Лабораторная работ 2. Airflow and docker.

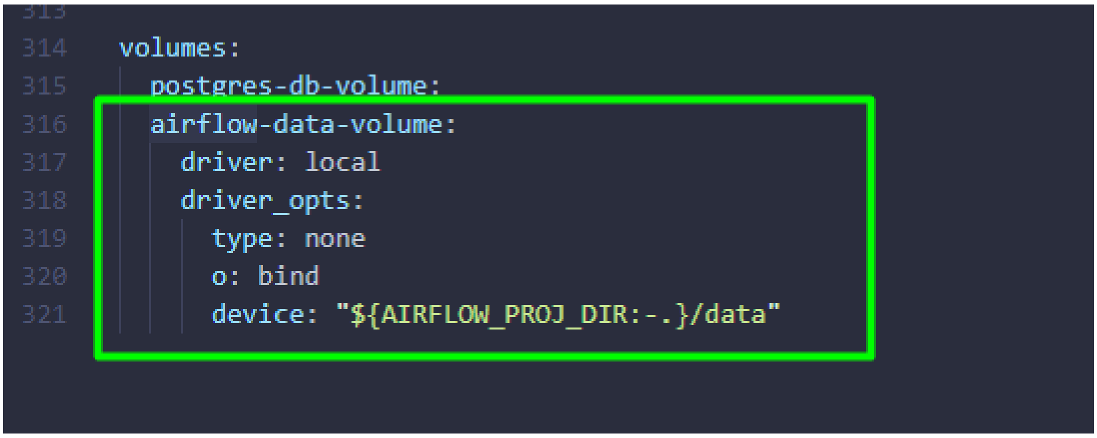
Грибанов Данил, 6233

Ход работы

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

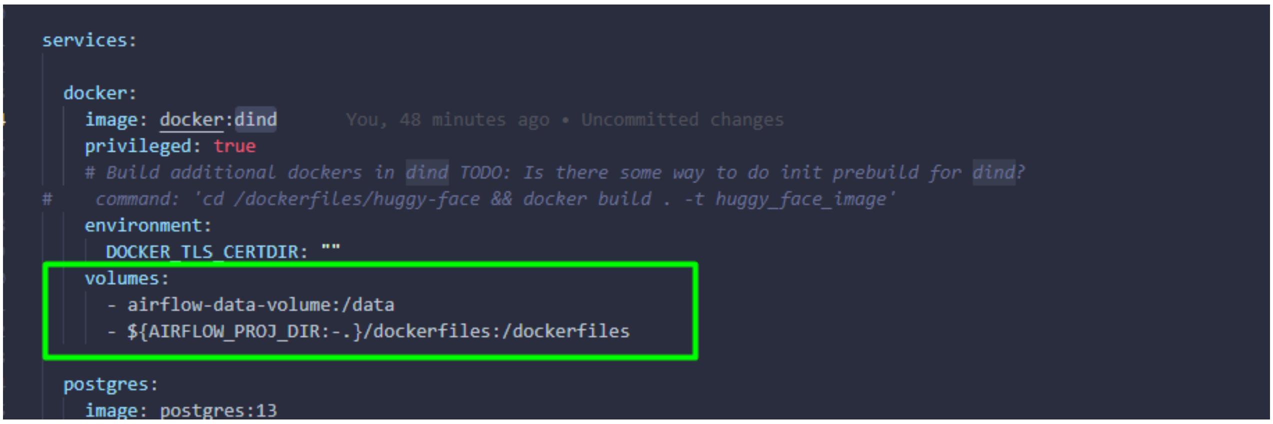
Перед запуском самой системы (докера airflow) следует подготовить следующее:

Создадим хранилище с данными для запуска и обучения всех систем:



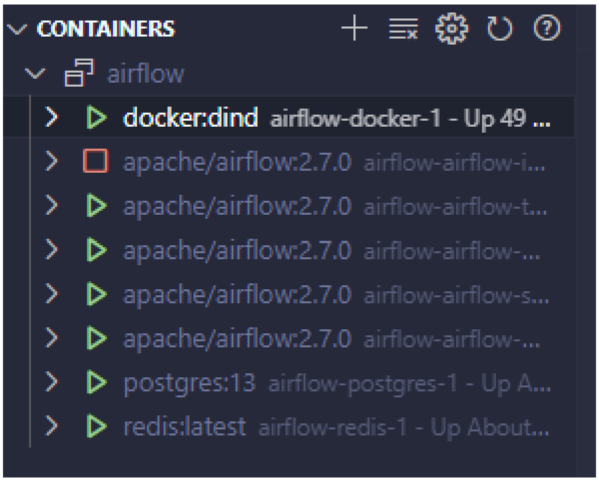
```
volumes:
  postgres-db-volume:
  airflow-data-volume:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: "${AIRFLOW_PROJ_DIR:-.}/data"
```

И добавим данное хранилище с папкой докер-файлов нужных в dind:

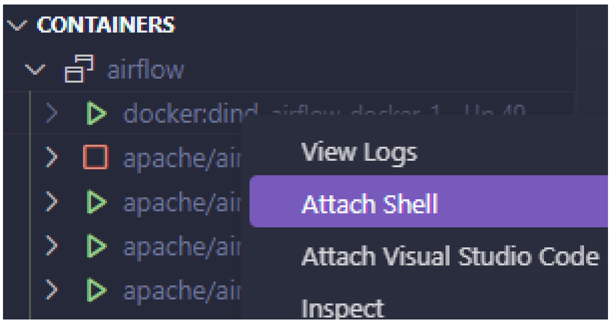


```
services:

  docker:
    image: docker:dind      You, 48 minutes ago • Uncommitted changes
    privileged: true
    # Build additional dockers in dind TODO: Is there some way to do init prebuild for dind?
#    command: 'cd /dockerfiles/huggy-face && docker build . -t huggy_face_image'
    environment:
      DOCKER_TLS_CERTDIR: ""
    volumes:
      - airflow-data-volume:/data
      - ${AIRFLOW_PROJ_DIR:-.}/dockerfiles:/dockerfiles

  postgres:
    image: postgres:13
```

Запустим docker-compose.yaml для сборки airflow и нужного:


CONTAINERS
airflow
docker:dind airflow-docker-1 - Up 49 ...
apache/airflow:2.7.0 airflow-airflow-i...
apache/airflow:2.7.0 airflow-airflow-t...
apache/airflow:2.7.0 airflow-airflow-...
apache/airflow:2.7.0 airflow-airflow-s...
apache/airflow:2.7.0 airflow-airflow-...
postgres:13 airflow-postgres-1 - Up A...
redis:latest airflow-redis-1 - Up About...

Дождемся инициализации (пару минут), и зайдем в docker:dind:



Теперь подготовим здесь докер для использования нейронных сетей:

cd /dockerfiles/huggy-face && docker build . -t huggy_face_image

Дождемся сборки (до 10 минут). Теперь мы готовы к запуску основных DAG разработанных для лабораторных работ. Код DAG к соотв. пунктам задания имеет комментарии и пояснения.

