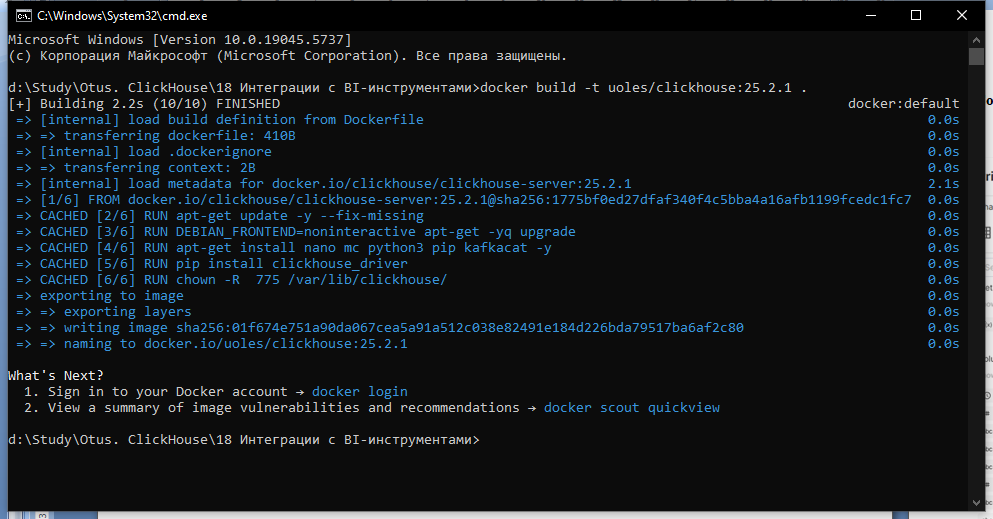
**16 ДЗ - Оркестраторы и DI Tools.**

**Собираем Dockerfile для clickhouse.**

Dockerfile:  
  
FROM clickhouse/clickhouse-server:25.2.1   
MAINTAINER Maksim Kulikov [max.uoles@rambler.ru](mailto:max.uoles@rambler.ru)  
  
RUN apt-get update -y --fix-missing   
RUN DEBIAN\_FRONTEND=noninteractive apt-get -yq upgrade  
RUN apt-get install nano mc python3 pip kafkacat –y  
RUN pip install clickhouse\_driver  
  
EXPOSE 8123 9000  
ENTRYPOINT ["/entrypoint.sh"]

Собираем образ командой:

docker build -t uoles/clickhouse:25.2.1 .

****

**Собираем docker-compose для запуска ClickHouse и Dagster.**

За основу брал статью <https://dagster.io/blog/interactive-debugging-with-dagster-and-docker>

Docker-compose.yml:

name: dagster\_clickhouse

services:

clickhouse-server:

container\_name: uoles-clickhouse-dagster-25

image: uoles/clickhouse:25.2.1

environment:

CLICKHOUSE\_DB: my\_database

CLICKHOUSE\_USER: username

CLICKHOUSE\_DEFAULT\_ACCESS\_MANAGEMENT: 1

CLICKHOUSE\_PASSWORD: password

ports:

- "18123:8123"

- "19000:9000"

ulimits:

nofile:

soft: 262144

hard: 262144

depends\_on:

- code\_location

links:

- code\_location

db:

image: postgres:16.4

environment:

POSTGRES\_USER: user

POSTGRES\_PASSWORD: password

POSTGRES\_DB: postgres

ports:

- "5432:5432"

webserver:

build:

context: .

dockerfile: .docker/webserver.Dockerfile

ports:

- "3000:3000"

env\_file:

- .env

code\_location:

build:

context: .

dockerfile: .docker/code-location.Dockerfile

env\_file:

- .env

ports:

- "5678:5678"

entrypoint: python

command:

- -m

- debugpy

- --listen

- 0.0.0.0:5678

- -m

- dagster

- code-server

- start

- -h

- 0.0.0.0

- -p

- "4000"

- -m

- defs

daemon:

build:

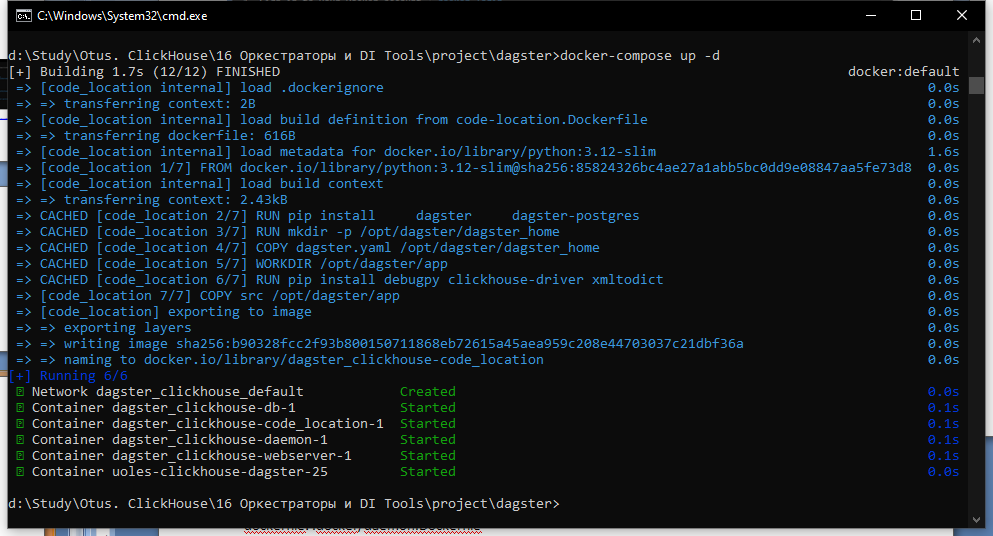
context: .

dockerfile: .docker/daemon.Dockerfile

env\_file:

- .env

Поднимаем приложения командой docker-compose up –d:



**Скрипты asset’ов.**

Скрипты лежат в файлах:

docker\dagster\src\assets.py

docker\dagster\src\defs.py

Получение данных по курсам валют:

@asset()

def asset\_load\_xml(context: AssetExecutionContext) -> dict:

url = "https://cbr.ru/scripts/XML\_daily.asp?date\_req=01/01/2020.xml"

data = '{}'

with requests.get(url) as response:

data = xmltodict.parse(response.content)

context.log.info(f"Output data is: {data}")

context.add\_output\_metadata({"valute": data})

return data

Отправка данных в clickhouse:

@asset(deps=[asset\_load\_xml])

def asset\_add\_to\_clickhouse(context: AssetExecutionContext, asset\_load\_xml: dict):

data = asset\_load\_xml

context.log.info(f"Prepared data is: {data["ValCurs"]["@name"]}")

client = Client(

host='clickhouse-server',

port=9000,

user='username',

password='password',

database='my\_database'

)

create\_table\_query = """

CREATE TABLE IF NOT EXISTS valute (

c\_date String NOT NULL,

c\_name String NOT NULL,

c\_str\_id String NOT NULL,

c\_num\_code String NOT NULL,

c\_char\_code String NOT NULL,

c\_nominal String NOT NULL,

c\_value String NOT NULL

) ENGINE = MergeTree()

ORDER BY (c\_date)

"""

client.execute(create\_table\_query)

context.log.info("Table exists")

c\_date = data["ValCurs"]["@Date"]

select\_query = "SELECT 1 FROM valute WHERE c\_date = %(c\_date)s"

result = client.execute(select\_query, {'c\_date': c\_date})

if len(result) == 0:

insert\_query = "INSERT INTO valute (c\_date, c\_name, c\_str\_id, c\_num\_code, c\_char\_code, c\_nominal, c\_value) VALUES"

insert\_data = [

(data["ValCurs"]["@Date"], item["Name"], item["@ID"], item["NumCode"], item["CharCode"], item["Nominal"], item["Value"])

for item in data["ValCurs"]["Valute"]

]

client.execute(insert\_query, insert\_data)

context.log.info("Data inserted")

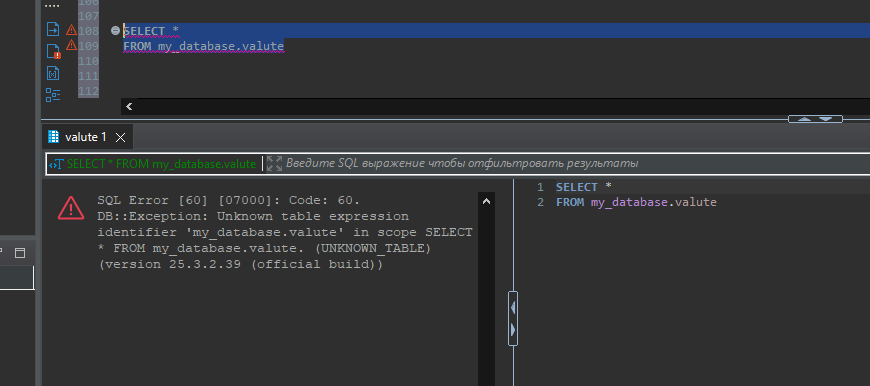
else:

context.log.info("Data already loaded")

