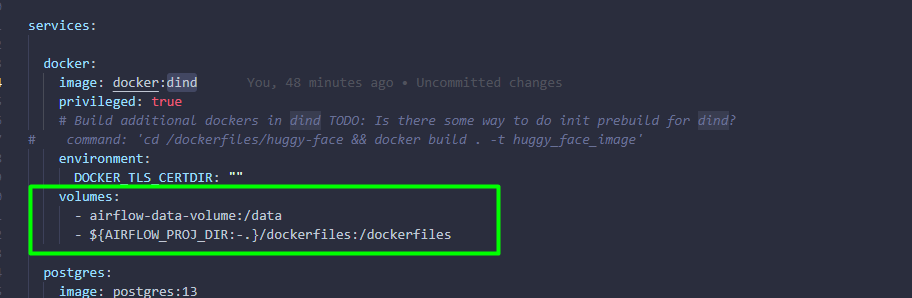
**Там находятся модификации для dind и отдельные хранилища для работы (часть из этого осталась еще со 2 лабораторной работы, и больших изменений там нет).**

**Далее будут описаны кратко основные изменения airflow compose файла. Перед запуском самой системы (докера airflow) следует подготовить следующее:**

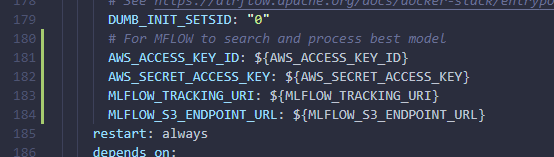
Создадим хранилище с данными для запуска и обучения всех систем:



Добавим данное хранилище с папкой докер-файлов нужных в dind:



**Также необходимо добавить переменные для поиска лучшей модели в MLflow:**



**Для последующего выполнения заданий, необходимо собрать докер airflow (./req/airflow/docker-compose.yaml) и mlflow docker-compose.mlflow.yaml.**

**Для докера airflow имеется свой .env с необходимыми параметрами:**

AIRFLOW\_PROJ\_DIR=.

AIRFLOW\_UID=50000

AWS\_ACCESS\_KEY\_ID=minio

AWS\_SECRET\_ACCESS\_KEY=minio123

#*MLFLOW\_S3\_ENDPOINT\_URL=http://minio:9000*

MLFLOW\_TRACKING\_URI=http://mlflow\_server:5000

MLFLOW\_S3\_ENDPOINT\_URL=http://minio:9000

MYSQL\_DATABASE=mlflow\_database

MYSQL\_USER=mlflow\_user

MYSQL\_PASSWORD=mlflow

MYSQL\_ROOT\_PASSWORD=mysql

**Для MLflow .env выглядит так:**

AIRFLOW\_PROJ\_DIR=.

AIRFLOW\_UID=50000

AWS\_ACCESS\_KEY\_ID=minio

AWS\_SECRET\_ACCESS\_KEY=minio123

#*MLFLOW\_S3\_ENDPOINT\_URL=http://minio:9000*

MLFLOW\_TRACKING\_URI=http://localhost:5000

MLFLOW\_S3\_ENDPOINT\_URL=http://localhost:9000

MYSQL\_DATABASE=mlflow\_database

MYSQL\_USER=mlflow\_user

MYSQL\_PASSWORD=mlflow

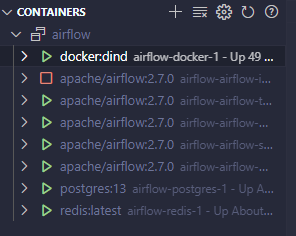
MYSQL\_ROOT\_PASSWORD=mysql

***Задание 1. Пайплайн, который обучает любой классификатор из sklearn по заданному набору параметров***

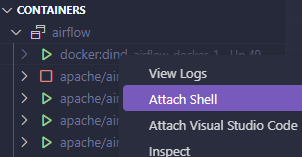
Для работы этого задания нужно сделать пару вещей, а именно:

* Собрать докер внутри dind для обучения
* Сделать соединение в airflow

Начнем с dind. Мы ранее запустили сборку, и имеем подобное:



На всякий случай после запуска дождемся инициализации (пару минут), и зайдем в docker:dind:



Теперь подготовим здесь докер для использования sklearn пакета, обучения и логирования результатов:

cd /dockerfiles/sklearn && docker build . -t sklearn\_image

Рассмотрим сам файл докера:

FROM python:3.10-slim

# *Set where tracing should be sended*

ENV MLFLOW\_TRACKING\_URI=http://mlflow\_server:5000

ENV MLFLOW\_S3\_ENDPOINT\_URL=http://minio:9000

ENV AWS\_ACCESS\_KEY\_ID=minio

ENV AWS\_SECRET\_ACCESS\_KEY=minio123

# *Set timeout otherwise long build will crash in pip*

RUN apt update && apt install -y git

RUN pip install --default-timeout=100 scikit-learn mlflow boto3

Здесь нам необходимо проставить путь до MLFLOW и необходимые параметры, а также поставить нужные пакеты. Важно заполнить путь именно таким образом (возможно можно еще заменить mlflow\_server на web)

Дождемся сборки. Теперь мы готовы к запуску основного DAG для обучения.

Сам DAG:

1. **import** os
2. **from** datetime **import** datetime
3. **from** airflow **import** DAG
4. **from** airflow.providers.docker.operators.docker **import** DockerOperator
5. **from** airflow.sensors.filesystem **import** FileSensor
7. **from** docker.types **import** Mount
9. default\_args **=** {
10. 'owner': 'airflow',
11. 'start\_date': datetime(2023, 1, 1),
12. 'retries': 1,
13. }

16. dag **=** DAG(
17. 'train\_nn',
18. default\_args**=**default\_args,
19. description**=**'DAG train NN',
20. schedule\_interval**=**None,
21. )
23. wait\_for\_new\_file **=** FileSensor(
24. task\_id**=**'wait\_for\_new\_train\_file',
25. poke\_interval**=**10,  # Interval to check for new files (in seconds)
26. filepath**=**'/opt/airflow/data/lab3\_configs',  # Target folder to monitor
27. fs\_conn\_id**=**'file\_train\_connection',
28. dag**=**dag,
29. )
31. train\_nn **=** DockerOperator(
32. task\_id**=**'train\_nn\_on\_new\_config',
33. image**=**'sklearn\_image',
34. docker\_url**=**"tcp://docker:2375", # For Dind usage case
35. mount\_tmp\_dir**=**False,
36. network\_mode**=**'host',
37. #    env\_file='/dockerfiles/.env',
38. entrypoint**=**'bash',
39. command**=**['-c', "python /data/scripts/train\_sklearn.py /data/lab3\_configs"],
40. mounts**=**[
41. Mount(source**=**'/data', target**=**'/data', type**=**'bind'),
42. ],
43. dag**=**dag,
44. )

47. wait\_for\_new\_file >> train\_nn

Важно отметить сборку докера для обучения, особенно на 36 строчке с установкой network\_mode. Нам необходимо чтобы был доступ к mlflow, значит нам надо прокинуть хост сеть для запускаемого докера, тогда по ранее установленным ссылкам (через .env для airflow-worker и для docker\_sklearn) будет доступен mlflow.

