Корноушкин Илья 6131

Ход выполнения лабораторной работы

Нам нужны следующие инструменты -

- Apache Airflow
- Apache NiFi
- Elasticsearch

Apache Airflow

В lab1.py реализует DAG, который сначала считывает и обрабатывает данные из определённой директории. Сохраняет все данные в едином файле. И после пересылает данные в ElasticSearch.

```
def read_and_transf_data():
 data = pd.DataFrame()
 for i in range(26):
     data = pd.concat([data, pd.read_csv(f"/opt/airflow/data/chunk{i}.csv")])

 data = data[(data['designation'].str.len() > 0) & (data['region_1'].str.len() > 0)]
 data['price'] = data['price'].replace(np.nan, 0)
 data = data.drop(['id'], axis=1)
 data.to_csv('/opt/airflow/data/data.csv', index=False)

read_and_transf_task = PythonOperator(

     task_id='read_and_transf_task',

     python_callable=read_and_transf_data,

     dag=dag)
```

Рисунок 1 – чтение и обработка данных

```python
def load_to_elastic():
    es = Elasticsearch("http://elasticsearch-kibana:9200")
    data = pd.read_csv(f"/opt/airflow/data/data.csv")

    for i, row in data.iterrows():
        doc = {
        "country": row["country"],
        "description": row["description"],
        "designation": row["designation"],
        "points": row["points"],
        "price": row["price"],
        "province": row["province"],
        "region_1": row["region_1"],
        "region_2": row["region_1"],
        "taster_name": row["taster_name"],
        "taster_twitter_handle": row["taster_twitter_handle"],
        "title": row["title"],
        "variety": row["variety"],
        "winery": row["winery"],
    }

    if i < data.shape[0] - 1:
        es.index(index="wines", id=i, document=doc)

load_to_elastic_task = PythonOperator(

    task_id='load_to_elastic_task',

    python_callable=load_to_elastic,

    dag=dag)
```

Рисунок 2 – пересылка данных в ElasticSearch

Далее указываем порядок выполнение DAG.

```python
read_and_transf_task >> load_to_elastic_task
```

APACHE NIFI

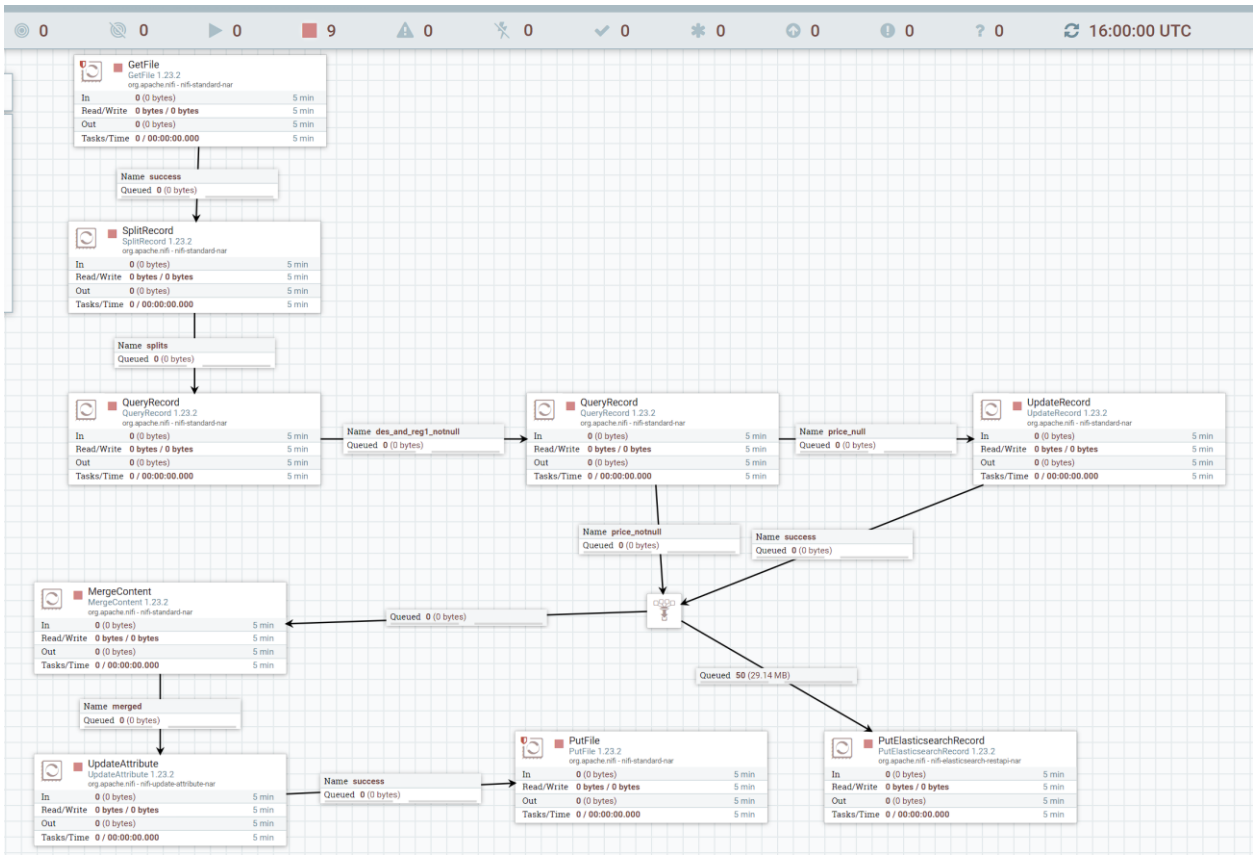


Рисунок 3 – схема NiFi

Делаем тоже что и в AirFlow. GetFile – считывает файлы. SplitRecord делит данные из файлов на батчи. QueryRecord – первая проверяет чтобы поля designation и region_1 не были пустыми, а вторая разделяет строки с price == null и price != null. UpdateRecord заменяет все значения price == null на price == 0.0. Funnel собирает значения вместе. MergeContent собирает из всех файлов один файл. UpdateAttribute заменяет название получившегося файла на output.csv (Иначе файл имеет название одного из исходных файлов). PutFile сохраняет получившийся файл в туже директорию. PutElasticSearchRecord пересылает данные в ElasticSearch.