Видео в краткий пересказ

Airflow/dags/airflow_lab2.py

```python
import os
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': 'airflow',
    'start_date': datetime(2023, 1, 1),
    'retries': 1,
}


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

# TODO: Connection could be done via PythonAPI - but I didnt found HOW - so, do this in Web instead...
#file_connection = Connection(
#    conn_id="file_connection",
#    conn_type="fs",
#    description="Connection to file-path",
#)

wait_for_new_file = FileSensor(
    task_id='wait_for_new_file',
    poke_interval=10,  # Interval to check for new files (in seconds)
    filepath='/opt/airflow/data/lab2',  # Target folder to monitor
    fs_conn_id='file_connection',
    dag=dag,
)

extract_audio = DockerOperator(
    task_id='extract_audio',
    image='jrottenberg/ffmpeg',
    docker_url="tcp://docker:2375", # For Dind usage case
    mount_tmp_dir=False,
    network_mode='bridge',
    entrypoint='bash',
    command=['-c', 'cd /data/lab2 && for single_video in ./*.mp4; do ffmpeg -y -i "${single_video}" -ss 1 -to 5 -vn "./../lab2_output/${single_video}.wav"; done'],
    mounts=[
        Mount(source='/data', target='/data', type='bind'),
    ],
    dag=dag,
)

audo2text = DockerOperator(
    task_id='audio2text',
    image='huggy_face_image',
    docker_url="tcp://docker:2375", # For Dind usage case
    mount_tmp_dir=False,
    network_mode='bridge',
    entrypoint='bash',
    command=['-c', "python /data/audio2text.py"],
    mounts=[
        Mount(source='/data', target='/data', type='bind'),
    ],
    dag=dag,
)

text2summary = DockerOperator(
    task_id='text2summary',
    image='huggy_face_image',
    docker_url="tcp://docker:2375", # For Dind usage case
    mount_tmp_dir=False,
    network_mode='bridge',
    entrypoint='bash',
    command=['-c', "python /data/text2summary.py"],
    mounts=[
        Mount(source='/data', target='/data', type='bind'),
    ],
    dag=dag,
)

wait_for_new_file >> extract_audio >> audo2text >> text2summary
```

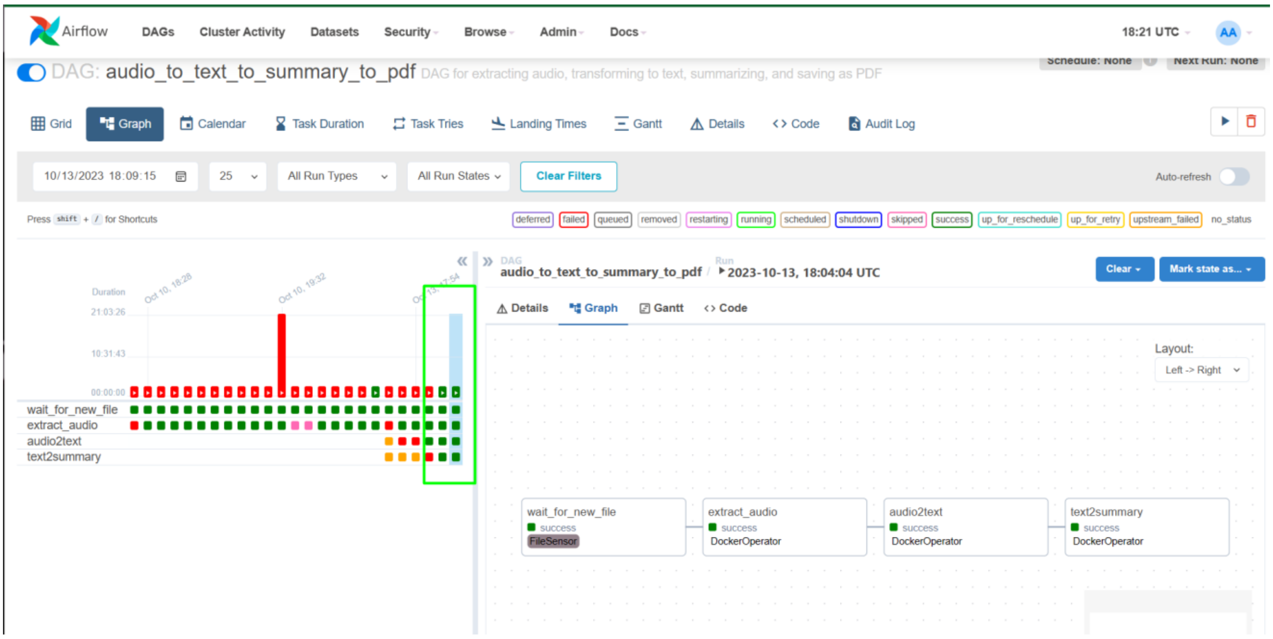
Код audio2text:

```
from transformers import pipeline
```

```python
import glob
import os

for audio_file_path in glob.glob('/data/lab2_output/*.wav'):
    _, filename = os.path.split(audio_file_path)
    output_txt_file_path = f'/data/lab2_output/text/{filename}.txt'
    if os.path.isfile(output_txt_file_path):
        continue # skip, already exist

    pipe = pipeline("automatic-speech-recognition", "openai/whisper-tiny")
    res = pipe(audio_file_path)

    with open(output_txt_file_path, "w") as text_file:
        text_file.write(res['text'])
```