Я также пытался здесь настроить через еще один.env, однако почему-то при сборке через DockerOperation он не видит этот файл при сборке, якобы он скрыт. Думал chmod проставить, но как-то не вышло, поэтому решил явно в dockerfile прописать.

Код обучения:

1. **import** argparse
2. **import** glob
3. **import** os
4. **import** json
5. **import** datetime
7. **from** sklearn.datasets **import** load\_digits
8. **from** sklearn.discriminant\_analysis **import** QuadraticDiscriminantAnalysis
9. **from** sklearn.ensemble **import** AdaBoostClassifier, RandomForestClassifier
10. **from** sklearn.gaussian\_process **import** GaussianProcessClassifier
11. **from** sklearn.gaussian\_process.kernels **import** RBF
12. **from** sklearn.inspection **import** DecisionBoundaryDisplay
13. **from** sklearn.model\_selection **import** train\_test\_split
14. **from** sklearn.naive\_bayes **import** GaussianNB
15. **from** sklearn.neighbors **import** KNeighborsClassifier
16. **from** sklearn.neural\_network **import** MLPClassifier
17. **from** sklearn.pipeline **import** make\_pipeline
18. **from** sklearn.preprocessing **import** StandardScaler
19. **from** sklearn.svm **import** SVC
20. **from** sklearn.tree **import** DecisionTreeClassifier
21. **from** sklearn.metrics **import** f1\_score, accuracy\_score
23. **from** mlflow.models **import** infer\_signature
24. **import** mlflow
26. AVAILABLE\_CLASSIFIERS\_NAME2MODEL\_DICT **=** {
27. 'KNeighborsClassifier': KNeighborsClassifier,
28. 'SVC': SVC,
29. 'GaussianProcessClassifier': GaussianProcessClassifier,
30. 'DecisionTreeClassifier': DecisionTreeClassifier,
31. 'RandomForestClassifier': RandomForestClassifier,
32. 'MLPClassifier': MLPClassifier,
33. 'AdaBoostClassifier': AdaBoostClassifier,
34. 'GaussianNB': GaussianNB,
35. 'QuadraticDiscriminantAnalysis': QuadraticDiscriminantAnalysis,
36. }
38. **def** train(config):
39. with open(config, 'r') as jr:
40. config\_data **=** json.load(jr)
42. **if** AVAILABLE\_CLASSIFIERS\_NAME2MODEL\_DICT.get(config\_data['model']) **is** None:
43. **raise** Exception('Unknown model name')
45. with mlflow.start\_run():
46. digits **=** load\_digits()
47. # flatten the images
48. n\_samples **=** len(digits.images)
49. data **=** digits.images.reshape((n\_samples, **-**1))
50. # Split data into 50% train and 50% test subsets
51. X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(
52. data, digits.target, test\_size**=**0.5, shuffle**=**False
53. )

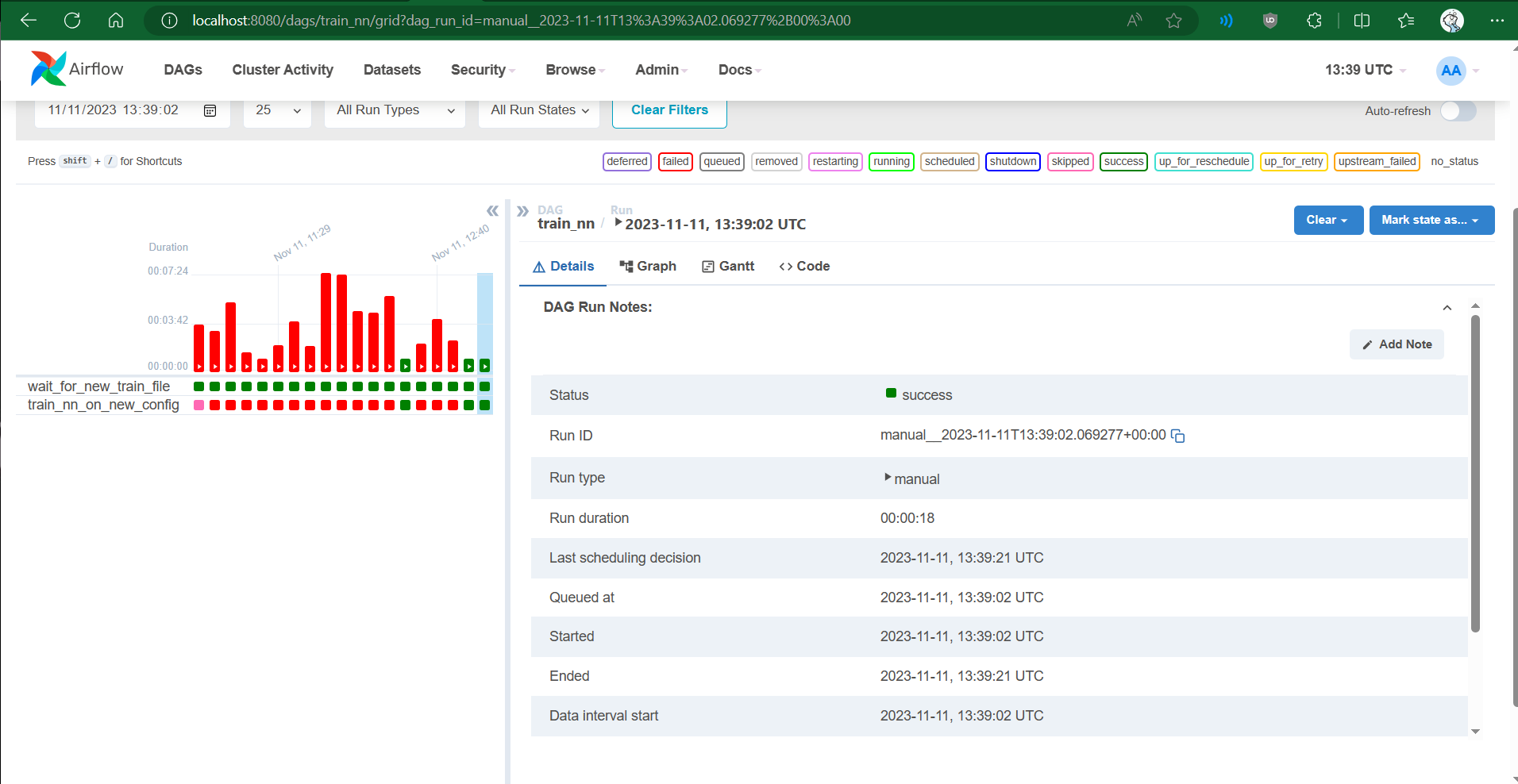
56. clf **=** AVAILABLE\_CLASSIFIERS\_NAME2MODEL\_DICT.get(config\_data['model'])(**\*\***config\_data['parameters'])
57. mlflow.log\_params(config\_data)
58. # Learn the digits on the train subset
59. clf.fit(X\_train, y\_train)
61. # Predict the value of the digit on the test subset
62. predicted **=** clf.predict(X\_test)
64. f1\_w **=** f1\_score(y\_test, predicted, average**=**'weighted')
65. f1\_avg **=** f1\_score(y\_test, predicted, average**=**'macro')
66. acc **=** accuracy\_score(y\_test, predicted)
67. mlflow.log\_metric("f1\_w", f1\_w)
68. mlflow.log\_metric("f1\_avg", f1\_avg)
69. mlflow.log\_metric("acc", acc)
71. signature **=** infer\_signature(X\_test, predicted)
72. mlflow.sklearn.log\_model(
73. clf, 'sk\_models',
74. signature**=**signature,
75. # Name must be unique, so just add datetime
76. registered\_model\_name**=**'sklearn-model-%s-%s' **%** (config\_data['model'], datetime.datetime.now()),
77. )