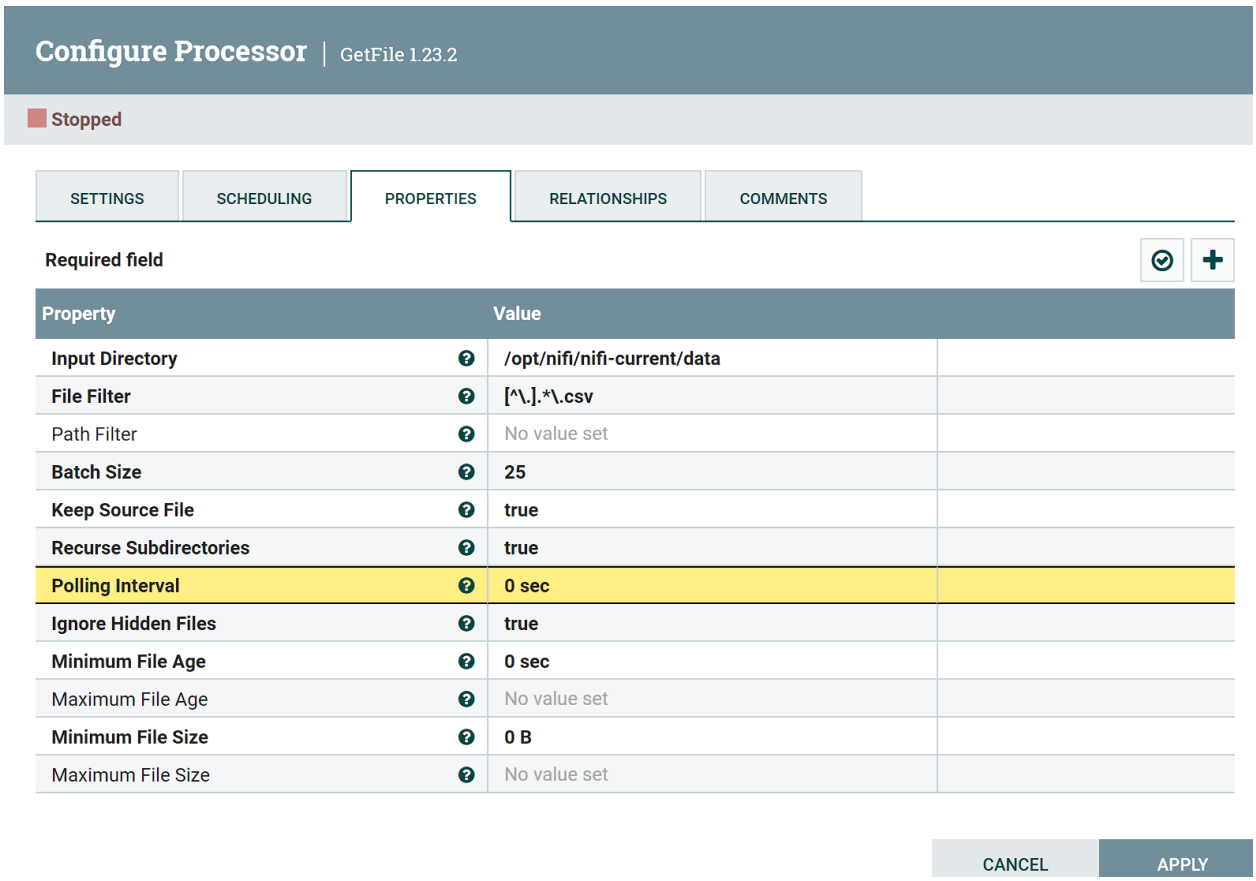


Рисунок 4 – Настройки GetFile

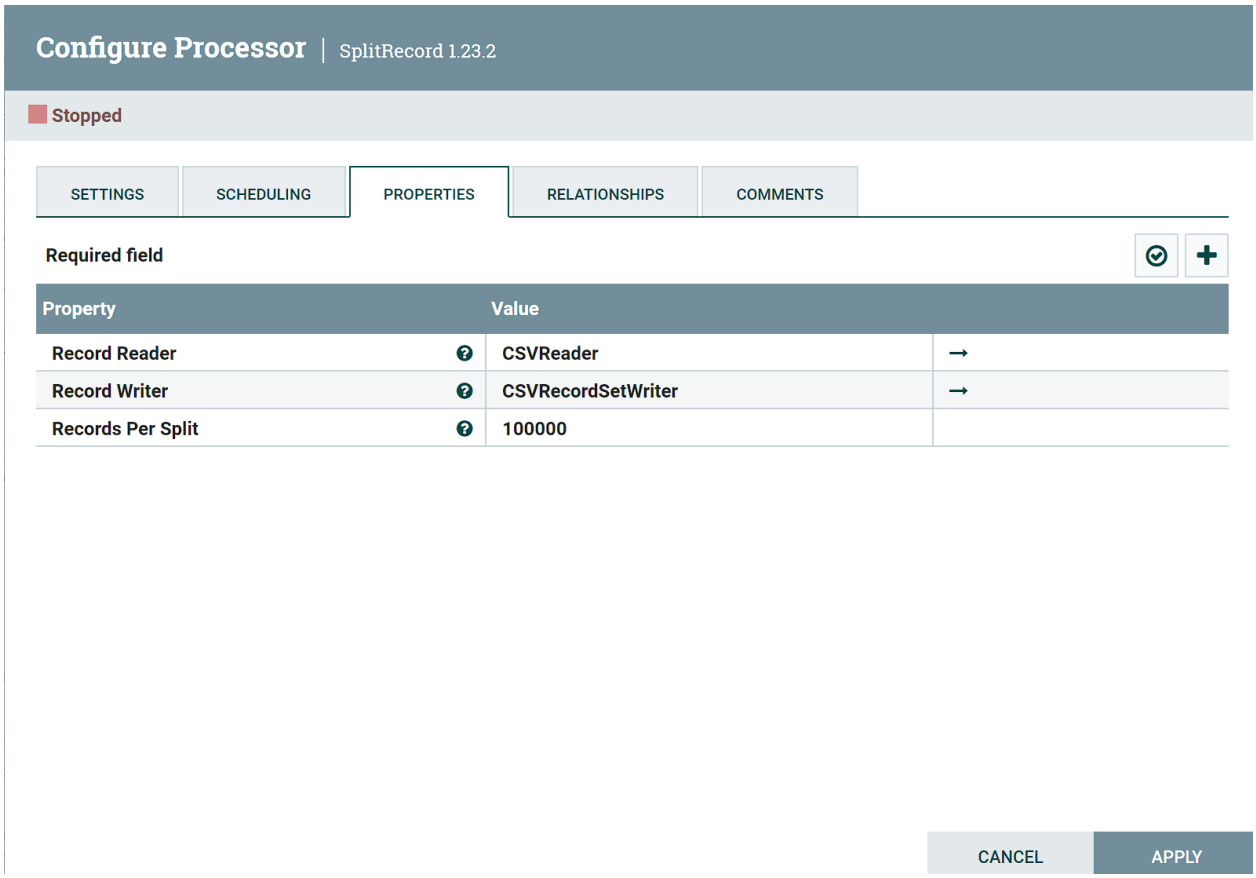


