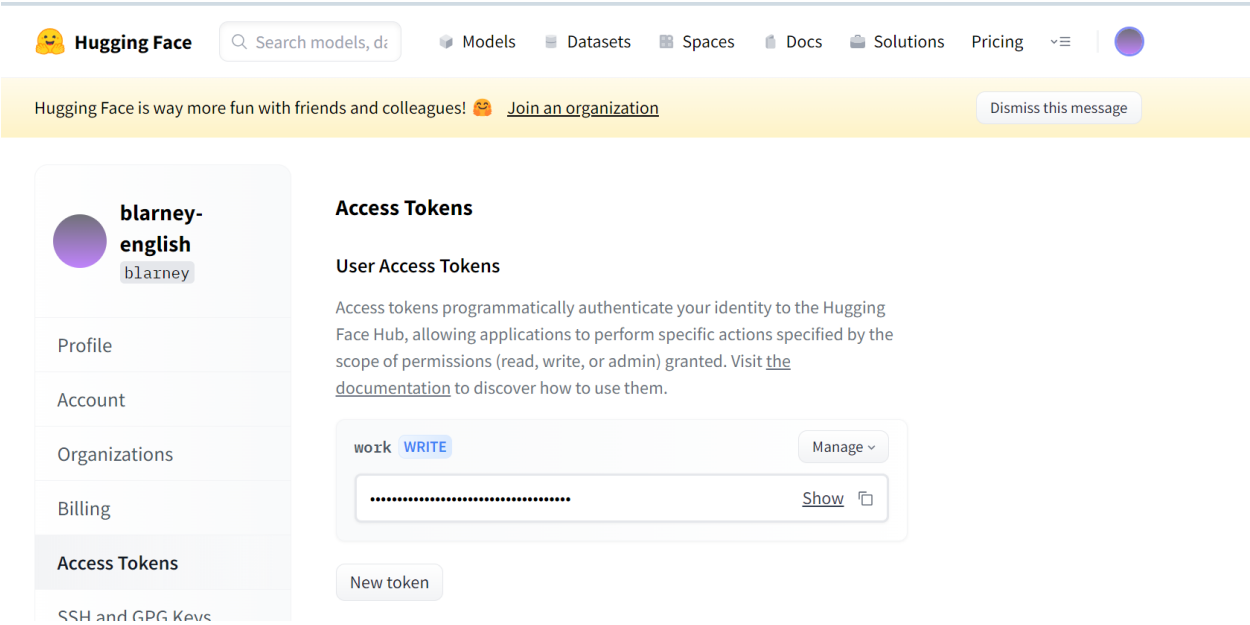
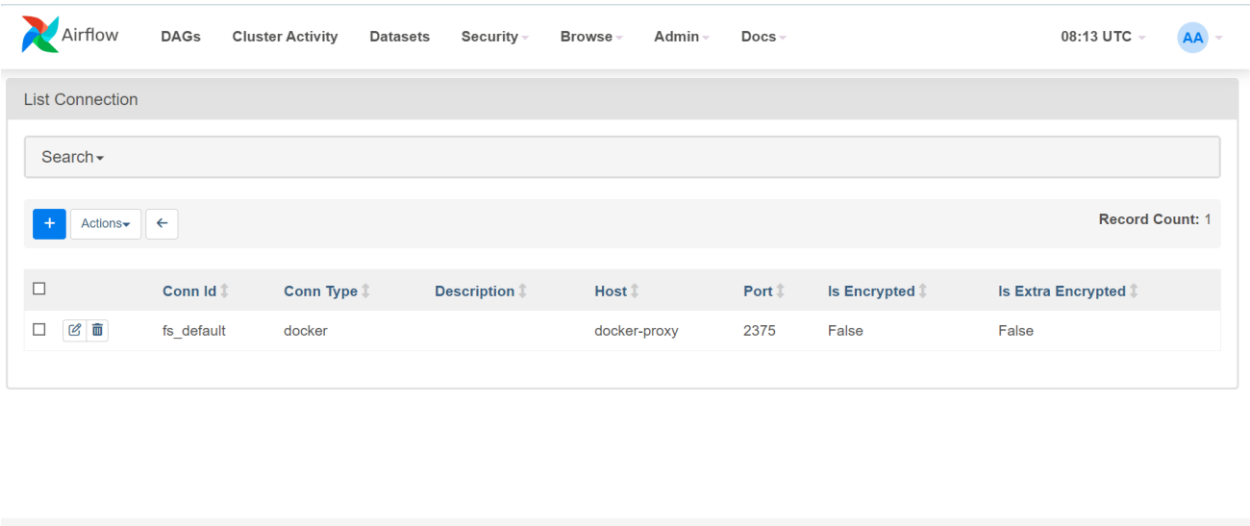
**Отчёт по второй лабораторной**