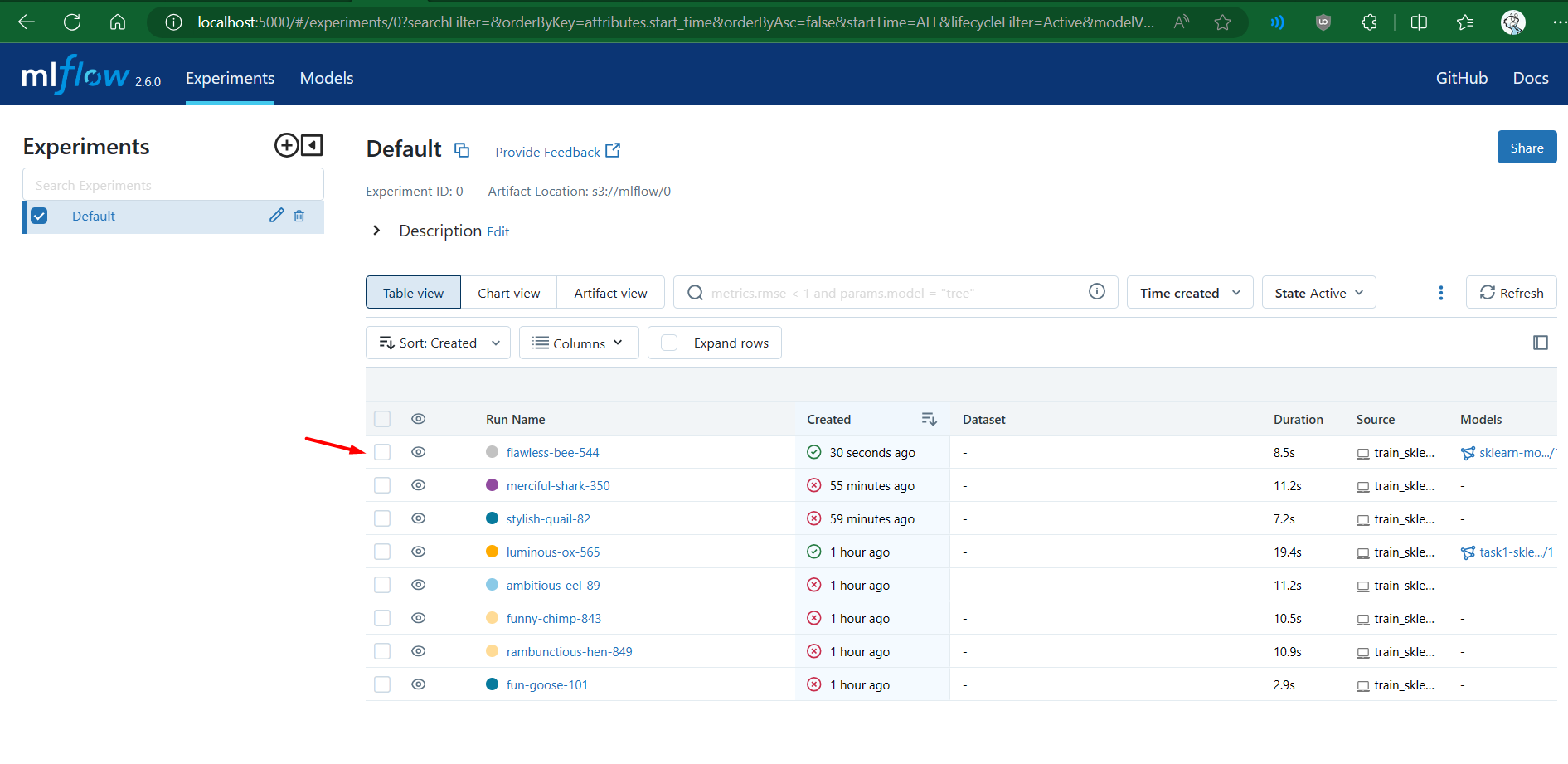
80. **if** \_\_name\_\_ **==** '\_\_main\_\_':
81. parser **=** argparse.ArgumentParser(description**=**'Train sklearn model on config file.')
82. parser.add\_argument('configs\_folder\_path', type**=**str,
83. help**=**'Path to folder with configs of the sklearn models.')
85. args **=** parser.parse\_args()
86. configs\_files\_path **=** glob.glob(os.path.join(args.configs\_folder\_path, '\*.json'))
87. **for** config\_file\_path **in** configs\_files\_path:
88. train(config\_file\_path)