Код text2summary:

```python
from transformers import pipeline
import glob
import os

for txt_file_path in glob.glob('/data/lab2_output/text/*.txt'):
    _, filename = os.path.split(txt_file_path)
    output_summary_file_path = f'/data/lab2_output/summary/{filename}'
    if os.path.isfile(output_summary_file_path):
        continue # skip, already exist

    with open(txt_file_path, 'r') as fr:
        text = fr.read()

    summarizer = pipeline("summarization", max_length=9) # Since our input small
    text_summ = summarizer(text)

    with open(output_summary_file_path, "w") as text_file:
        text_file.write(text_summ[0]['summary_text'])
```


DAG: audio_to_text_to_summary_to_pdf DAG for extracting audio, transforming to text, summarizing, and saving as PDF

Schedule: None Next Run: None

10/13/2023 18:09:15 · 25 · All Run Types · All Run States

DAG audio_to_text_to_summary_to_pdf / Run 2023-10-13, 18:04:04 UTC

Layout: Left -> Right

wait_for_new_file ■ success FileSensor

extract_audio ■ success DockerOperator

audio2text ■ success DockerOperator

text2summary ■ success DockerOperator

Обучение сети при обнаружении нового файла на примере MNIST

Для тестирования такой ситуации, я разбил обучающую выборку MNIST на 10 случайных частей. В папке airflow/data/lab2_nn_train/full_data – полные файлы данных, сама загрузка и слежка идет за папкой airflow/data/lab2_nn_train/data, куда можно закидывать или удалять файла для тестирования.

В качестве обучения взят набор данных MNIST и пару легких сверточных сетей.