Для выполнения данной лабораторной необходимо получить API токен Для этого я зарегистрировался на Hugging face




Для работы в меню Connections создадим подключение




Создадим DAG lab2_dag.py

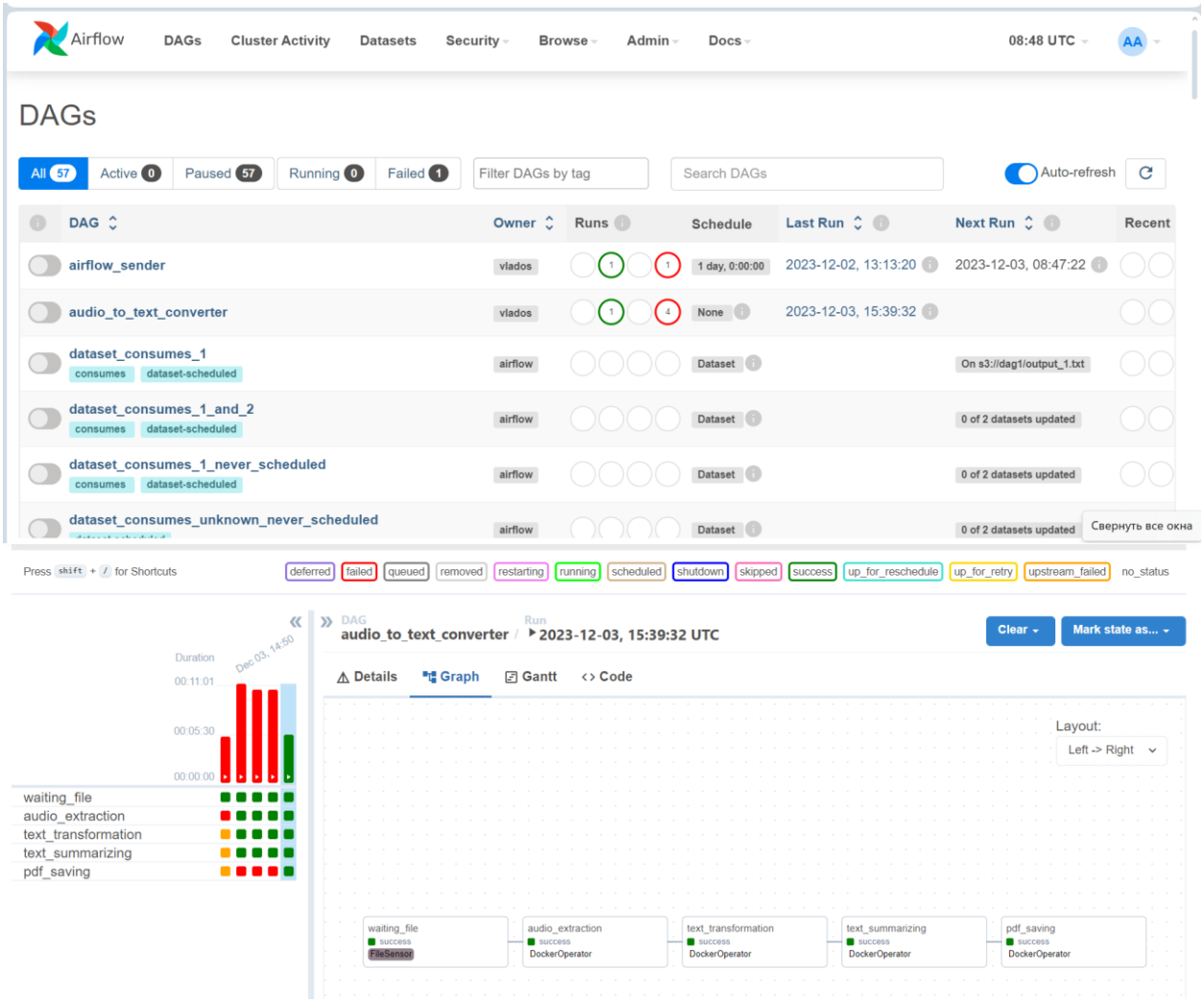
airflow > dags > lab2_dag.py

```python
from datetime import datetime
from airflow import DAG
from docker.types import Mount
from airflow.providers.docker.operators.docker import DockerOperator
from airflow.sensors.filesystem import FileSensor

default_args = {
    'owner': 'vlados',
    'start_date': datetime(2023, 1, 3),
    'retries': 1,
}

dag = DAG(
    'audio_to_text_converter',
    default_args=default_args,
    description='DAG for extracting audio, transforming to text, summarizing, and saving as PDF',
    schedule_interval=None,
)

waiting_file = FileSensor(
    task_id='waiting_file',
    poke_interval=10,  # Interval to check for new files (in seconds)
    filepath='/opt/airflow/data',  # Target folder to monitor
    fs_conn_id='fs_default',
    dag=dag,
)

