Отчет по лабораторной работе №1

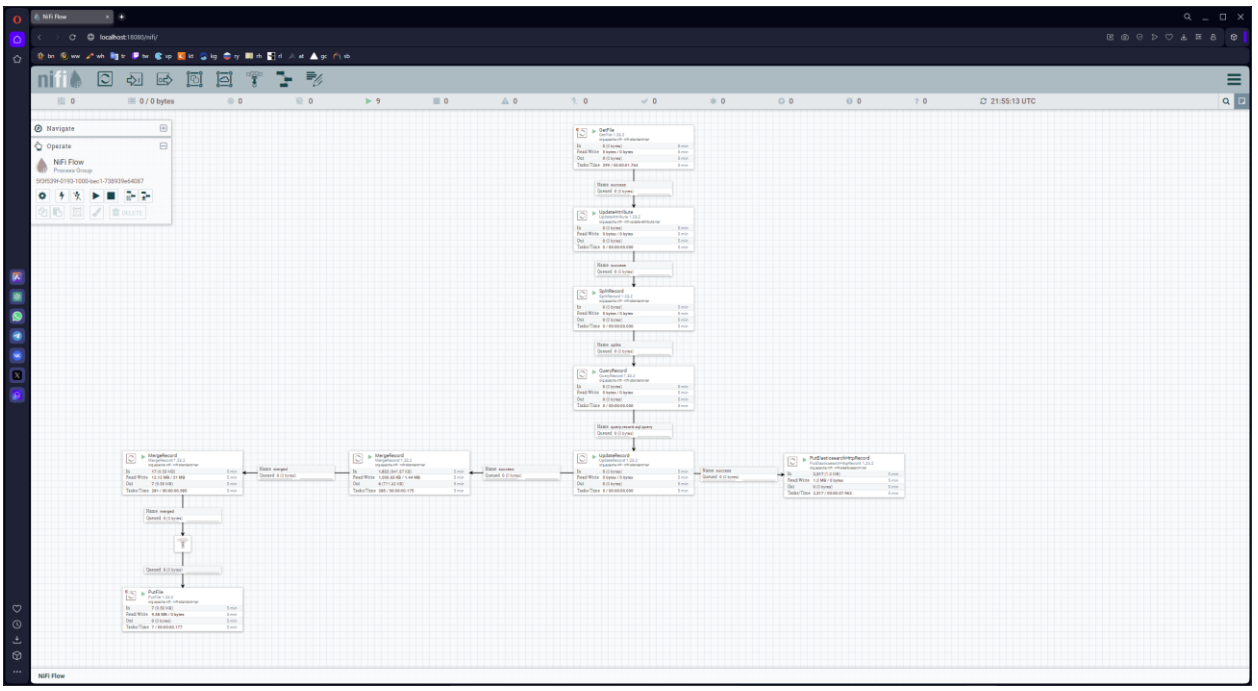
Выполнила Агафонова Елизавета, группа 6233

Apache Nifi

Для реализации пайплайна были сконфигурированы процессоры:

- GetFile – процессор для чтения файлов. В конфигурации настроена директория с файлами (data/lr1/in) и паттерн для поиска файлов ([^\.].*\.csv)
- UpdateAttribute – процессор для переименовывания имени файла. Необходим чтобы в конце был один файл
- SplitRecord – процессор для разделения файлов на строки и обеспечения дальнейшей построчной работы
- QueryRecord – процессор для фильтрации строк (SELECT * FROM FLOWFILE WHERE designation IS NOT NULL AND region_1 IS NOT NULL)
- UpdateRecord – процессор для замены всех null на 0.0 в price. В конфигурацию добавлен параметр /price со значением ${field.value:replaceNull(0.0)}
- MergeRecord (происходит в два этапа) – процессор для объединения. В первом этапе объединяем в крупные блоки. Во втором этапе объединяем все в один файл.
- PutFile – процессор для сохранения итогового файла в заданную директорию (data/lr1/out)
- PutElasticsearchHttpRecord – процессор для сохранения строк в Elasticsearch

Общая схема всего пайплайна представлена на рисунке.