Рисунок 5 – Настройки SplitRecord

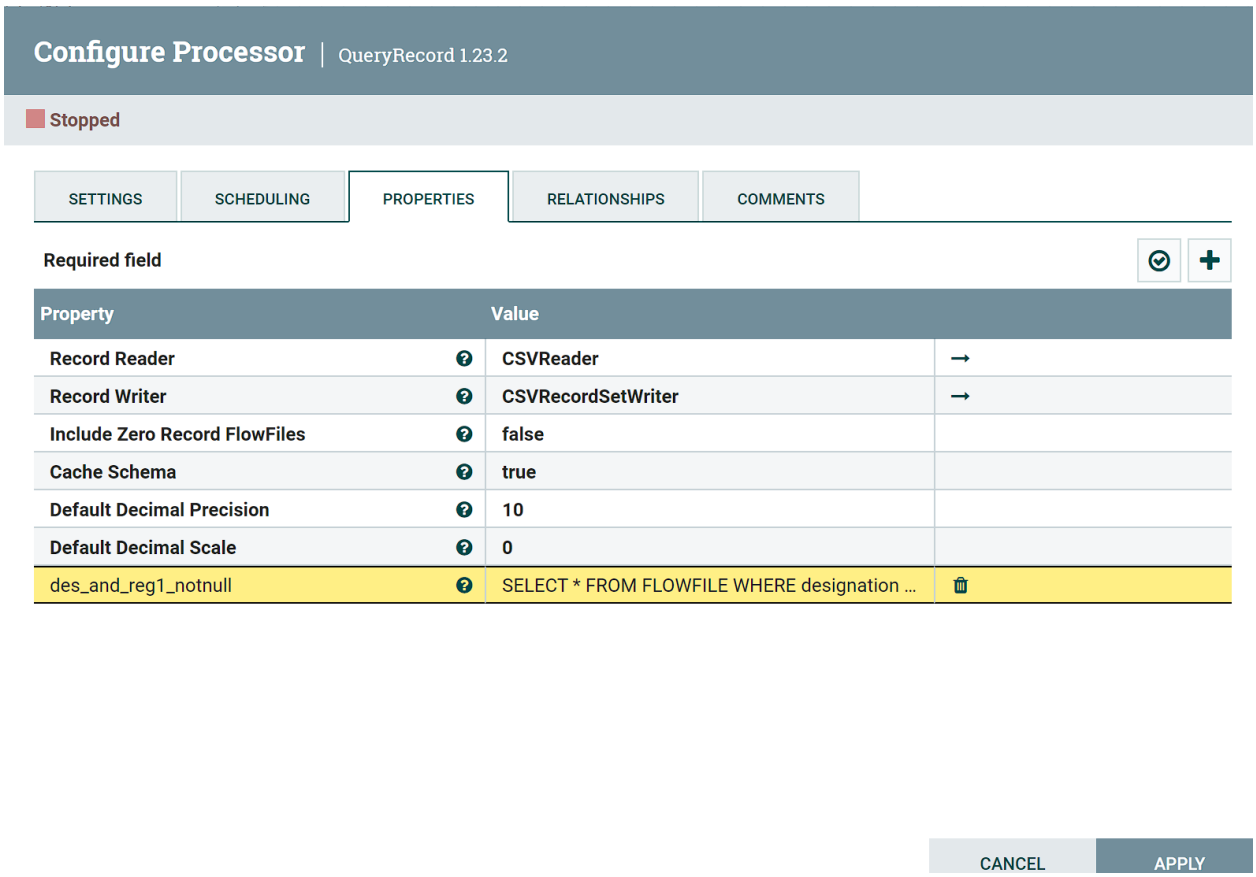


Рисунок 6 – Настройки QueryRecord

