**Лабораторная работ 2. Airflow and docker.**

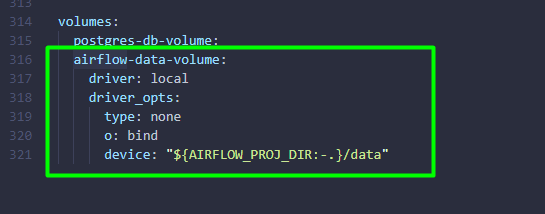
**Грибанов Данил, 6233**

**Ход работы**

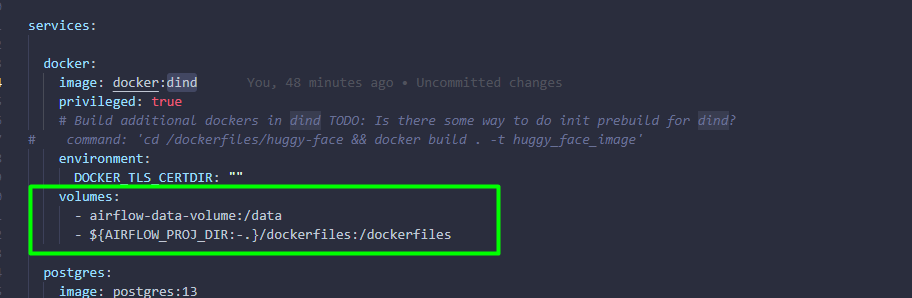
**Все файлы (dockerfile, docker-compose и прочее) находятся в папке airflow (так же dags и скрипты для запуска соотв. частей).**

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

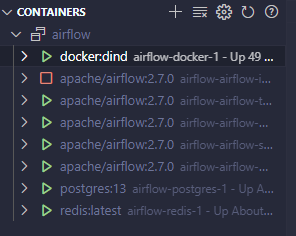
Создадим хранилище с данными для запуска и обучения всех систем:



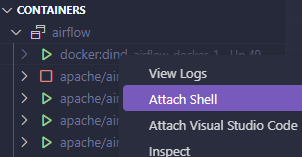
И добавим данное хранилище с папкой докер-файлов нужных в dind:



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



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

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. }

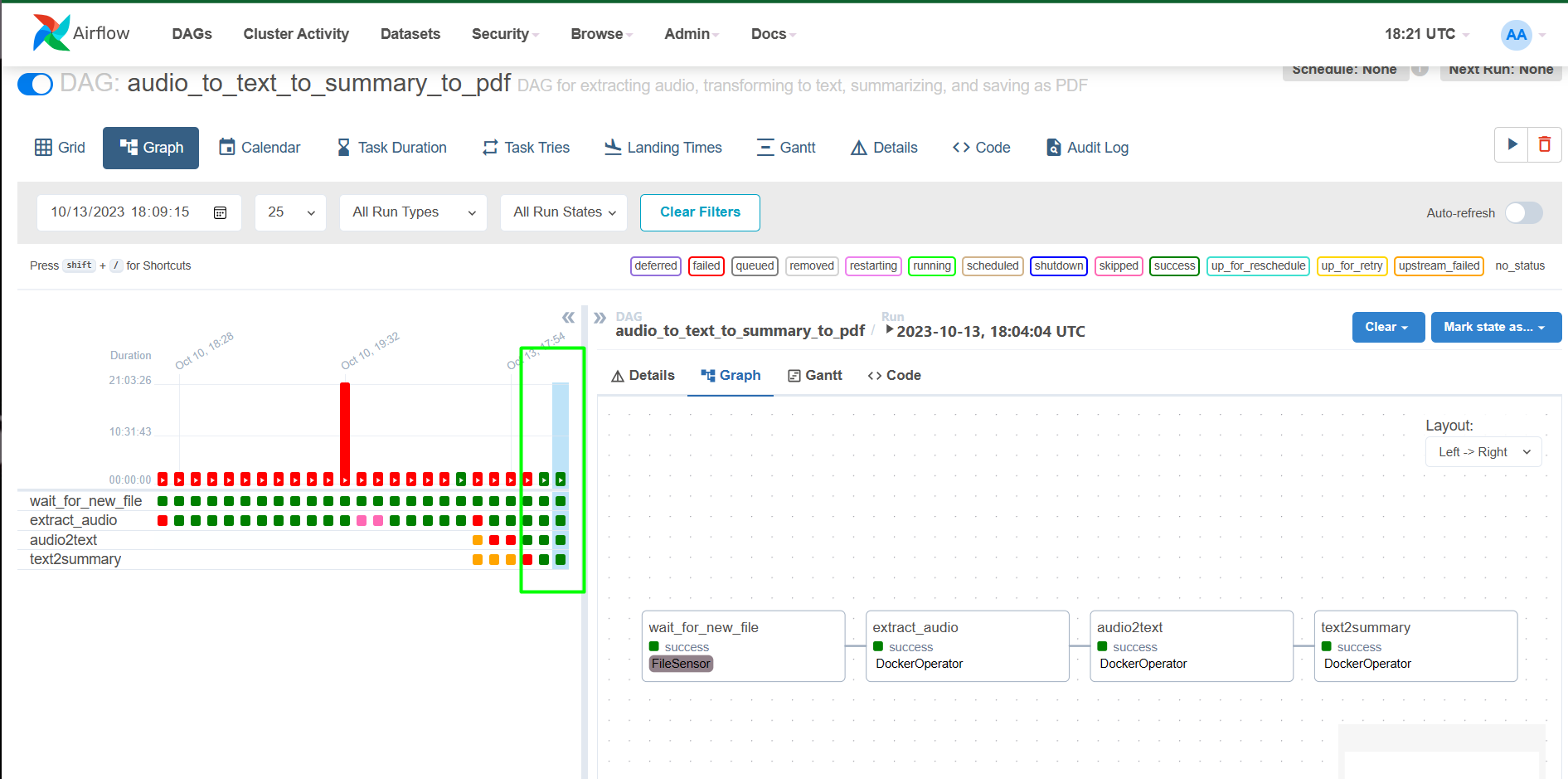
16. dag **=** DAG(
17. 'audio\_to\_text\_to\_summary\_to\_pdf',
18. default\_args**=**default\_args,
19. description**=**'DAG for extracting audio, transforming to text, summarizing, and saving as PDF',
20. schedule\_interval**=**None,
21. )
23. # TODO: Connection could be done via PythonAPI - but I didnt found HOW - so, do this in Web instead...
24. #file\_connection = Connection(
25. #    conn\_id="file\_connection",
26. #    conn\_type="fs",
27. #    description="Connection to file-path",
28. #)
30. wait\_for\_new\_file **=** FileSensor(
31. task\_id**=**'wait\_for\_new\_file',
32. poke\_interval**=**10,  # Interval to check for new files (in seconds)
33. filepath**=**'/opt/airflow/data/lab2',  # Target folder to monitor
34. fs\_conn\_id**=**'file\_connection',
35. dag**=**dag,
36. )
38. extract\_audio **=** DockerOperator(
39. task\_id**=**'extract\_audio',
40. image**=**'jrottenberg/ffmpeg',
41. docker\_url**=**"tcp://docker:2375", # For Dind usage case
42. mount\_tmp\_dir**=**False,
43. network\_mode**=**'bridge',
44. entrypoint**=**'bash',
45. 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'],
46. mounts**=**[
47. Mount(source**=**'/data', target**=**'/data', type**=**'bind'),
48. ],
49. dag**=**dag,
50. )
52. audo2text **=** DockerOperator(
53. task\_id**=**'audio2text',
54. image**=**'huggy\_face\_image',
55. docker\_url**=**"tcp://docker:2375", # For Dind usage case
56. mount\_tmp\_dir**=**False,
57. network\_mode**=**'bridge',
58. entrypoint**=**'bash',
59. command**=**['-c', "python /data/audio2text.py"],
60. mounts**=**[
61. Mount(source**=**'/data', target**=**'/data', type**=**'bind'),
62. ],
63. dag**=**dag,
64. )
66. text2summary **=** DockerOperator(
67. task\_id**=**'text2summary',
68. image**=**'huggy\_face\_image',
69. docker\_url**=**"tcp://docker:2375", # For Dind usage case
70. mount\_tmp\_dir**=**False,
71. network\_mode**=**'bridge',
72. entrypoint**=**'bash',
73. command**=**['-c', "python /data/text2summary.py"],
74. mounts**=**[
75. Mount(source**=**'/data', target**=**'/data', type**=**'bind'),
76. ],
77. dag**=**dag,
78. )
80. wait\_for\_new\_file >> extract\_audio >> audo2text >> text2summary