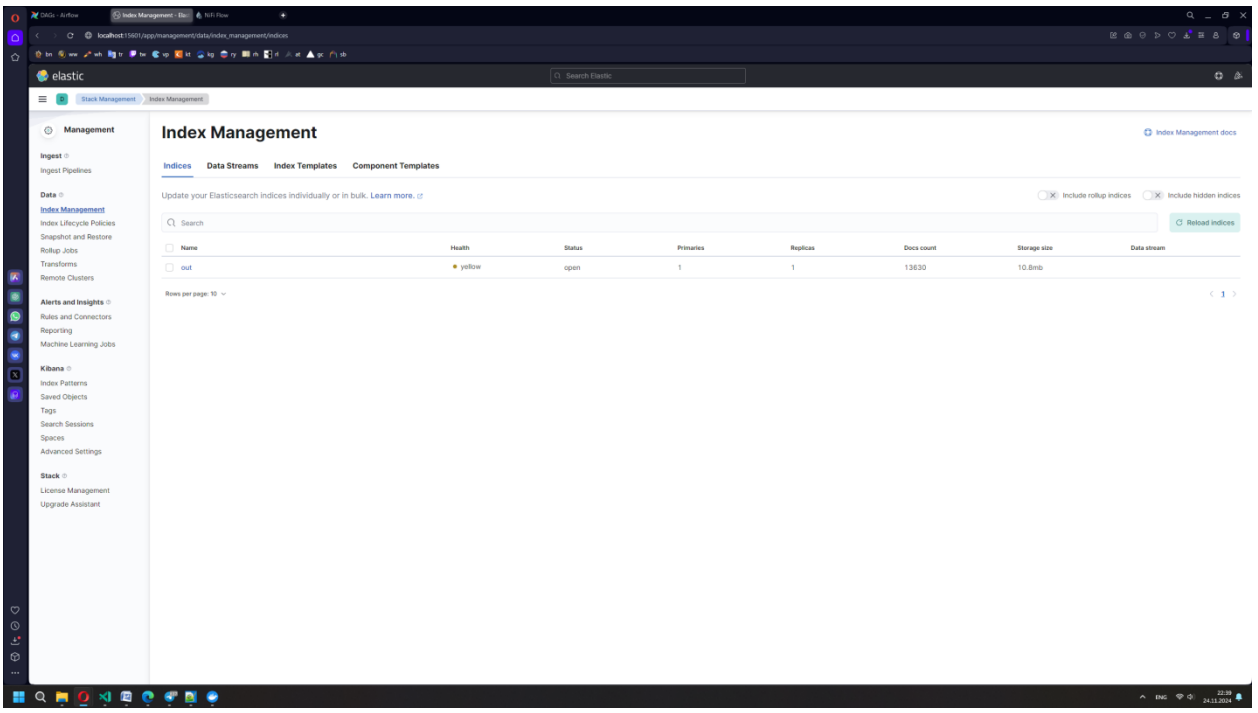
С полной конфигурацией пайплайна можно ознакомиться в файле lr1_nifi.xml.

При запуске возникла проблема с недостатком памяти (OutOfMemoryError: Java heap space). Решением оказалось изменение параметров в файле bootstrap.conf.