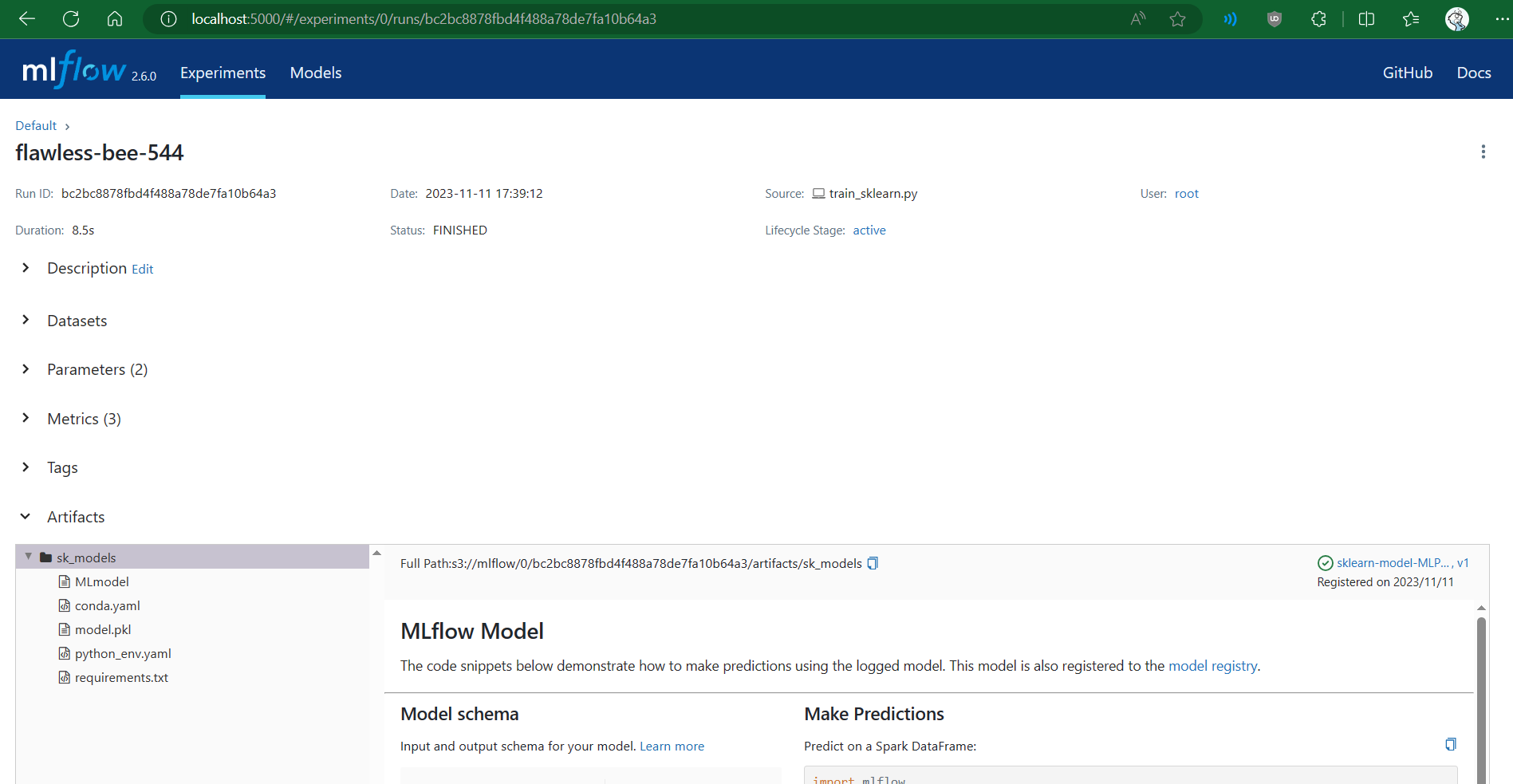
Пример json конфига:

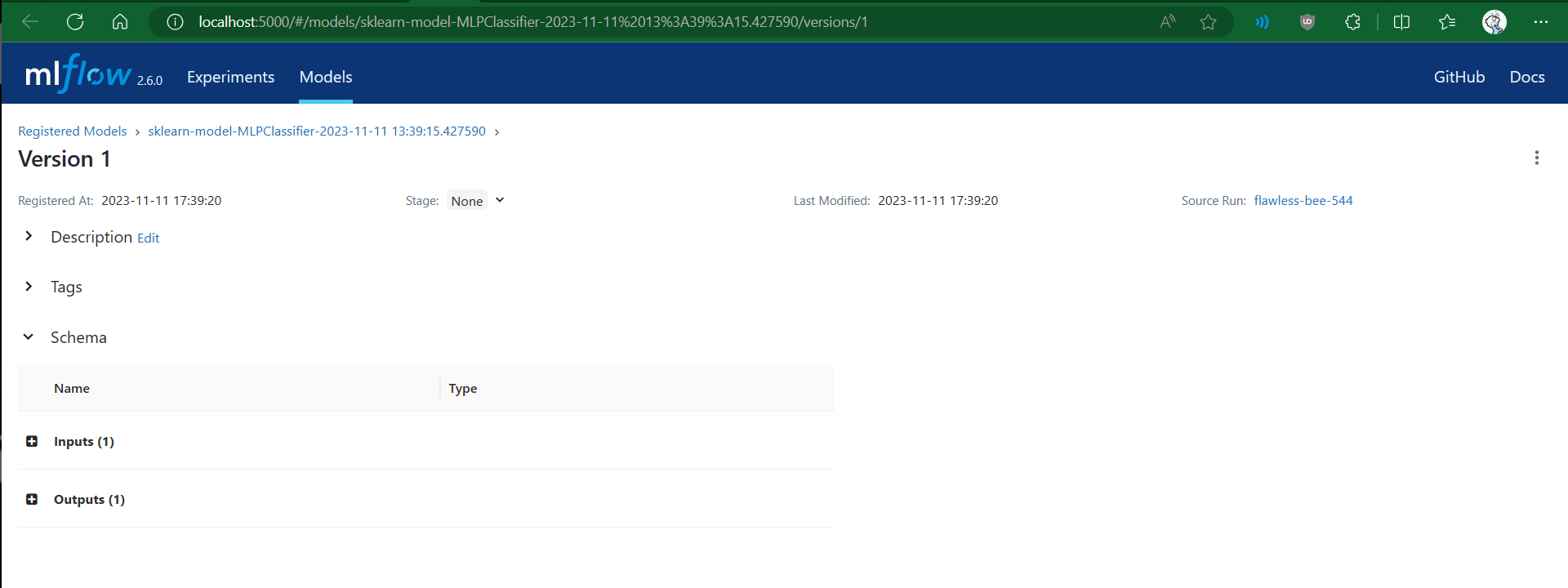
1. {
2. "model": "MLPClassifier",
3. "parameters": {
4. "hidden\_layer\_sizes": 200,
5. "learning\_rate": "adaptive"
6. }
7. }