```

```python
audio_extraction = DockerOperator(
    task_id='audio_extraction',
    image='jrottenberg/ffmpeg',
    command='-i /data/video.mp4 -vn -acodec copy /data/audio.aac',
    mounts=[Mount(source='/data', target='/data', type='bind')],
    docker_url="tcp://docker-proxy:2375",
    dag=dag,
)

text_transformation = DockerOperator(
    task_id='text_transformation',
    image='nyurik/alpine-python3-requests',
    command='python /data/text_transformation.py',
    mounts=[Mount(source='/data', target='/data', type='bind')],
    docker_url="tcp://docker-proxy:2375",
    dag=dag,
)

text_summarizing = DockerOperator(
    task_id='text_summarizing',
    image='nyurik/alpine-python3-requests',
    command='python /data/text_summarizing.py',
    mounts=[Mount(source='/data', target='/data', type='bind')],
    docker_url="tcp://docker-proxy:2375",
    dag=dag,
)
```

```python
pdf_saving = DockerOperator(
    task_id='pdf_saving',
    image='blarney/tensorflow_learner:1.0',
    command='python /data/save_to_pdf.py',
    mounts=[Mount(source='/data', target='/data', type='bind')],
    docker_url="tcp://docker-proxy:2375",
    dag=dag,
)

waiting_file >> audio_extraction >> text_transformation >> text_summarizing >> pdf_saving
```

Теперь заходим в DAG audio_to_text_converter (файл lab2_dag.py приложен) и запускаем его