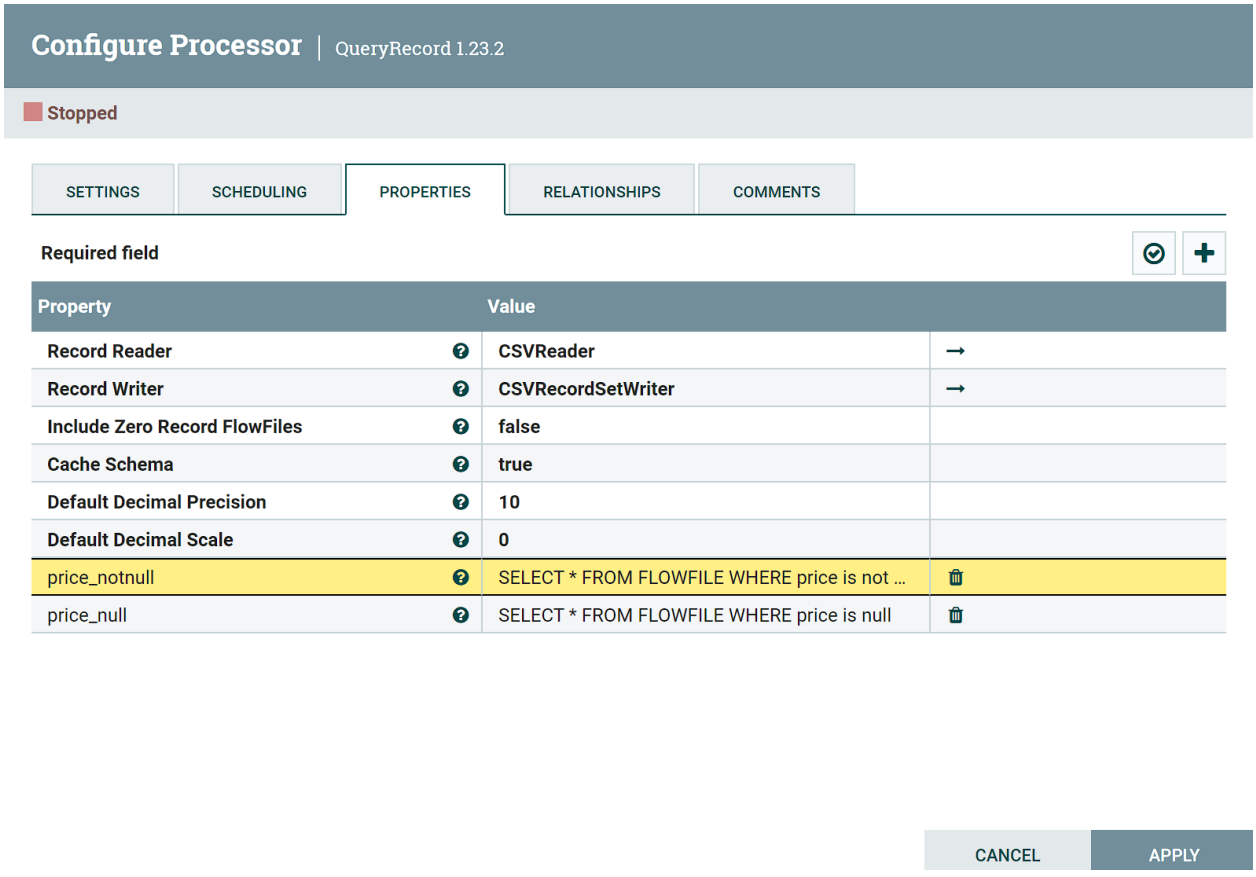


Рисунок 7 – Настройки QueryRecord

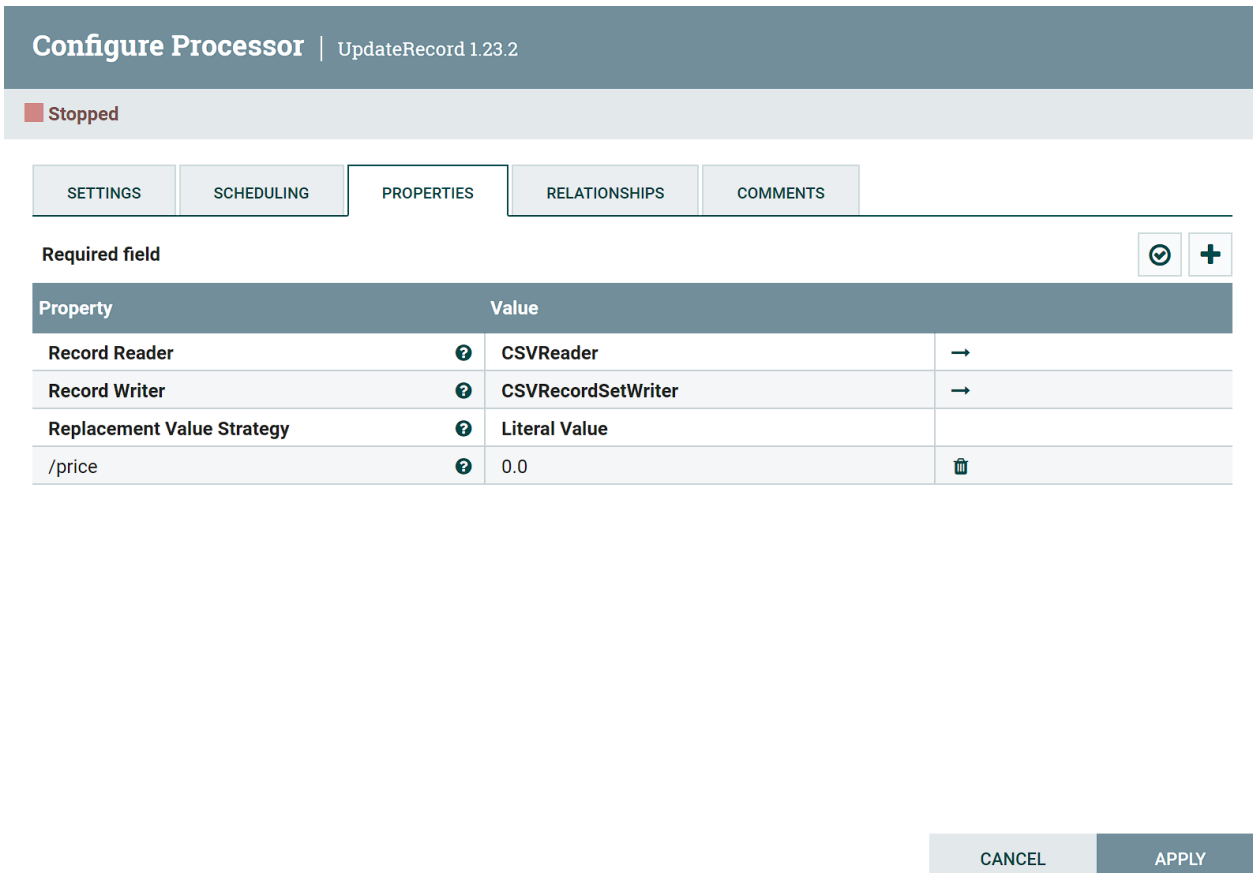