Пример запуска:









***Задание 2. Пайплайн, который выбирает лучшую модель из обученных и производит её хостинг.***

**Теперь необходимо чтобы автоматически проверялись модели и лучшей ставился статус продакшена (Production).**

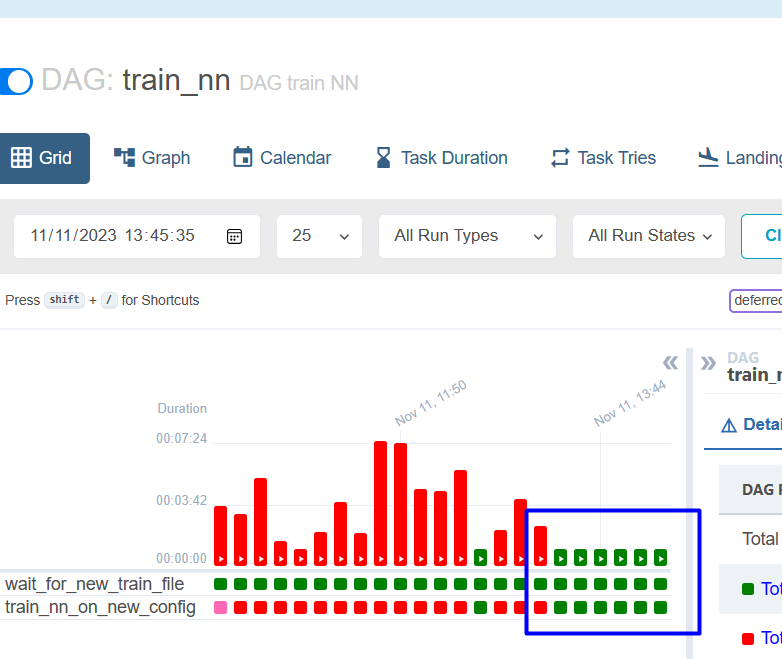
**Для этого используется следующий DAG:**

1. **from** datetime **import** datetime, timedelta
2. **from** airflow.sensors.time\_delta **import** TimeDeltaSensorAsync
3. **from** airflow.decorators **import** task, dag

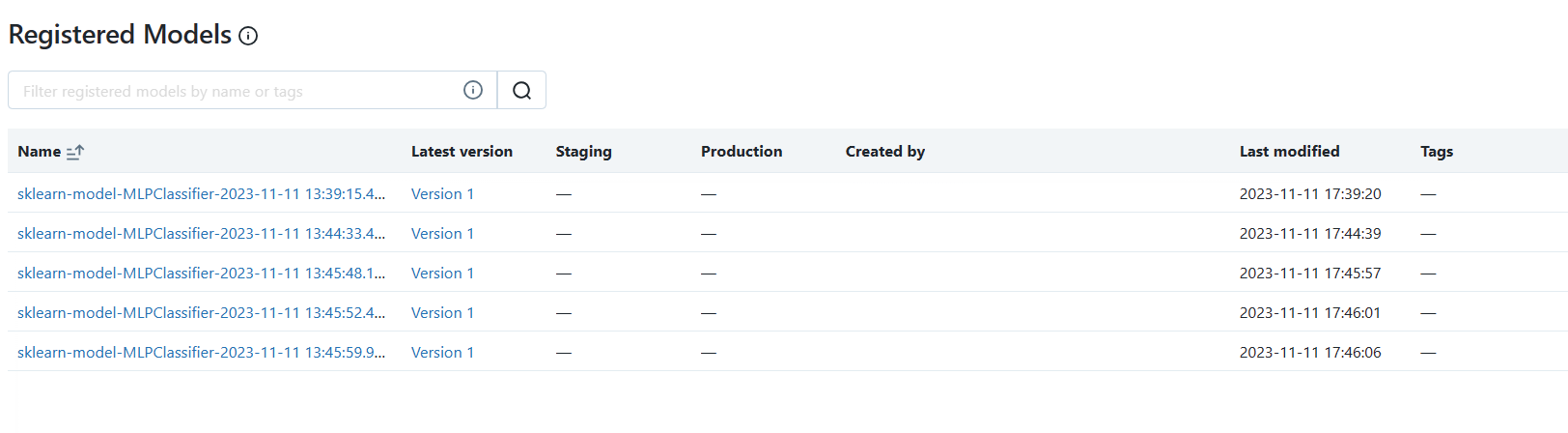
6. @dag(
7. dag\_id**=**'choose\_prod\_model',
8. start\_date**=**datetime(2021, 1, 1),
9. schedule\_interval**=**None,
10. catchup**=**False,
11. doc\_md**=**\_\_doc\_\_,
12. )
13. **def** choose\_prod\_model():
14. wait\_interval **=** TimeDeltaSensorAsync(
15. task\_id**=**"wait\_interval",
16. delta**=**timedelta(seconds**=**10),
17. )
19. @task
20. **def** check\_mlflow\_models():
21. **from** mlflow **import** MlflowClient
23. client **=** MlflowClient()
24. best\_f1\_w **=** **-**1.0
25. best\_model **=** None
27. **for** rm **in** client.search\_registered\_models():
28. latest\_model\_version **=** rm.latest\_versions[**-**1]
29. exp\_info **=** client.get\_run(latest\_model\_version.run\_id)
31. **if** exp\_info.data.metrics['f1\_w'] > best\_f1\_w:
32. best\_f1\_w **=** exp\_info.data.metrics['f1\_w']
33. best\_model **=** latest\_model\_version
35. **if** best\_model **is** **not** None:
36. client.transition\_model\_version\_stage(
37. name**=**best\_model.name, version**=**best\_model.version, stage**=**"Production"
38. )
39. **else**:
40. print('Models not found')
42. wait\_interval >> check\_mlflow\_models()
44. choose\_prod\_model\_pipeline **=** choose\_prod\_model()