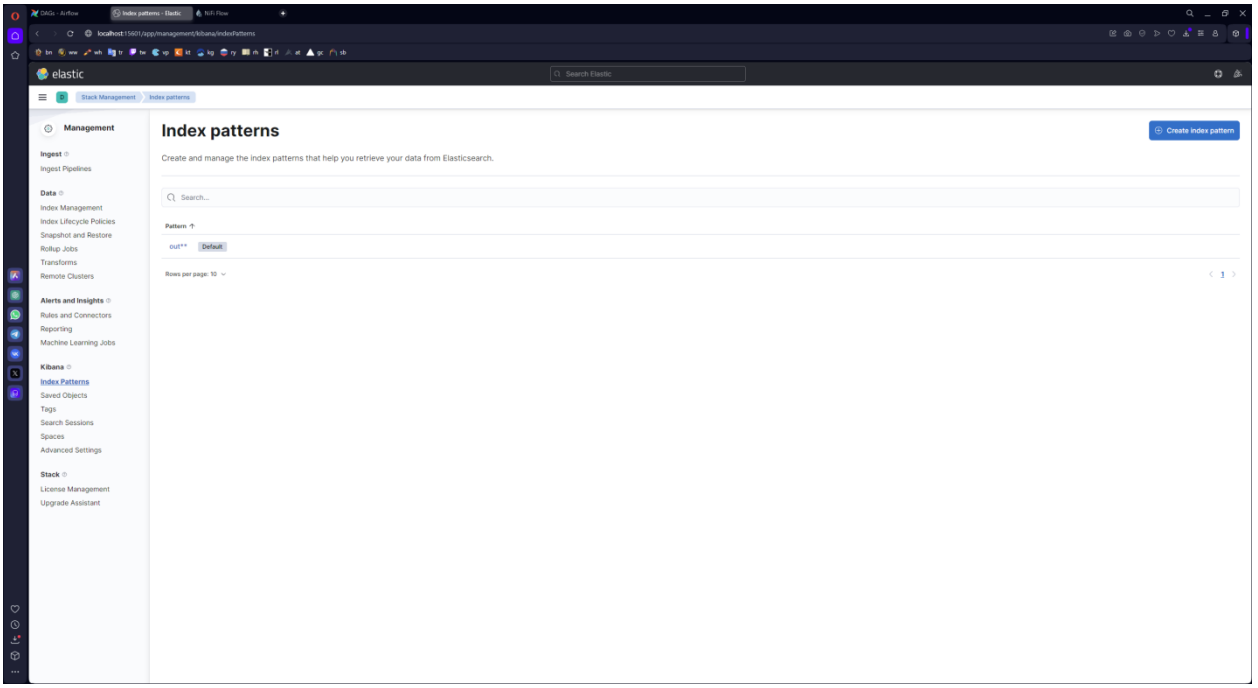
/opt/nifi/nifi-current/conf/bootstrap.conf

```
# Configure where NiFi's lib and conf directories live
lib.dir=./lib
conf.dir=./conf

# How long to wait after telling NiFi to shutdown before explicitly killing the Process
graceful.shutdown.seconds=20

# Disable JSR 199 so that we can use JSP's without running a JDK
java.arg.1=-Dorg.apache.jasper.compiler.disablejsr199=true

# JVM memory settings
java.arg.2=-Xms1024m
java.arg.3=-Xmx1024m

# Enable Remote Debugging
#java.arg.debug=-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8000

java.arg.4=-Djava.net.preferIPv4Stack=true

# allowRestrictedHeaders is required for Cluster/Node communications to work properly
java.arg.5=-Dsun.net.http.allowRestrictedHeaders=true
java.arg.6=-Djava.protocol.handler.pkgs=sun.net.www.protocol