Рисунок 8 – Настройки UpdateRecord

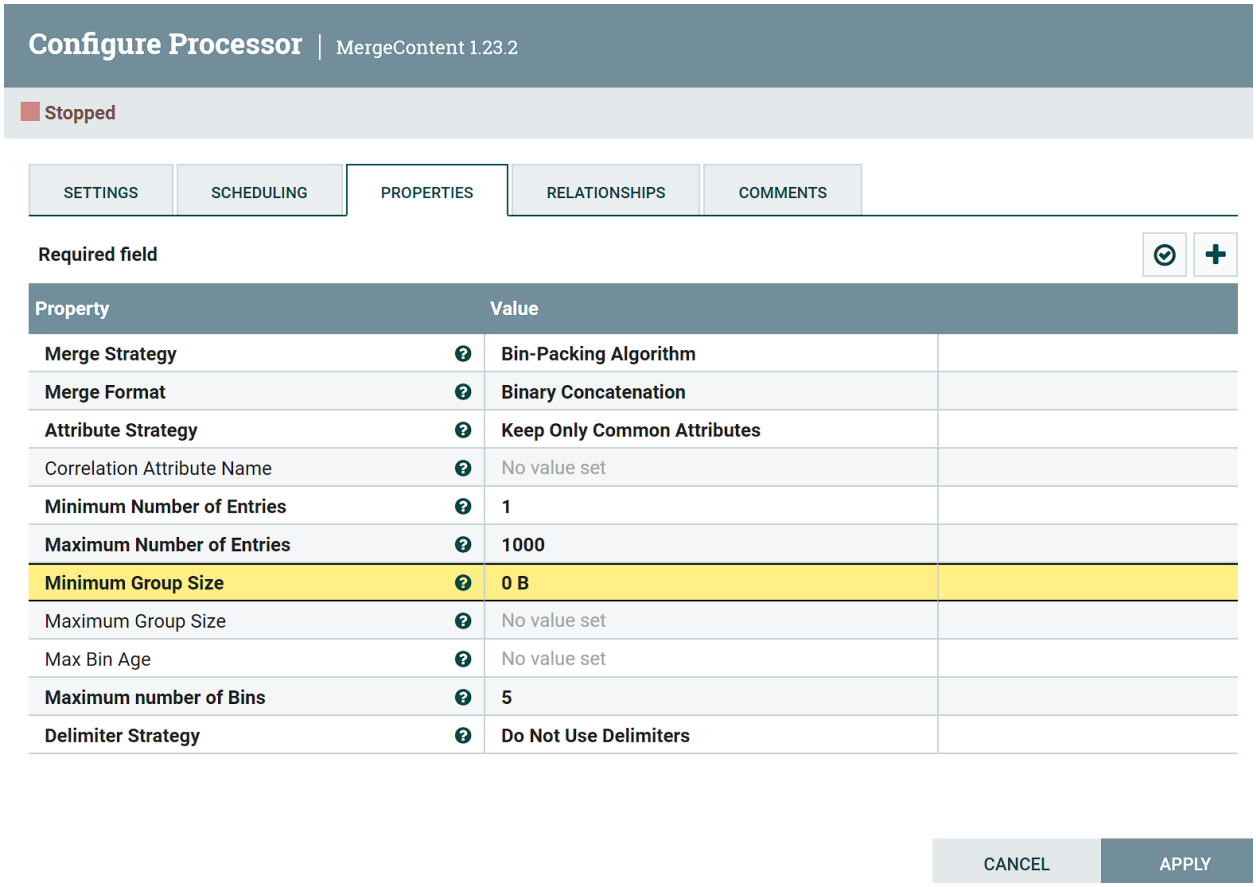


Рисунок 9 – Настройки MergeContent

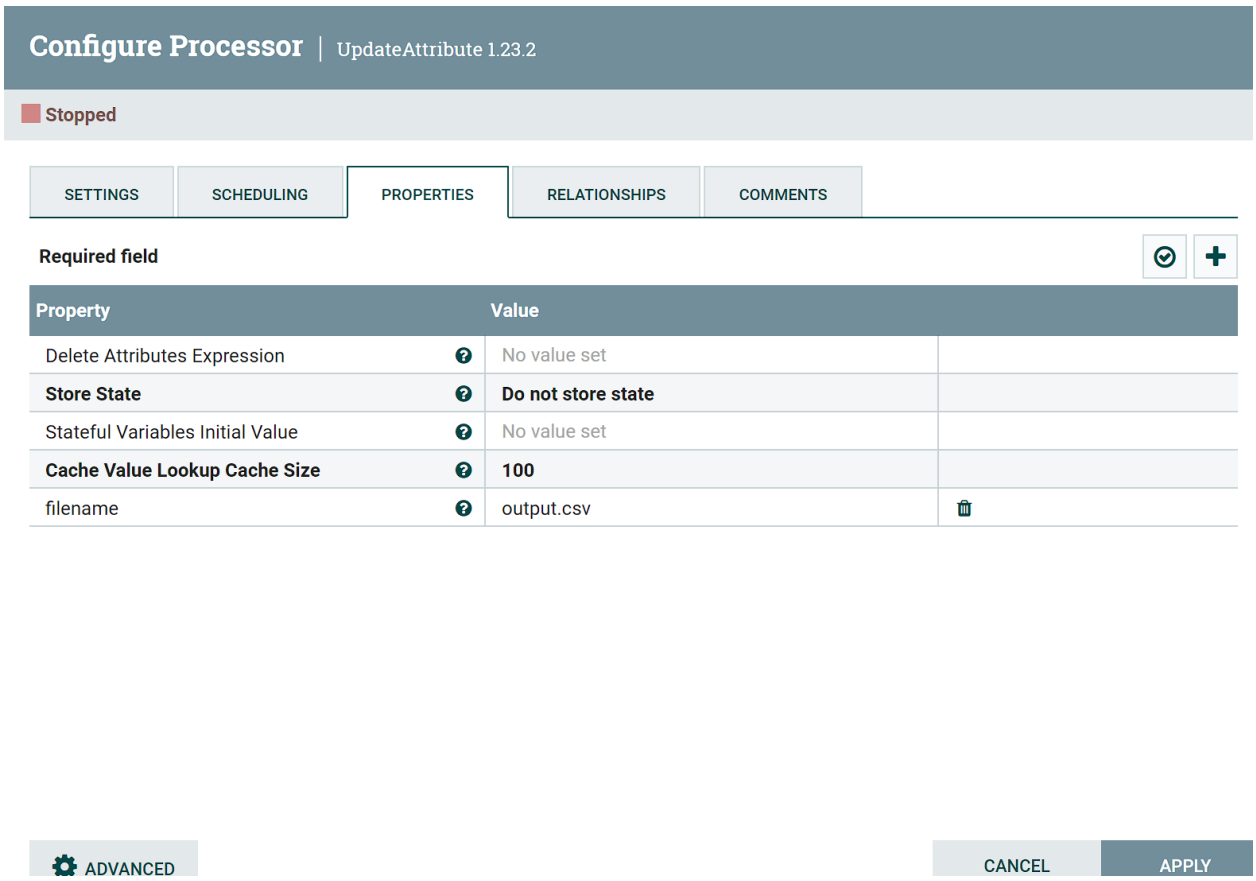


Рисунок 10 – Настройки UpdateAttribute

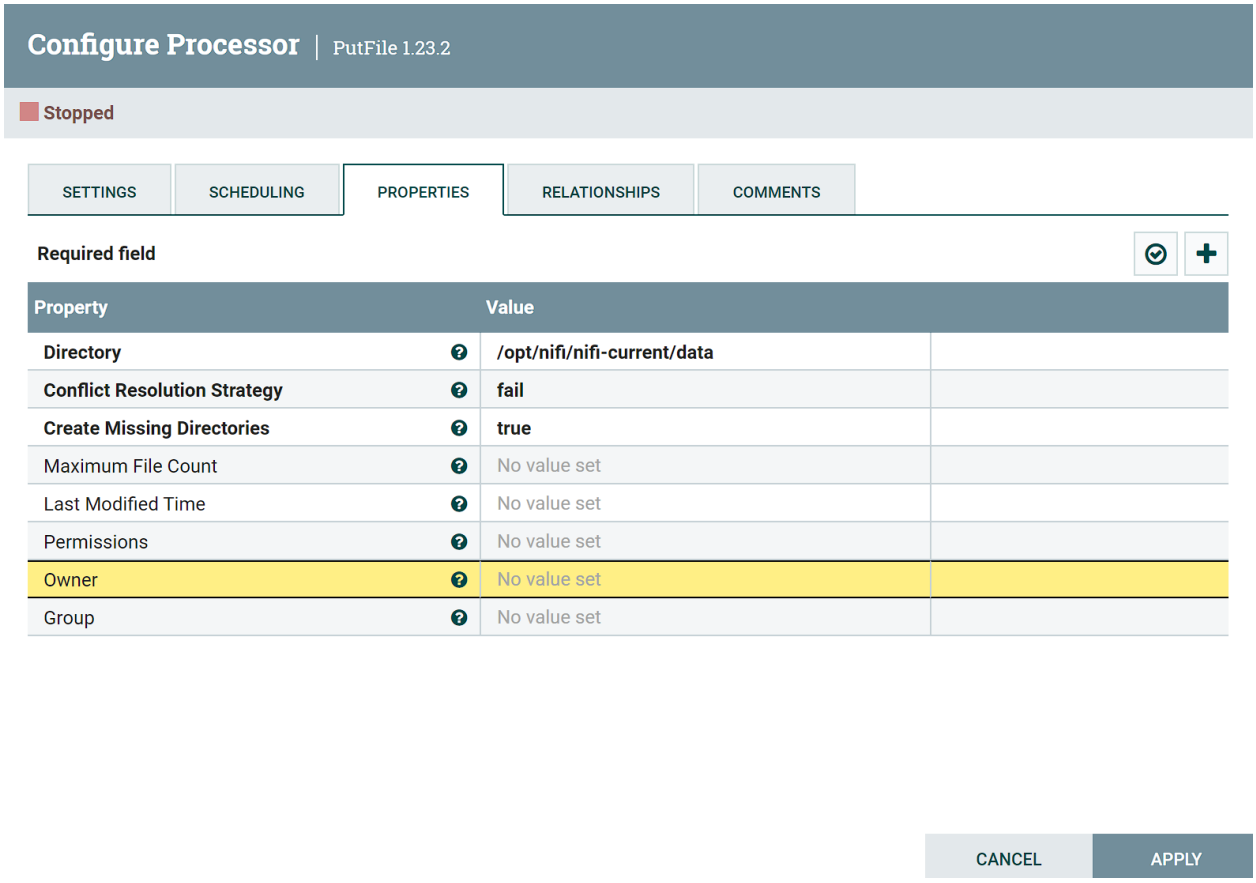


Рисунок 11 – Настройки PutFile

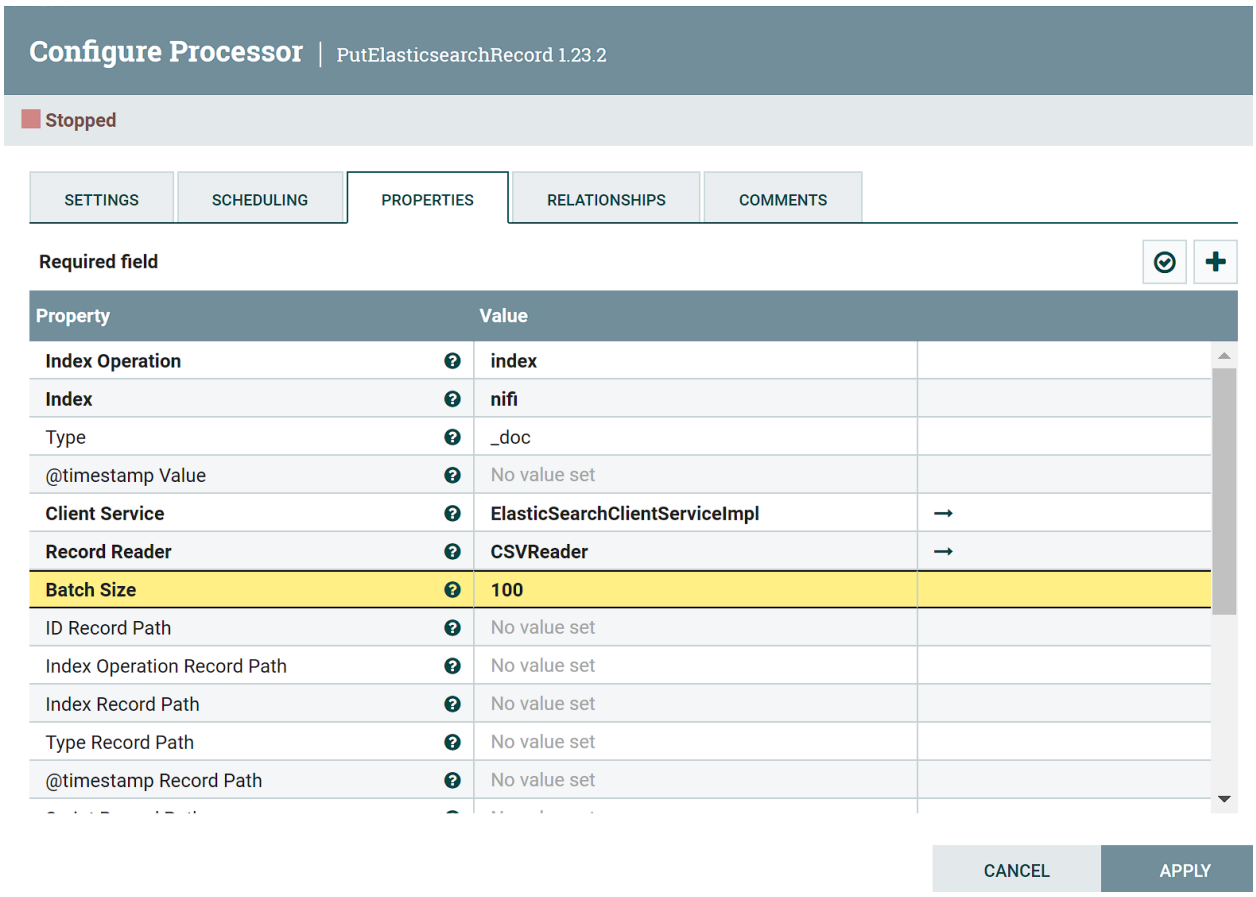
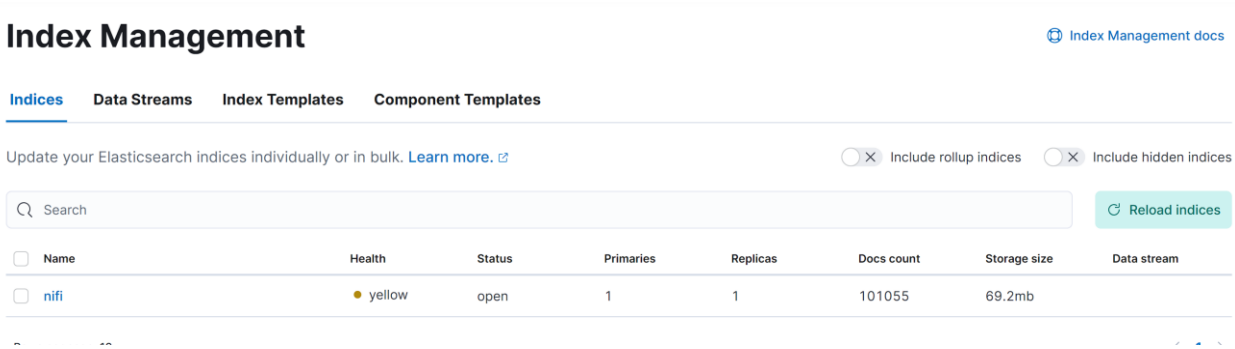


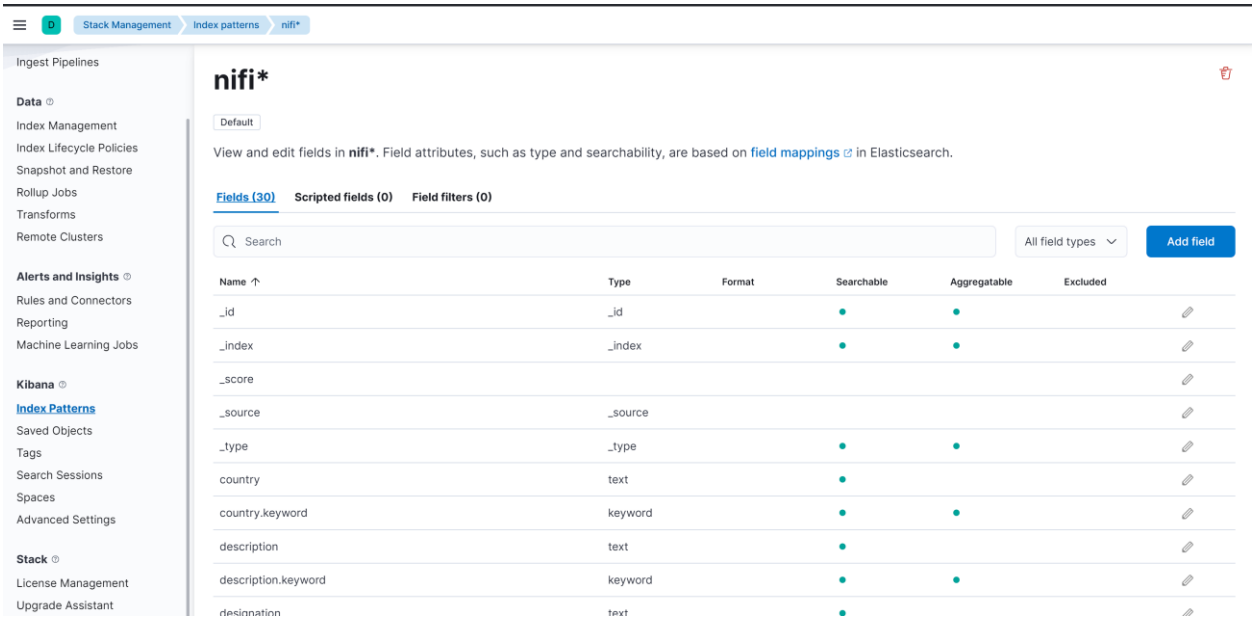
Рисунок 12 – Настройки PutElasticSearchRecord

Kibana

После отправки данных в ElasticSearch переходим в Stack Management\Index Management. Можем увидеть поступившие данные.



На этой же странице переходим в kibana\index patterns. Создаём паттерн.



Для визуализации переходим в visualize library. Строим график гистограммы стоимости напитка к баллам поставленными дегустаторами