**Ранее был запущено обучение и сохранилась модель, запустим подобный скрипт несколько раз и затем запустим DAG описанный выше.**

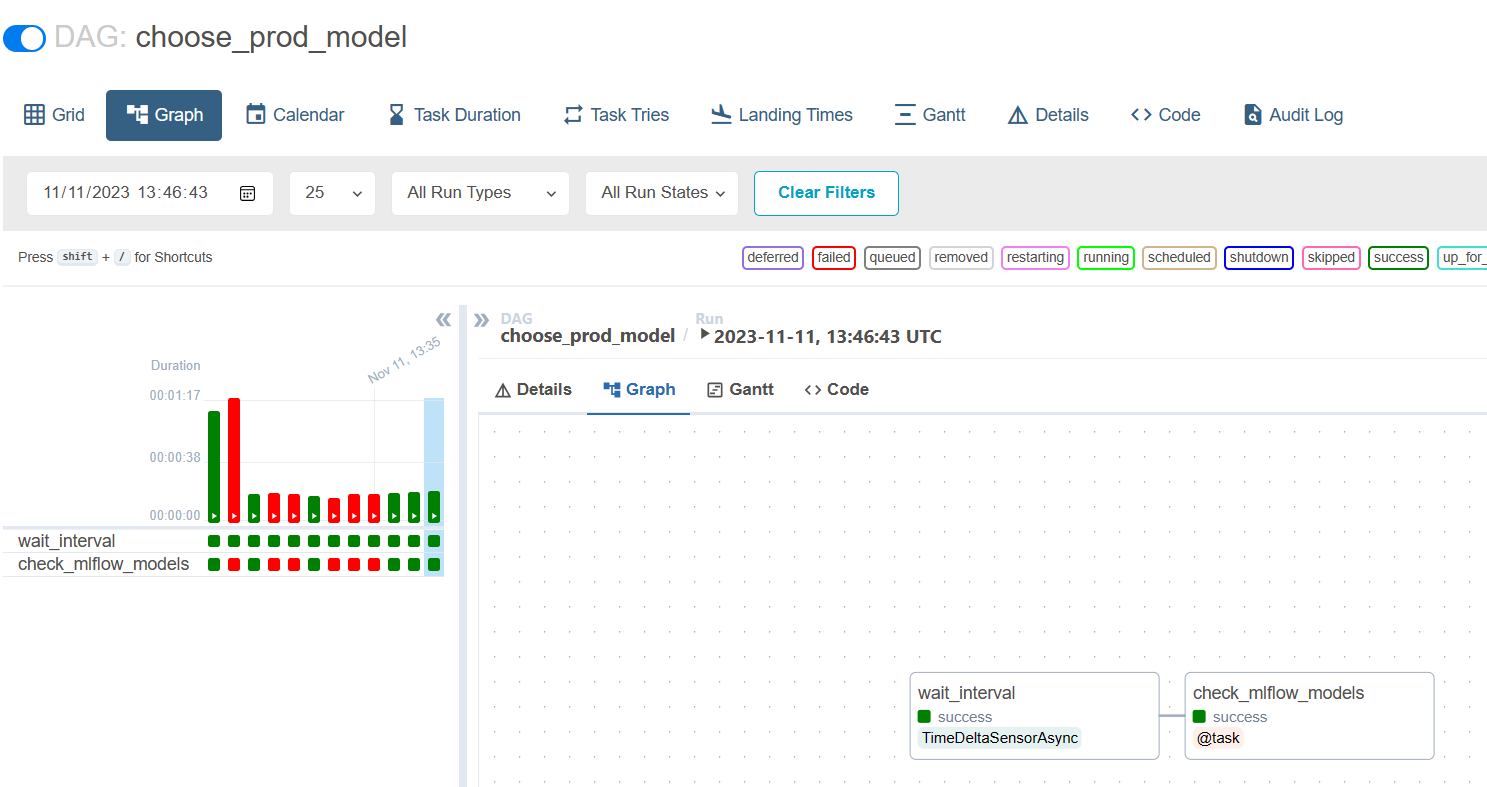
**Запускаем несколько раз обучение:**



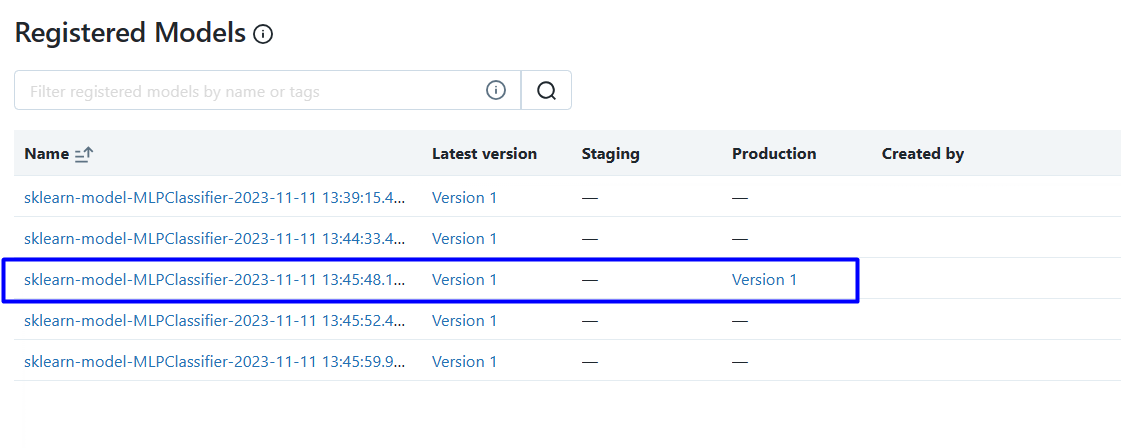
**Видим несколько моделей в итоге:**

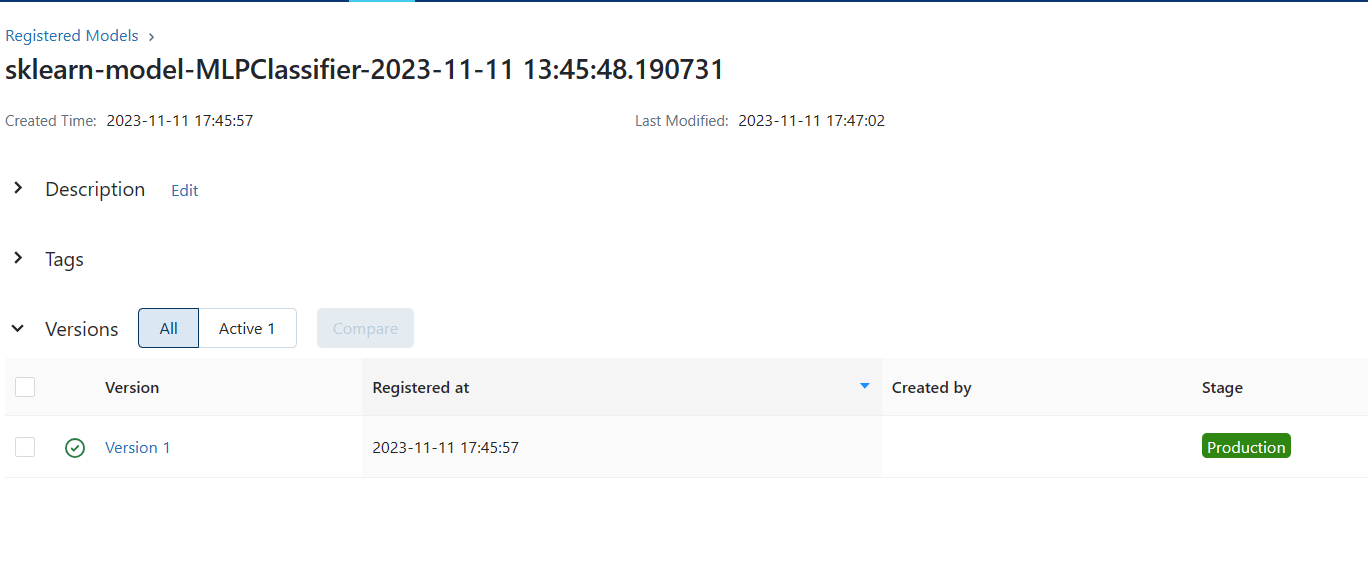


**Запускаем DAG для поиска лучшей:**

****

**Проверяем:**

****

****

**Другие модели не имеют подобное статуса, пример:**