```

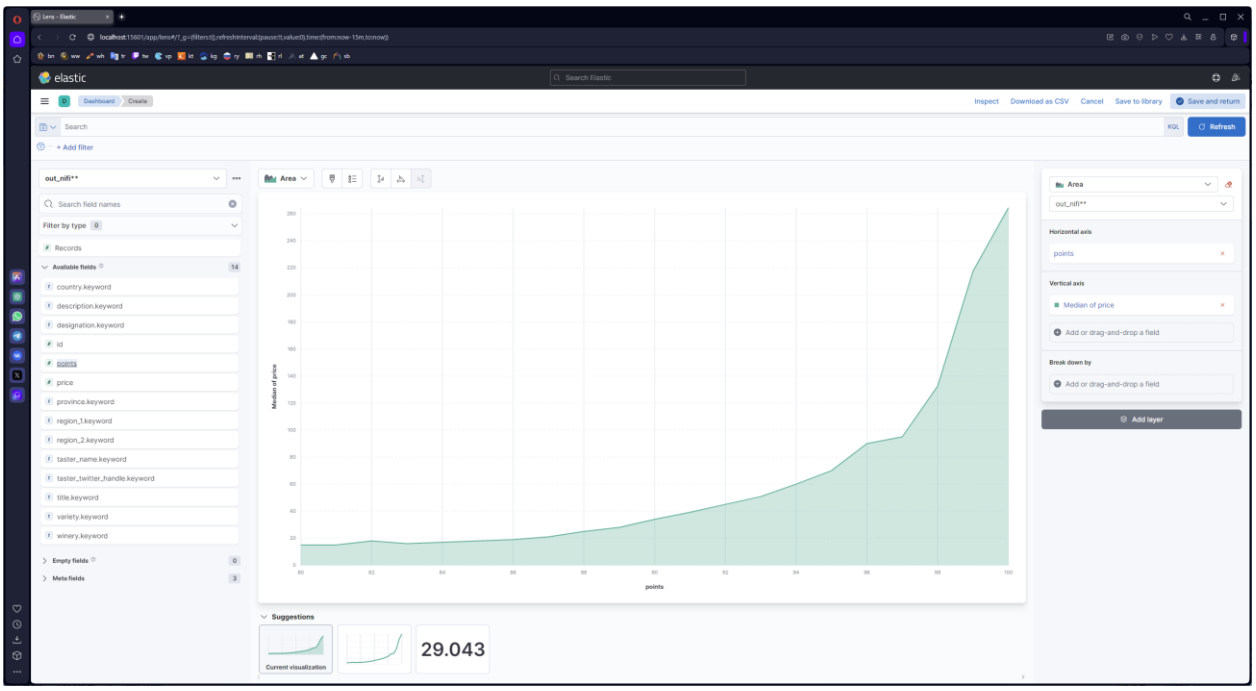
Проверим наши данные



Создадим Index patterns



На основе полученных данных построим график. Передаем на вертикальную ось поле price на горизонтальную points.



Apache airflow

DAG состоит из tasks:

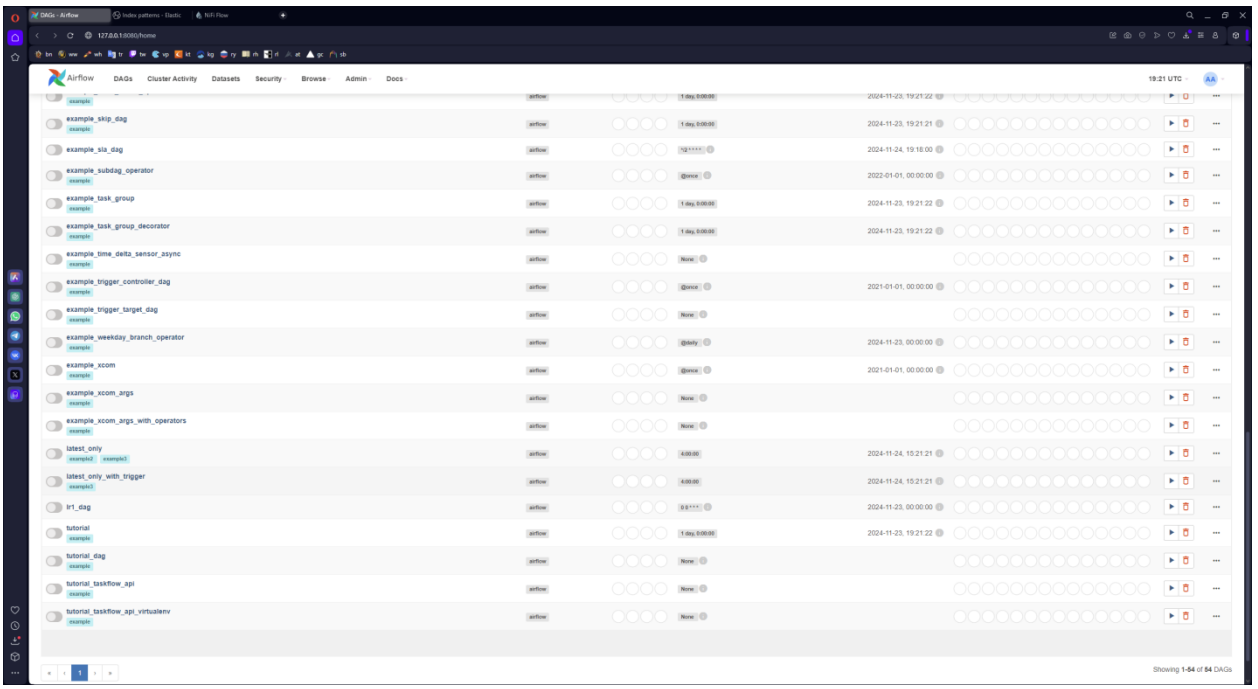
- read_files_task – считывает все csv-файлы и сохраняет их в один датафрейм
- filter_task – удаляет все строки у которых designation или region_1 null, результат сохраняет в файл buffer.csv
- fill_task – заменяет все значения null в price на 0.0, результат сохраняет в файл buffer.csv
- save_csv_task – сохраняет результат в итоговый csv-файл
- save_es_task – сохраняет в Elacticsearch. Каждая строка датафрейма преобразуется в json