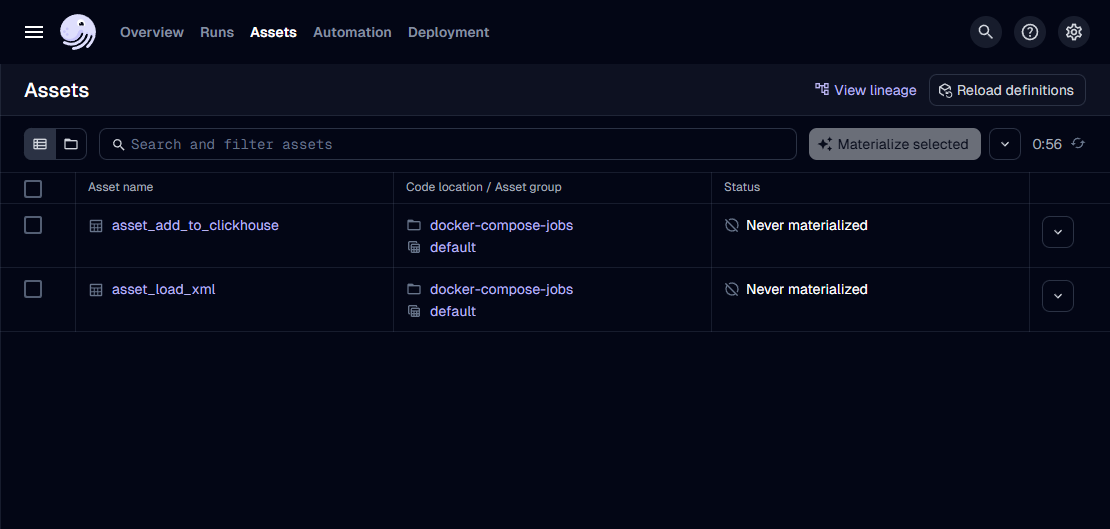
Скрипт отправки данных сначала создает таблицу, если ее нет. Потом проверяет наличие данных по дате из XML. Если даты нет, то добавляет данные.

**Запуск скриптов.**

Подключаемся к базе и проверяем, что таблицы нет:



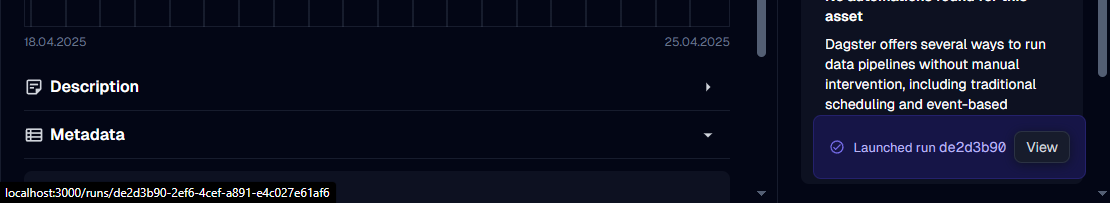
Заходим на страницу dagster, на вкладку со скриптами - <http://localhost:3000/assets> .