Airflow/dags/airflow_train_lab2.py

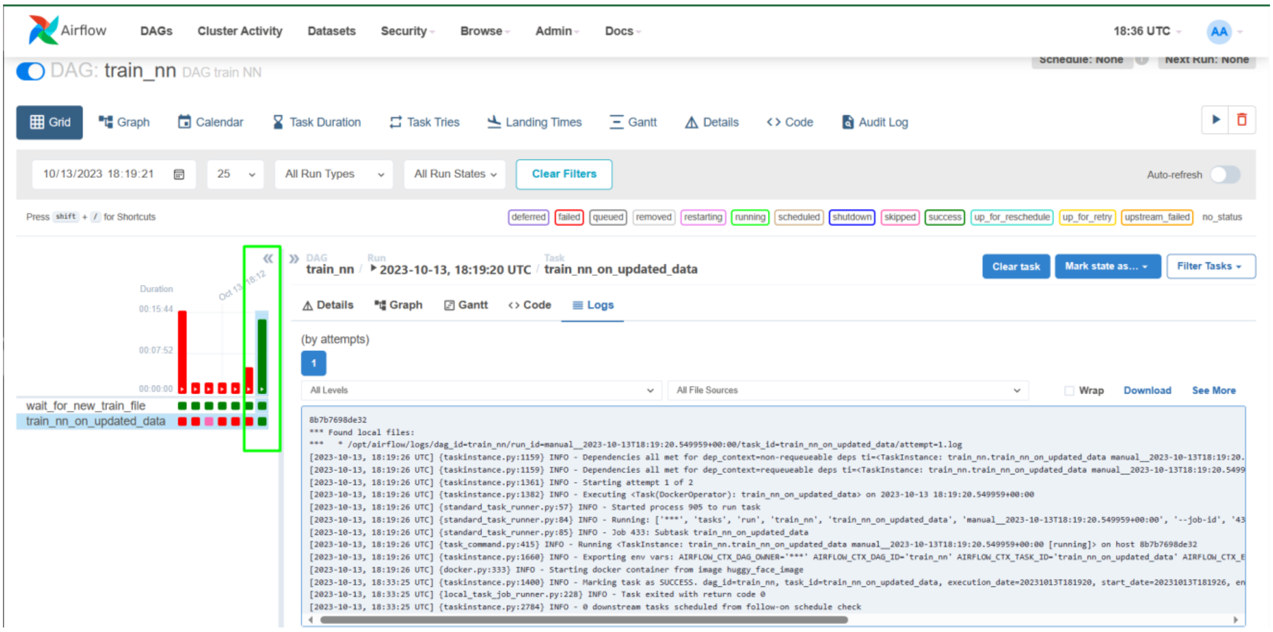
```python
import os
from datetime import datetime
from airflow import DAG
from airflow.providers.docker.operators.docker import DockerOperator
from airflow.sensors.filesystem import FileSensor

from docker.types import Mount

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


dag = DAG(
    'train_nn',
    default_args=default_args,
    description='DAG train NN',
    schedule_interval=None,
)

wait_for_new_file = FileSensor(
    task_id='wait_for_new_train_file',
    poke_interval=10,  # Interval to check for new files (in seconds)
    filepath='/opt/airflow/data/lab2_nn_train/data',  # Target folder to monitor
    fs_conn_id='file_train_connection',
    dag=dag,
)

train_nn = DockerOperator(
    task_id='train_nn_on_updated_data',
    image='huggy_face_image',
    docker_url="tcp://docker:2375", # For Dind usage case
    mount_tmp_dir=False,
    network_mode='bridge',
    entrypoint='bash',
    command=['-c', "python /data/lab2_nn_train/train_nn_mnist.py"],
    mounts=[
        Mount(source='/data', target='/data', type='bind'),
    ],
    dag=dag,
)

```

```

wait_for_new_file >> train_nn
```

Код обучения:

```
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset
from torchvision import transforms
from torch.optim.lr_scheduler import StepLR

import numpy as np
import glob
import os
import sys
from datetime import datetime

SAVE_MODEL_PATH = '/data/lab2_nn_train/models'
DATA_PATH = '/data/lab2_nn_train/data'
TEST_FILENAME = 'test_data.npz'
TRAIN_PREFIX = 'train_'


BATCH_SIZE = 32
EPOCHS = 15
LR = 1e-4
LR_STEP = 0.7
CUDA = False
SEED = 2023
LOG_INTERVAL = 10
SAVE_MODEL = True


class MnistDataset(Dataset):

    def __init__(self, x, y, transform=None):
        assert len(x) == len(y)
        self.x = x
        self.y = y
        self.transform = transform

    def __len__(self):
        return len(self.x)

    def __getitem__(self, idx):
        x_sample = self.x[idx]

        if self.transform:
            x_sample = self.transform(x_sample)

        return x_sample, self.y[idx]


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % LOG_INTERVAL == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))


def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))


def main():
    use_cuda = CUDA and torch.cuda.is_available()

    torch.manual_seed(SEED)
    exp_folder_path = os.path.join('/data/lab2_nn_train/res', str(datetime.now()))
    os.makedirs(exp_folder_path, exist_ok=True)
    sys.stdout = open(os.path.join(exp_folder_path, "print.log"), 'w')

    if use_cuda:
        device = torch.device("cuda")
    else:
        device = torch.device("cpu")

    train_kwargs = {'batch_size': BATCH_SIZE}
    test_kwargs = {'batch_size': BATCH_SIZE}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
```

```python
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])
    x_train_list = []
    y_train_list = []

    for single_train_file in glob.glob(os.path.join(DATA_PATH, f'{TRAIN_PREFIX}*.npz')):
        single_train_loaded = np.load(single_train_file)
        x_train_list.append(single_train_loaded['x_train'])
        y_train_list.append(single_train_loaded['y_train'])

    x_train_np = np.concatenate(x_train_list, axis=0)
    y_train_np = np.concatenate(y_train_list, axis=0)
    dataset_train = MnistDataset(
        x=x_train_np, y=y_train_np,
        transform=transform
    )

    test_loaded = np.load(os.path.join(DATA_PATH, TEST_FILENAME))
    dataset_test = MnistDataset(
        x=test_loaded['x_test'], y=test_loaded['y_test'],
        transform=transform
    )
    train_loader = torch.utils.data.DataLoader(dataset_train,**train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset_test, **test_kwargs)

    model = Net().to(device)
    optimizer = optim.Adadelta(model.parameters(), lr=LR)

    scheduler = StepLR(optimizer, step_size=1, gamma=LR_STEP)
    for epoch in range(1, EPOCHS + 1):
        train(model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    if SAVE_MODEL:
        torch.save(model.state_dict(), os.path.join(exp_folder_path, 'mnist_cnn.pt'))


if __name__ == '__main__':
    main()
```