****

**Теперь, данную модель можно захостить через сам MLflow и MLServer.**

**Некоторая помарка, я не делал отдельных скриптов и отдельных вызовов для следующий дествий, так как они довольно скромные и делаются быстро. Я установил все нужное и делал все в консоли в airflow-worker для удобства.**

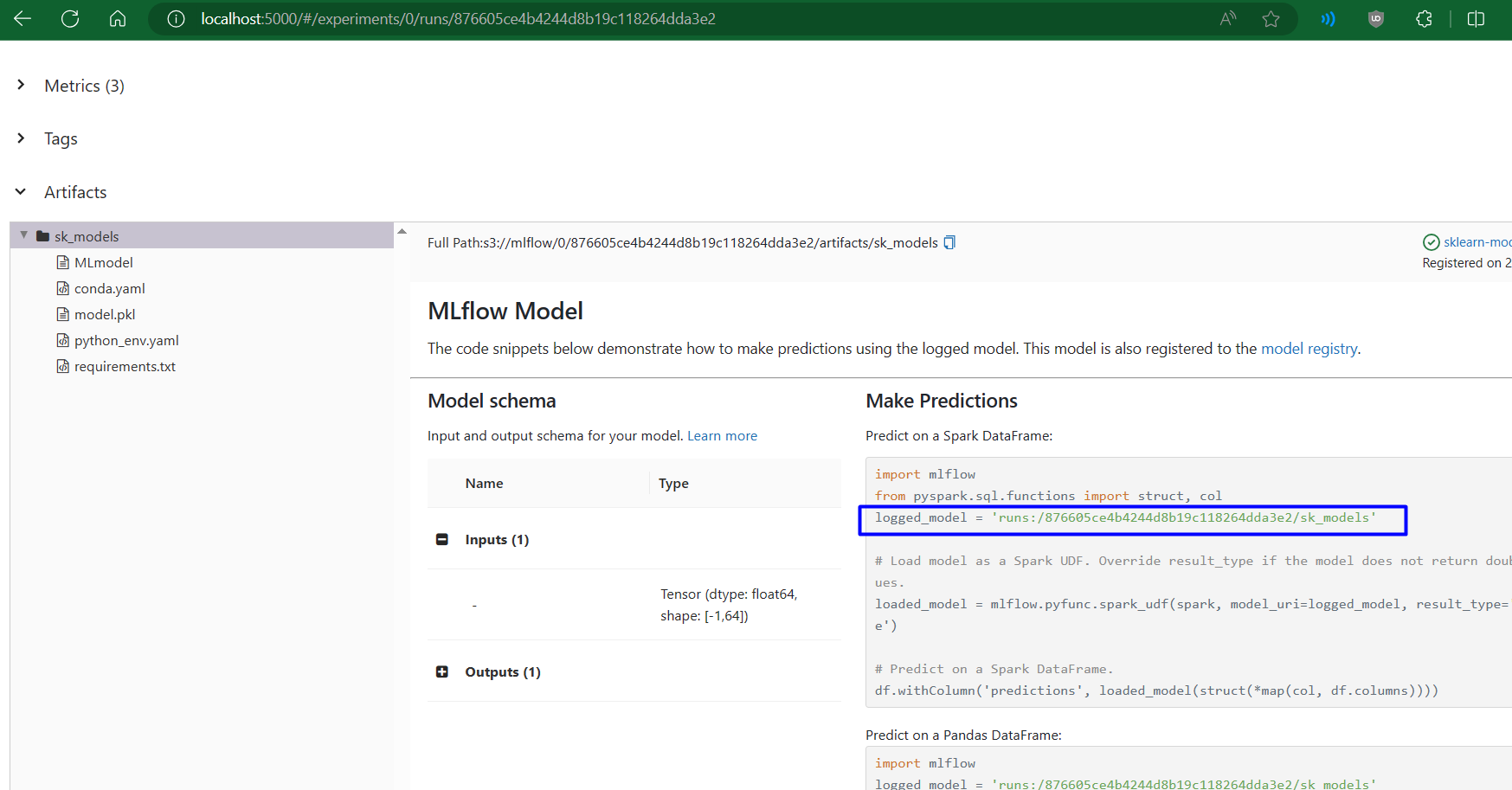
**Во-первых установим MLFlow с extra пакетами (помним, просто в airflow-worker для удобства и быстроты, далее все там же):**