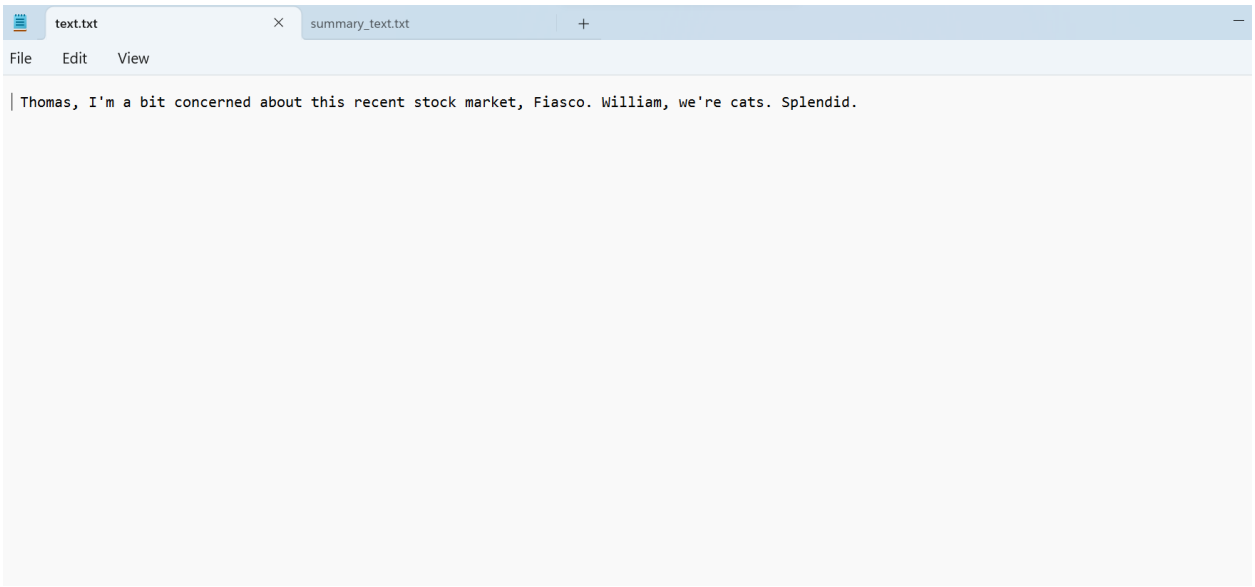


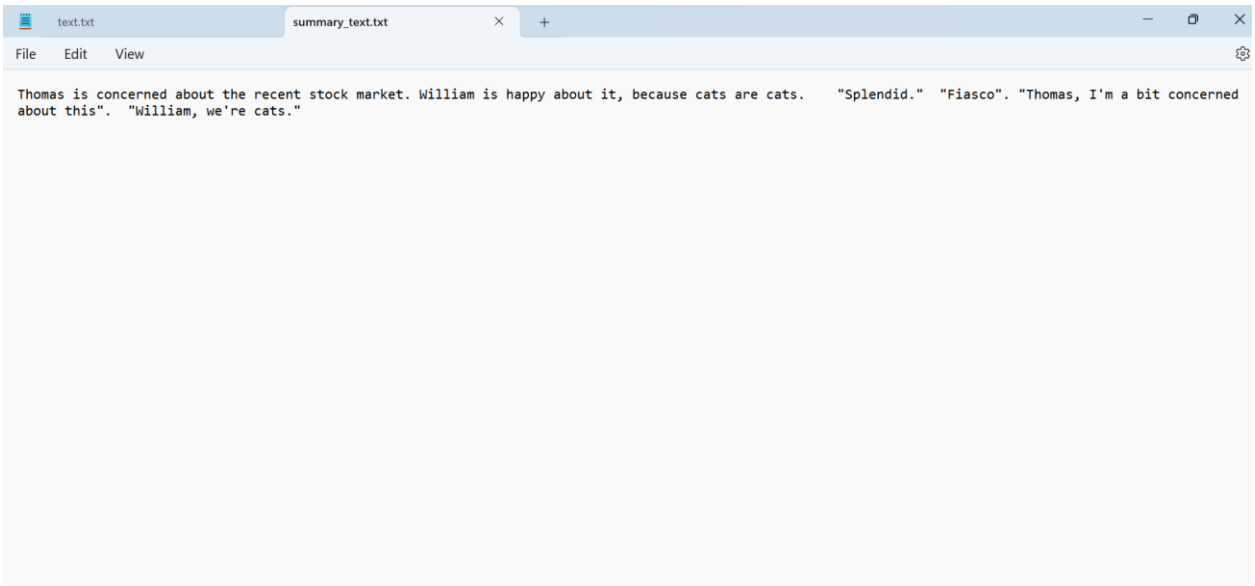

Все нужные файлы создались (во вложении). Генерация произведена на основе этого видео

https://www.youtube.com/watch?v=vTzeQ2HXVXw

| Имя | Дата изменения | Тип | Размер |
|---|---|---|---|
| audio.aac | 03.12.2023 19:44 | AAC Audio File (VLC) | 242 КБ |
| csv_logger.py | 03.12.2023 16:56 | Исходный файл Pyth... | 0 КБ |
| log.csv | 04.12.2023 11:44 | Файл Microsoft Excel,... | 15 КБ |
| mnist_training.py | 04.12.2023 11:43 | Исходный файл Pyth... | 2 КБ |
| model.h5 | 04.12.2023 11:43 | Файл "H5" | 17 КБ |
| result.pdf | 03.12.2023 19:44 | Документ Adobe Acr... | 2 КБ |
| save_to_pdf.py | 03.12.2023 19:37 | Исходный файл Pyth... | 1 КБ |
| summary_text.txt | 03.12.2023 19:44 | Text Document | 1 КБ |
| text.txt | 03.12.2023 19:44 | Text Document | 1 КБ |
| text_summarizing.py | 03.12.2023 19:37 | Исходный файл Pyth... | 1 КБ |
| text_transformation.py | 03.12.2023 17:20 | Исходный файл Pyth... | 1 КБ |
| video.mp4 | 03.12.2023 12:36 | MP4 Video File (VLC) | 420 КБ |

## Посмотрим, что получилось



Для использования библиотеки fpdf (позволяет создать pdf файл), а также библиотек я создал образ Docker и загрузил необходимые библиотеки. Этот же образ используется при обучении нейросети

C: > docker > Dockerfile

```
#Deriving the latest base image
FROM tensorflow/tensorflow:latest

RUN pip install scikit-learn numpy pandas fpdf

#Labels as key value pair
LABEL Maintainer="vlados.tensorflow_learner"


# Any working directory can be chosen as per choice like '/' or '/home' etc
# i have chosen /usr/app/src
WORKDIR /usr/app/src
```