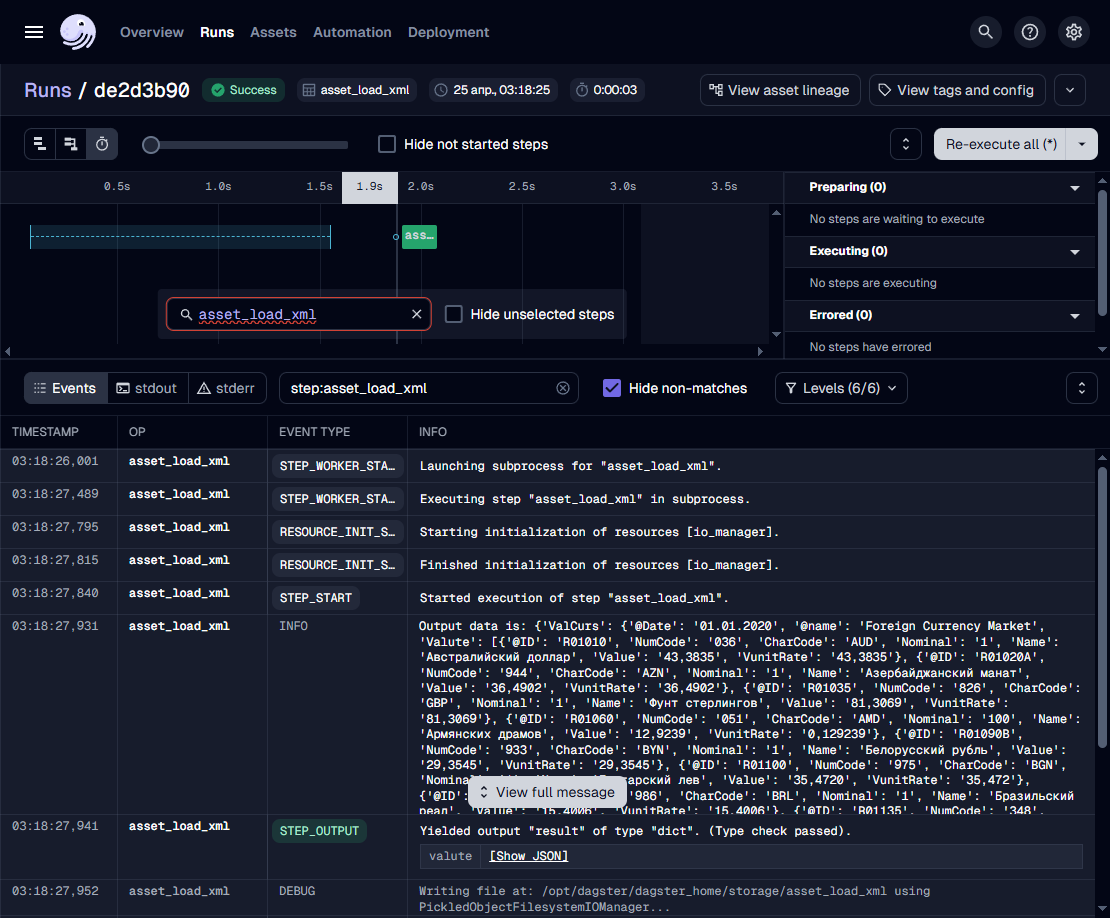
Заходим сначала в скрипт «asset\_load\_xml»:



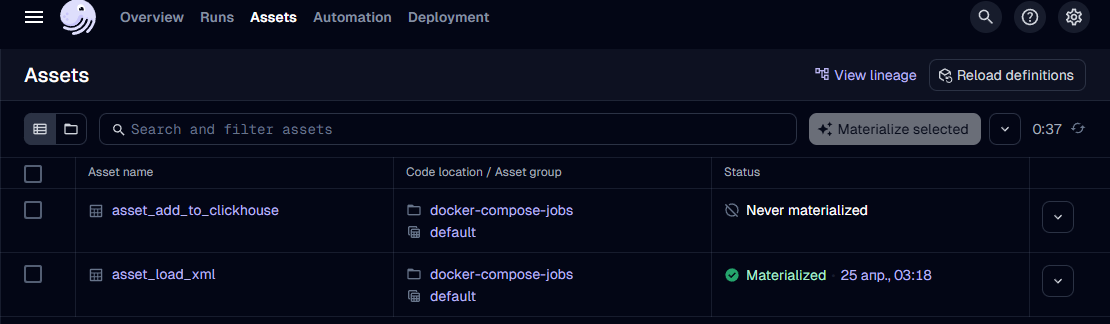
и жмем кнопку «Materialize» для запуска скрипта.