pip install mlflow**[**extras**]**

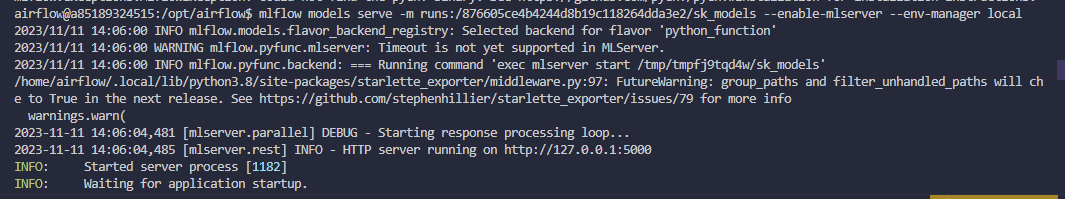
**Теперь необходимо запустить модель с:**

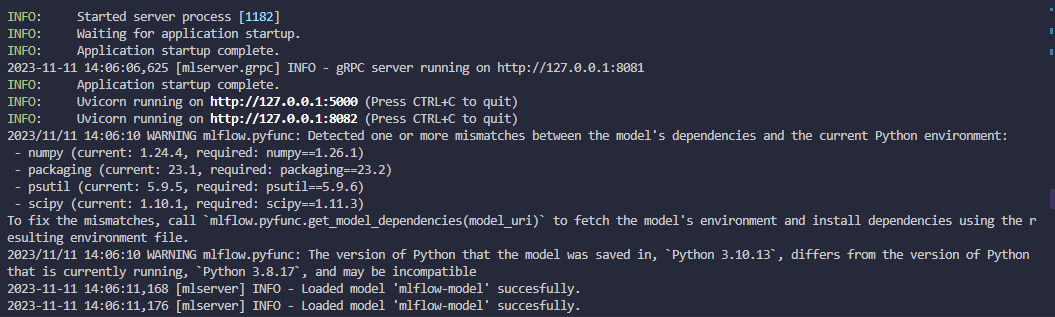
mlflow models serve -m runs:/876605ce4b4244d8b19c118264dda3e2/sk\_models --enable-mlserver --env-manager local

**В качестве параметра -m указывается model\_uri (значение runs:/876605ce4b4244d8b19c118264dda3e2/sk\_models), которую можно получить, например отсюда:**



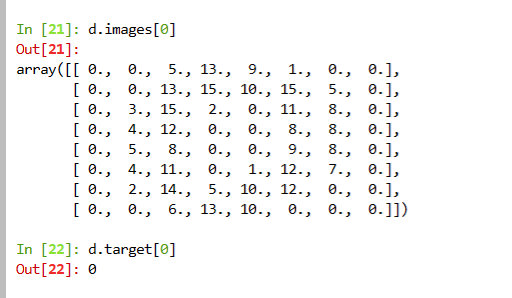
**Я взял это значение для Production сети.**





**Модель готова к работе.**

**Возьмем следующий вход:**

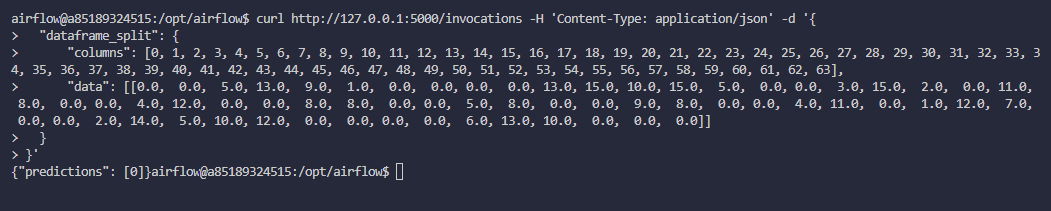


**По очертаниям, это нуль, что подтверждает значение метки (нулевой класс, или нуль имеется в виду цифра из набора данных мини версии «mnist»).**

**Перепишем входные данные в виде запроса:**

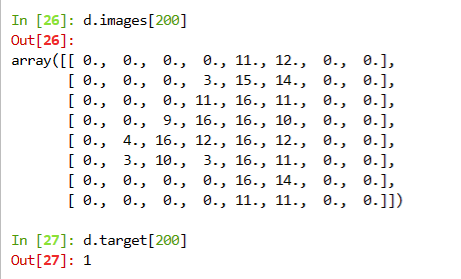
1. curl http://127.0.0.1:5000/invocations -H 'Content-Type: application/json' -d '{
2. "dataframe\_split": {
3. "columns": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63],
4. "data": [[0.0,  0.0,  5.0, 13.0,  9.0,  1.0,  0.0,  0.0, 0.0,  0.0, 13.0, 15.0, 10.0, 15.0,  5.0,  0.0, 0.0,  3.0, 15.0,  2.0,  0.0, 11.0,  8.0,  0.0, 0.0,  4.0, 12.0,  0.0,  0.0,  8.0,  8.0,  0.0, 0.0,  5.0,  8.0,  0.0,  0.0,  9.0,  8.0,  0.0, 0.0,  4.0, 11.0,  0.0,  1.0, 12.0,  7.0,  0.0, 0.0,  2.0, 14.0,  5.0, 10.0, 12.0,  0.0,  0.0, 0.0,  0.0,  6.0, 13.0, 10.0,  0.0,  0.0,  0.0]]
5. }
6. }'

**В другой консоли отправим этот запрос:**



**И видим, что мы получаем ответ класс 0.**

**Для другого числа (единица):**



**Ответ:**