*Код audio2text:*

1. **from** transformers **import** pipeline
2. **import** glob
3. **import** os
5. **for** audio\_file\_path **in** glob.glob('/data/lab2\_output/\*.wav'):
6. \_, filename **=** os.path.split(audio\_file\_path)
7. output\_txt\_file\_path **=** f'/data/lab2\_output/text/{filename}.txt'
8. **if** os.path.isfile(output\_txt\_file\_path):
9. **continue** # skip, already exist
11. pipe **=** pipeline("automatic-speech-recognition", "openai/whisper-tiny")
12. res **=** pipe(audio\_file\_path)
14. with open(output\_txt\_file\_path, "w") as text\_file:
15. text\_file.write(res['text'])

*Код text2summary:*

1. **from** transformers **import** pipeline
2. **import** glob
3. **import** os
5. **for** txt\_file\_path **in** glob.glob('/data/lab2\_output/text/\*.txt'):
6. \_, filename **=** os.path.split(txt\_file\_path)
7. output\_summary\_file\_path **=** f'/data/lab2\_output/summary/{filename}'
8. **if** os.path.isfile(output\_summary\_file\_path):
9. **continue** # skip, already exist
11. with open(txt\_file\_path, 'r') as fr:
12. text **=** fr.read()
14. summarizer **=** pipeline("summarization", max\_length**=**9) # Since our input small
15. text\_summ **=** summarizer(text)
17. with open(output\_summary\_file\_path, "w") as text\_file:
18. text\_file.write(text\_summ[0]['summary\_text'])



*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

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/lab2\_nn\_train/data',  # 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\_updated\_data',
33. image**=**'huggy\_face\_image',
34. docker\_url**=**"tcp://docker:2375", # For Dind usage case
35. mount\_tmp\_dir**=**False,
36. network\_mode**=**'bridge',
37. entrypoint**=**'bash',
38. command**=**['-c', "python /data/lab2\_nn\_train/train\_nn\_mnist.py"],
39. mounts**=**[
40. Mount(source**=**'/data', target**=**'/data', type**=**'bind'),
41. ],
42. dag**=**dag,
43. )

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

Код обучения:

1. **import** argparse
2. **import** torch
3. **import** torch.nn as nn
4. **import** torch.nn.functional as F
5. **import** torch.optim as optim
6. **from** torch.utils.data **import** Dataset
7. **from** torchvision **import** transforms
8. **from** torch.optim.lr\_scheduler **import** StepLR
10. **import** numpy as np
11. **import** glob
12. **import** os
13. **import** sys
14. **from** datetime **import** datetime
16. SAVE\_MODEL\_PATH **=** '/data/lab2\_nn\_train/models'
17. DATA\_PATH **=** '/data/lab2\_nn\_train/data'
18. TEST\_FILENAME **=** 'test\_data.npz'
19. TRAIN\_PREFIX **=** 'train\_'

22. BATCH\_SIZE **=** 32
23. EPOCHS **=** 15
24. LR **=** 1e**-**4
25. LR\_STEP **=** 0.7
26. CUDA **=** False
27. SEED **=** 2023
28. LOG\_INTERVAL **=** 10
29. SAVE\_MODEL **=** True

32. **class** MnistDataset(Dataset):
34. **def** \_\_init\_\_(self, x, y, transform**=**None):
35. **assert** len(x) **==** len(y)
36. self.x **=** x
37. self.y **=** y
38. self.transform **=** transform
40. **def** \_\_len\_\_(self):
41. **return** len(self.x)
43. **def** \_\_getitem\_\_(self, idx):
44. x\_sample **=** self.x[idx]
46. **if** self.transform:
47. x\_sample **=** self.transform(x\_sample)
49. **return** x\_sample, self.y[idx]

52. **class** Net(nn.Module):
53. **def** \_\_init\_\_(self):
54. super(Net, self).\_\_init\_\_()
55. self.conv1 **=** nn.Conv2d(1, 32, 3, 1)
56. self.conv2 **=** nn.Conv2d(32, 64, 3, 1)
57. self.dropout1 **=** nn.Dropout(0.25)
58. self.dropout2 **=** nn.Dropout(0.5)
59. self.fc1 **=** nn.Linear(9216, 128)
60. self.fc2 **=** nn.Linear(128, 10)
62. **def** forward(self, x):
63. x **=** self.conv1(x)
64. x **=** F.relu(x)
65. x **=** self.conv2(x)
66. x **=** F.relu(x)
67. x **=** F.max\_pool2d(x, 2)
68. x **=** self.dropout1(x)
69. x **=** torch.flatten(x, 1)
70. x **=** self.fc1(x)
71. x **=** F.relu(x)
72. x **=** self.dropout2(x)
73. x **=** self.fc2(x)
74. output **=** F.log\_softmax(x, dim**=**1)
75. **return** output