В конце все tasks объединяются в DAG по схеме:

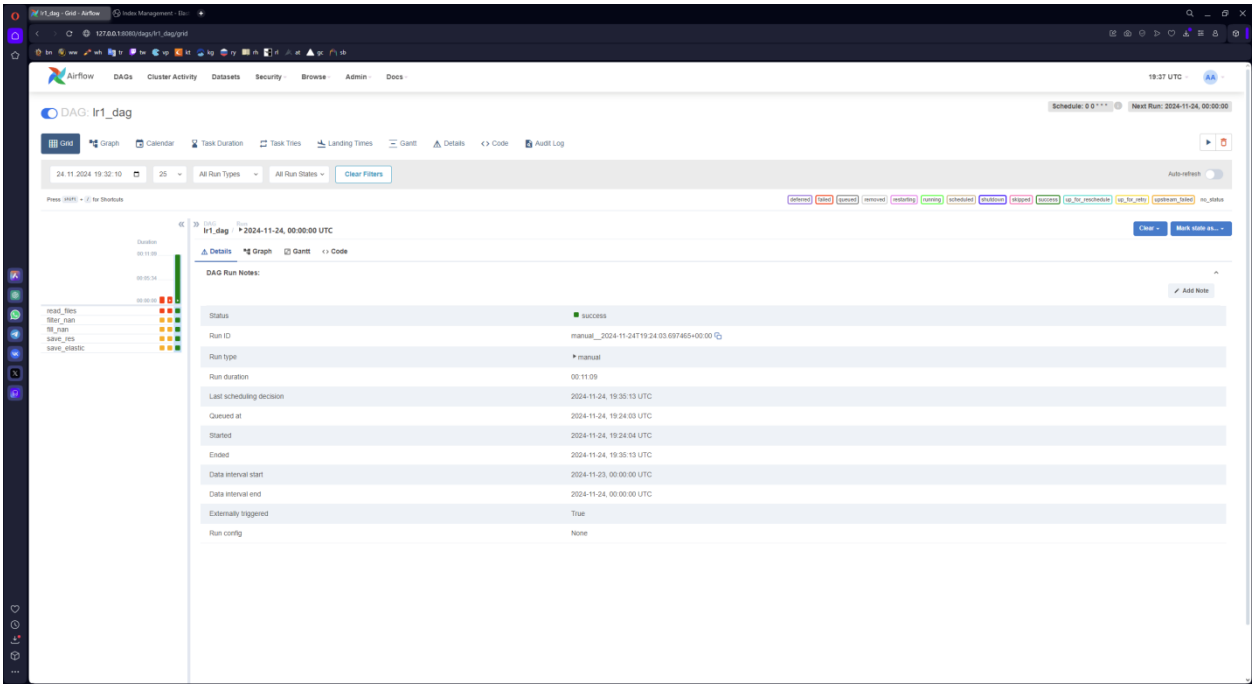
read_files_task >> filter_task >> fill_task >> [save_csv_task, save_es_task]