Теперь отправим образ в докерхаб

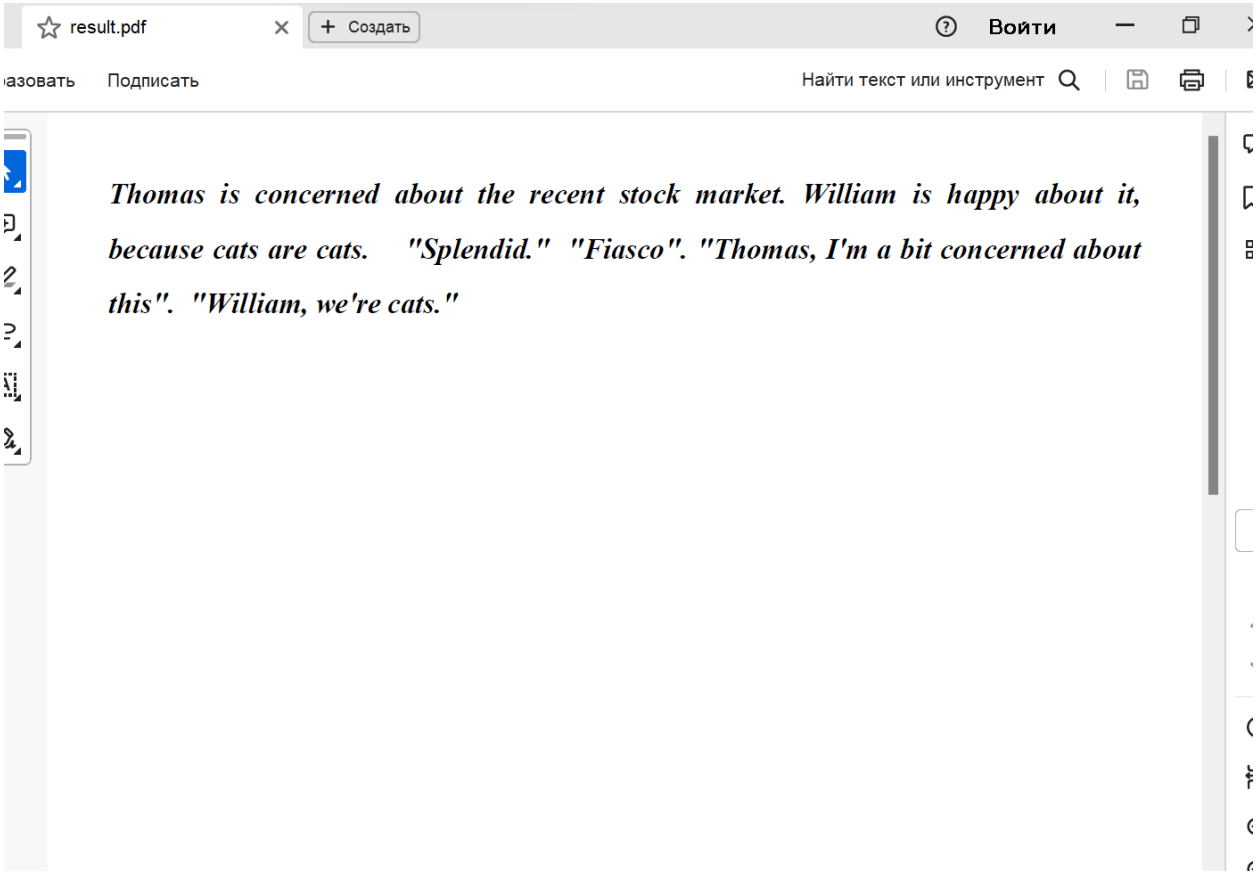
```
Start a build
PS C:\docker> docker build . -t my_container
[+] Building 299.4s (8/8) FINISHED                                                                                 docker:default
 => [internal] load build definition from Dockerfile                                                                          0.0s
 => => transferring dockerfile: 363B                                                                                          0.0s
 => [internal] load .dockerignore                                                                                             0.0s
 => => transferring context: 2B                                                                                               0.0s
 => [internal] load metadata for docker.io/tensorflow/tensorflow:latest                                                       2.7s
 => [auth] tensorflow/tensorflow:pull token for registry-1.docker.io                                                          0.0s
 => [1/3] FROM docker.io/tensorflow/tensorflow:latest@sha256:4689c724a7d65a7d289cc2ae536fa3cd6b636b2df3e23da3a4  193.4s
 => => resolve docker.io/tensorflow/tensorflow:latest@sha256:4689c724a7d65a7d289cc2ae536fa3cd6b636b2df3e23da3a44f  0.0s
 => => sha256:4689c724a7d65a7d289cc2ae536fa3cd6b636b2df3e23da3a44ffdd6ac3af46f 2.83kB / 2.83kB                                0.0s
 => => sha256:6a8c4ad355bebd8ef0bea19569618f584ccb4a8adf5bed962e3db9aa4c0010d1 4.54kB / 4.54kB                                0.0s
 => => sha256:345716e6b3b788c2ad1d84667e57458de4bfe33d247000c7b55cbbe90c1a971d 861B / 861B                                    0.9s
 => => sha256:5cc77b1d99a1e092e17bf48f5ead9028d5ac88e76729a10f02f74cb72a82d870 769B / 769B                                    1.3s
 => => sha256:aece8493d3972efa43bfd4ee3cdba659c0f787f8f59c82fb3e48c87cbb22a12e 29.54MB / 29.54MB                             54.4s
 => => sha256:5cd1e9a0284bf5a49731ecf8d7436d199fef8adc18745d487843caa92d93e9b1 163B / 163B                                    1.5s
 => => sha256:f190ee1fa1d7baf03b64c1a00e4417777c2ba26c3478a1fdf46049b8415f919a 37.63MB / 37.63MB                             54.0s
 => => sha256:909fb360ac91957ba7f421486cec96a01a9d732795d9c754f9d8cdbc4e792370 133.67MB / 133.67MB                          117.0s
 => => sha256:bcbc616aad4fac3c21d24d39c503ae697731ba183e0bef0ac576ff529fc1baec 970B / 970B                                   54.6s
 => => sha256:77cb63a520e5d19dd7e7e1a3b29b35aca75db372592dd4b14d144ed9e6ff5273 128B / 128B                                   54.8s
```

```
 => => extracting sha256:345716e6b3b788c2ad1d84667e57458de4bfe33d247000c7b55cbbe90c1a971d                                   0.0s
 => => extracting sha256:5cc77b1d99a1e092e17bf48f5ead9028d5ac88e76729a10f02f74cb72a82d870                                   0.0s
 => => extracting sha256:5cd1e9a0284bf5a49731ecf8d7436d199fef8adc18745d487843caa92d93e9b1                                   0.0s
 => => extracting sha256:f190ee1fa1d7baf03b64c1a00e4417777c2ba26c3478a1fdf46049b8415f919a                                   0.6s
 => => sha256:f2e46a0a82df7f2a17e81b6c585aae77a18510b5e96186888c75b54ef277bf82 1.13kB / 1.13kB                              87.3s
 => => sha256:831122755b02c47a8a94d5a9ad0da7717c9cbfe7f8a5230d24f764f7d35e9925 1.13kB / 1.13kB                              88.0s
 => => extracting sha256:909fb360ac91957ba7f421486cec96a01a9d732795d9c754f9d8cdbc4e792370                                   6.5s
 => => extracting sha256:bcbc616aad4fac3c21d24d39c503ae697731ba183e0bef0ac576ff529fc1baec                                   0.0s
 => => extracting sha256:77cb63a520e5d19dd7e7e1a3b29b35aca75db372592dd4b14d144ed9e6ff5273                                   0.0s
 => => extracting sha256:3268aa633e510fe5595e2faade355f5ca532d32987ed8087a6e1ffcb92b98f9b                                   2.3s
 => => extracting sha256:c6cbc212b10ddd3e38925320760e6bbe7fb3047ddae8ce92fb893dde185e95e1                                  16.6s
 => => extracting sha256:f2e46a0a82df7f2a17e81b6c585aae77a18510b5e96186888c75b54ef277bf82                                   0.0s
 => => extracting sha256:831122755b02c47a8a94d5a9ad0da7717c9cbfe7f8a5230d24f764f7d35e9925                                   0.0s
 => [2/3] RUN pip install scikit-learn numpy pandas fpdf                                                                  101.1s
 => [3/3] WORKDIR /usr/app/src                                                                                              0.0s
 => exporting to image                                                                                                      2.0s
 => => exporting layers                                                                                                     2.0s
 => => writing image sha256:e678fd59c718222517c8b8a4e037708fd21952497aade61a53ec9cc9c6d53b70                                0.0s
 => => naming to docker.io/library/my_container                                                                             0.0s
```