DAG: train_nn DAG train NN

10/13/2023 18:19:21 · 25 · All Run Types · All Run States

wait_for_new_train_file
train_nn_on_updated_data

DAG train_nn / Run 2023-10-13, 18:19:20 UTC / Task train_nn_on_updated_data

(by attempts)

All Levels · All File Sources

```
8b7b7698de32
*** Found local files:
***   * /opt/airflow/logs/dag_id=train_nn/run_id=manual__2023-10-13T18:19:20.549959+00:00/task_id=train_nn_on_updated_data/attempt=1.log
[2023-10-13, 18:19:26 UTC] {taskinstance.py:1159} INFO - Dependencies all met for dep_context=non-requeueable deps ti=<TaskInstance: train_nn.train_nn_on_updated_data manual__2023-10-13T18:19:20.
[2023-10-13, 18:19:26 UTC] {taskinstance.py:1159} INFO - Dependencies all met for dep_context=requeueable deps ti=<TaskInstance: train_nn.train_nn_on_updated_data manual__2023-10-13T18:19:20.5499
[2023-10-13, 18:19:26 UTC] {taskinstance.py:1361} INFO - Starting attempt 1 of 2
[2023-10-13, 18:19:26 UTC] {taskinstance.py:1382} INFO - Executing <Task(DockerOperator): train_nn_on_updated_data> on 2023-10-13 18:19:20.549959+00:00
[2023-10-13, 18:19:26 UTC] {standard_task_runner.py:57} INFO - Started process 905 to run task
[2023-10-13, 18:19:26 UTC] {standard_task_runner.py:84} INFO - Running: ['***', 'tasks', 'run', 'train_nn', 'train_nn_on_updated_data', 'manual__2023-10-13T18:19:20.549959+00:00', '--job-id', '43
[2023-10-13, 18:19:26 UTC] {standard_task_runner.py:85} INFO - Job 433: Subtask train_nn_on_updated_data
[2023-10-13, 18:19:26 UTC] {task_command.py:415} INFO - Running <TaskInstance: train_nn.train_nn_on_updated_data manual__2023-10-13T18:19:20.549959+00:00 [running]> on host 8b7b7698de32
[2023-10-13, 18:19:26 UTC] {taskinstance.py:1660} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='***' AIRFLOW_CTX_DAG_ID='train_nn' AIRFLOW_CTX_TASK_ID='train_nn_on_updated_data' AIRFLOW_CTX_E
[2023-10-13, 18:19:26 UTC] {docker.py:333} INFO - Starting docker container from image huggy_face_image
[2023-10-13, 18:33:25 UTC] {taskinstance.py:1400} INFO - Marking task as SUCCESS. dag_id=train_nn, task_id=train_nn_on_updated_data, execution_date=20231013T181920, start_date=20231013T181926, en
[2023-10-13, 18:33:25 UTC] {local_task_job_runner.py:228} INFO - Task exited with return code 0
[2023-10-13, 18:33:25 UTC] {taskinstance.py:2784} INFO - 0 downstream tasks scheduled from follow-on schedule check
```