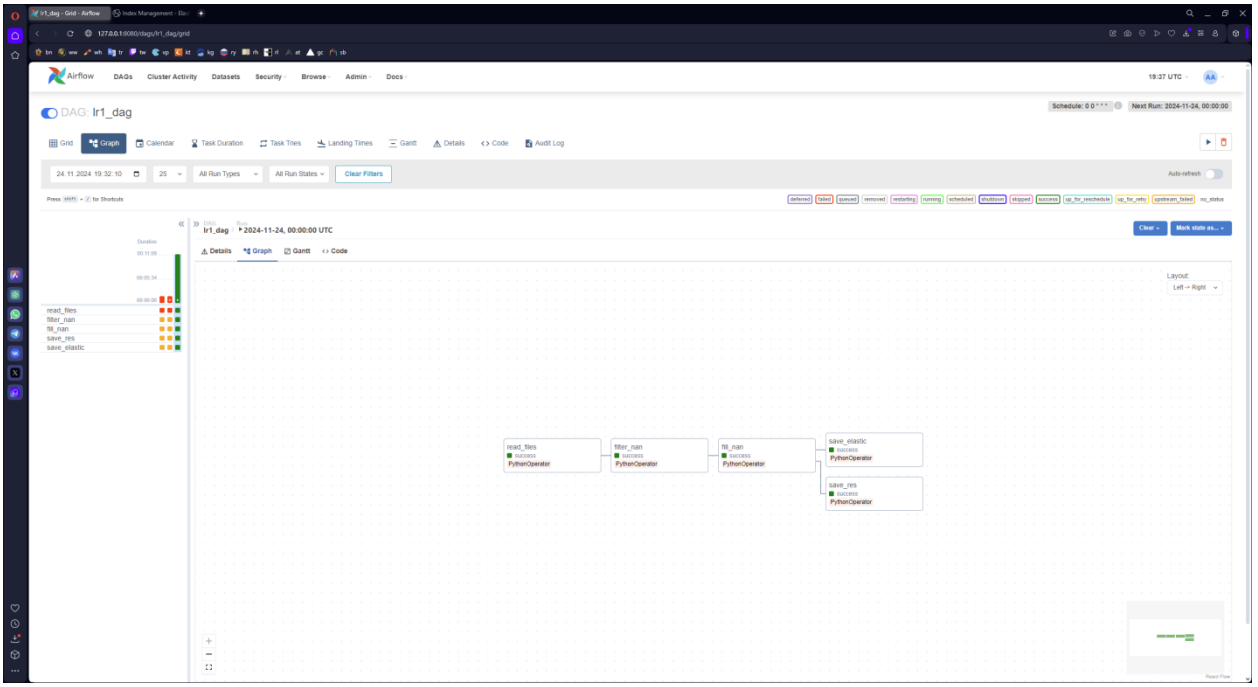
С программным кодом реализующим DAG можно ознакомиться в файле main.py.