Пушим в докерхаб

```
PS C:\docker> docker push blarney/tensorflow_learner:1.0
The push refers to repository [docker.io/blarney/tensorflow_learner]
ac04e3abc1f7: Pushed
b4e26ff48ccf: Pushed
75acb1242fe3: Mounted from tensorflow/tensorflow
1d7a2a211a6b: Mounted from tensorflow/tensorflow
2db699de670e: Mounted from tensorflow/tensorflow
0cf31f98a4b6: Mounted from tensorflow/tensorflow
f663f4c9c5b6: Mounted from tensorflow/tensorflow
104e4c35057a: Mounted from tensorflow/tensorflow
eb864c00a034: Mounted from tensorflow/tensorflow
94235a128255: Mounted from tensorflow/tensorflow
0ac81db158f3: Mounted from tensorflow/tensorflow
8e8c3d39273b: Mounted from tensorflow/tensorflow
f99aba8580cb: Mounted from tensorflow/tensorflow
256d88da4185: Mounted from tensorflow/tensorflow
1.0: digest: sha256:8bf49649451c2fb64f85ac69022dc743911268463f851c92c4cc64056ce32111 size: 3250
```

Итоговый результат работы DAG представлен ниже




Как видно, DAG отработал **превосходно**

Теперь перейдём к созданию DAG для обучения нейросети lab2_dag2.py

airflow > dags > lab2_dag2.py

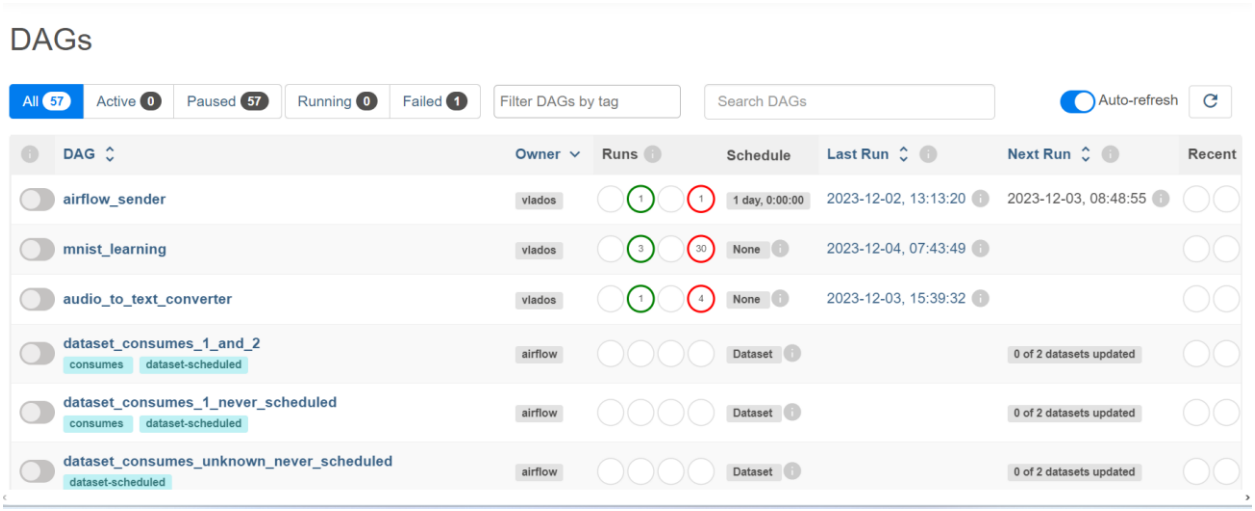
```python
from datetime import datetime
from airflow import DAG
from airflow.providers.docker.operators.docker import DockerOperator
from airflow.sensors.filesystem import FileSensor
from docker.types import Mount

default_args = {
    'owner': 'vlados',
    'start_date': datetime(2023, 12, 3),
}

dag = DAG(
    'mnist_learning',
    default_args=default_args,
    schedule_interval=None,
)

load_data_train_model = DockerOperator(
    task_id='load_data_train_model',
    image='blarney/tensorflow_learner:1.0',
    command='python /data/mnist_training.py',
    mounts=[Mount(source='/data', target='/data', type='bind')],
    docker_url="tcp://docker-proxy:2375",
    dag=dag,
)

load_data_train_model
```

Модель mnist_training.py

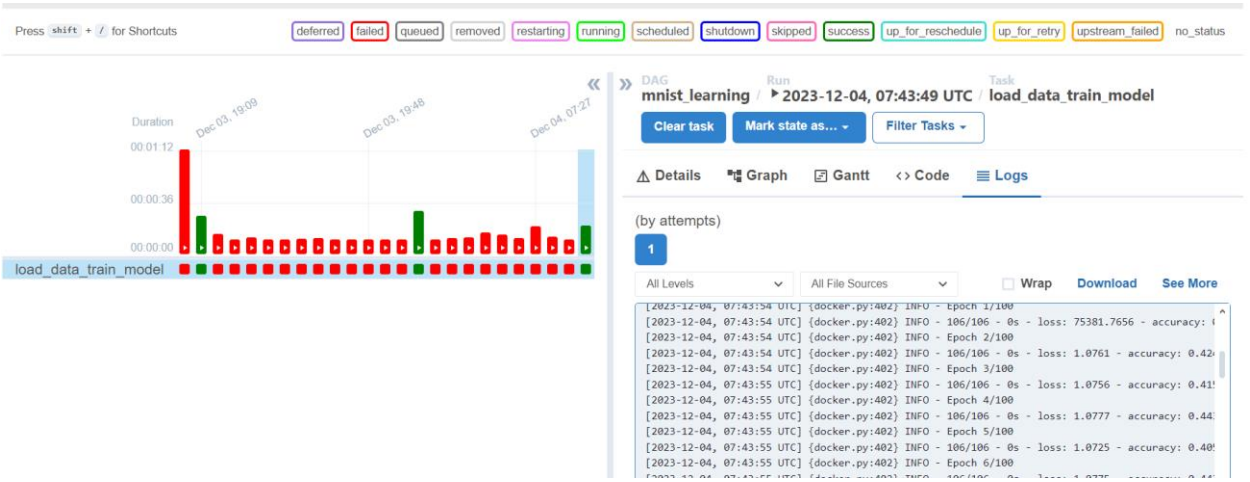
airflow > data > mnist_training.py

```python
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
import tensorflow
from keras import models
from keras import layers
from tensorflow.keras.utils import to_categorical

wine = load_wine(as_frame=True)
wine.frame.head()
X = wine.data
y = wine.target
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.4,random_state=123)

train_labels = to_categorical(y_train)
test_labels = to_categorical(y_test)
def declare_model_for_learning():
    model = models.Sequential()
    model.add(layers.Dense(12, activation='relu', input_shape=(13,)))
    model.add(layers.Dense(12, activation='relu'))
    model.add(layers.Dense(3, activation='softmax'))
    return model

my_model = declare_model_for_learning()
my_model.compile(optimizer=tensorflow.keras.optimizers.SGD(learning_rate=0.05), loss='categorical_crossent
my_model.save_weights('/data/model.h5')

my_model.load_weights('/data/model.h5')
csv_logger = [tensorflow.keras.callbacks.CSVLogger('/data/log.csv', append=True, separator=';')]
```

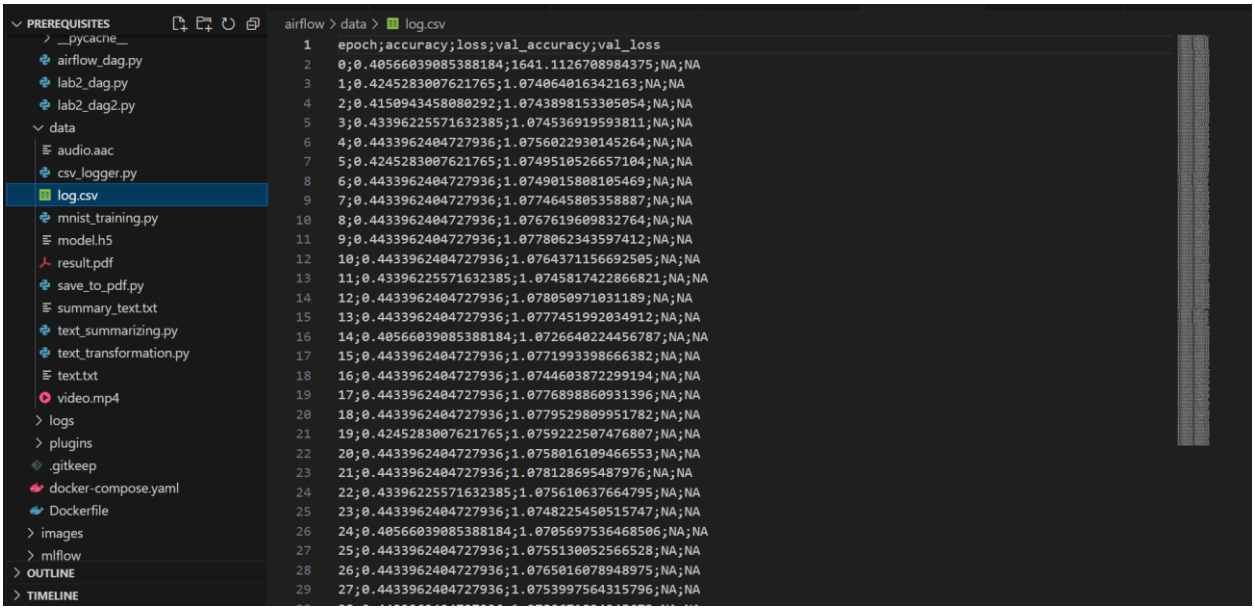
Заходим в Dag mnist_learning




Как видно, работает




Логи также сохранились




Отлично! Всё работает!