78. **def** train(model, device, train\_loader, optimizer, epoch):
79. model.train()
80. **for** batch\_idx, (data, target) **in** enumerate(train\_loader):
81. data, target **=** data.to(device), target.to(device)
82. optimizer.zero\_grad()
83. output **=** model(data)
84. loss **=** F.nll\_loss(output, target)
85. loss.backward()
86. optimizer.step()
87. **if** batch\_idx **%** LOG\_INTERVAL **==** 0:
88. print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
89. epoch, batch\_idx **\*** len(data), len(train\_loader.dataset),
90. 100. **\*** batch\_idx **/** len(train\_loader), loss.item()))

93. **def** test(model, device, test\_loader):
94. model.eval()
95. test\_loss **=** 0
96. correct **=** 0
97. with torch.no\_grad():
98. **for** data, target **in** test\_loader:
99. data, target **=** data.to(device), target.to(device)
100. output **=** model(data)
101. test\_loss **+=** F.nll\_loss(output, target, reduction**=**'sum').item()  # sum up batch loss
102. pred **=** output.argmax(dim**=**1, keepdim**=**True)  # get the index of the max log-probability
103. correct **+=** pred.eq(target.view\_as(pred)).sum().item()
105. test\_loss **/=** len(test\_loader.dataset)
107. print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
108. test\_loss, correct, len(test\_loader.dataset),
109. 100. **\*** correct **/** len(test\_loader.dataset)))

112. **def** main():
113. use\_cuda **=** CUDA **and** torch.cuda.is\_available()
115. torch.manual\_seed(SEED)
116. exp\_folder\_path **=** os.path.join('/data/lab2\_nn\_train/res', str(datetime.now()))
117. os.makedirs(exp\_folder\_path, exist\_ok**=**True)
118. sys.stdout **=** open(os.path.join(exp\_folder\_path, "print.log"), 'w')
120. **if** use\_cuda:
121. device **=** torch.device("cuda")
122. **else**:
123. device **=** torch.device("cpu")
125. train\_kwargs **=** {'batch\_size': BATCH\_SIZE}
126. test\_kwargs **=** {'batch\_size': BATCH\_SIZE}
127. **if** use\_cuda:
128. cuda\_kwargs **=** {'num\_workers': 1,
129. 'pin\_memory': True,
130. 'shuffle': True}
131. train\_kwargs.update(cuda\_kwargs)
132. test\_kwargs.update(cuda\_kwargs)
134. transform**=**transforms.Compose([
135. transforms.ToTensor(),
136. transforms.Normalize((0.1307,), (0.3081,))
137. ])
138. x\_train\_list **=** []
139. y\_train\_list **=** []
141. **for** single\_train\_file **in** glob.glob(os.path.join(DATA\_PATH, f'{TRAIN\_PREFIX}\*.npz')):
142. single\_train\_loaded **=** np.load(single\_train\_file)
143. x\_train\_list.append(single\_train\_loaded['x\_train'])
144. y\_train\_list.append(single\_train\_loaded['y\_train'])
146. x\_train\_np **=** np.concatenate(x\_train\_list, axis**=**0)
147. y\_train\_np **=** np.concatenate(y\_train\_list, axis**=**0)
148. dataset\_train **=** MnistDataset(
149. x**=**x\_train\_np, y**=**y\_train\_np,
150. transform**=**transform
151. )
153. test\_loaded **=** np.load(os.path.join(DATA\_PATH, TEST\_FILENAME))
154. dataset\_test **=** MnistDataset(
155. x**=**test\_loaded['x\_test'], y**=**test\_loaded['y\_test'],
156. transform**=**transform
157. )
158. train\_loader **=** torch.utils.data.DataLoader(dataset\_train,**\*\***train\_kwargs)
159. test\_loader **=** torch.utils.data.DataLoader(dataset\_test, **\*\***test\_kwargs)
161. model **=** Net().to(device)
162. optimizer **=** optim.Adadelta(model.parameters(), lr**=**LR)
164. scheduler **=** StepLR(optimizer, step\_size**=**1, gamma**=**LR\_STEP)
165. **for** epoch **in** range(1, EPOCHS **+** 1):
166. train(model, device, train\_loader, optimizer, epoch)
167. test(model, device, test\_loader)
168. scheduler.step()
170. **if** SAVE\_MODEL:
171. torch.save(model.state\_dict(), os.path.join(exp\_folder\_path, 'mnist\_cnn.pt'))

174. **if** \_\_name\_\_ **==** '\_\_main\_\_':
175. main()