DAG lr1_dag

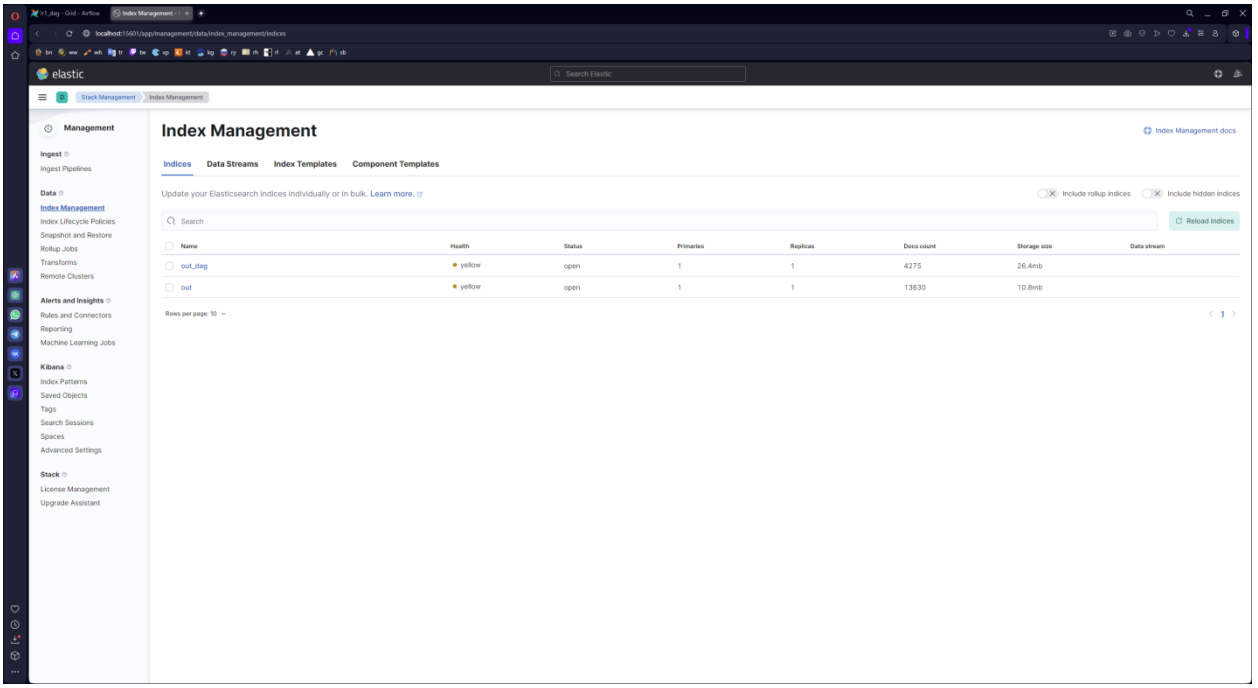


Success на весь DAG после окончания запуска

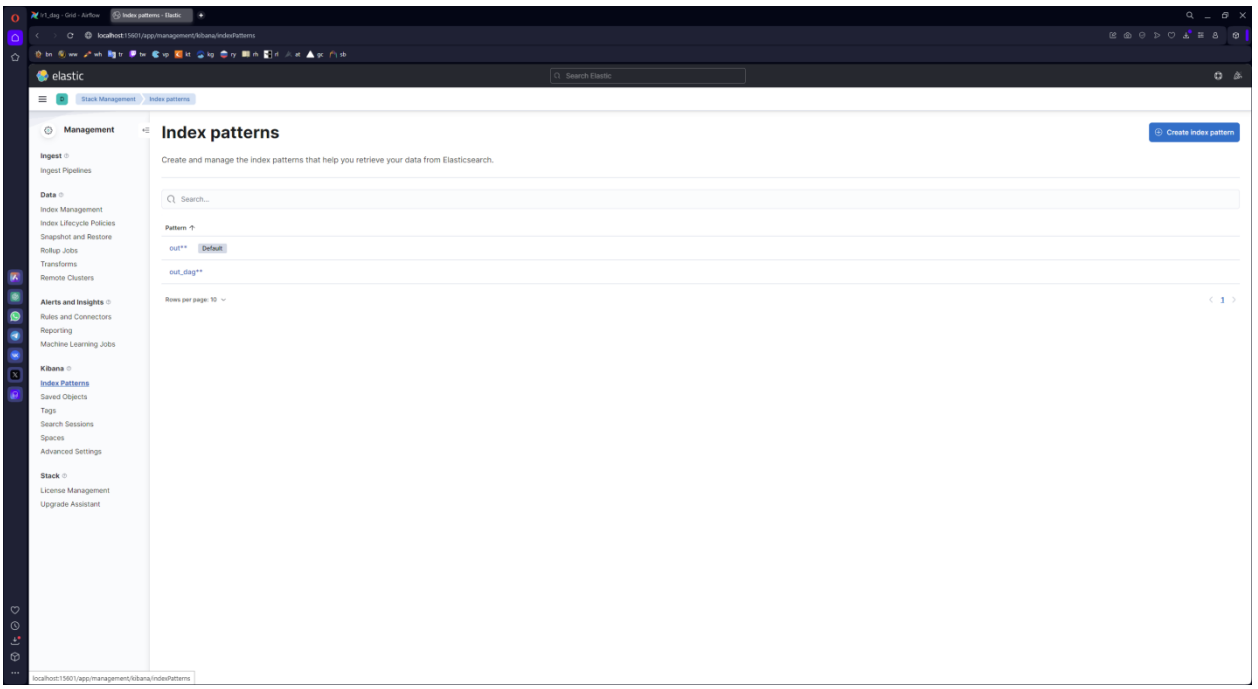




Проверим наши данные



Создадим Index patterns



На основе полученных данных построим график. Передаем на вертикальную ось поле price на горизонтальную points.