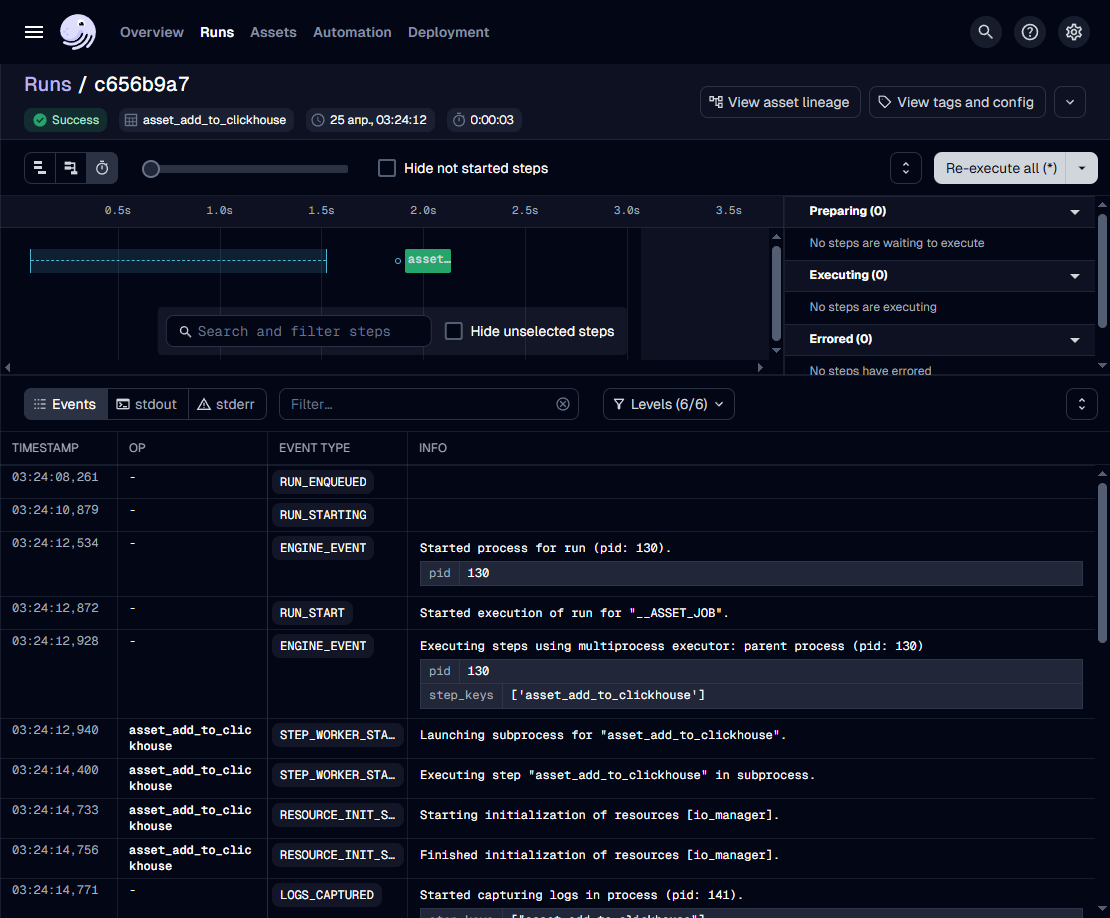
Появляется сообщение о запуске процесса. Жмем кнопу «View» и переходим к логам выполнения.



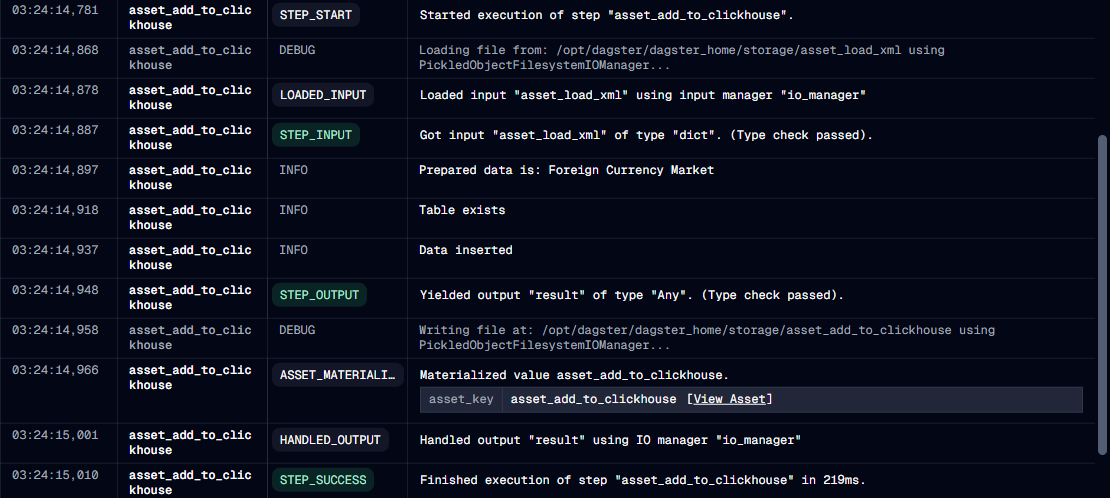
Видно, что скрипт отработал успешно и данные подтянулись. Далее возвращаемся к списку скриптов и запускаем так же второй скрипт «asset\_add\_to\_clickhouse» для отправки данных в clickhouse.



Можно запустить скрипты так же и с вкладки «Assets».



Скрипт отработал успешно.



Проверяем наличие данных в clickhouse:

