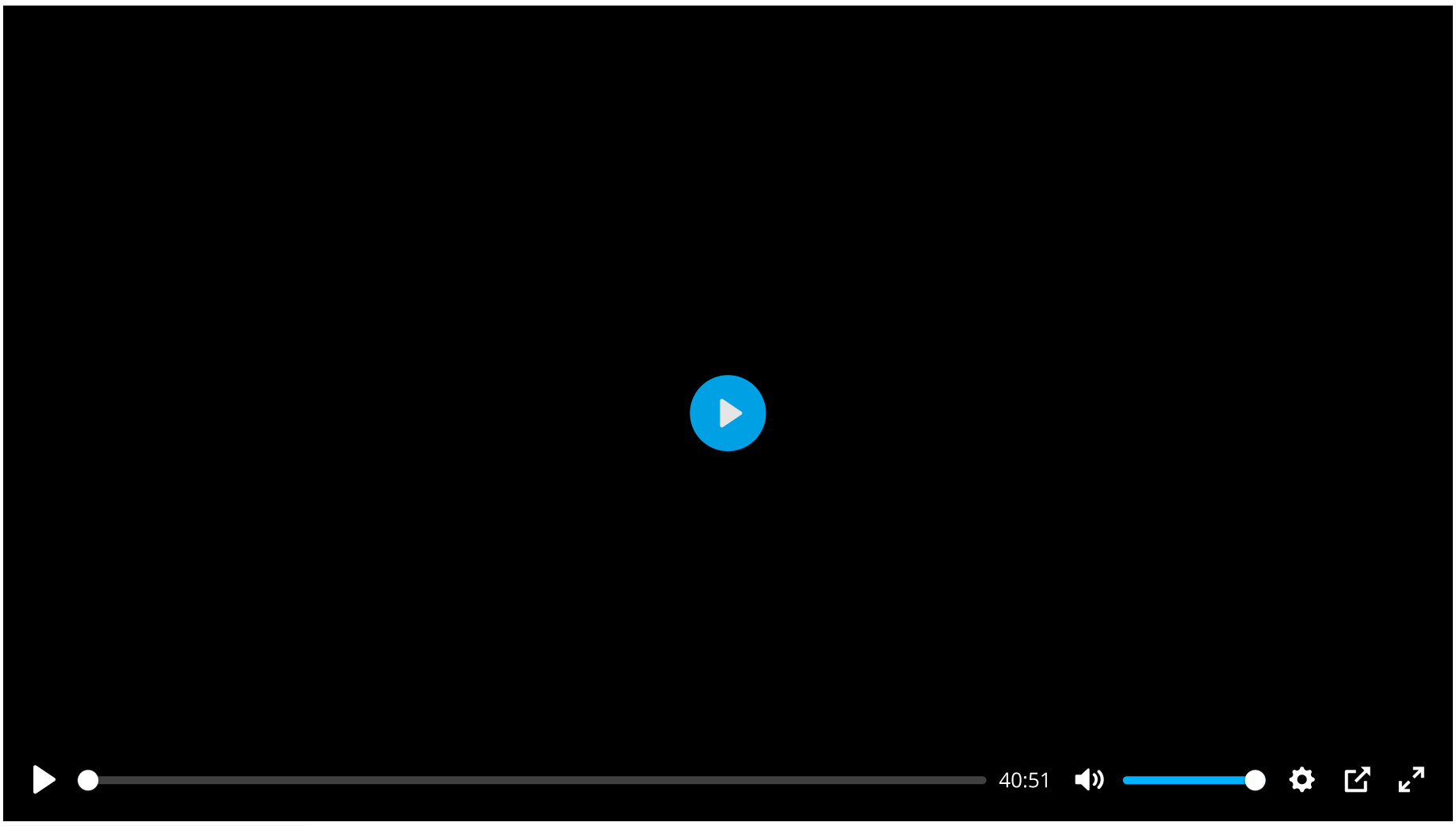
10.4 Построение потока



Презентация к уроку 10.4

Тайм-код (мин:сек)

0:00 Введение
1:39 Построение потока данных
11:45 Предварительный тест
14:36 Построение потока данных продолжение JoltTransformJSON
16:29 Промежуточный тест
17:10 Построение потока данных продолжение QueryRecord
22:22 Промежуточный тест
22:46 Построение потока данных продолжение PutDatabaseRecord
29:23 Промежуточный тест с ошибкой
30:59 Построение потока данных продолжение ExecuteSQL
32:48 Промежуточный тест
34:11 Replay
35:03 Чтение из БД ExecuteSQLRecord
36:30 Промежуточный тест
36:56 Формат даты и AVRO схема
38:51 Промежуточный тест
39:39 Экспорт темплейта
40:28 Итоги

Текстовая расшифровка видео:

ПОСТРОЕНИЕ ПОТОКА ДАННЫХ

План занятия:

- Построение потока данных;
- Запуск потока;
- Изучение результатов;
- Контроль ошибок;
- Сохранение потока данных;
- Использование Data Provenance.

Загрузка и обработка данных

Мы прочитаем данные из внешнего источника (ссылка в описании к данному уроку) и преобразуем их.

Данные представляют из себя XML-файл, который распространяет Европейский Центральный банк с уже закрытыми курсами валют:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<gesmes:Envelope xmlns:gesmes="http://www.gesmes.org/xml/2002-08-01" xmlns=
"http://www.ecb.int/vocabulary/2002-08-01/eurofxref">
    <gesmes:subject>Reference rates</gesmes:subject>
    <gesmes:Sender>
        <gesmes:name>European Central Bank</gesmes:name>
    </gesmes:Sender>
    <Cube>
        <Cube time='2022-11-08'>
            <Cube currency='USD' rate='0.9996'/>
            <Cube currency='JPY' rate='146.25'/>
            ...
            <Cube currency='THB' rate='37.220'/>
            <Cube currency='ZAR' rate='17.8397'/>
        </Cube>
    </Cube>
</gesmes:Envelope>
```

Нас интересует прочтение этих данных.

Что мы сделаем:

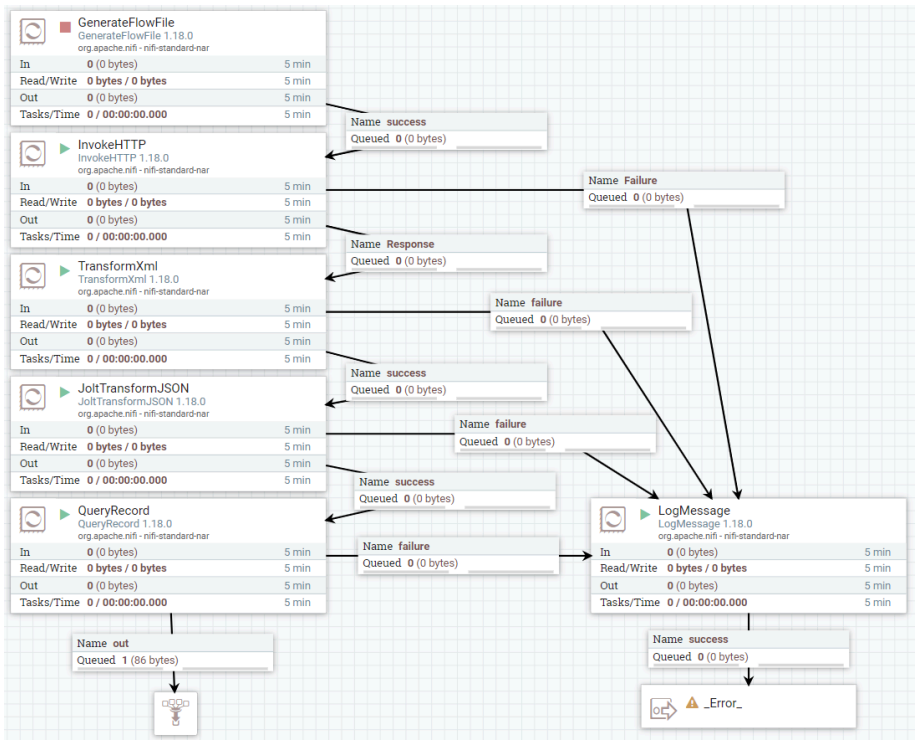
- Преобразуем их в JSON-формат и сделаем преобразование в рамках этого формата;
- Отфильтруем необходимые данные и представим их в формате CSV.

Пример того, как могут выглядеть итоговые данные:

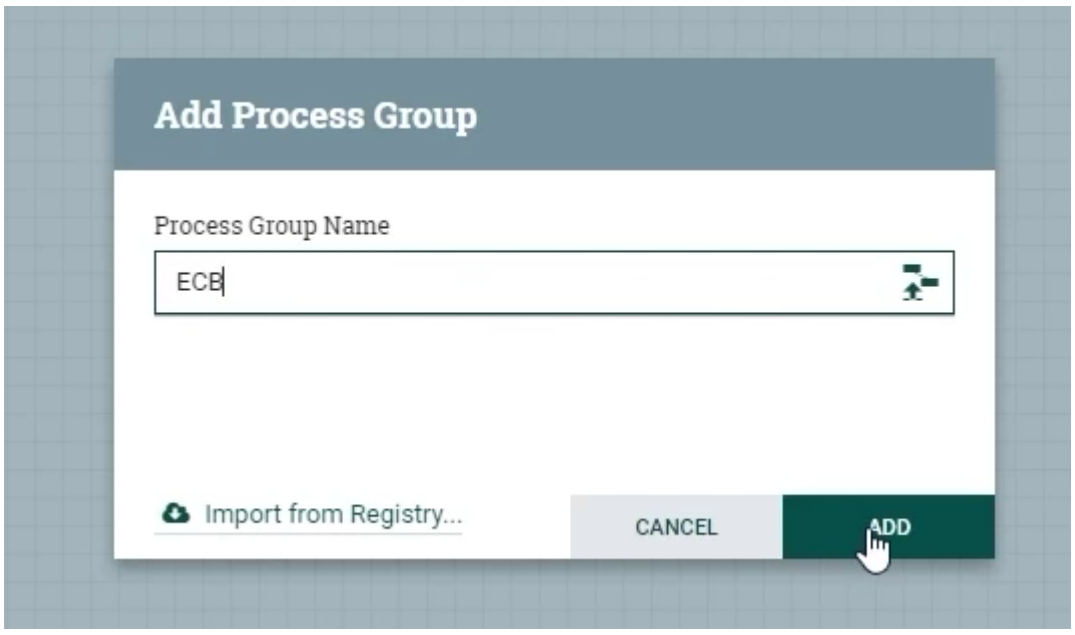
```
time,currency,rate
2022-11-08,USD,0.9996
2022-11-08,JPY,146.25
2022-11-08,GBP,0.87378
```

Эти данные мы запишем в программу PostgreSQL. Это необходимый навык для дальнейшей работы с данными.

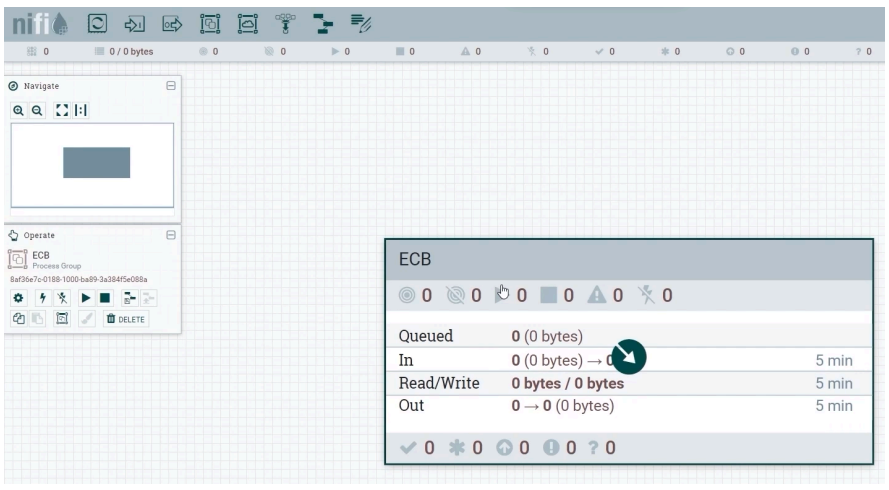
Подобным образом будет выглядеть наш поток (здесь не хватает сохранения в базу данных, рассмотрим это позже):

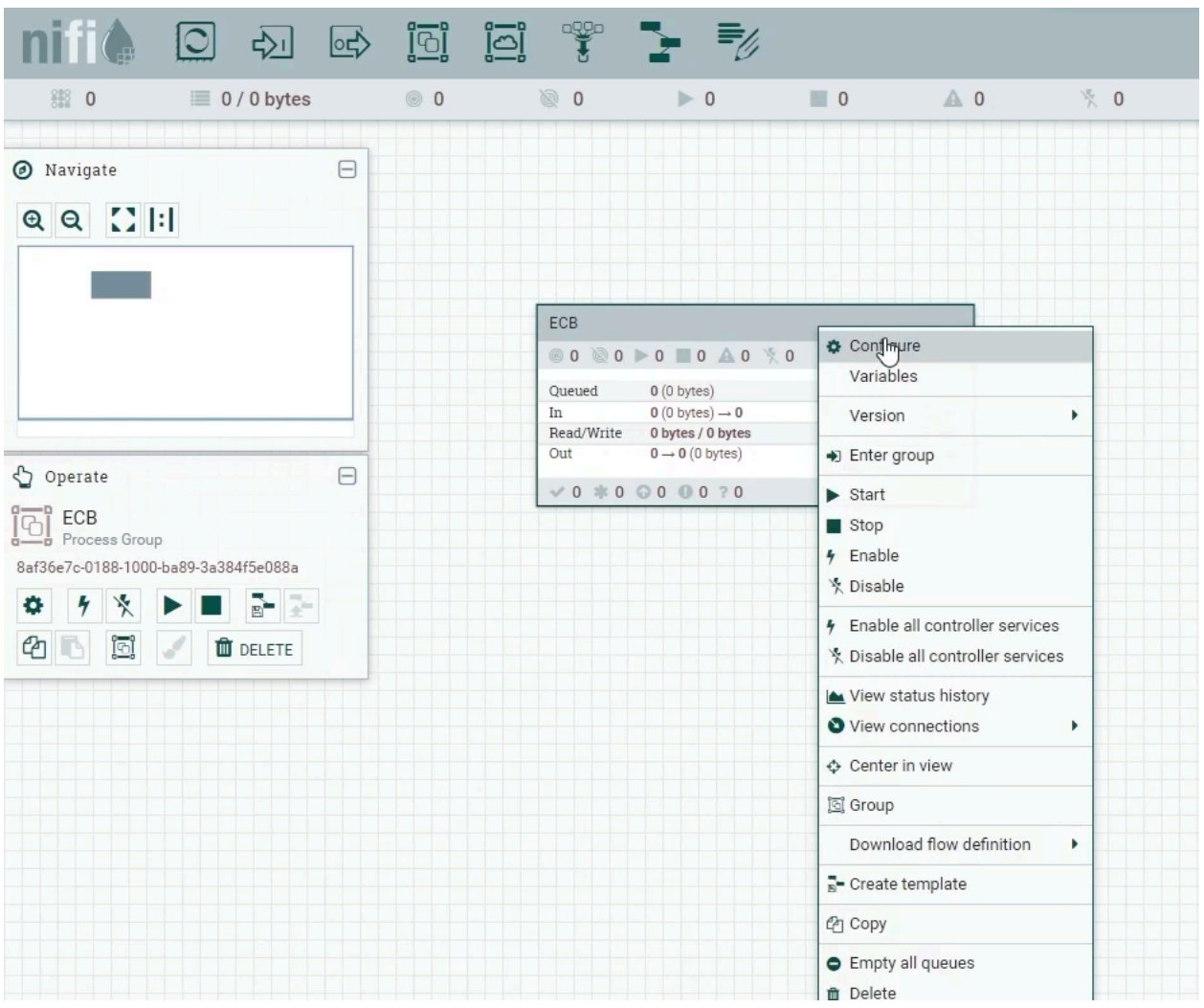


Перейдем в приложение NiFi и расположим группу на рабочем столе. Из тулбара перетаскиваем соответствующую иконку и пишем название (у нас будет «ECB» от Европейского Центрального банка):

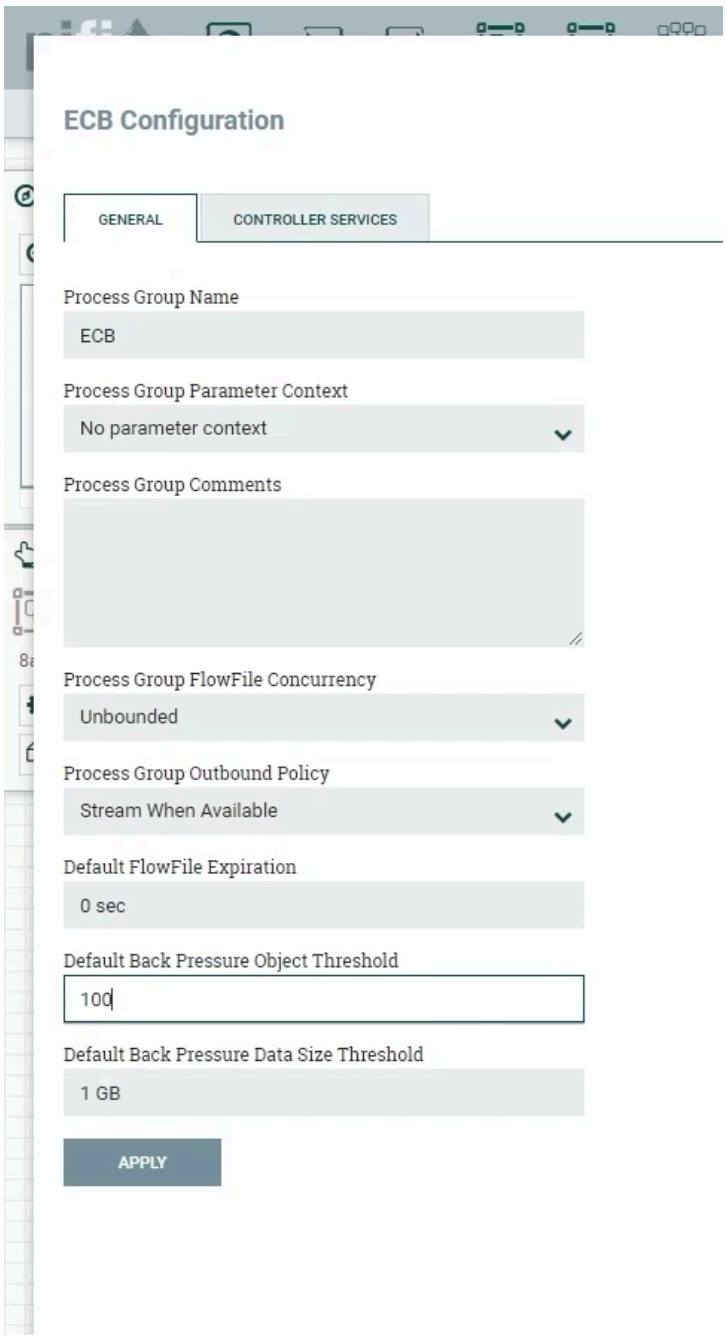


Мы создаем группу и заходим в её настройки:





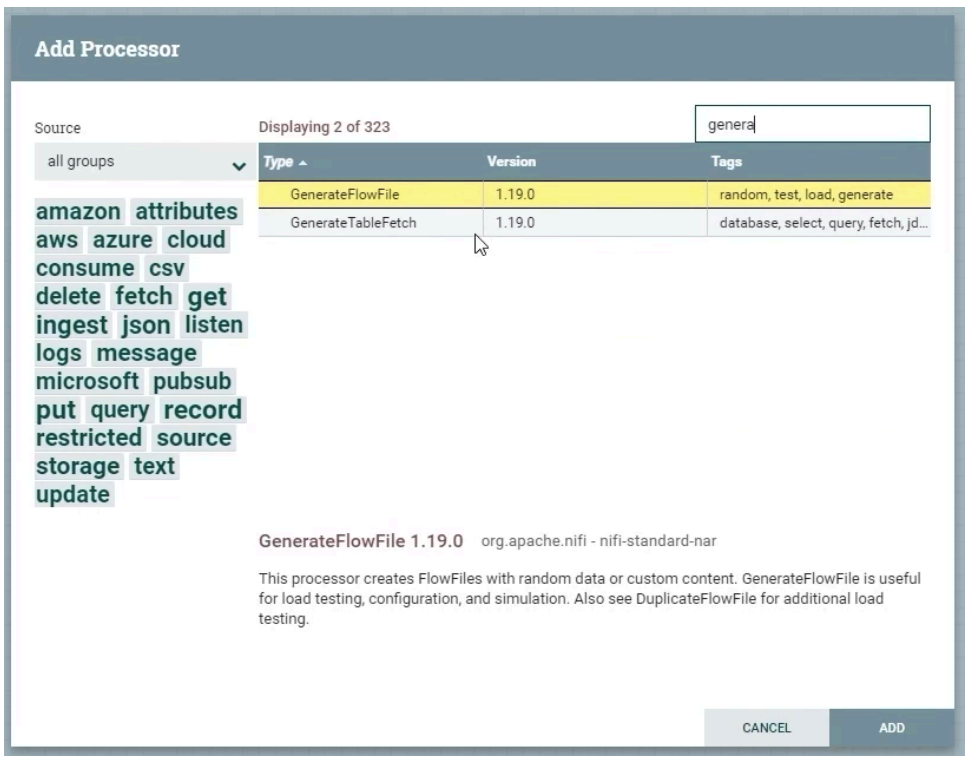
Рекомендуем поменять параметр «10 000» на «100» FlowFile (максимальное количество в очередях):



На данную страницу мы вернемся позже и настроим на уровне этой группы в корне, соответствующие сервисы:

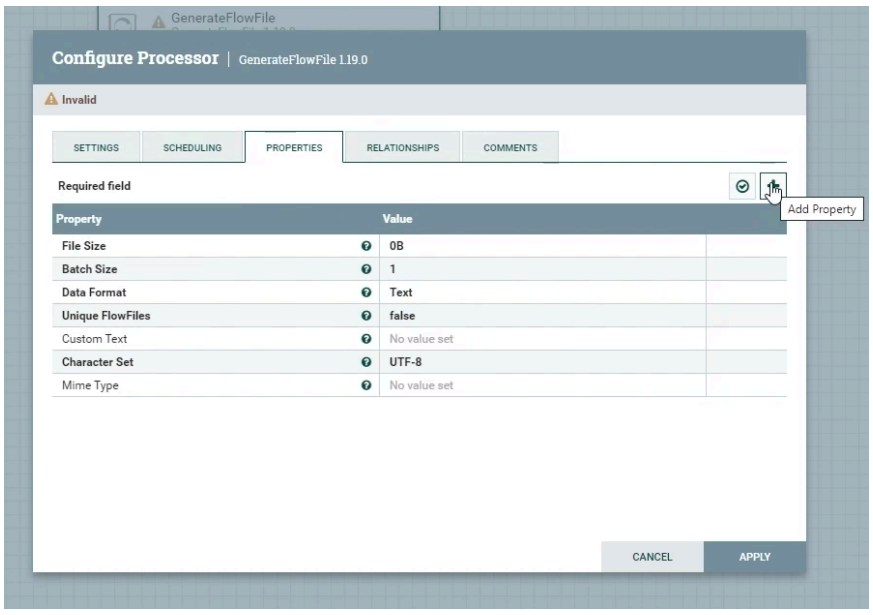
(2:25)

Заходим в группу через двойной клик и начинаем строить DataFlow. Внутри группы нужно расположить процессор «GenerateFlowFile»:

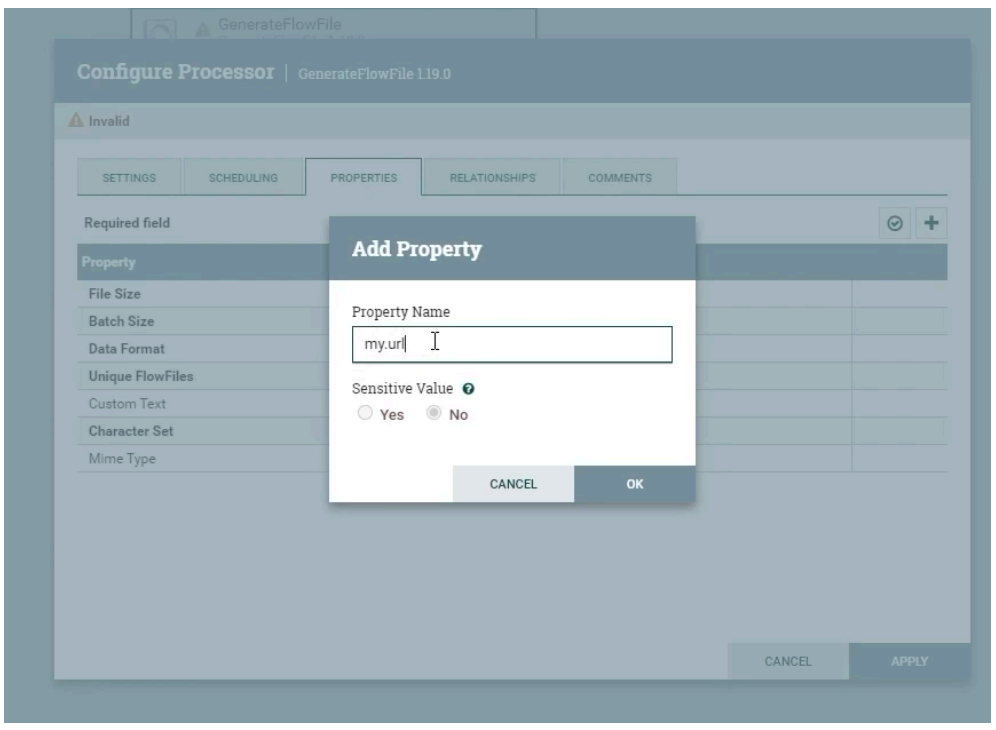


Это процессор, который может запускаться по расписанию или вручную. В этот процессор можно дописать атрибуты, необходимые для потока:

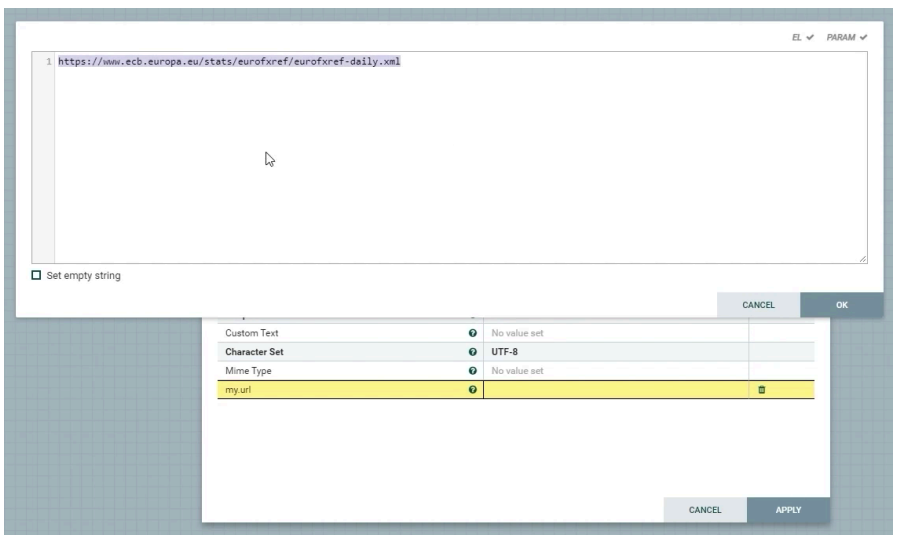
Внутри процессора нажимаем сверху справа плюсик:



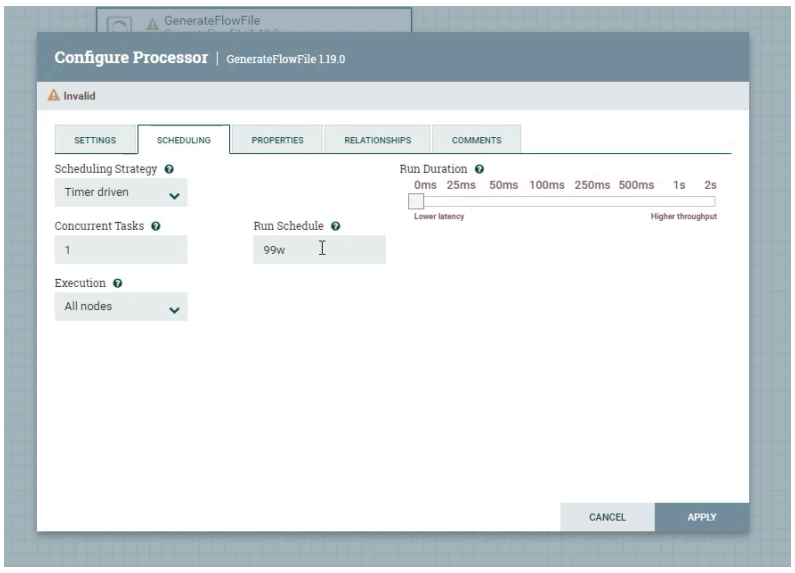
Первый атрибут, который будет использоваться, назовем «My URI»:



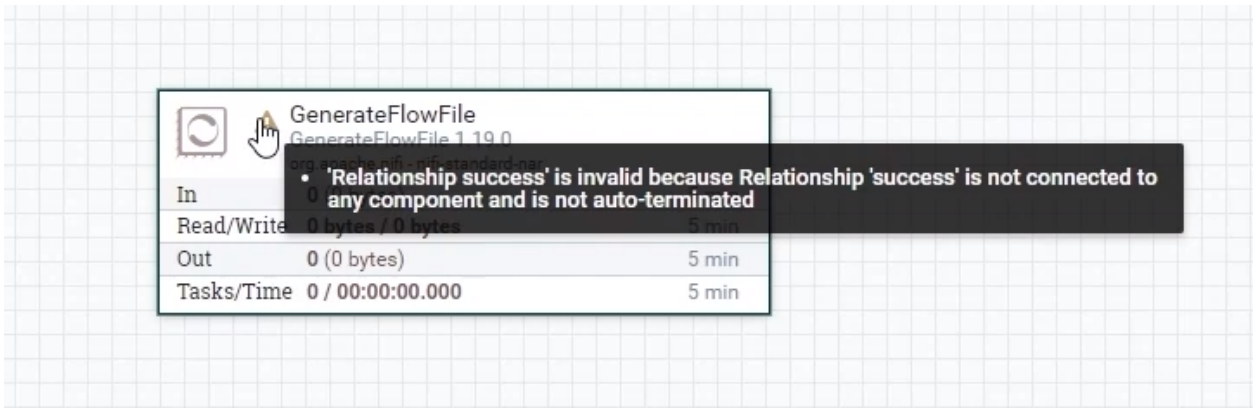
Префикс «My» будет удобен для работы с атрибутами. Если список большой, то все атрибуты будут группироваться в алфавитном порядке. Префикс позволяет сгруппировать все атрибуты. Вставляем ссылку XML-файл из буфера обмена. Выглядит это следующим образом:



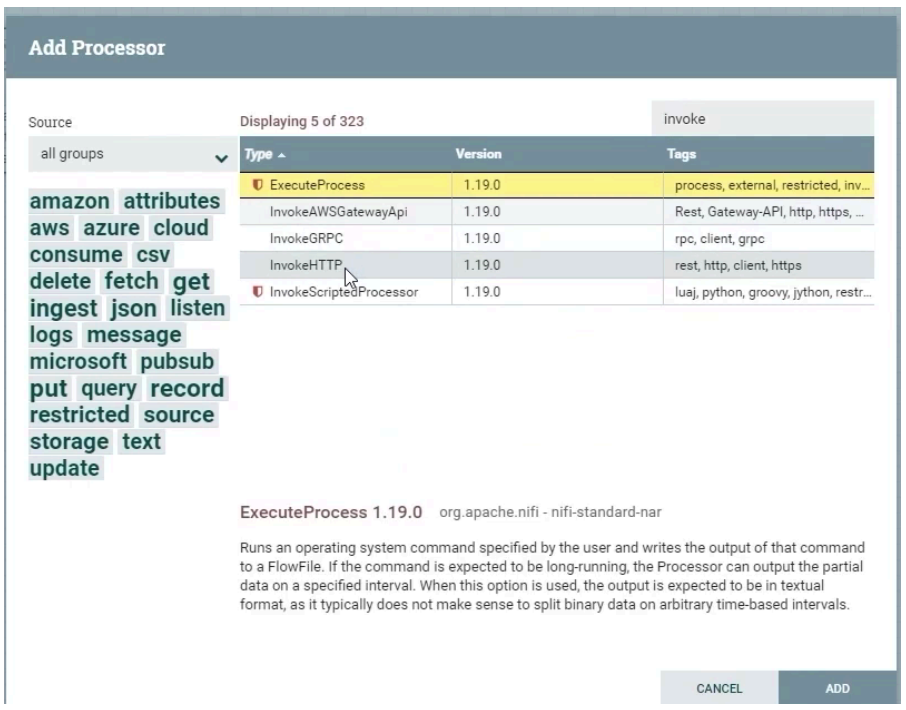
Рекомендуем сразу в разделе «Scheduling» задать частоту запуска процессора: пишем «99w». Это очень большое значение. Если мы оставим процесс в запущенном виде, то Flowfile не будут часто генерится, и DataFlow не переполнится. Можно ещё задать запуск по CRON driven с значением Run Schedule = * * * * * ? 2029. В таком случае процессор можно будет запустить только вручную через контекстное меню, выбрав Run Once:

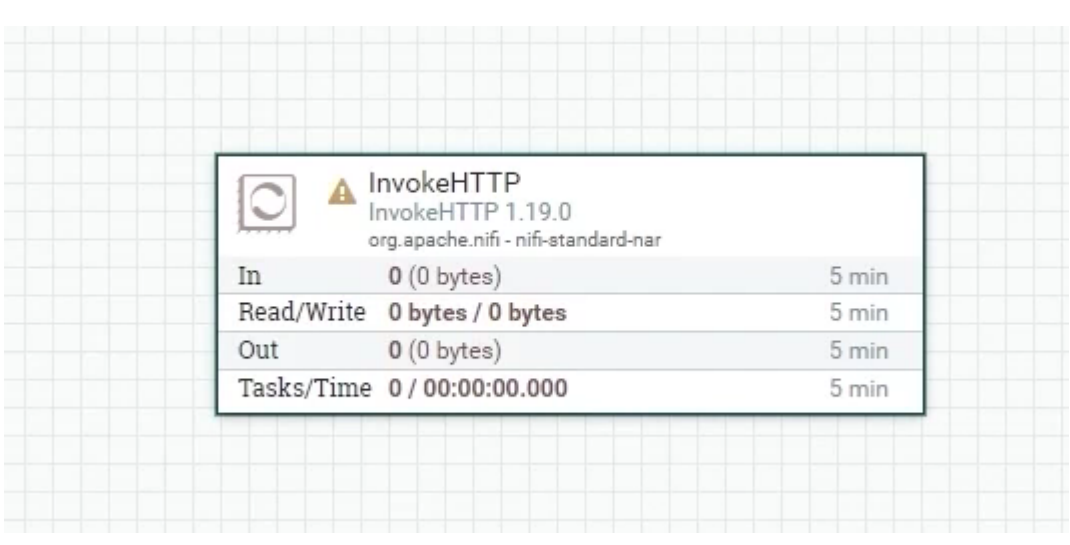


Нажимаем «Apply» и видим, что процесс расположился на канвасе. Он находится в статусе «Invalid». Мы можем подвести курсор к иконке и прочитать причину инвалидного состояния процессора:

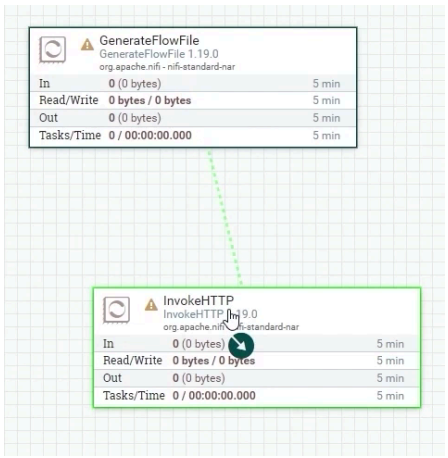


В данном случае процессор ни с чем не соединен, исправим это, так как дальше нам необходимо будет строить поток. Следующим будет процессор «InvokeHTTP»:

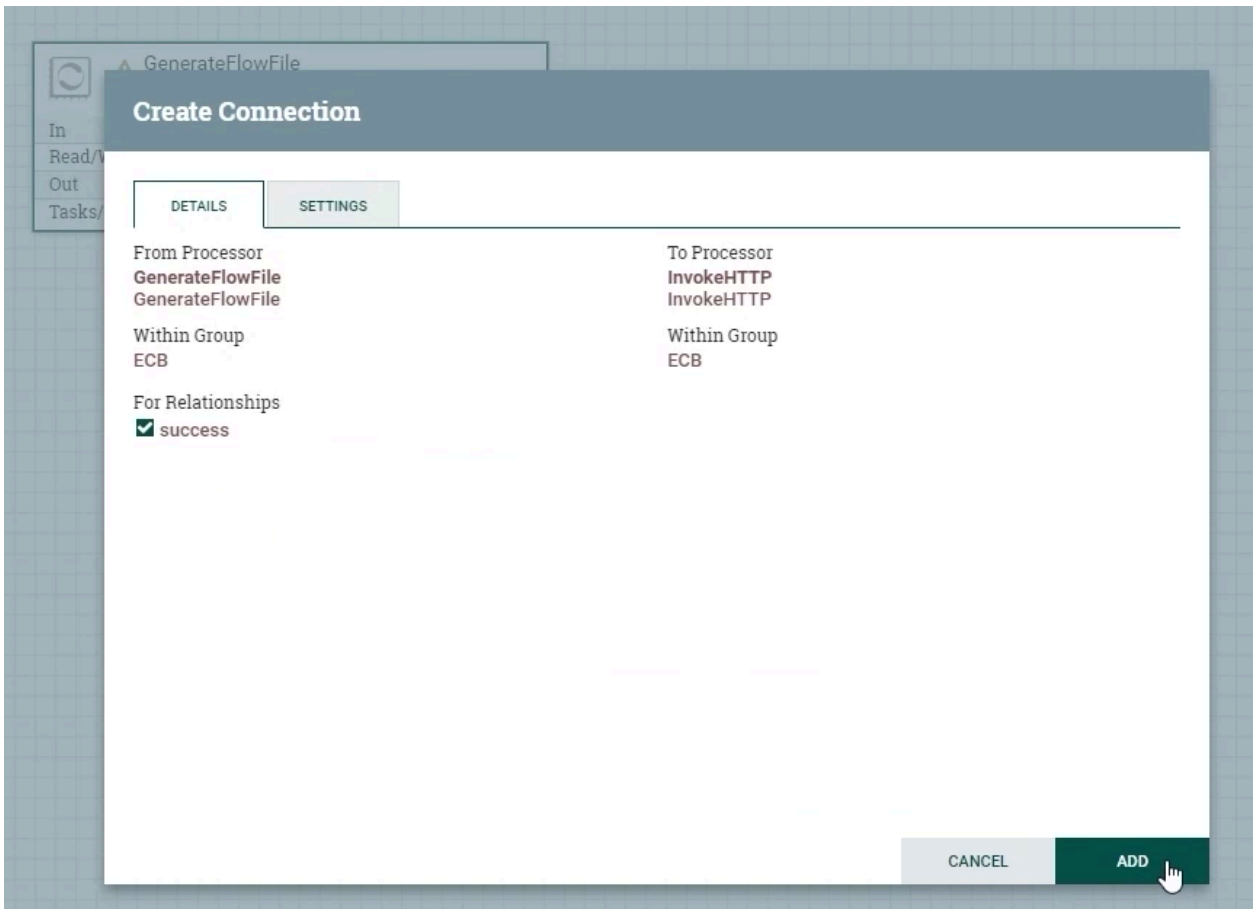




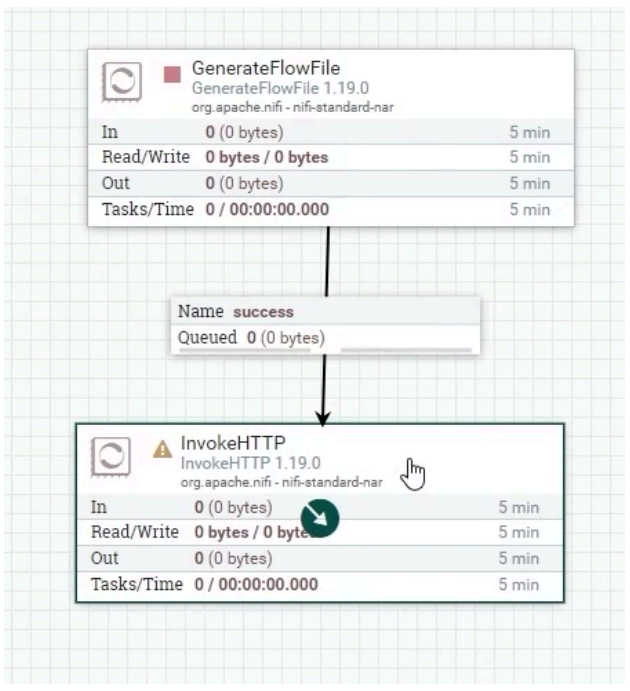
Из центра «GenerateFlowFile» перетаскиваем соединительную линию к следующему процессору:



Соглашаемся с дефолтными настройками:

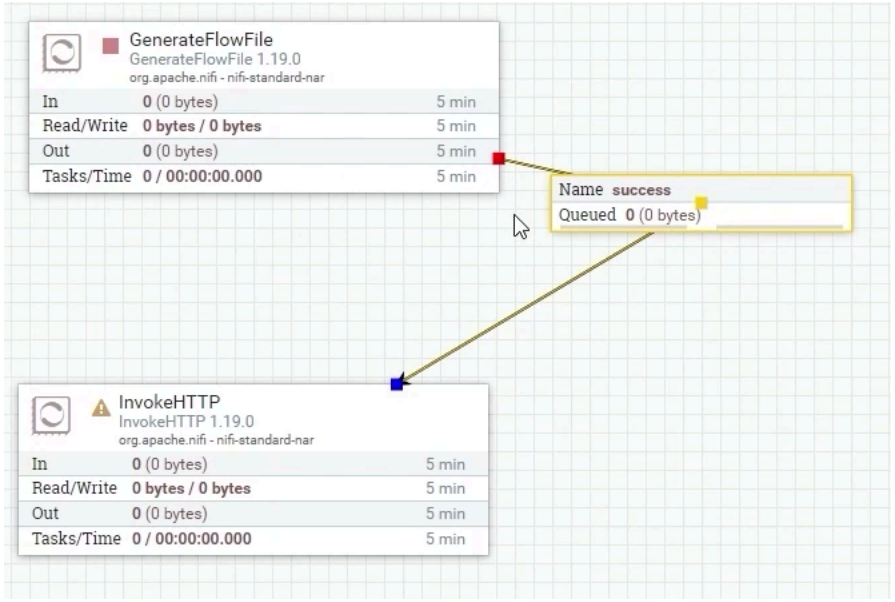


Соединяем два процессора между собой:



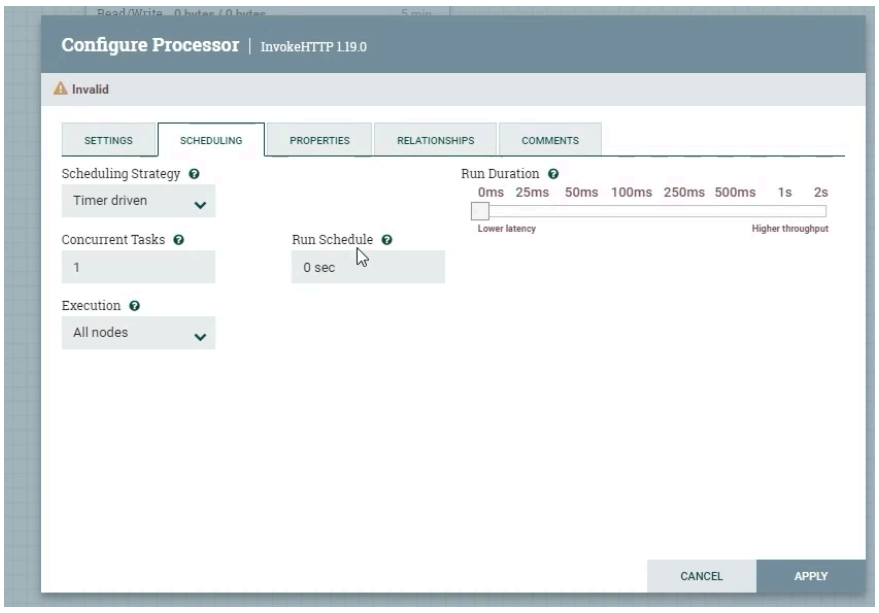
Для дальнейшего комфорта рекомендуем сразу строить поток ровным и компактным.

Сделав двойной клик, можно заметить желтый квадрат, с помощью которого можно двигать объект очереди:

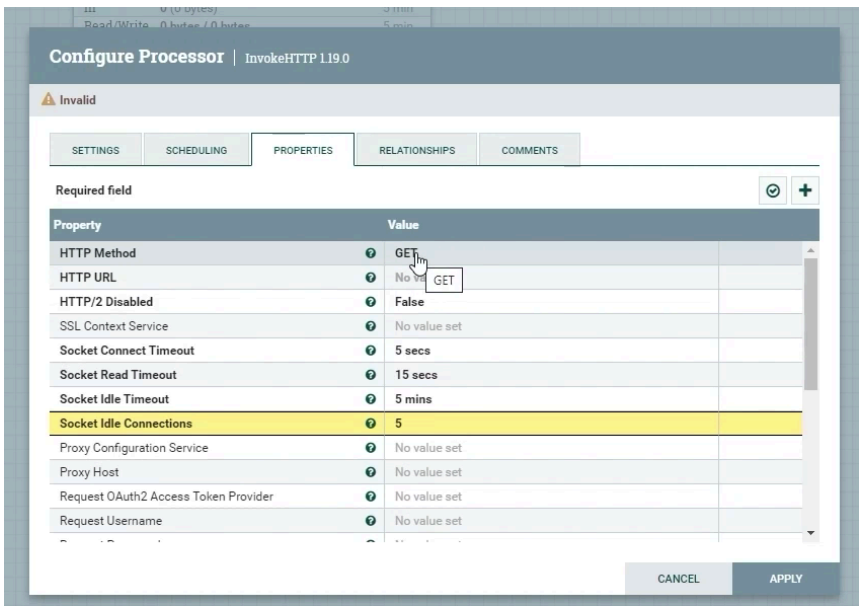


Делаем компактное расположение процессоров.

Следующий объект – «InvokeHTTP». Он никак не настроен. В разделе «Scheduling» ему ничего менять не нужно, он должен запускаться максимально быстро, то есть оставляем «0 sec»:

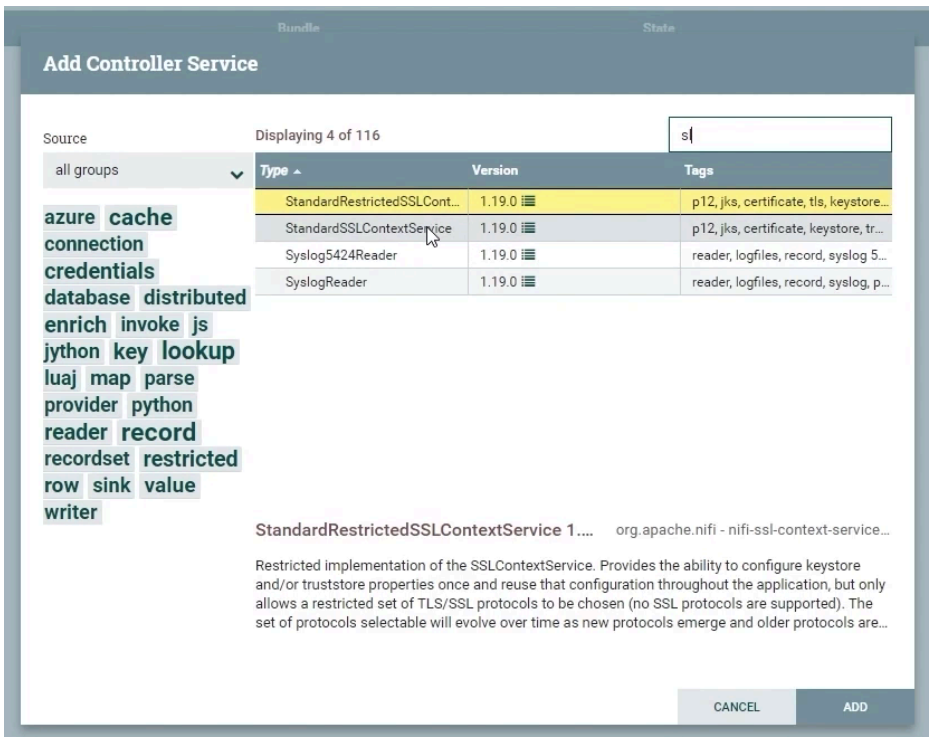


В свойствах проверяем настройки:



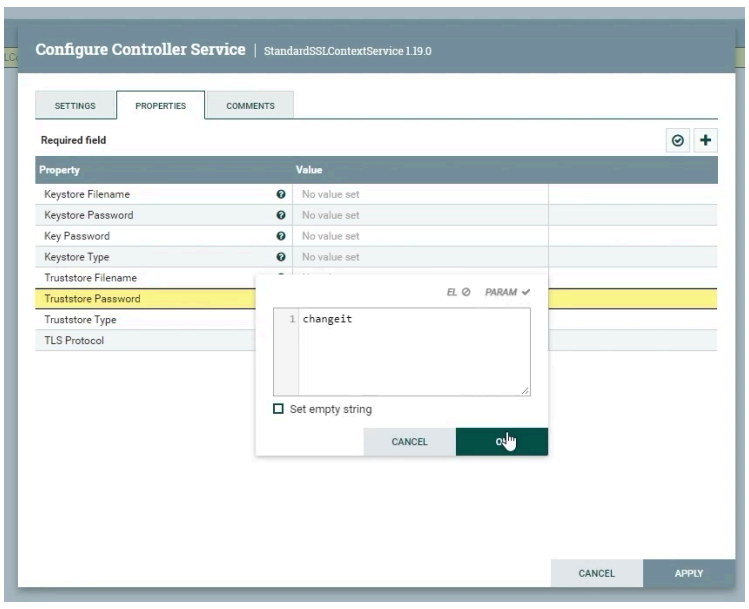
Метод – Get, адрес задан в атрибуте (если нам нужен атрибут, то мы начинаем так: `«${my.url}»)`. Все атрибуты удобнее собирать в первом процессоре для дальнейшего обслуживания.

Следует проверить в процессоре «InvokeHTTP» настройку SSL Context Service. Все сервисы лучше настраивать на уровне группы или же в корне, а именно через контекстное меню и configuration. В закладке «Controller Services» задавать новый контроллер, который, в нашем случае, будет называться «StandardSSLContextService».



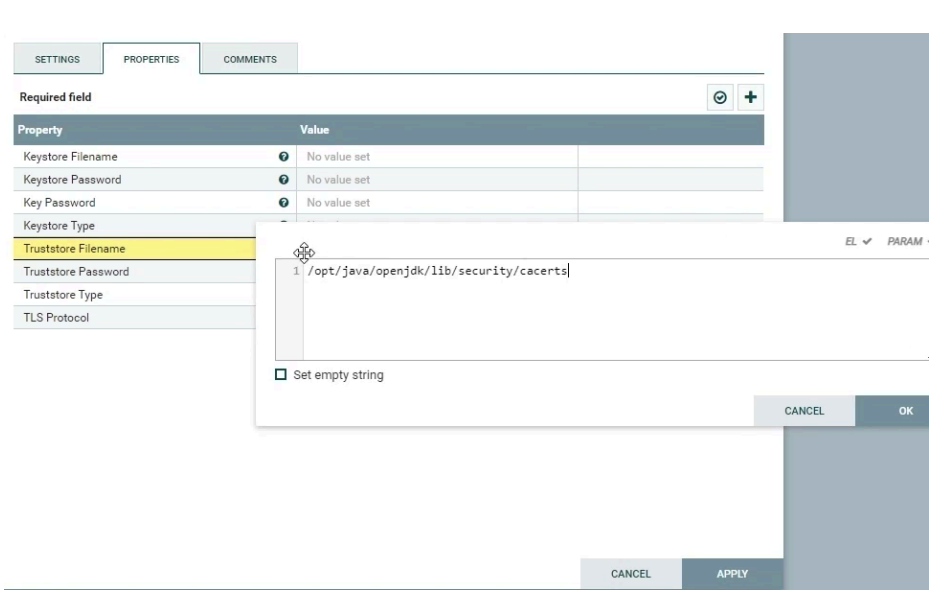
Настроим его:

Нажимаем на «шестеренку» справа в этой строчке. Нам необходимо задать тип хранилища публичных ключей для JAVA. Нужно задать дефолтный пароль:

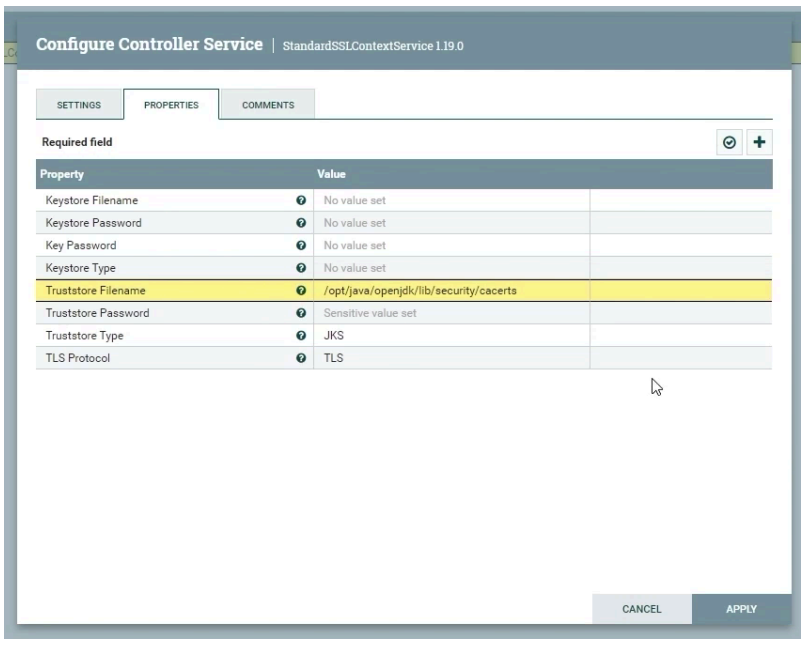


В данном случае пароль – «Changeit» (его никто не менял).

Также создаем файл для хранилища (его путь указываем полным для системы):

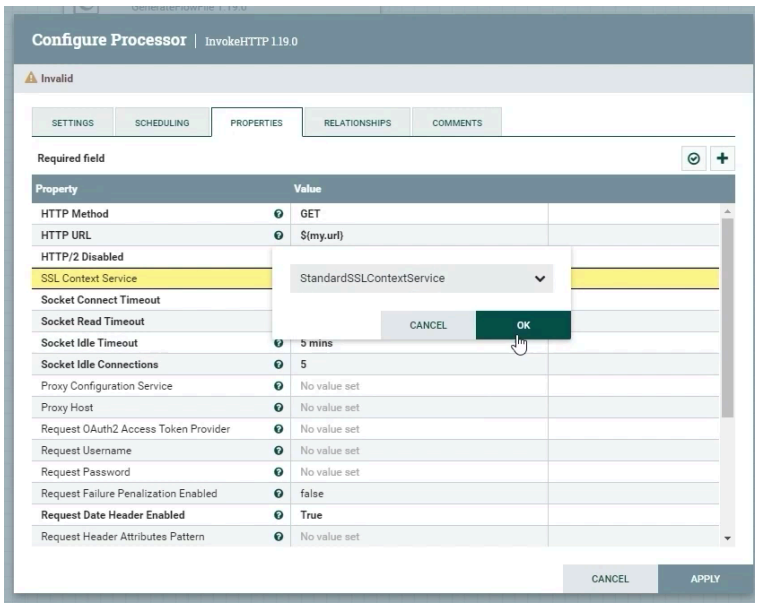


В нашем примере NiFi работает в докере и, соответственно, нужно заранее определить путь к этому файлу и указать:

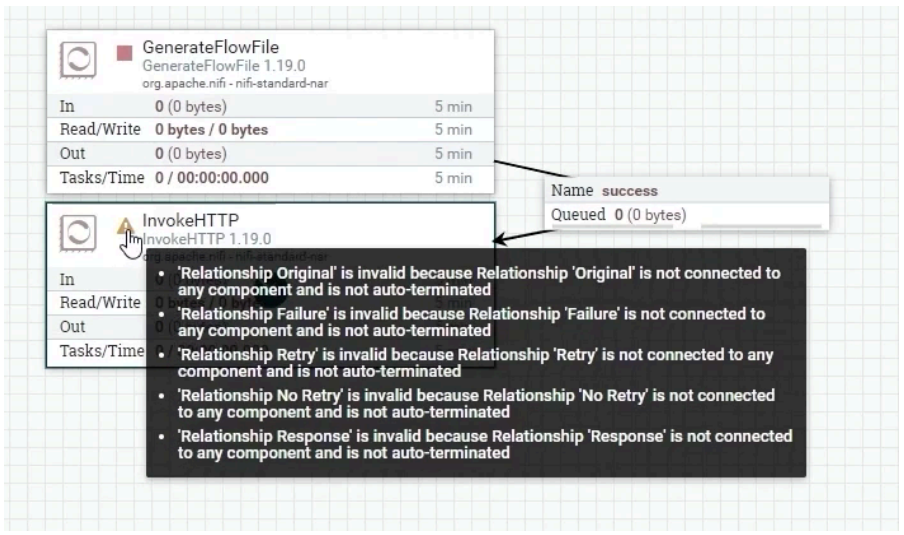


После того, как мы нажали кнопку «Apply», сервис находится в статусе «Неактивен». Через иконку «Молния» необходимо сделать его активным. Сделав вышеперечисленное, сервис готов к использованию.

Возвращаемся в процессор «InvokeHTTP» и выбираем уже настроенный сервис из списка доступных:

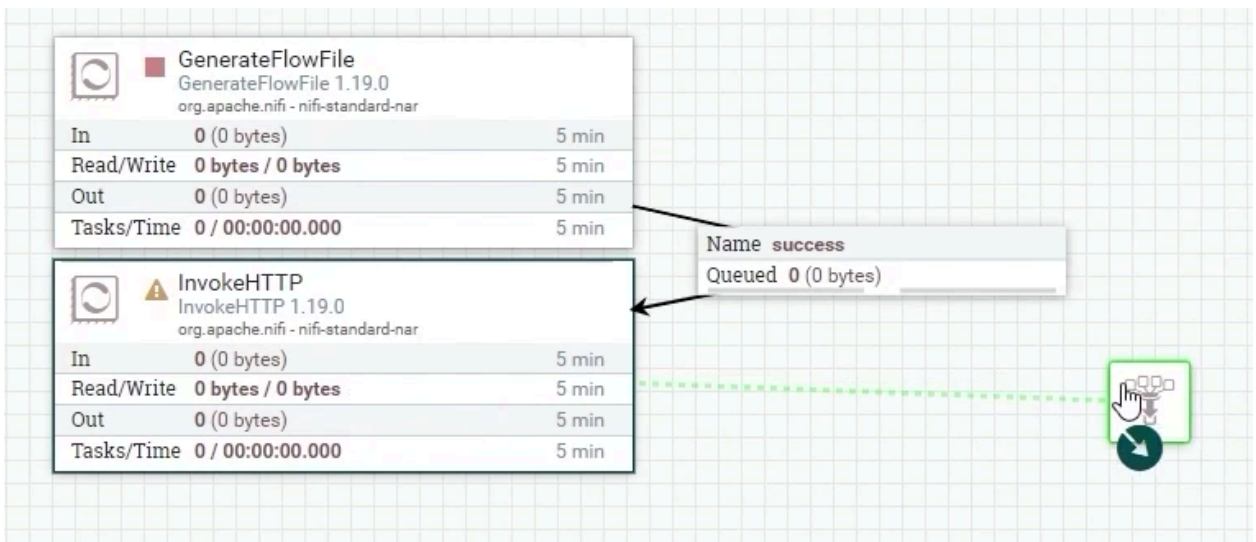


На данном этапе мы видим, что процессор настроен:

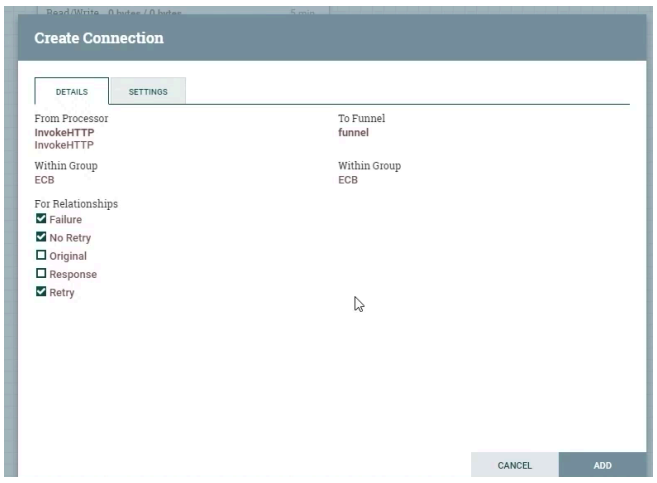


В дальнейшем нужно будет сделать соединение и вывести поток ошибок.

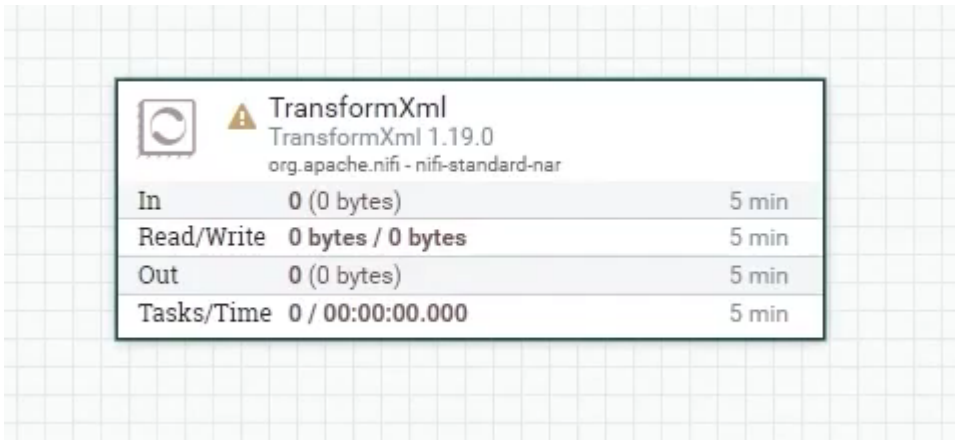
Для потока ошибок мы рекомендуем использовать «Воронку» (строить линию от процессора к воронке и выбирать уже там):



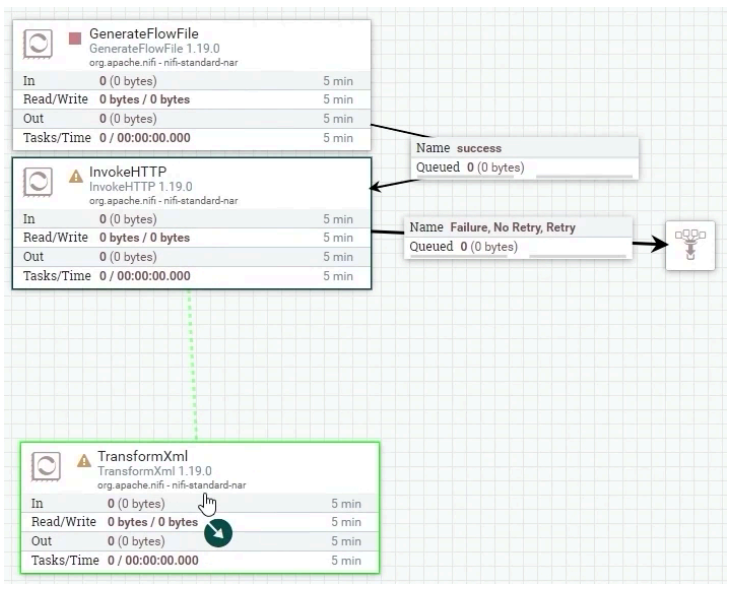
Выберем «Failure», «No Retry», «Retry» (если будет повторная попытка обращения к какому-то веб-узлу), а «Original» задизейблим:



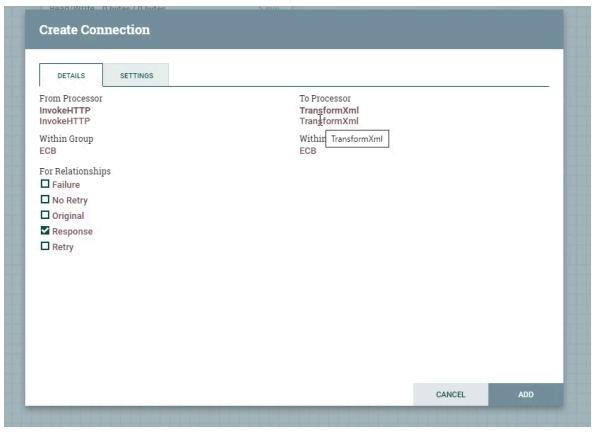
Необходимо обработать процессор после прочтения файла. В данном случае – «Xml». Сделаем преобразование данного файла в JSON, используя процессор «Transform». Здесь должен быть «Transform Xml»:

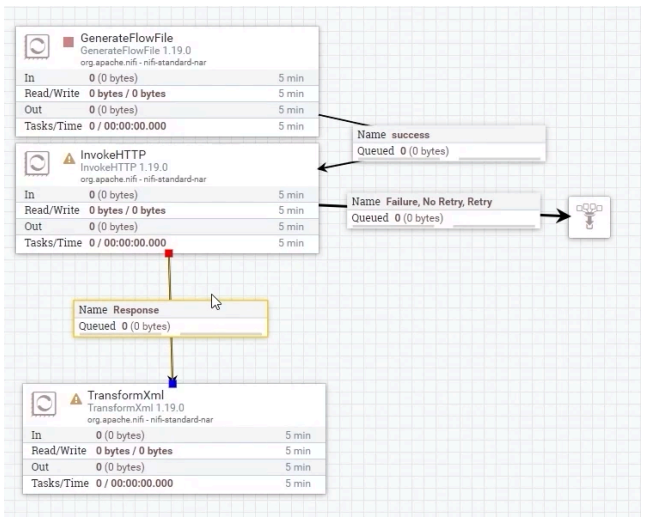


Добавляем этот процессор, после чего соединяем один с другим:

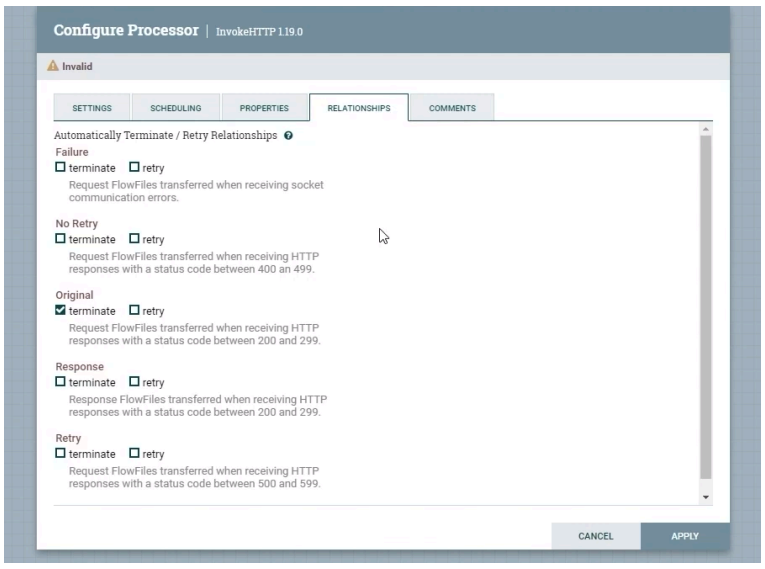


Наш «Response», когда мы получили ответ от сервера, идет в эту очередь:

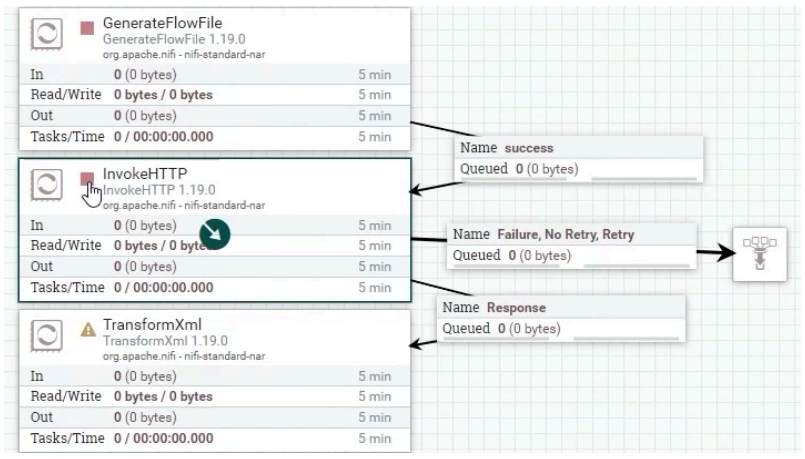




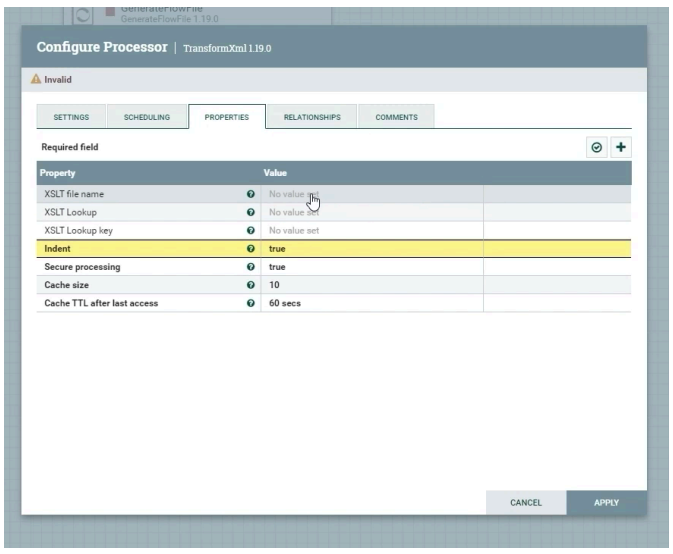
Проблема: InvokeHTTP что-то не достает для того, чтобы быть валидным. В данном случае загвоздка заключается в неопределенном «Relationship Original». В закладке «Relationship» выделяем «Original» и делаем ему «Terminate»:



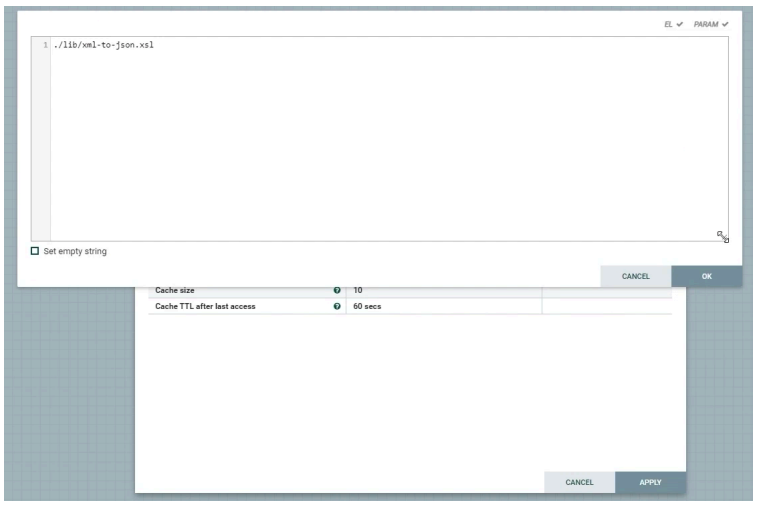
Оригинальный файл не последует за рамки данного процессора. На примере мы видим, что процессор стал валидным:



Настроим процессор «Transform Xml». Существует очень удобный файл шаблона, который позволяет из Xml сделать JSON -формат:

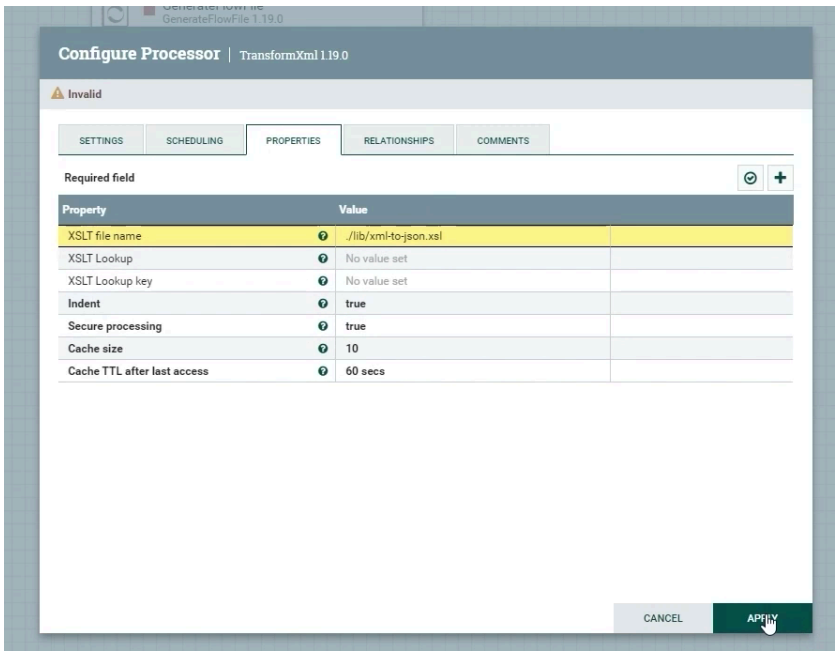


Нужно указать путь к этому файлу (сам файл находится в приложении к данному уроку):

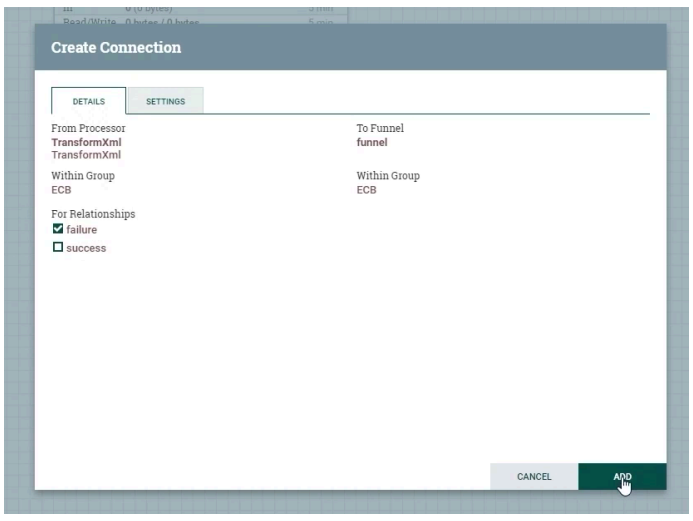


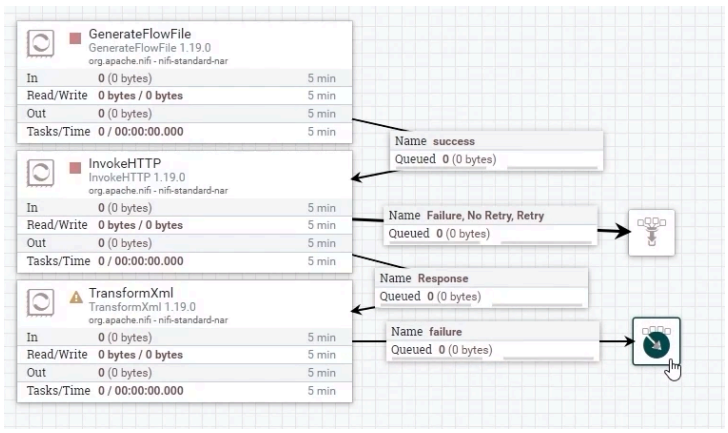
Начинаем с «./». NiFi посчитает, что это с корневого каталога, где установлен NiFi. Будет использоваться папка «lib», где будут расположены все библиотеки. Во избежание проблем с изменчивостью версий файлов делаем символическую ссылку, чтобы в NiFi при апгрейде файла путь никогда не менялся.

После указания filename все остальные настройки – по дефолту:



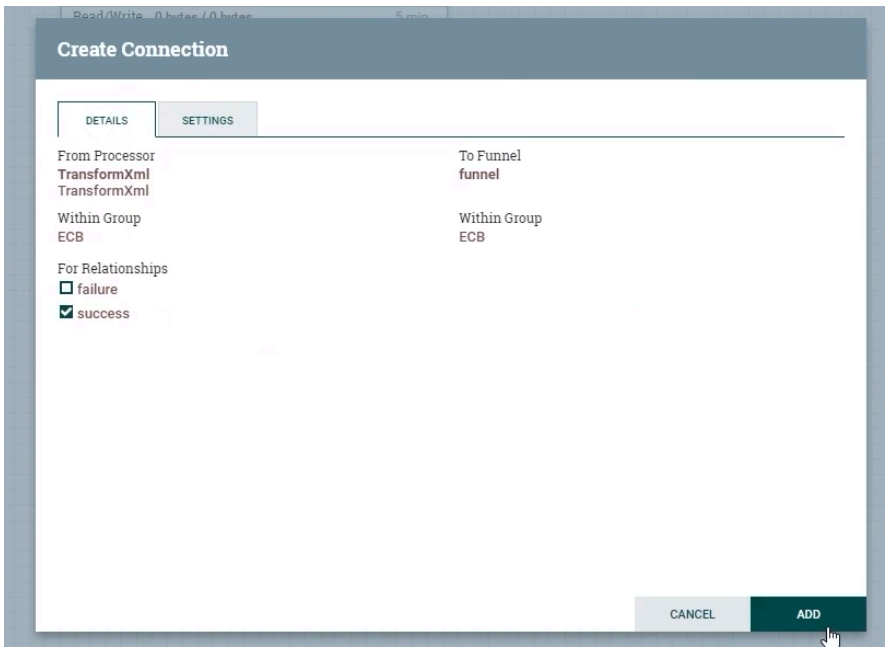
Добавляем еще одну воронку, в которую выводим поток ошибок:



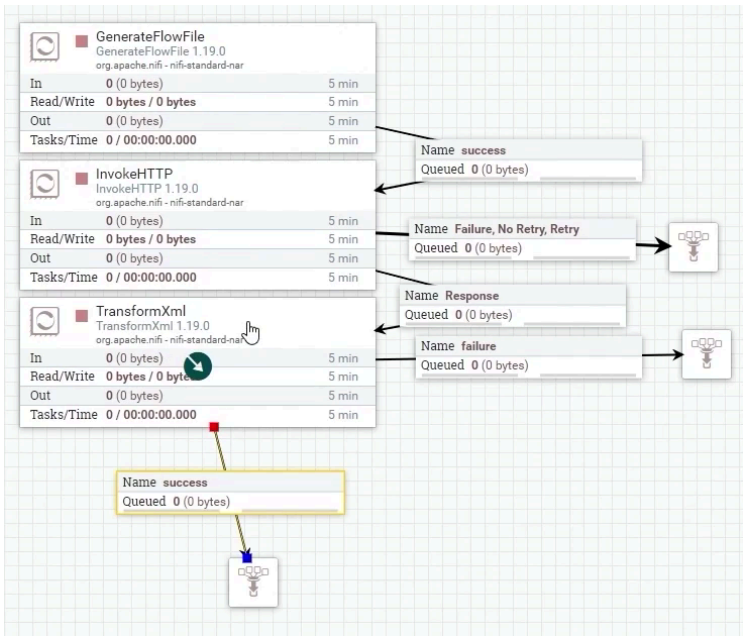


Подобное расположение наиболее удобно и эффективно, так как файлы с ошибками будут находиться справа от процессора. Таким образом можно быстро вычислить процессор, который выдал ошибку.

Проверим «Transform Xml». Nak как мы не знаем какой процесс будет следующим, установим воронку и поток «Success» выводим туда:

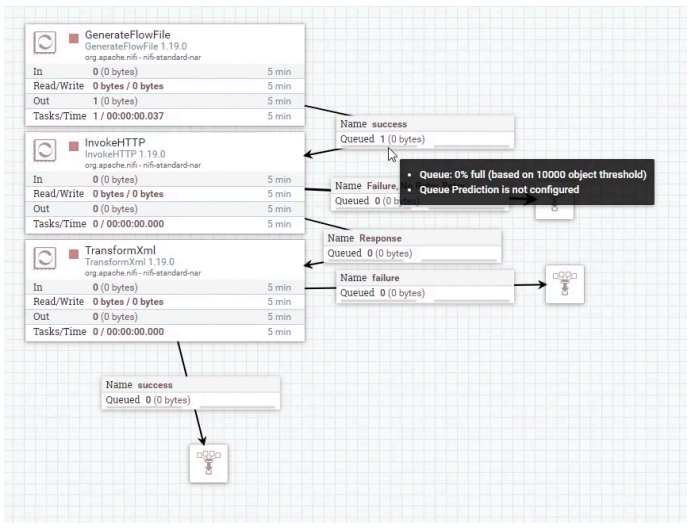


Мы видим, что «Transform Xml» стал валидным и остановленным:

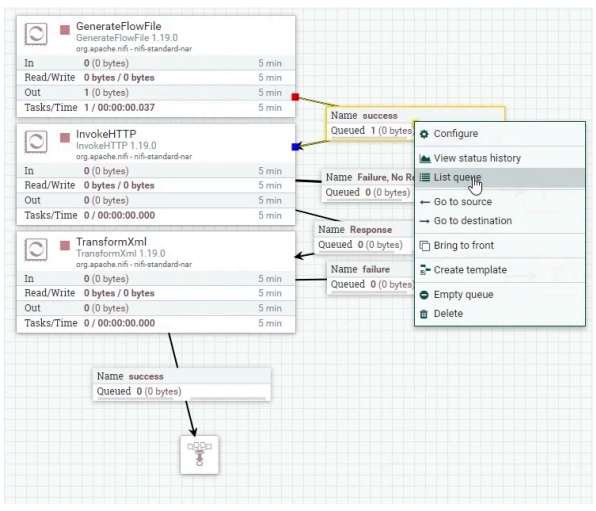


Вопрос: как убедиться на раннем этапе работает ли поток?

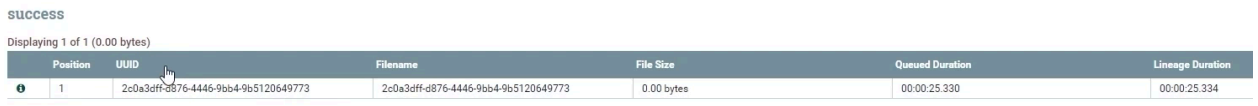
Ответ: для этого в контекстном меню первого процессора «GenerateFlowFile» выбираем пункт «Run Once», так мы запускаем процесс не целиком, а лишь тестовый, единичный запуск. Появляется файл нулевой длины:



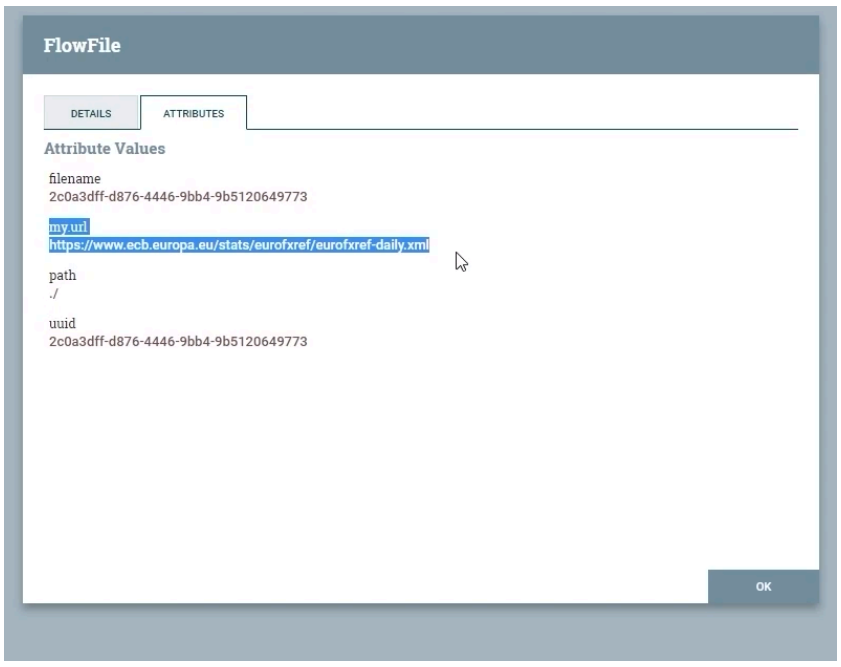
У этого файла есть необходимые атрибуты. Правой кнопкой мыши нажимаем на очередь, в которой появился этот файл, делаем просмотр:



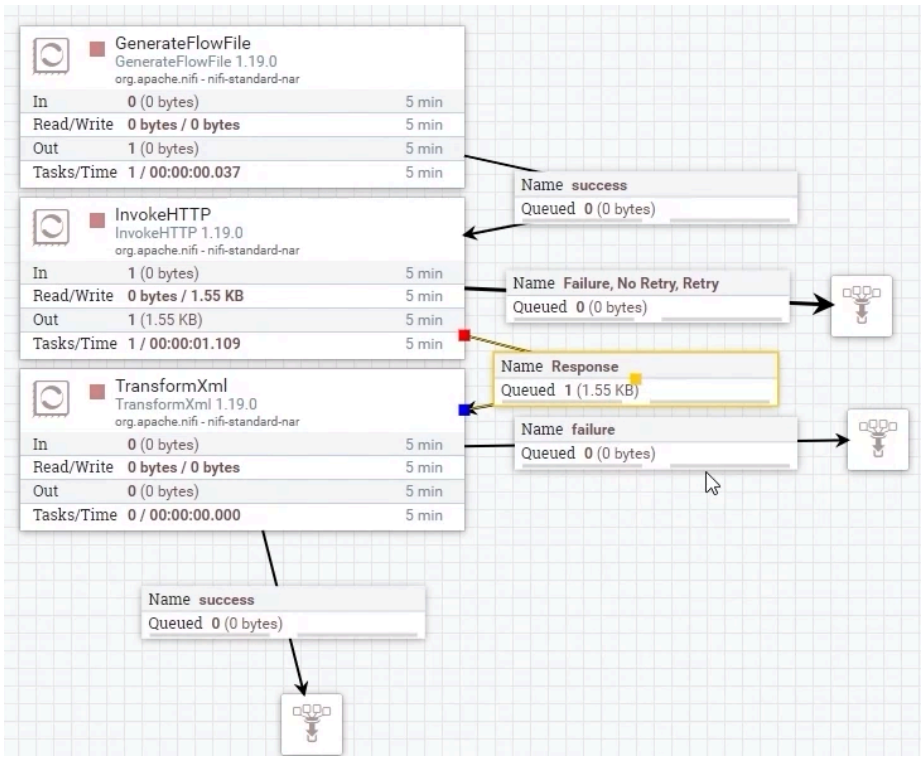
Изображенный файл имеет рандомный UUID в названии filename и идентификационный номер. Нас интересует «кнопочка» с информацией слева и «глазик», который позволяет посмотреть содержимое FlowFile:



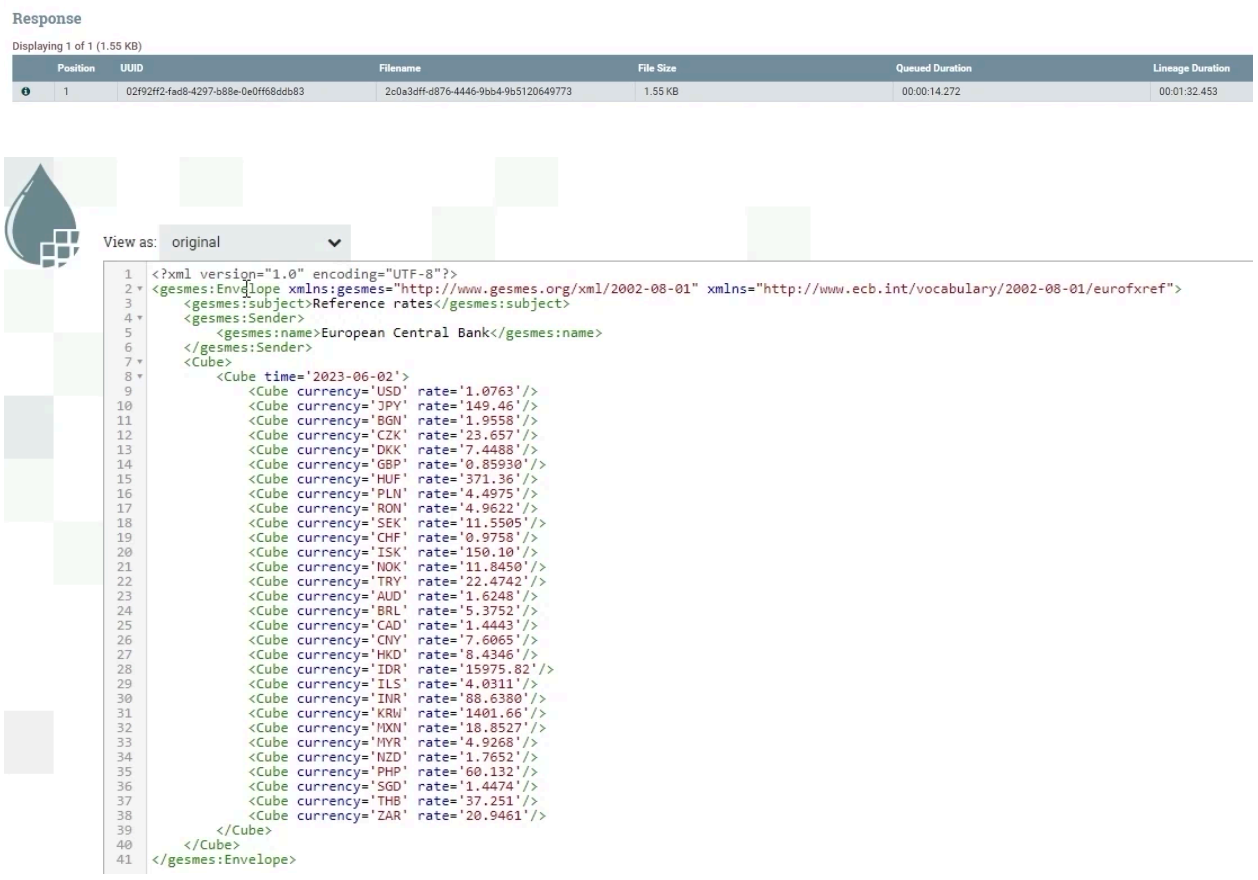
По информации мы можем проверить наличие атрибутов:



Запускаем через Start или Run Once скачивание файла из сети. Соответствующий процессор отработал, и мы видим, что файл отработал (он не нулевой):

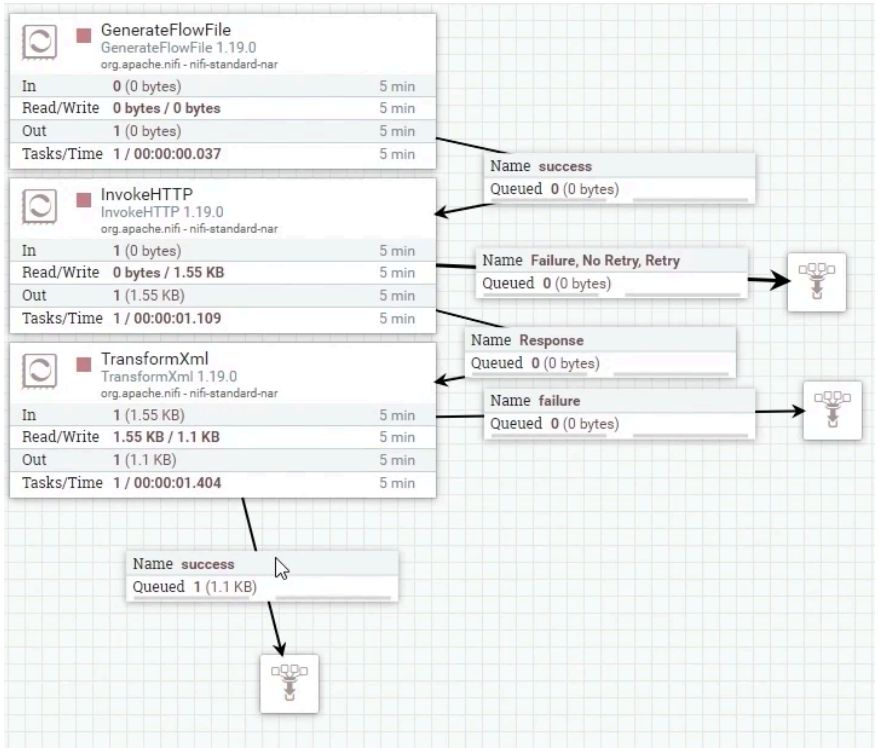


Мы можем посмотреть эту очередь и файл:

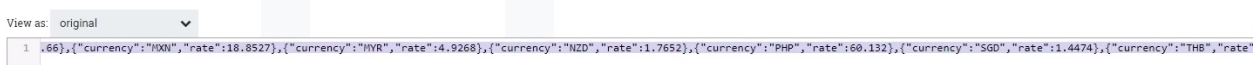


Мы видим, что это Xml-файл с наличием названий валют и их кодов, а также rate.

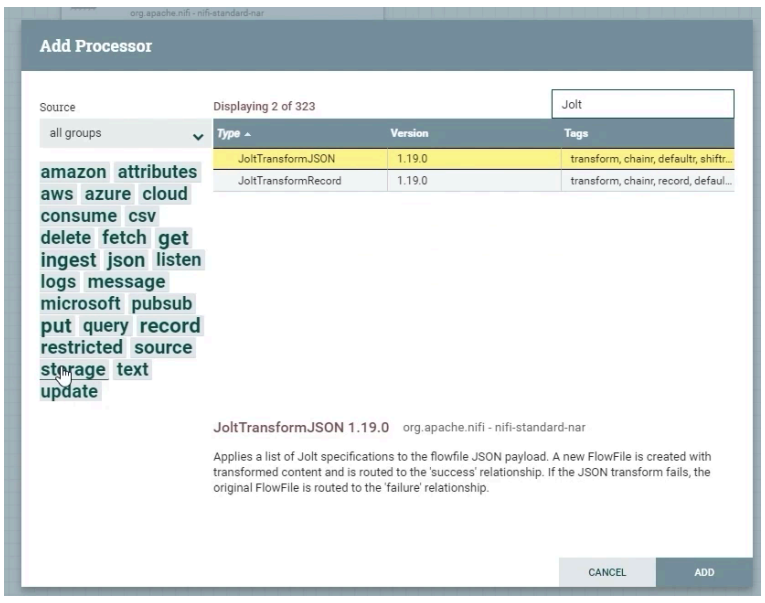
Закрываем просмотр, переходим к следующему шагу – трансформации Xml-файла с использованием заготовленного template, взятого из сети. Запускаем тем же способом и видим, что файл вышел в поток «Success»:

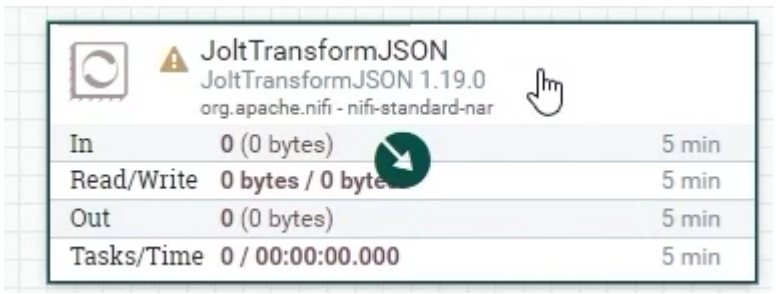


Данный файл мы не можем отформатировать, однако по структуре видно, что это JSON-формат, так как current type все еще Xml (это нужно поменять, но это не первоочередно):



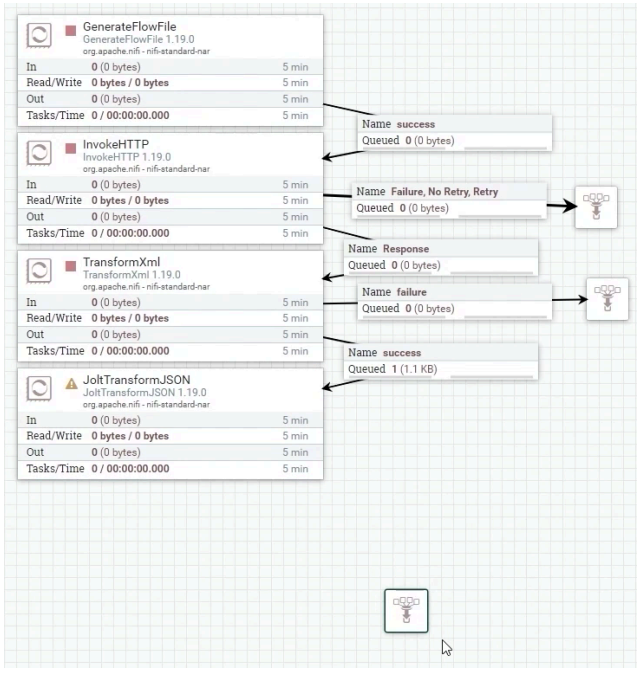
Приступим к трансформации JSON-файла с помощью процессора «JoltTransformJSON»:



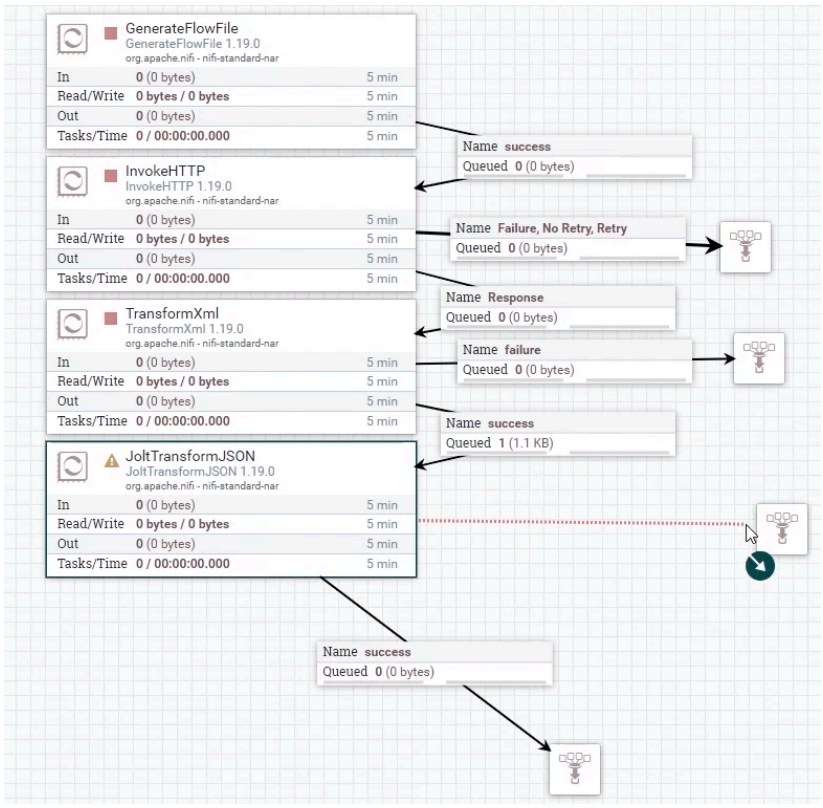


JoltTransformJSON – мощный и часто используемый процессор. Язык Jolt многим сложен для понимания, его нужно изучать отдельно, поэтому воспользуемся готовой наработкой.

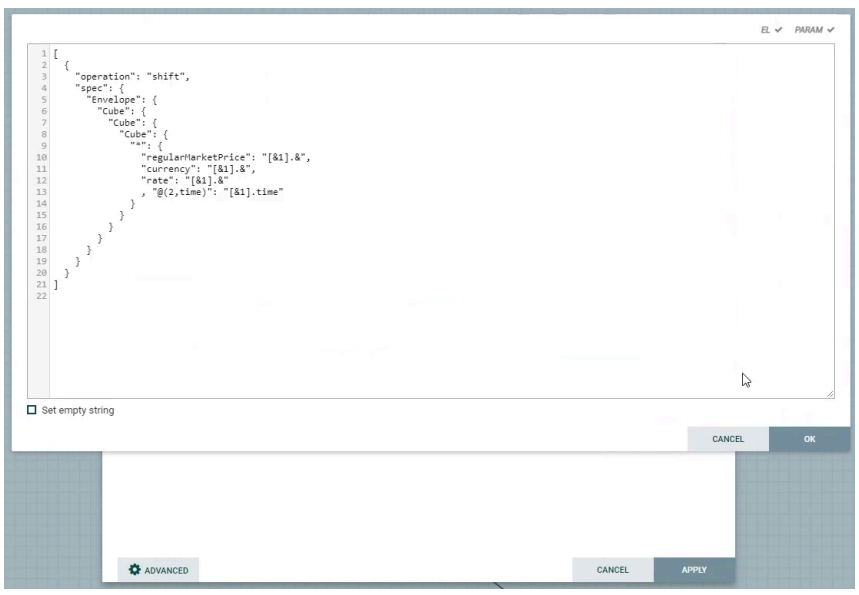
В данном случае мы соединяем «Transform Xml» с «JoltTransform» (воронку отводим в сторону):



Заранее соединим Success с воронкой. На период отладки могут возникать ошибки, отводим их к соответствующей воронке:



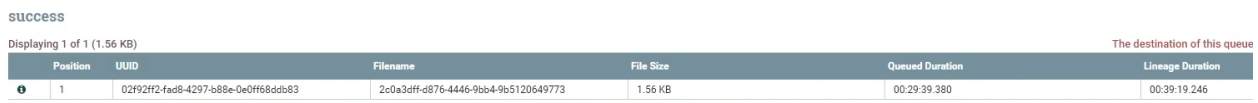
Для данного процессора важна спецификация, выглядит заготовка следующим образом:



Кратко: есть структура в JSON, по ней мы доходим до данных, которые представим в виде плоских таблиц (у нас будет массив, содержащий элементы «Название валюты», «Время», «Кодировка»).

Данный файл также будет приложен к материалам урока в следущем шаге.

Как мы видим, процессор стал валидным. Запускаем его и смотрим на результат:



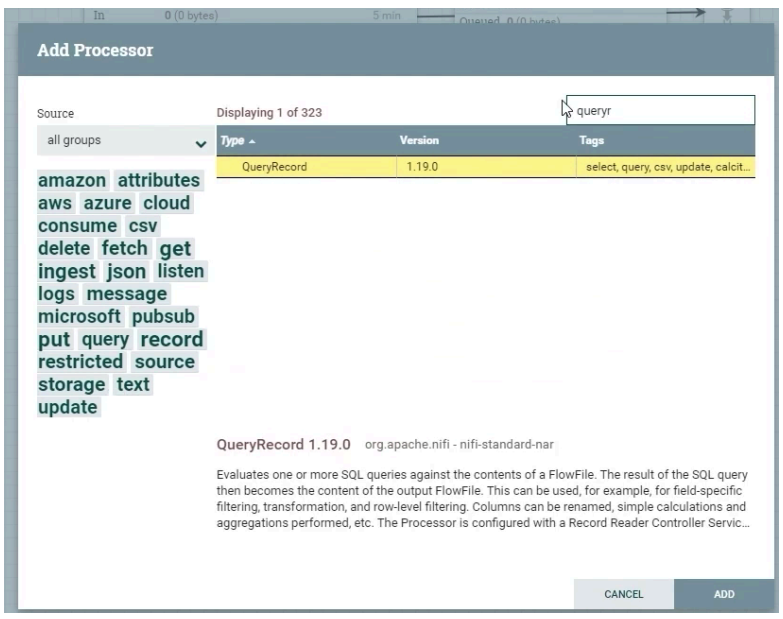
Чтобы посмотреть данные, нажимаем справа на иконку с «глазиком». Теперь в форматированном варианте можем посмотреть данные:

View as: formatted

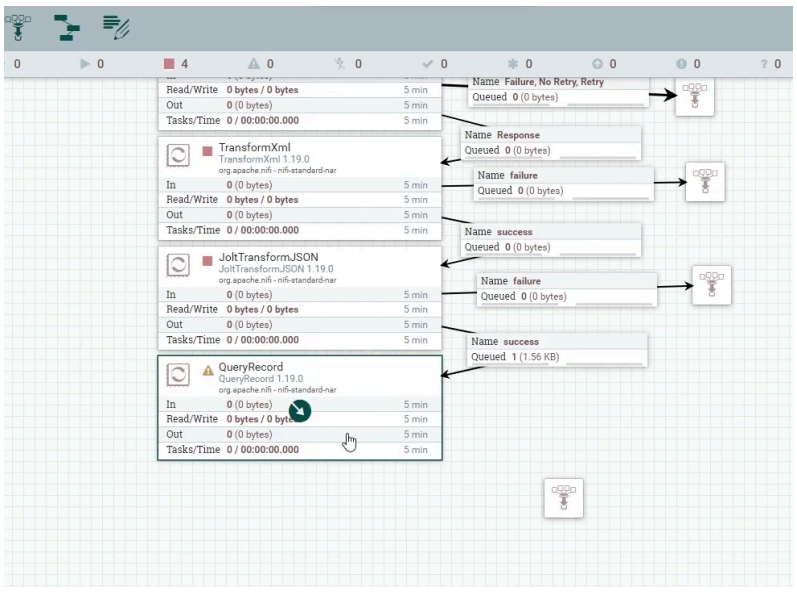

```
[ {
  "time" : "2023-06-02",
  "currency" : "USD",
  "rate" : 1.0763
}, {
  "time" : "2023-06-02",
  "currency" : "JPY",
  "rate" : 149.46
}, {
  "time" : "2023-06-02",
  "currency" : "BGN",
  "rate" : 1.9558
}, {
  "time" : "2023-06-02",
  "currency" : "CZK",
  "rate" : 23.657
}, {
  "time" : "2023-06-02",
  "currency" : "DKK",
  "rate" : 7.4488
}, {
  "time" : "2023-06-02",
  "currency" : "GBP",
  "rate" : 0.8593
}, {
  "time" : "2023-06-02",
  "currency" : "HUF",
  "rate" : 371.36
}, {
  "time" : "2023-06-02",
  "currency" : "PLN",
  "rate" : 4.4975
}, {
  "time" : "2023-06-02",
  "currency" : "RON",
  "rate" : 4.9622
}, {
  "time" : "2023-06-02",
  "currency" : "SEK",
  "rate" : 11.5505
}, {
  "time" : "2023-06-02",
  "currency" : "CHF",
  "rate" : 0.9758
}, {
  "time" : "2023-06-02",
  "currency" : "ISK",
  "rate" : 150.1
}, {
  "time" : "2023-06-02",
  "currency" : "NOK",
  "rate" : 11.845
}, {
  "time" : "2023-06-02",
  "currency" : "TRY",
  "rate" : 22.4742
}, {
  "time" : "2023-06-02",
  "currency" : "AUD",
  "rate" : 1.6248
}, {
  "time" : "2023-06-02",
  "currency" : "BRL",
  "rate" : 5.3752
}, {
  "time" : "2023-06-02",
  "currency" : "CAD",
  "rate" : 1.4443
}, {
  "time" : "2023-06-02",
  "currency" : "CNY",
```


Перед нами JSON-массив со множеством элементов, объектов, каждые из которых содержат код валюты, дату и котировки. Однако нас интересуют сами данные. Их мы будем упрощать.

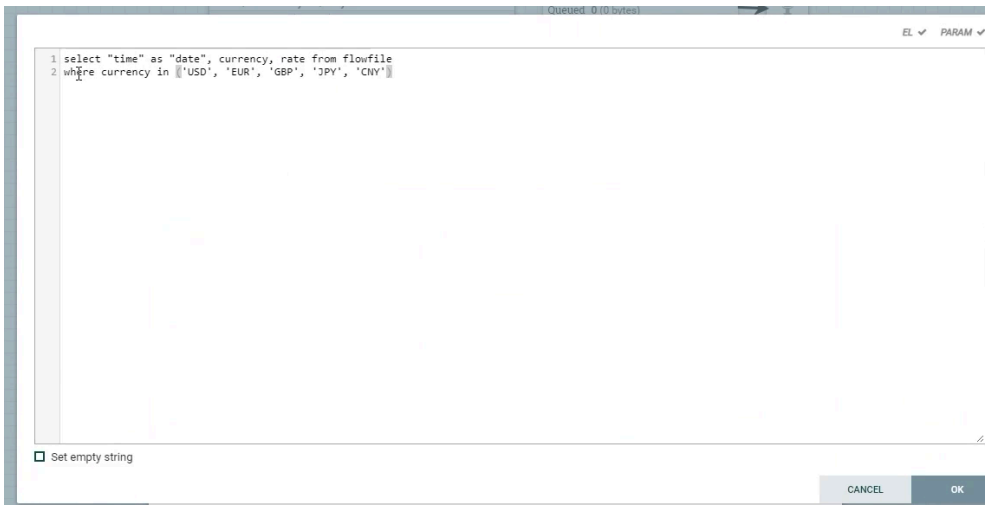
Чтобы упростить данные, воспользуемся процессором «QueryRecord»:



Перетаскиваем его на рабочий стол, выбираем из списка (используйте фильтр), соединяем процессор и линкуем его к новому процессору:



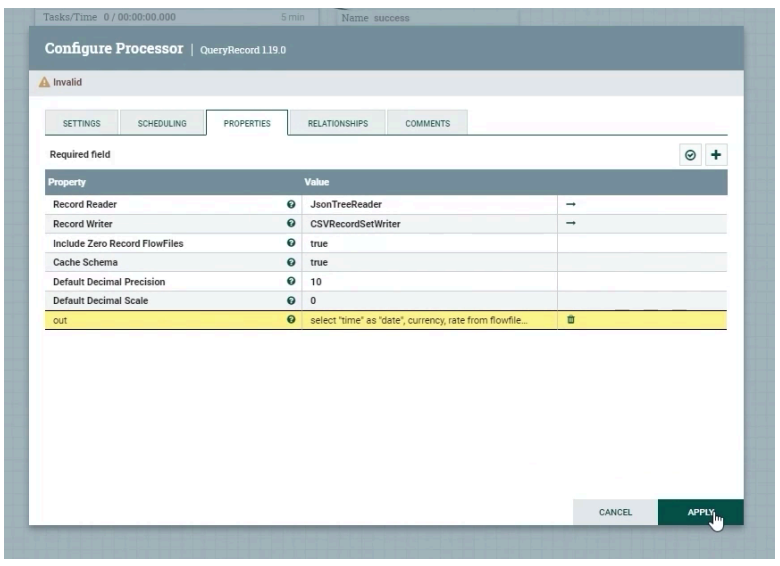
Добавляем еще одну воронку, чтобы пускать туда поток ошибок. Задаем QueryRecord новый атрибут, который назовем «Out». Это будут наши выходящие данные, которые пойдут из этого процессора. Для них будет использоваться SQL-конструкция следующего вида:



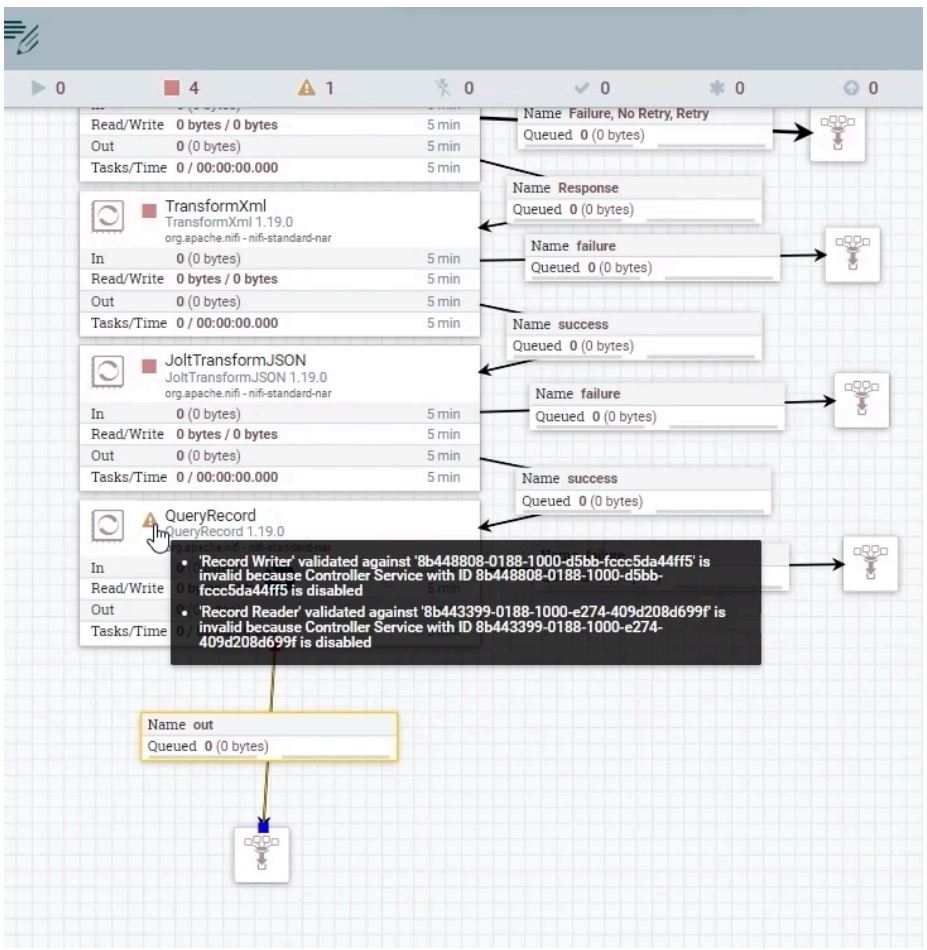
Здесь находятся атрибуты из FlowFile. В прошлом шаге мы смотрели на JSON, там был ключ «Time», мы же хотим, чтобы ключ имел название «Data». Для этого пишем «Time as Data». **Обратите внимание, что они заключены в двойные кавычки. Это особенность синтаксиса. Это служебные слова, которые нельзя оставлять без кавычек.**

В данном select'е вы можете видеть «from FlowFile». FlowFile выступает в виде таблицы, которая возникла после JoltTransform. Из нее мы делаем select и указываем из знакомого SQL-синтаксиса, что currency должен быть в перечисленном списке.

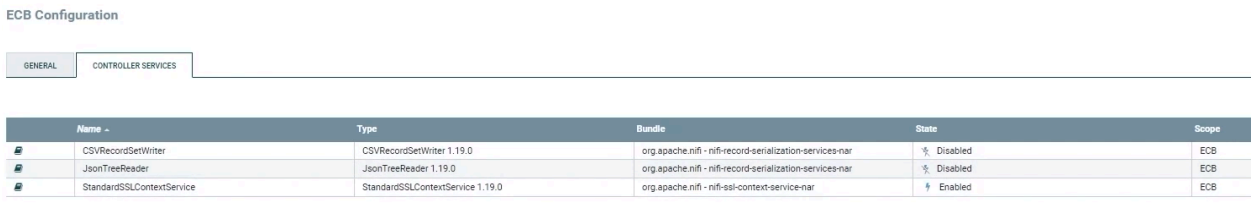
Теперь зададим два сервиса. Так как мы знаем, что на входе идут JSON-данные, создаем новый RecordReader, который будет называться «JSON3Reader». Теперь нам нужно представить данные в виде CSV. Мы задаем еще один новый сервис RecordWriter с названием «CSVRecordSetWriter» и нажимаем Apply:



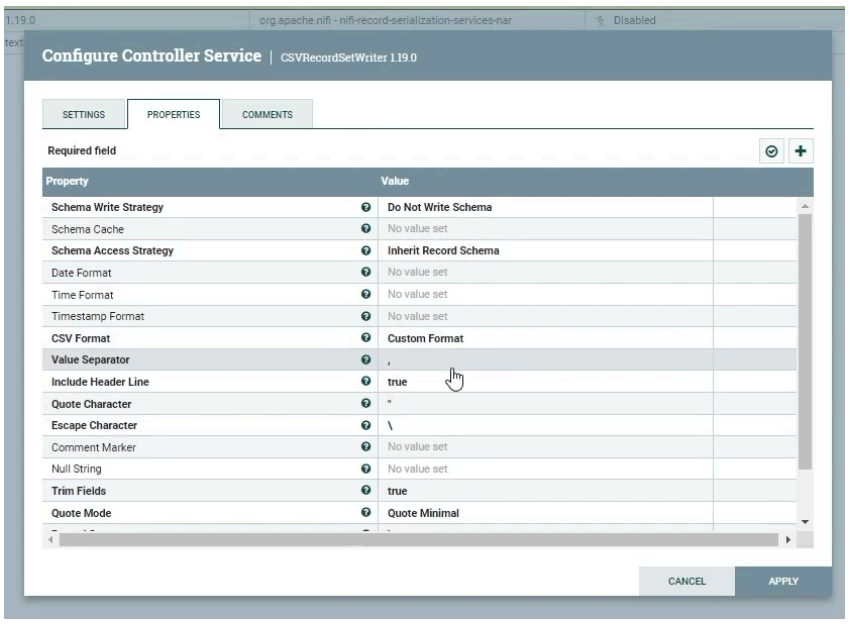
Соединяем наш «Out» с процессором, используя воронку. Снова видим, что процессор невалидный. Причина – процессор находится в статусе «Disabled»:



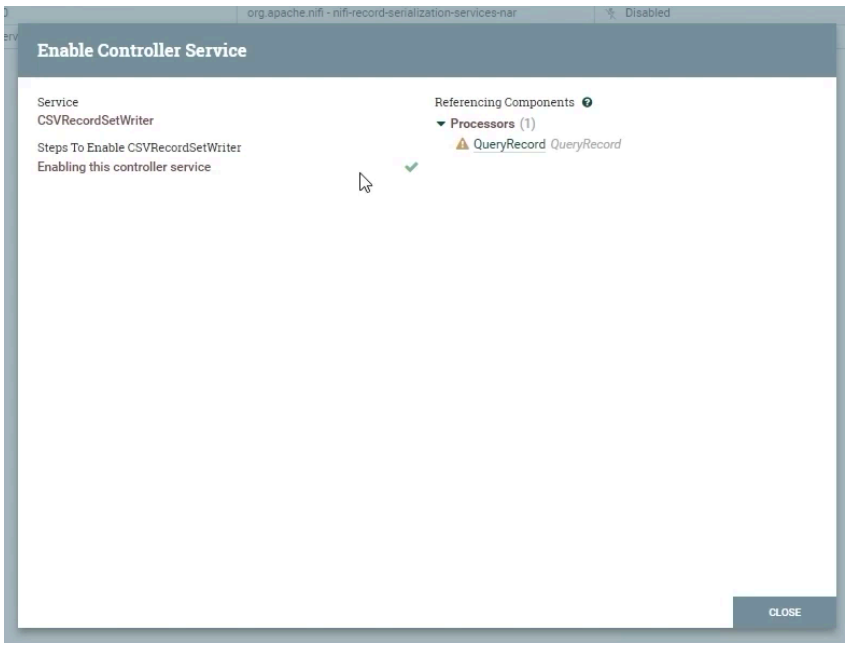
Через процессор следуем за сервисами. Помним, что сервисы **также доступны через конфигурацию** (есть иконка в виде молнии, которая позволяет сделать сервис активным):



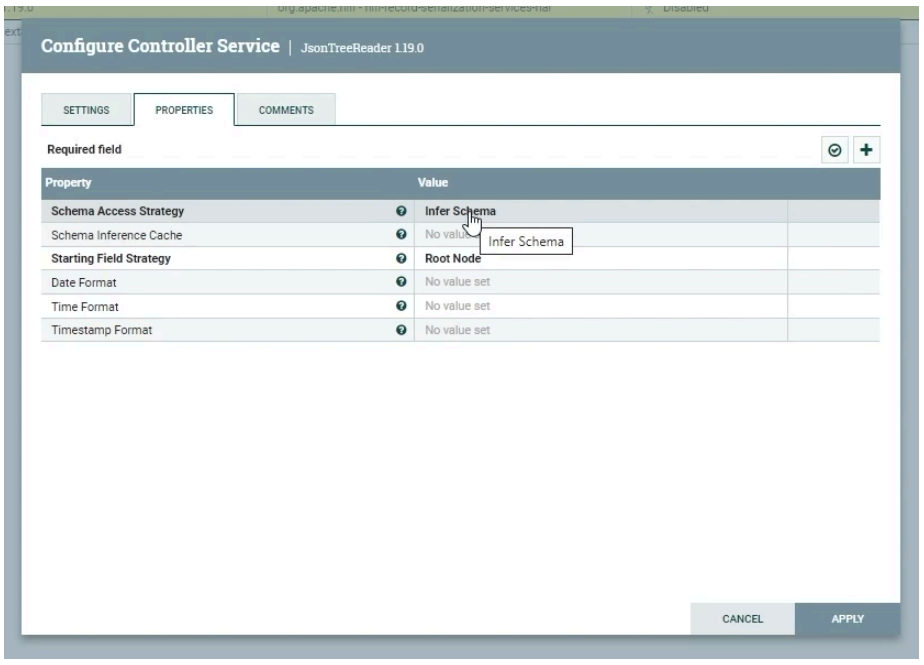
Итак, начинаем с верхнего по списку. Важно понимать, что у CSV Writer есть дефолтные настройки, но нам надо обратить внимание на сепаратор («,» или чаще всего «;»):



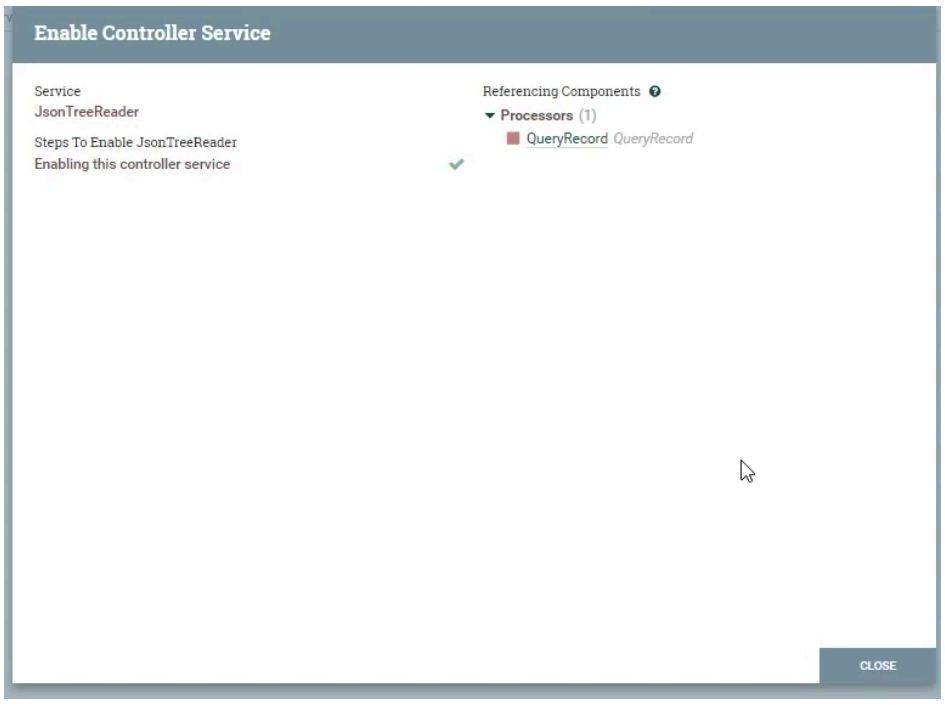
Остальное оставляем как есть. При помощи значка «Молния» делаем процессор активным. Процессор готов к работе:



Следующий процессор – JSONTreeReader. Его важный параметр – стратегия. Мы оставляем дефолтное значение (это значит, что данные будут автоматически распознаны):

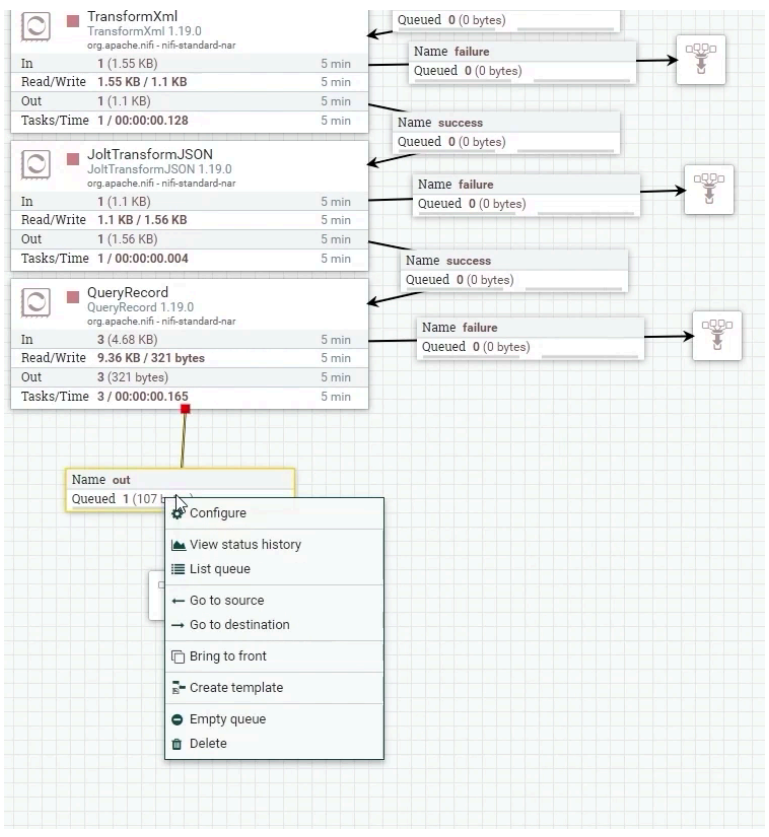


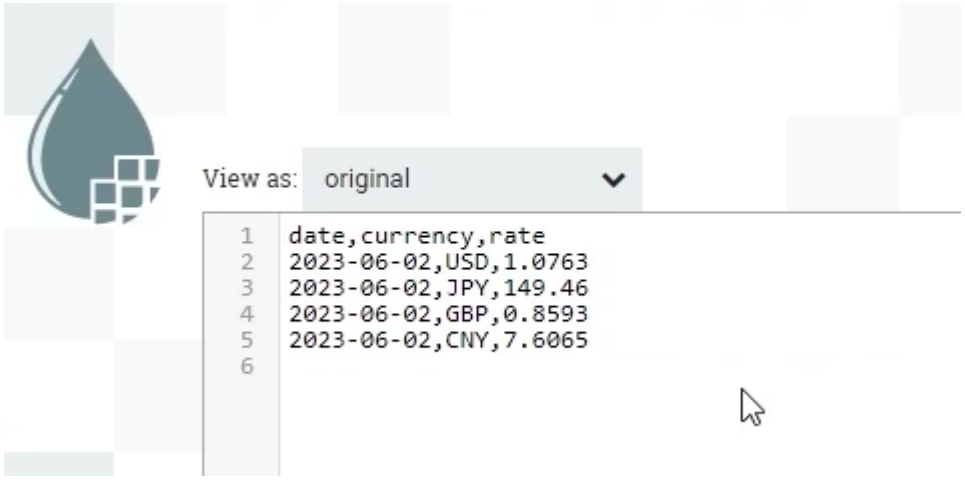
Делаем процессор активным:



Менять настройки процессора можно только в статусе «Disabled».

Закрываем окно со свойствами сервисов и смотрим на работу процессора. Подтверждаем все изменения и нажимаем на процессоре контекстное меню, ищем Run Once. В итоге получается файл:





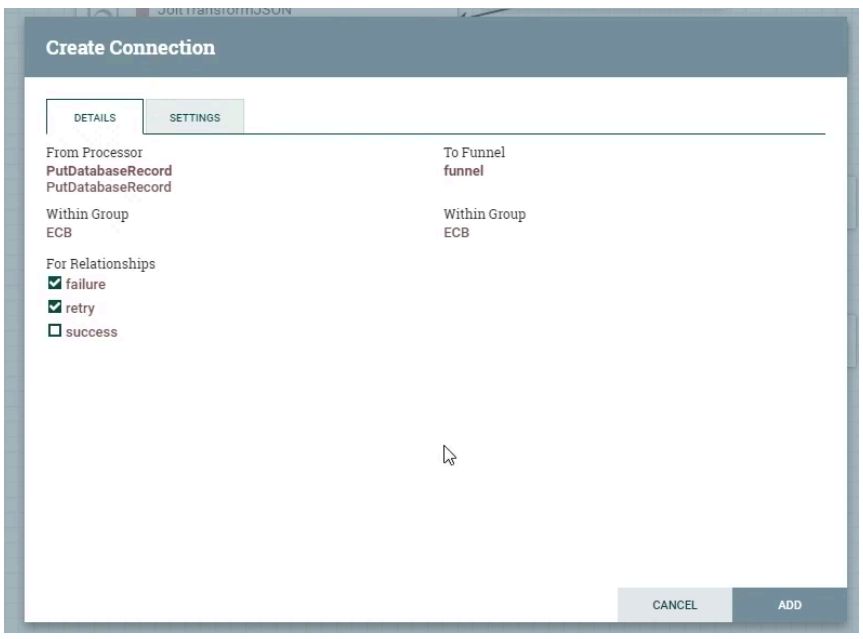
Мы видим, что это CSV-файл с заголовком.

Видим данные:

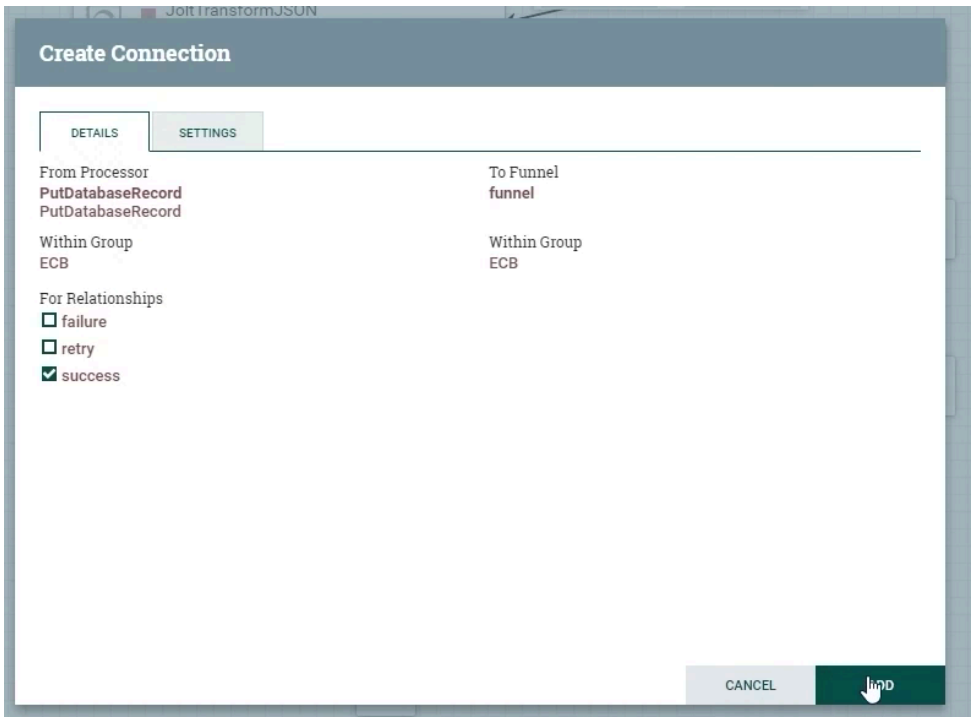
- дата;
- код валюты;
- rate.

Финальный шаг – записать данные в Postgre.

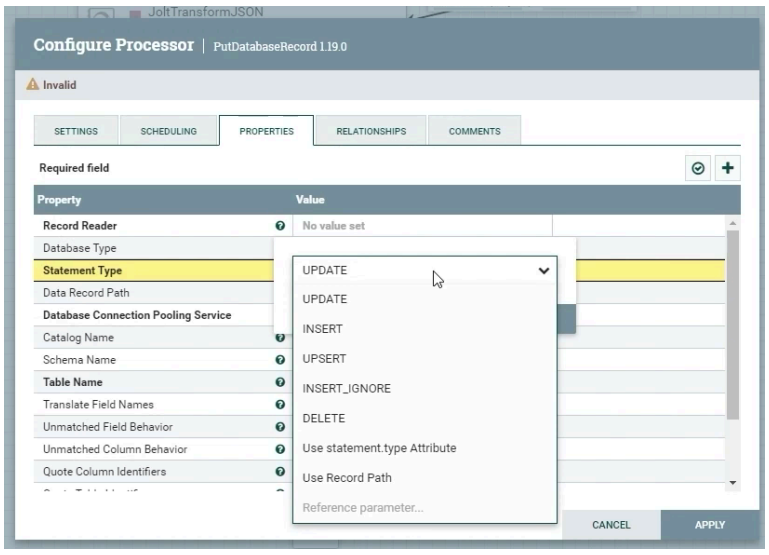
Воспользуемся процессором «PutDatabaseRecord». Соединяем только что созданный процессор с предыдущим, правильно его располагаем. Оттаскиваем в сторону воронку и красиво располагаем процессоры. Сохраняем вывод ошибок на воронке. В ошибку выносим повтор:



Добавим еще одну воронку для успешного вывода процессора:

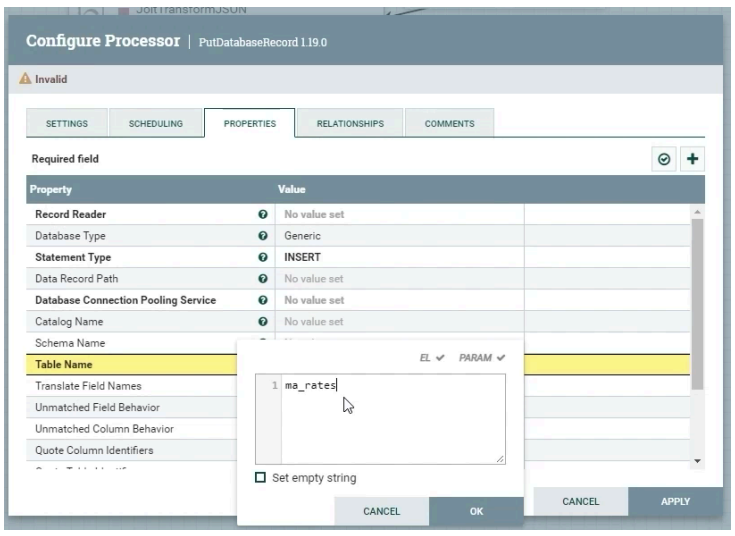


Для PutDatabaseRecord важно то, какие данные будут в него входить. В нашем случае – CSV-файл. Сначала нужно будет задать Reader. Также важно, какой у нас Statement Type (то, как мы будем передавать данные в базу данных). Нас интересует либо INSERT, либо UPSERT:

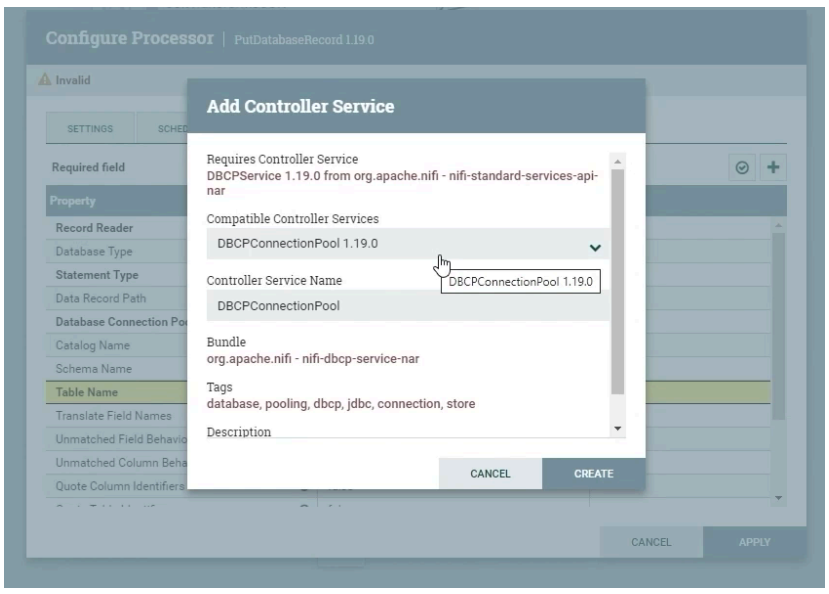


Мы подготовили таблицу, где каждая попытка записи будет записываться в таблицу, соответствующий primary-ключ меняться, а также повышаться индекс на 1, 2, 3.

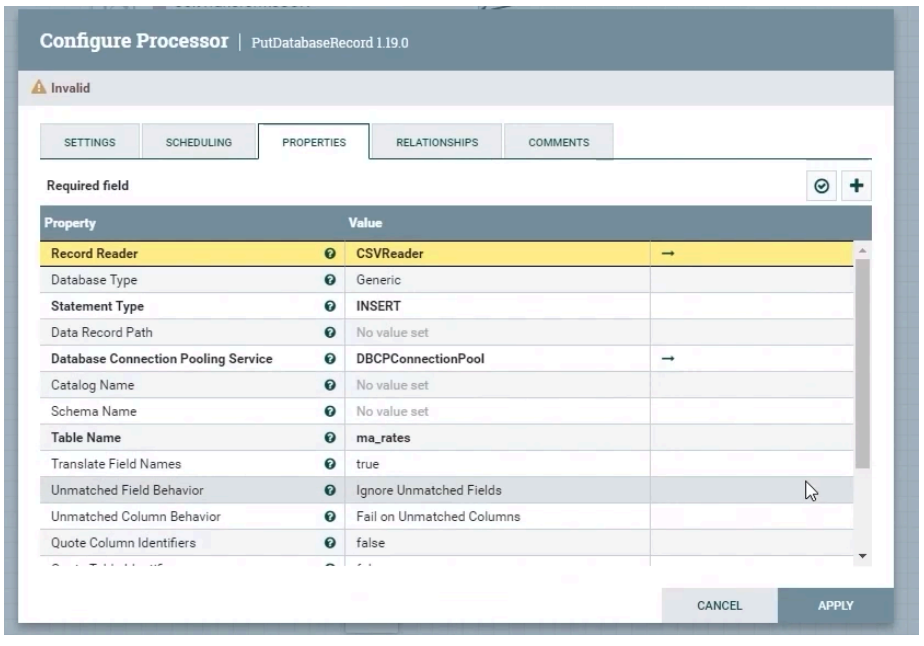
Мы используем стратегию INSERT. Нам нужно записать название таблицы. В нашем случае – «Ma_rates»:



Также, помимо Reader’а, настраиваем соответствующий сервис. Для этого создаем новый сервис:

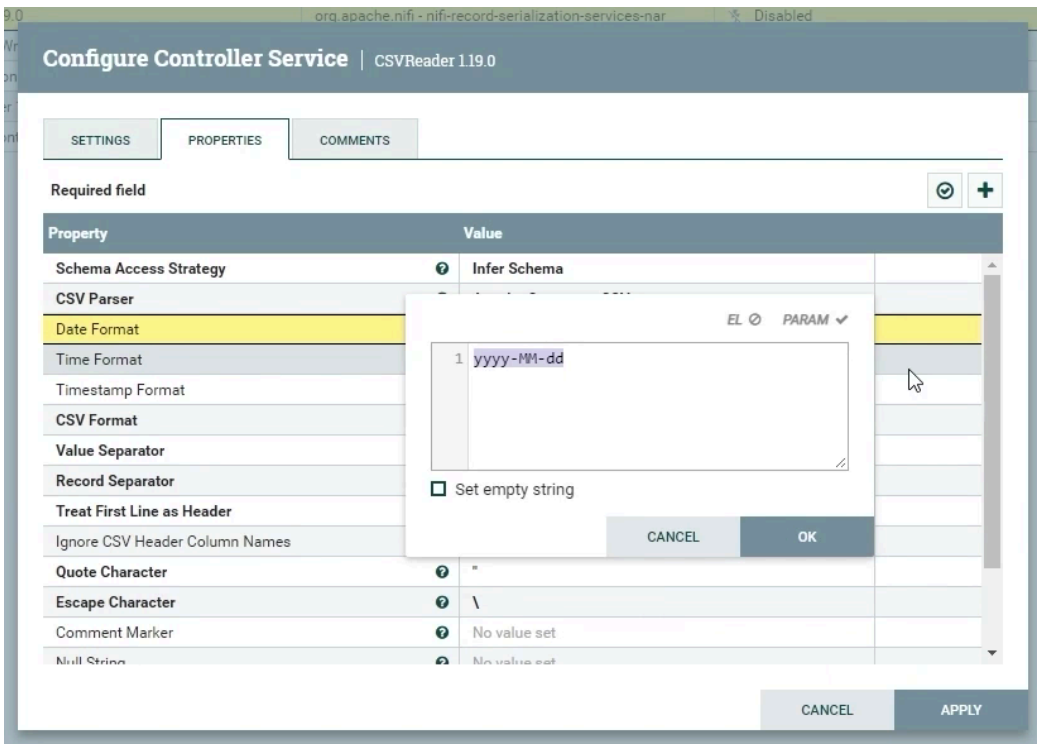


В данном случае – «DBCPConnectionPool». Создаем его и сразу задаем необходимый Reader. Поскольку данные идут в формате CSV, нам нужен соответствующий сервис – «CSVReader»:



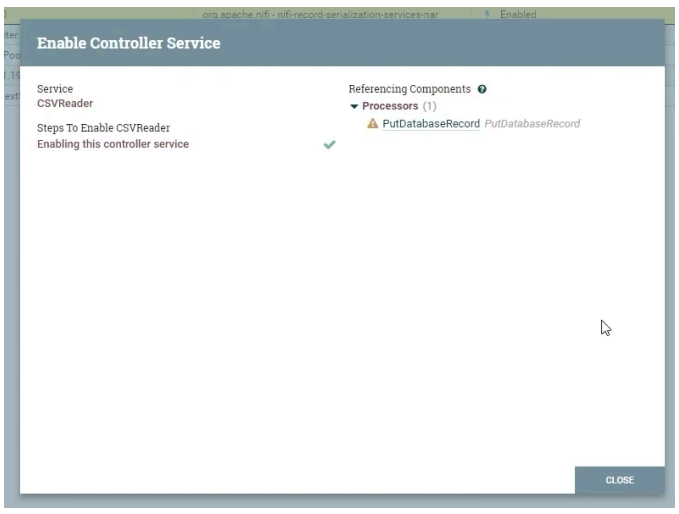
Переходим к настройкам сервисов и сохраняем промежуточные результаты:

Что важно для CSVReader: мы можем использовать «Infer Schema», то есть из заголовка будут браться названия полей. Типы полей будут автоматически распознаны. **Для записи дат необходимо указать маску даты,** иначе мы получим стринг, это не даст записать данные. Маска выглядит следующим образом:

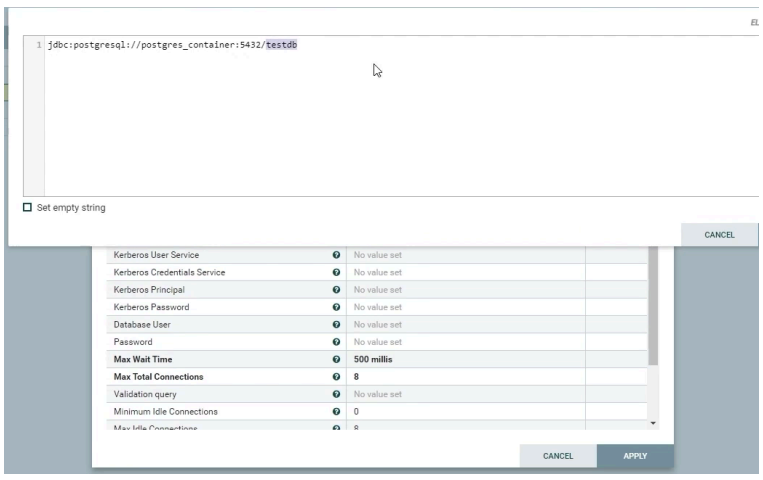


Остальное оставляем по дефолту.

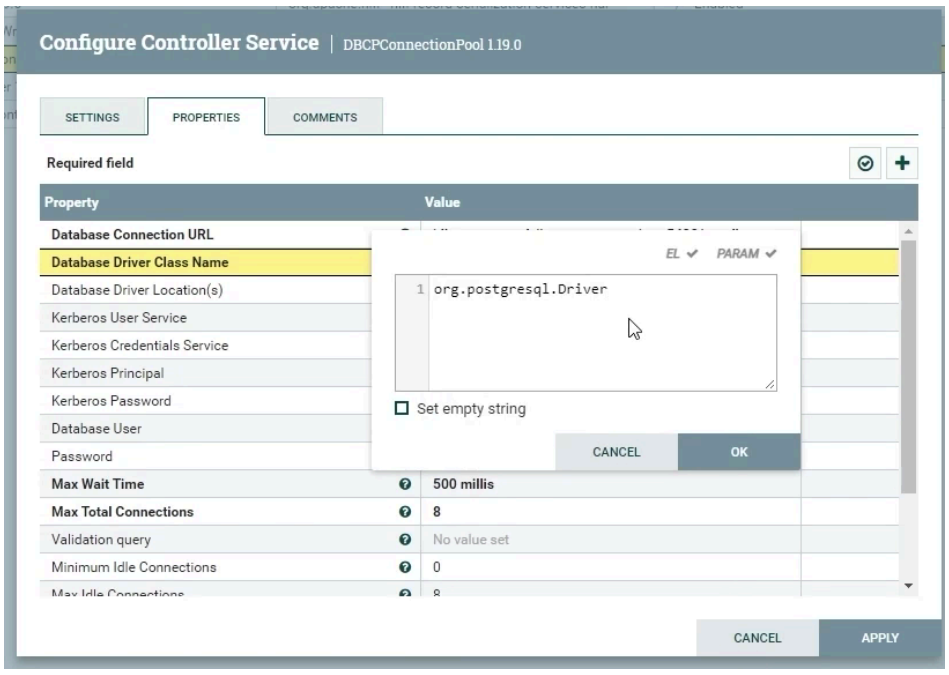
Активируем и получаем зеленую «галочку»:



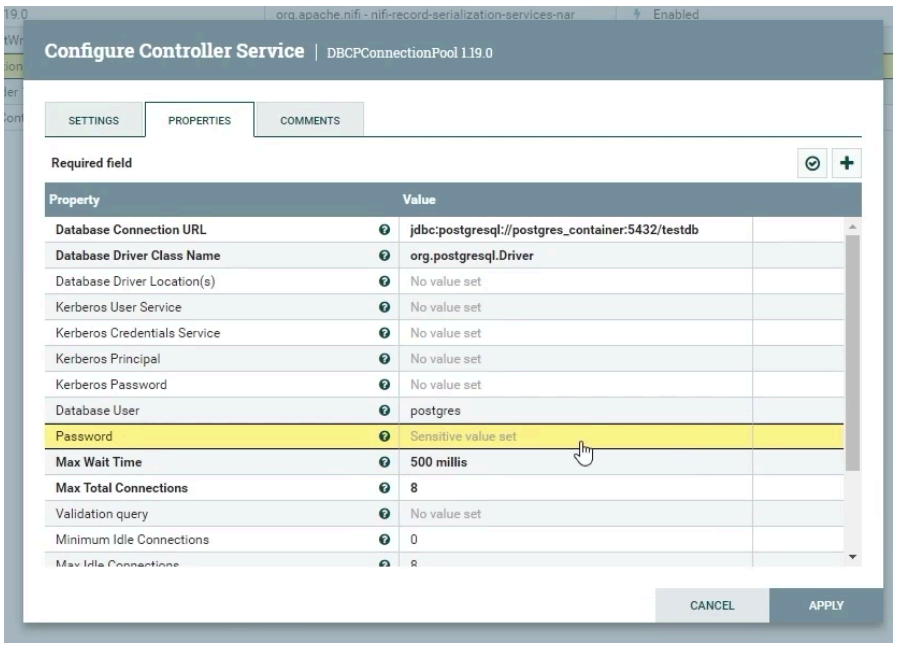
Настраиваем ConnectionPool. В этом случае нам необходим URL для базы данных. Выглядит он следующим образом:

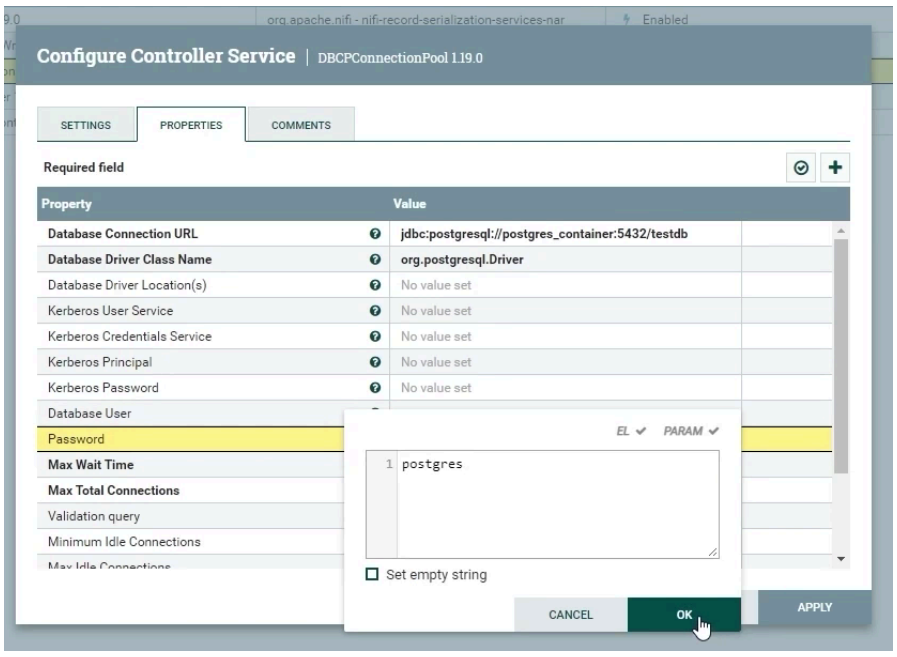


Это jdbc-коннектор для базы данных «Testdb». Далее, нам нужен соответствующий Database Driver Class Name, выбираем его из заготовок. Он будет называться «Driver»:



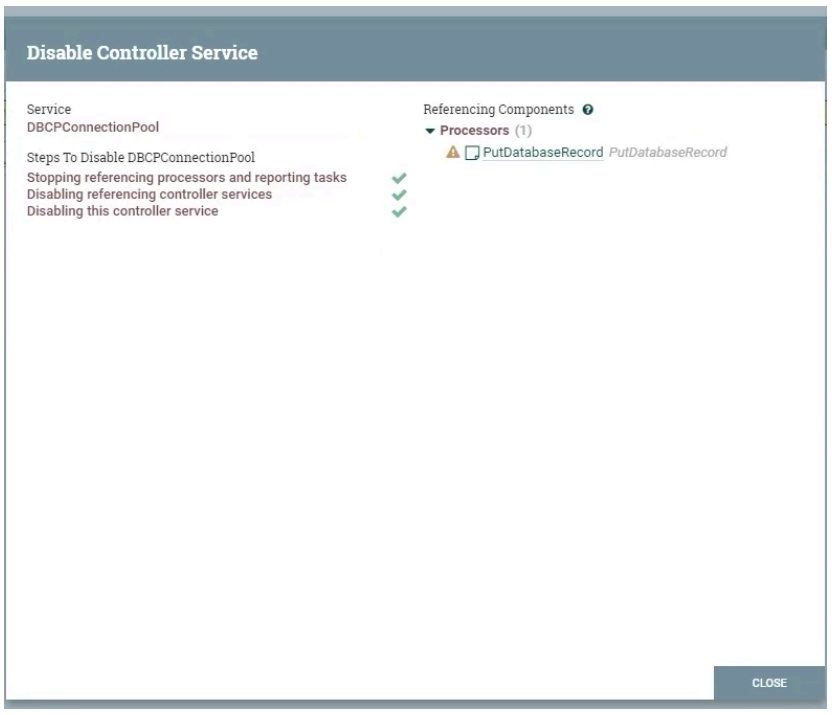
Задаем Database User, который будет «postgres», а также password, который для простоты в нашем примере будет соответствовать названию:



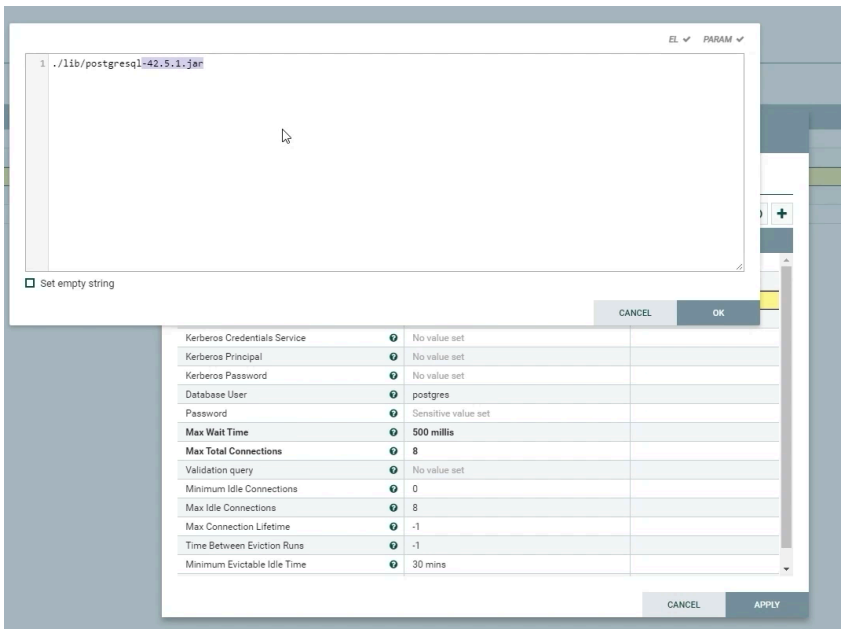


Применяем настройки и делаем сервис активным.

Важно: для данного сервиса необходим драйвер для базы данных postgre. Мы возвращаемся в настройки**.**
Помним, что их нельзя менять, не сделав «disable»:

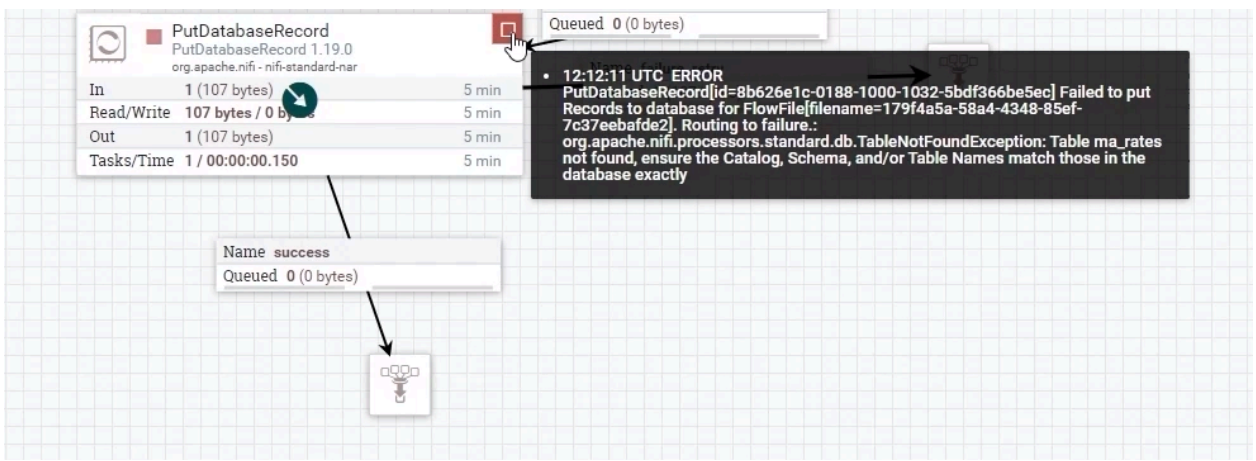


Заходим в конфигурацию и указываем путь к файлу с драйвером. Драйвер скачан с [сайта Postgre SQL](#) и выглядит следующим образом:



В файле указана версия, чего мы делать не рекомендуем. **Версию для NiFi нужно убирать и вставлять символическую ссылку на уровне файловой системы там, где вы положили драйвер**. Можете видеть, что путь указан относительно корневой папке NiFi. Если ваш template передастся с одной системы на другую, а пути не будут соответствовать, то вам не придется переделывать в NiFi DataFlow.

Применяем изменения и делаем сервис с настроенным драйвером активным. Проверяем его отработку, заведомо зная о возможной выдаче ошибки. Этот случай мы и рассмотрим:



Процессор выдал ошибку. Наведя курсор на красный флажок, мы можем ознакомиться с ошибкой.

Можно зайти в раздел через «Гамбургер-меню». Вверху справа есть Bulletin Board, открыв его, можно скопировать сообщение об ошибке и подробно с ней ознакомиться.

Рассмотрим ошибку:

```
12:12:11 UTC
ERROR
8b626e1c-0188-1000-1032-5bdf366be5ec
```

```
PutDatabaseRecord[id=8b626e1c-0188-1000-1032-5bdf366be5ec] Failed to put
Records to database for FlowFile[filename=179f4a5a-58a4-4348-85ef-
7c37eebafde2]. Routing to failure.:
org.apache.nifi.processors.standard.db.TableNotFoundException: Table ma_rates
not found, ensure the Catalog, Schema, and/or Table Names match those in the
database exactly
```

Auto-refresh

Мы можем выявить причины ошибки. Так, например, таблица «ma_rates» не создана, а также содержит в себе опечатку.

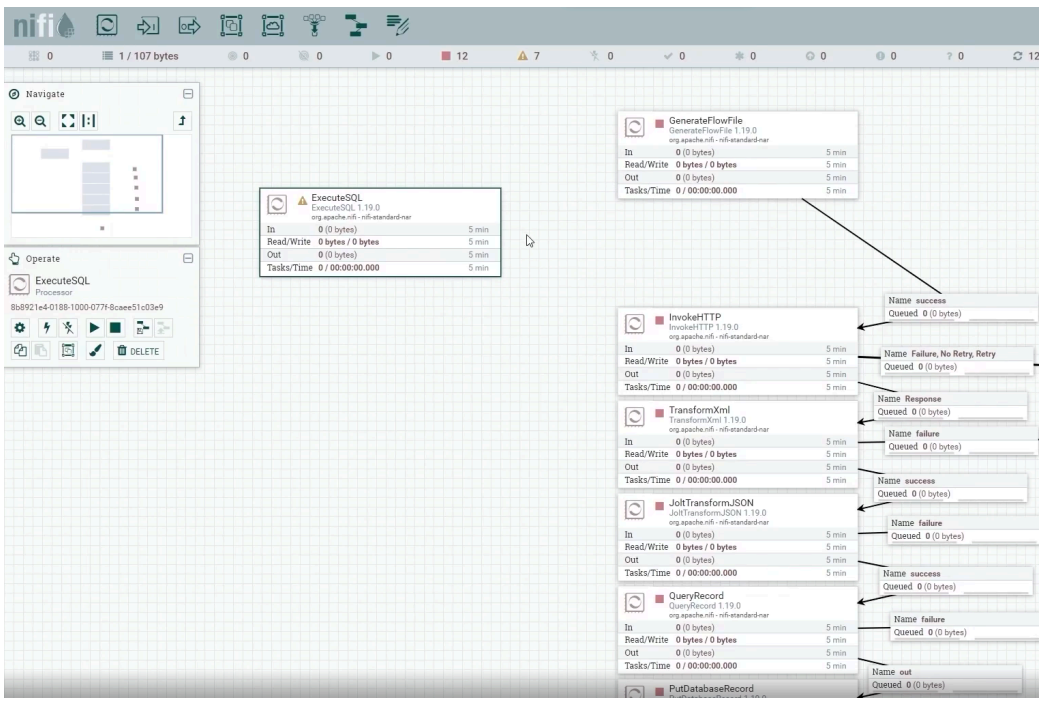
Чтобы создать таблицу, нужно будет выполнить соответствующий скрипт:

```
CREATE TABLE IF NOT EXISTS my_rates2
( id BIGSERIAL primary key
, date date NOT NULL
, currency varchar(3) NOT NULL
, rate decimal);
```

- Создаем таблицу с названием «my_rates2»;
- Primary key будет integer-значение;
- Data будет data;
- Для currency будет varchar(3);
- Для rate decimal.

Выполним эту инструкцию средствами NiFi.

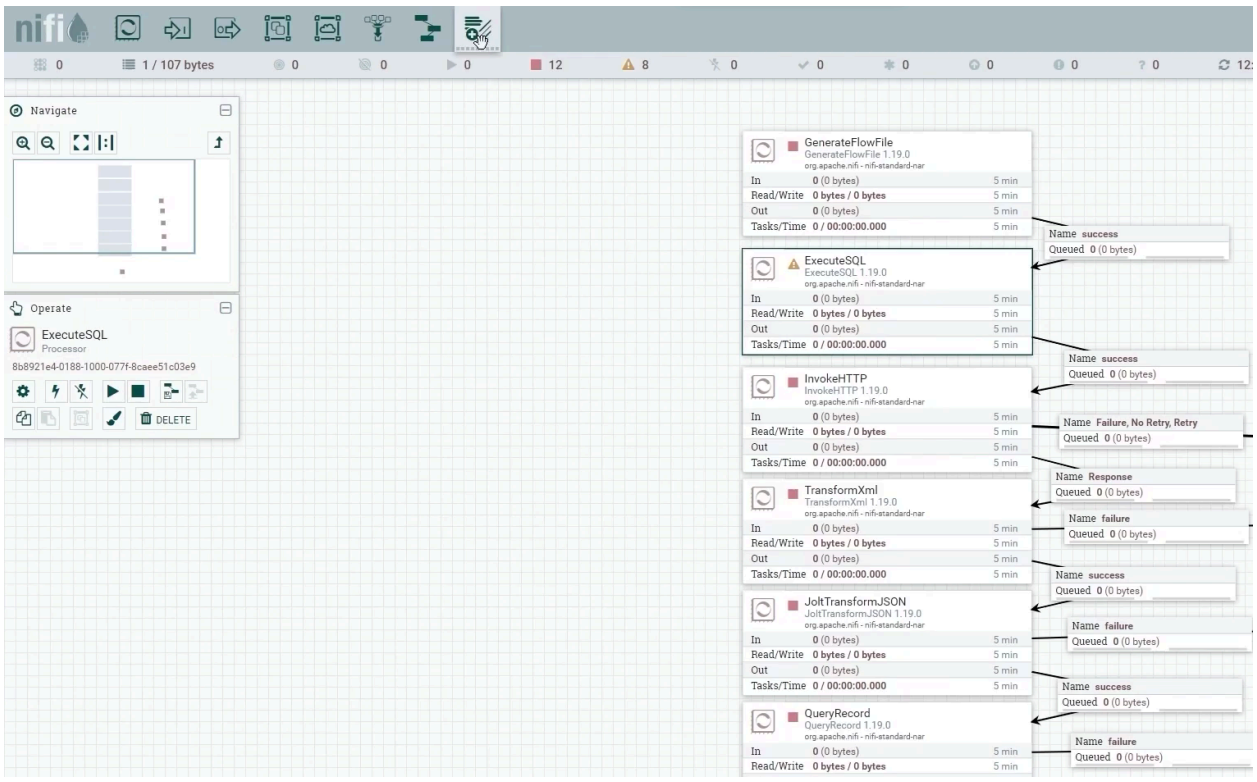
В самое начало для нашего потока мы добавим процессор «ExecuteSQL»:



Вопрос: почему именно такое расположение?

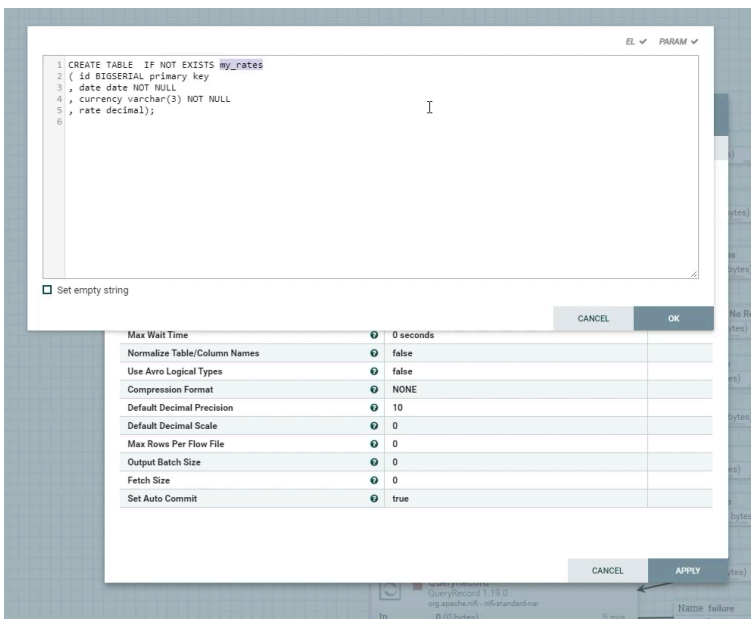
Ответ: после GenerateFlowFile идет InvokeHTTP, который забирает данные извне. Процесс ExecuteSQL тоже заключается в заборе данных из БД и в переписывании FlowFile. Данные БД нас не интересуют, нам важно запустить скрипт, чтобы создать соответствующую таблицу. Именно поэтому данное место для него наиболее оптимально.

Соединяем все как обычно и загибаем должным образом линии:



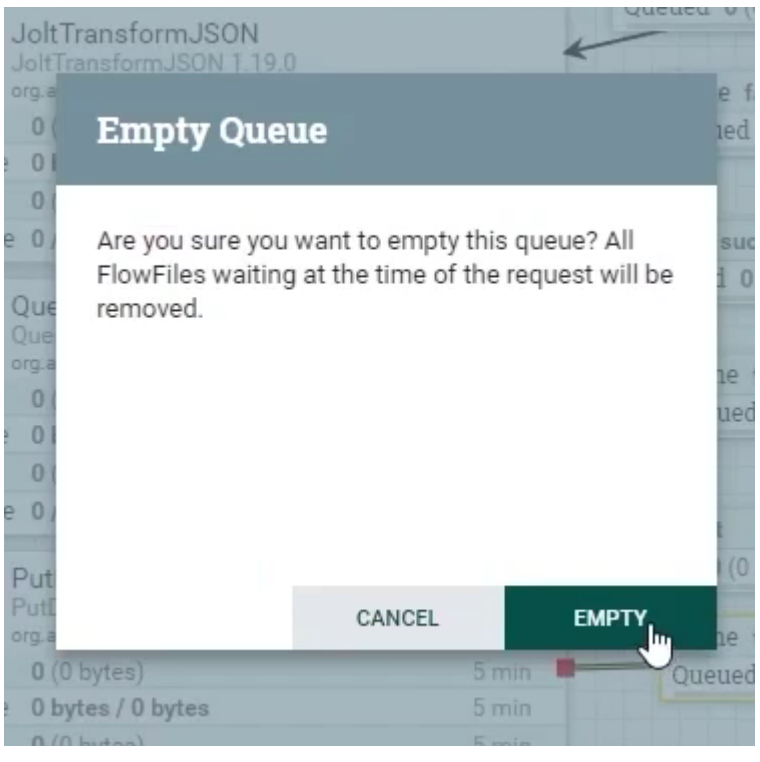
Так как этот процессор тоже может выдать ошибку, мы назначаем ему соответствующую воронку.

Настроим процессор, поскольку он находится в состоянии «Invalid». DBCPConnectionPool уже настроен, поэтому новый создавать не нужно. Также важно в SQL Pre-Query добавить скрипт:

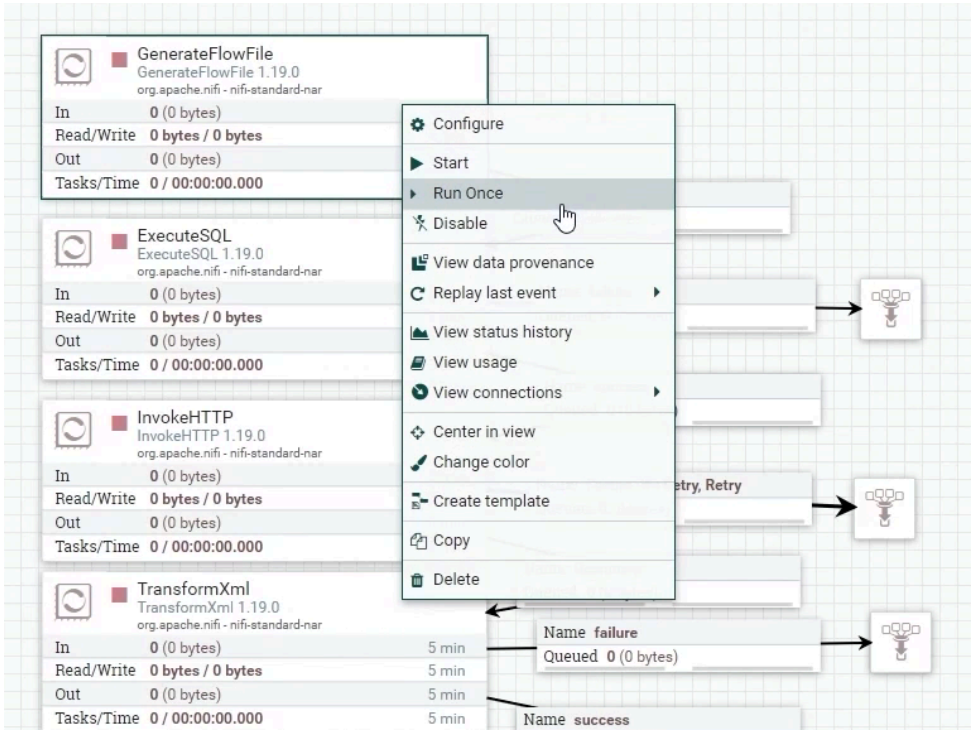


Нажимаем Apply и видим, что процессор валиден.

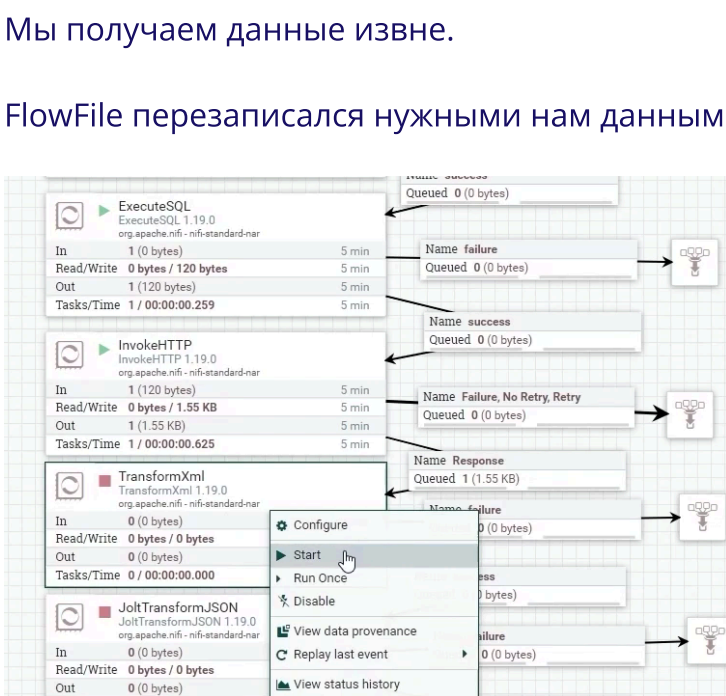
Для удобства сделаем очистку очереди с ошибками:

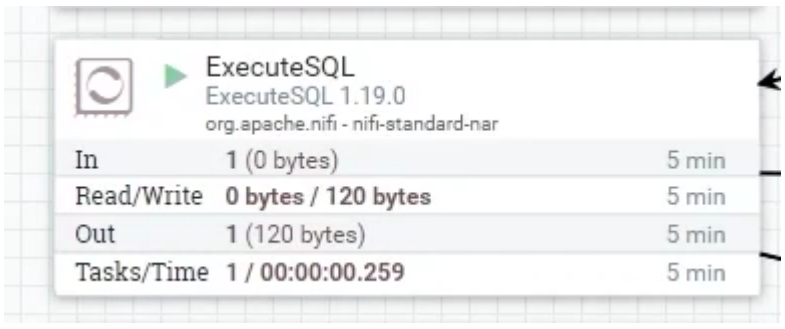


Повторим запуск всего Flow заново. Первый процессор мы выключаем, через Run Once делаем генерацию FlowFile:



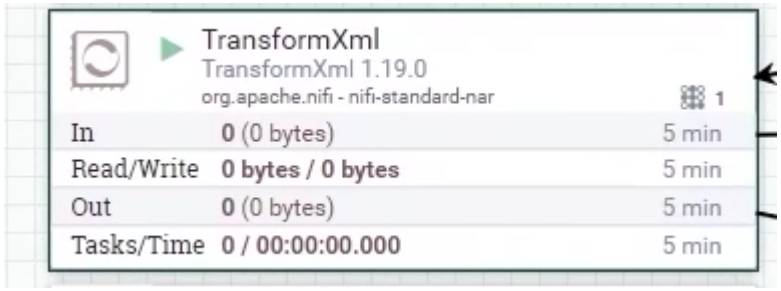
Запускаем ExecuteSQL (благополучно отработал):



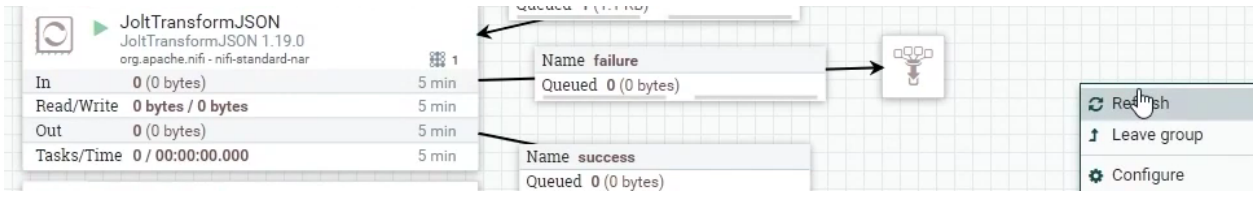


Мы получаем данные извне.

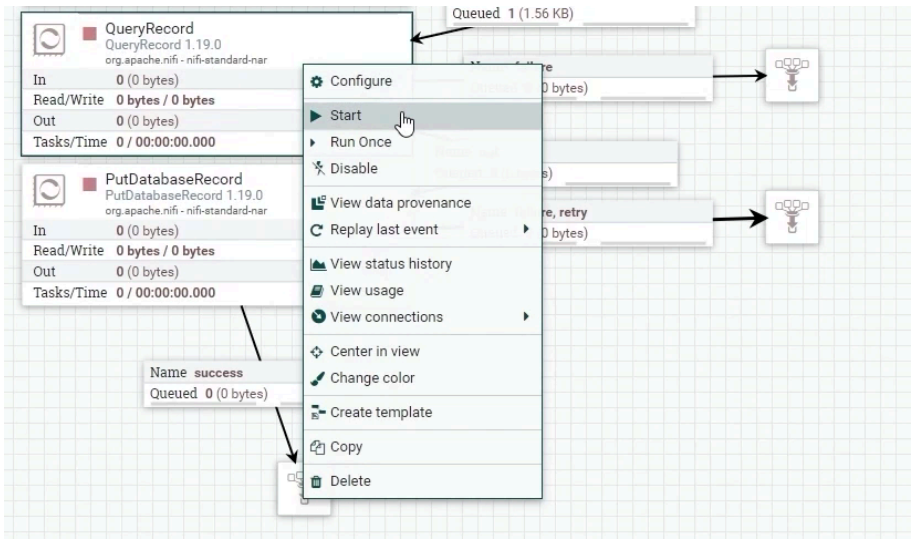
FlowFile перезаписался нужными нам данными; делаем трансформацию их Xml в JSON:

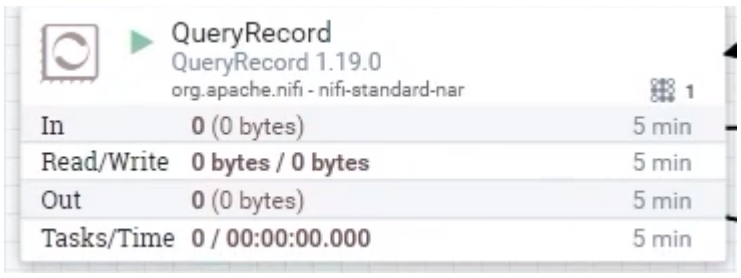


Следующий шаг – преобразование неплоского JSON в плоский:



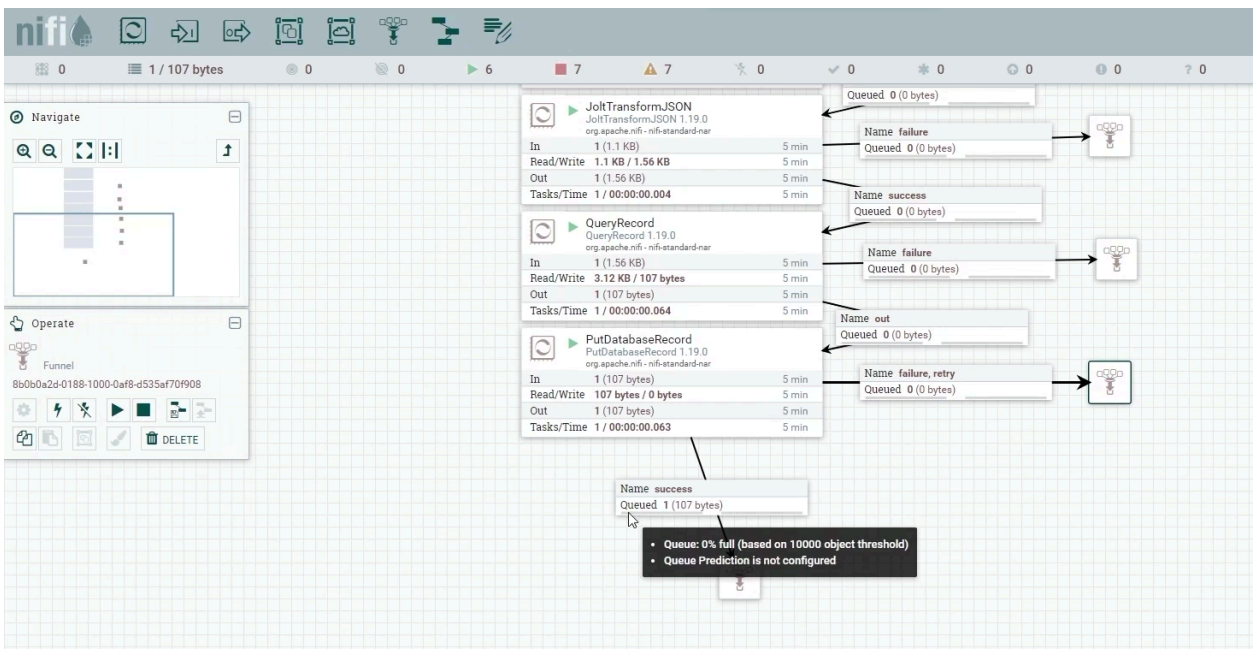
Упрощаем record за счет QueryRecord:





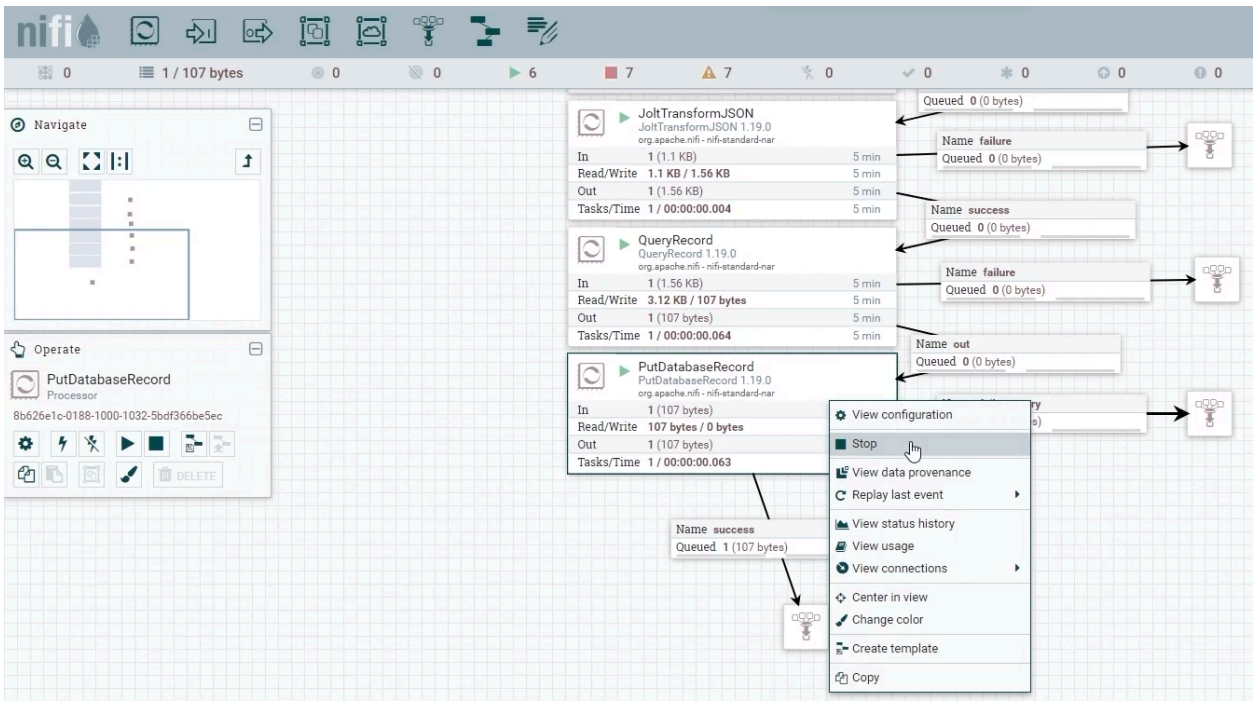
Вносим последние корректировки, устраняем опечатки. Запускаем процессор на Start.

Все сработало без ошибок:

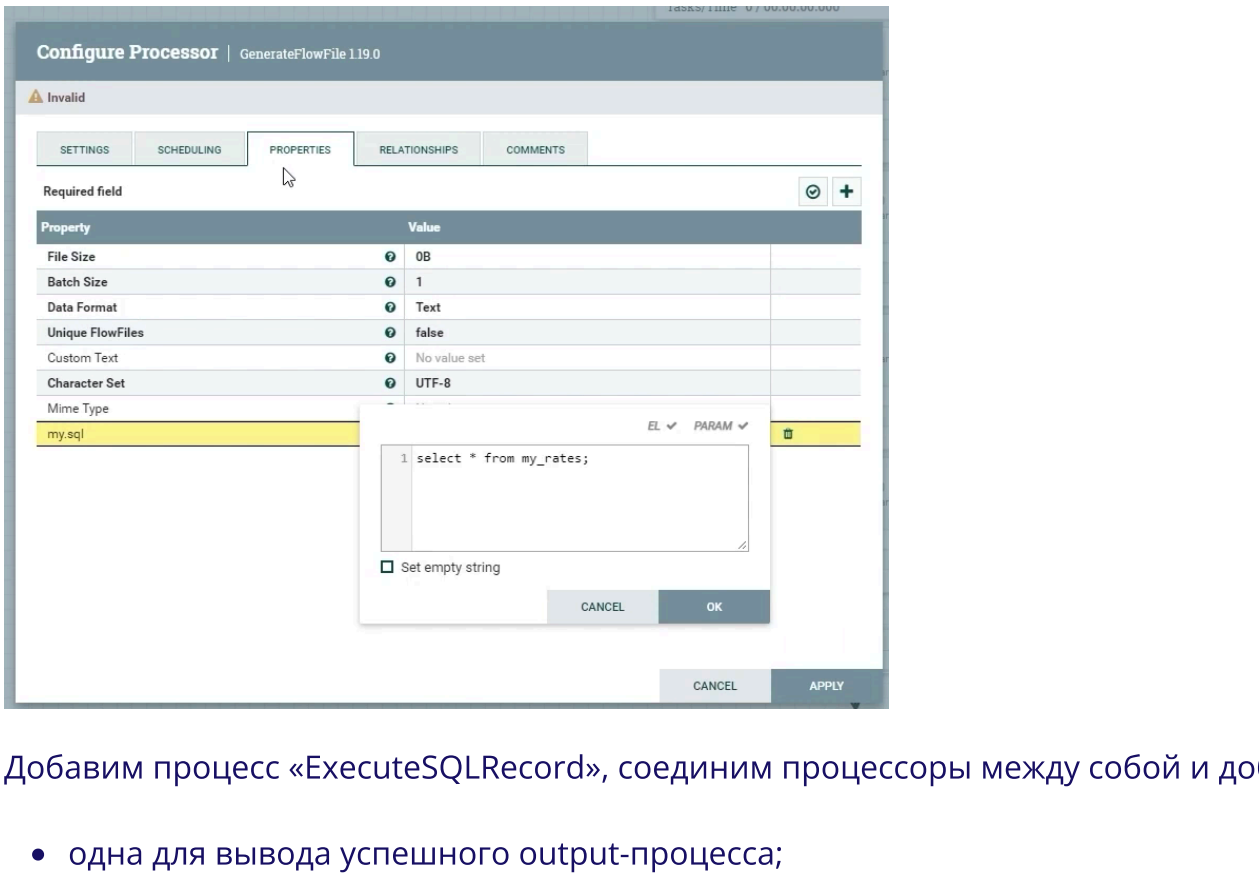


Рассмотрим, как правильно обрабатывать, если все же произошла ошибка.

Процессор можно остановить:



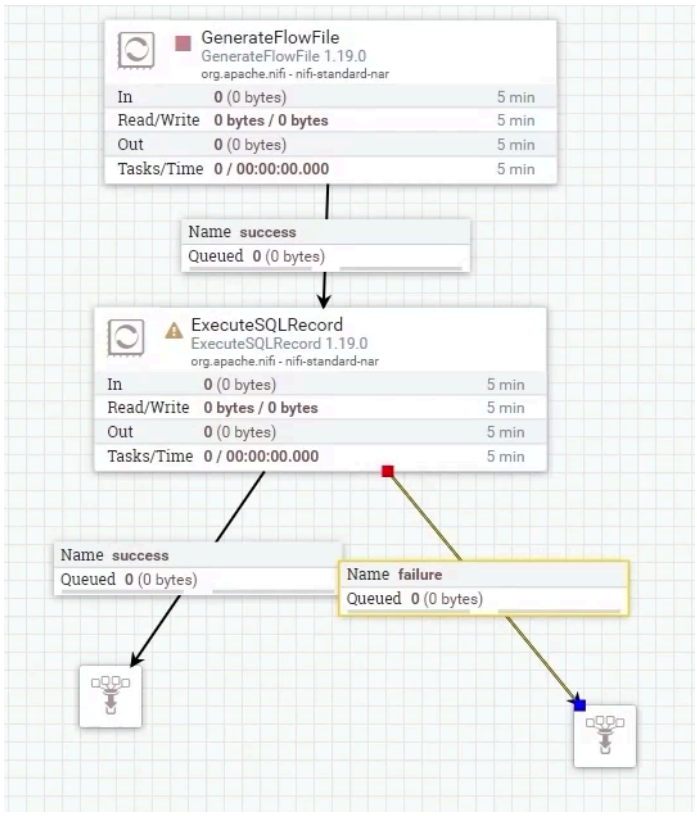
В контекстном меню есть пункт «Replay», повторяем последние действия:



В данном случае данные легли в таблицу, проверим данные средствами NiFi. Для этого сделаем еще один DataFlow. Начнём с GenerateFlowFile, укажем атрибут и назовем его «my.sql». В него мы разместим простой запрос:

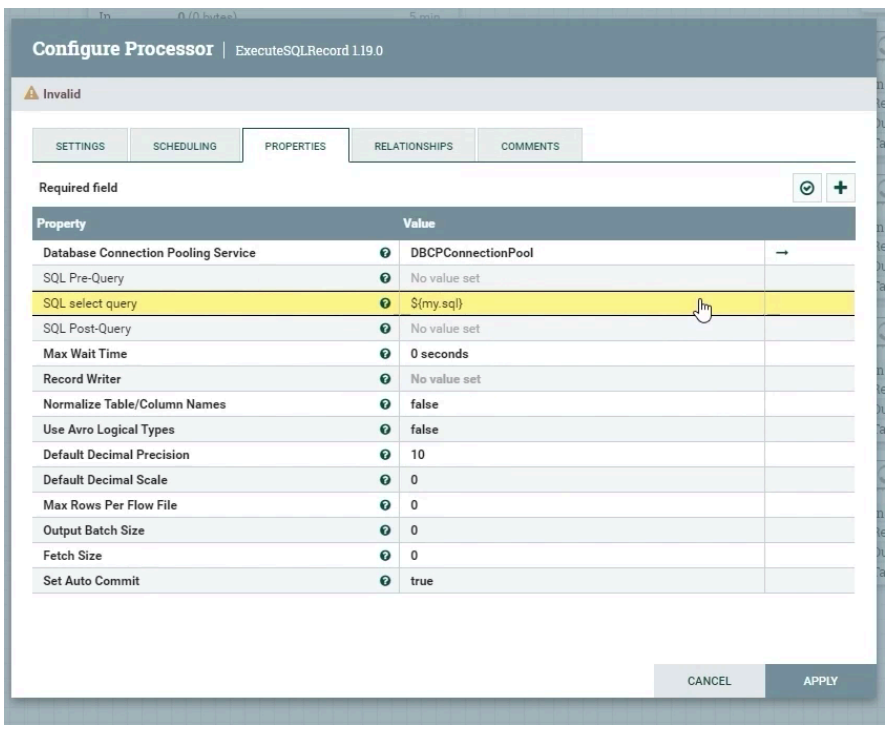
Добавим процесс «ExecuteSQLRecord», соединим процессоры между собой и добавим две воронки:

- одна для вывода успешного output-процесса;
- другая – на случай ошибки:

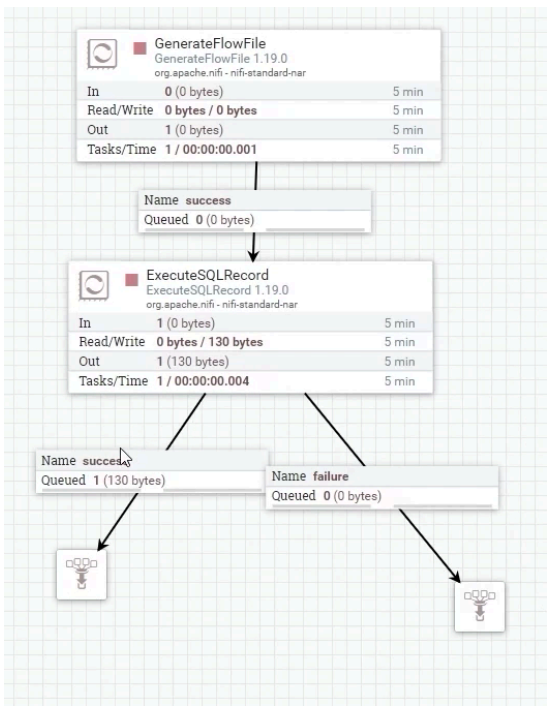


Настраиваем ExecuteSQLRecord. Чтобы он заработал, необходимо установить коннектор, поскольку он уже настроен, используем повторно. **Один сервис можно использовать на несколько процессоров без ограничений.**

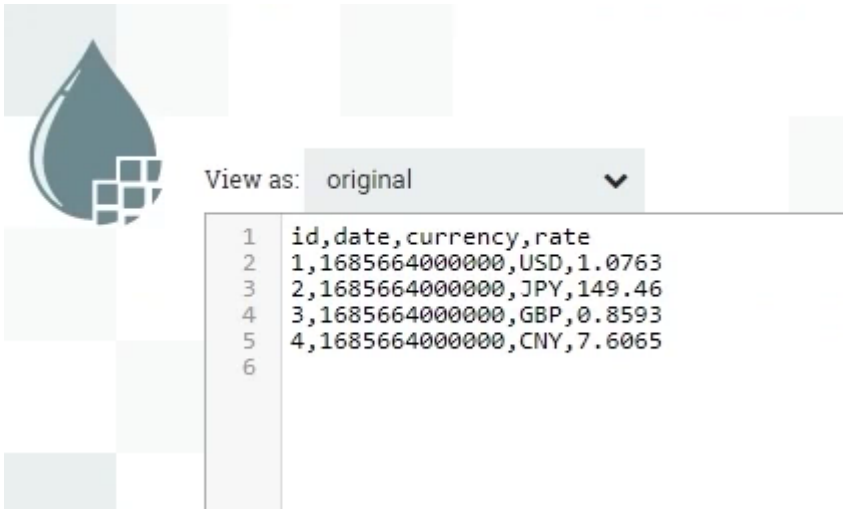
Выполняем запрос, который уже находится в атрибуте, прописываем его следующим образом:



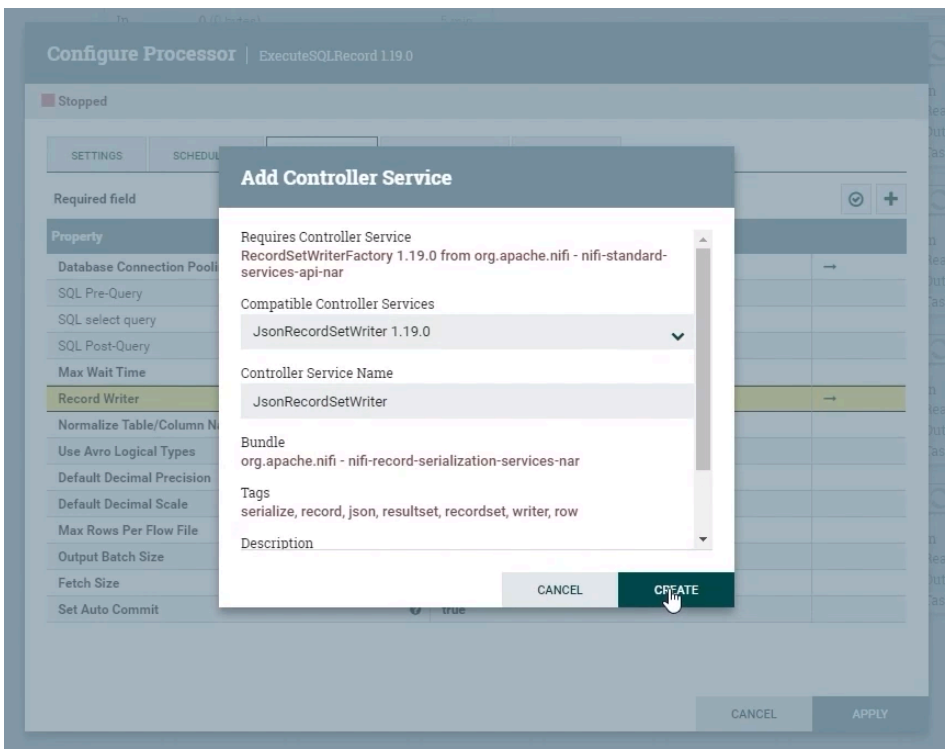
Определимся с форматом выходных данных: в RecordWriter укажем уже имеющийся CSVRecordSetWriter. Проверяем, запуская процессоры, и видим, что данные успешно отработаны:



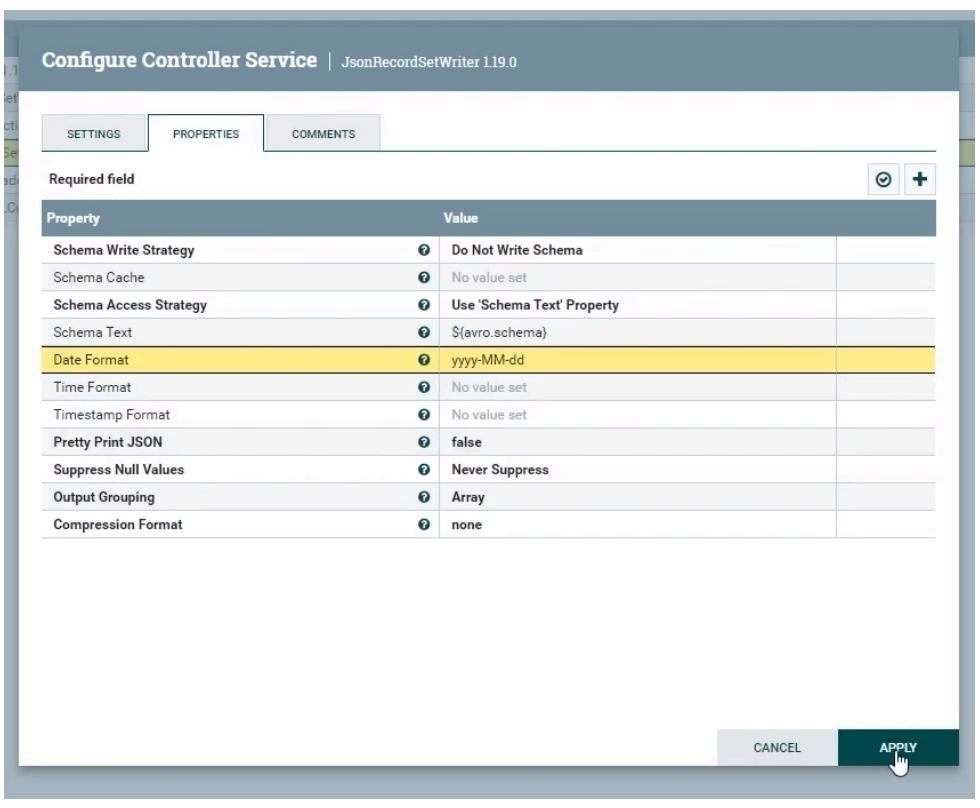
У нас получился CSV-файл:



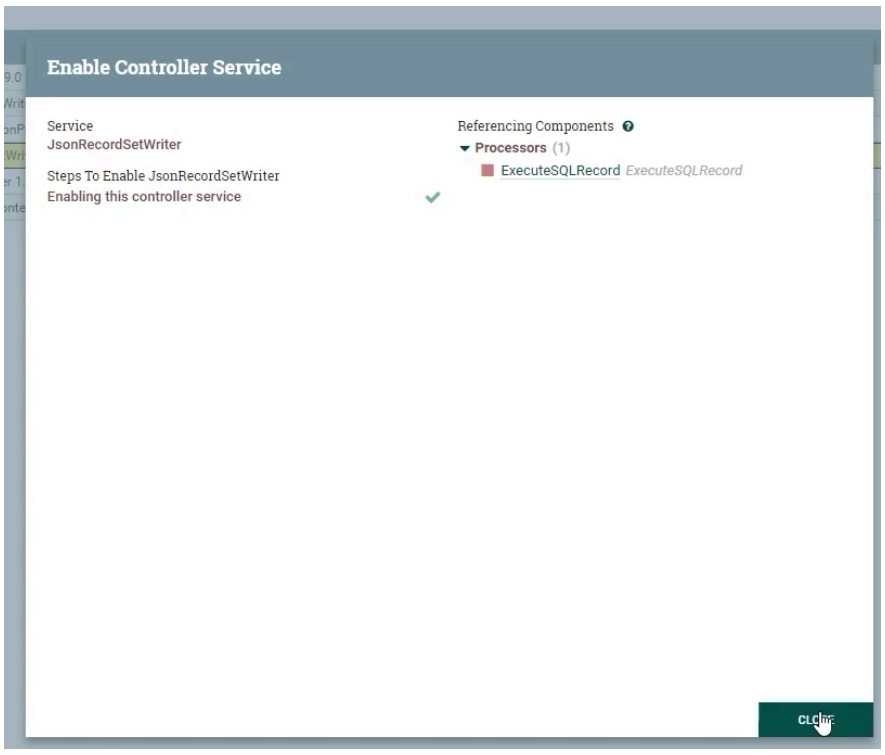
Нюанс: data просто в виде большого числа. Исправим это. Для этого сделаем новый процессор «JsonRecordWriter1.19.0», перейдем в его настройку:



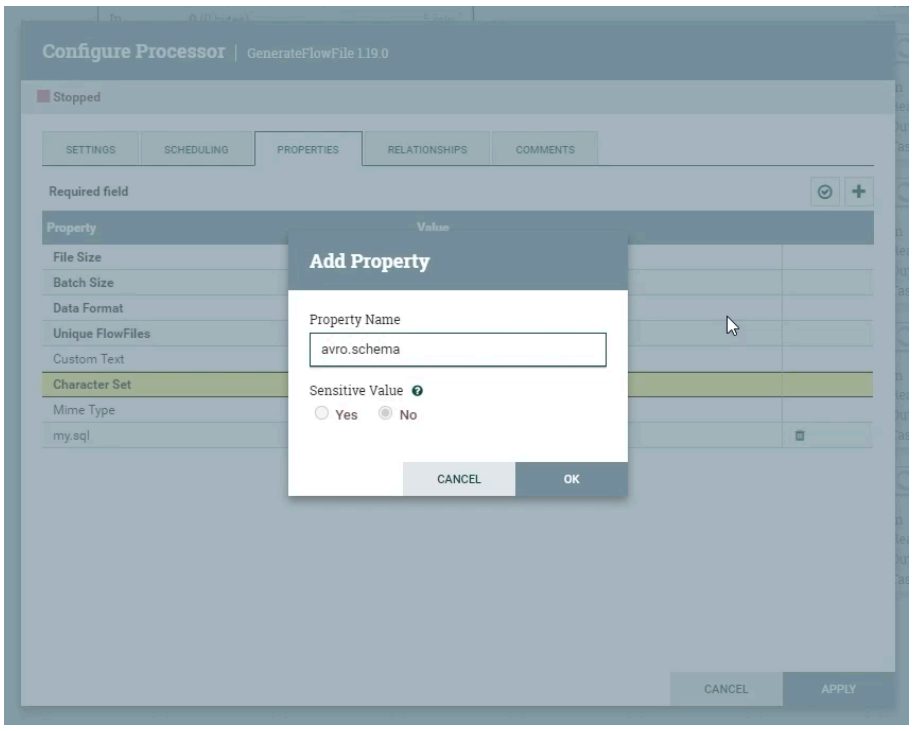
Здесь мы используем «Use ´Schema Text´ Property. Укажем формат даты, где маска будет следующей:



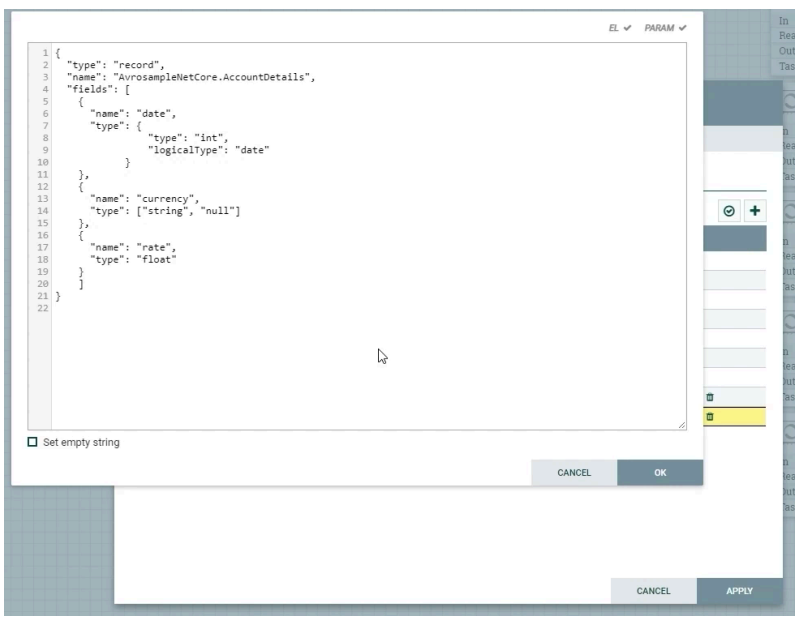
Поскольку мы хотим сделать схему, нам необходимо ее предварительно составить. Фокус: если есть Json-данные, можно онлайн сконвертировать их в Avro-схему. Она подготовлена заранее:



Возвращаемся и добавляем в атрибут «avro.schema»:

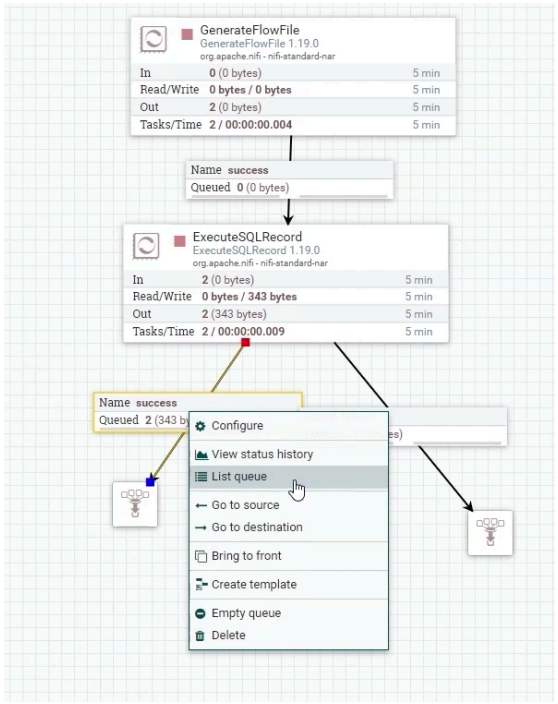


Разместим туда подготовленную схему, которая выглядит следующим образом:

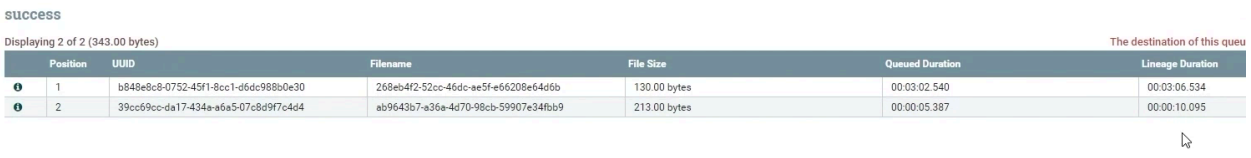


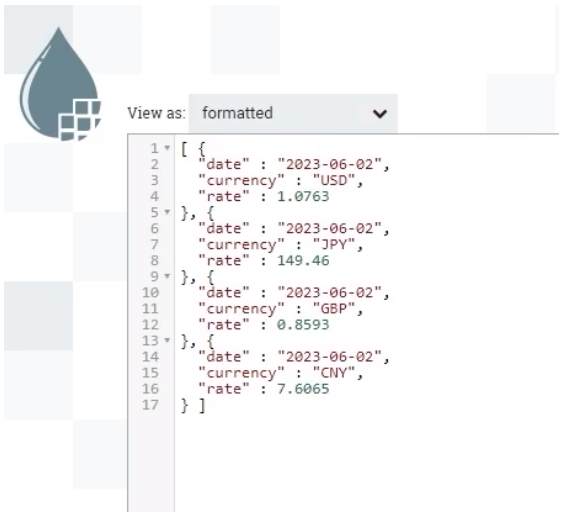
Для даты мы указываем специальный формат (ссылки об этом будут в описании к уроку).

После создания схемы мы запускаем процессор и проверяем его работу. Мы видим, что данные ушли в Success:



Из двух файлов выбираем тот, у которого меньше секунд. Смотрим, что получилось:





У нас получился Json-массив, который мы можем отформатировать.

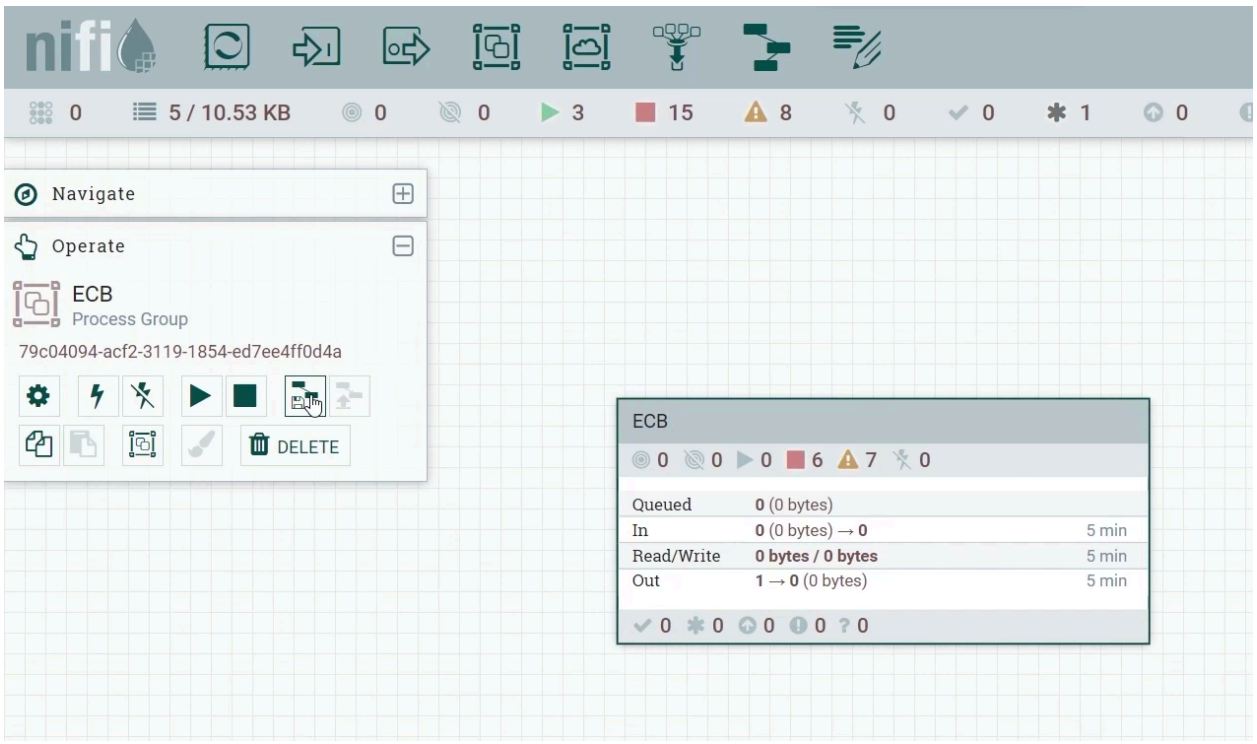
Отличный результат!

Подводим итоги:

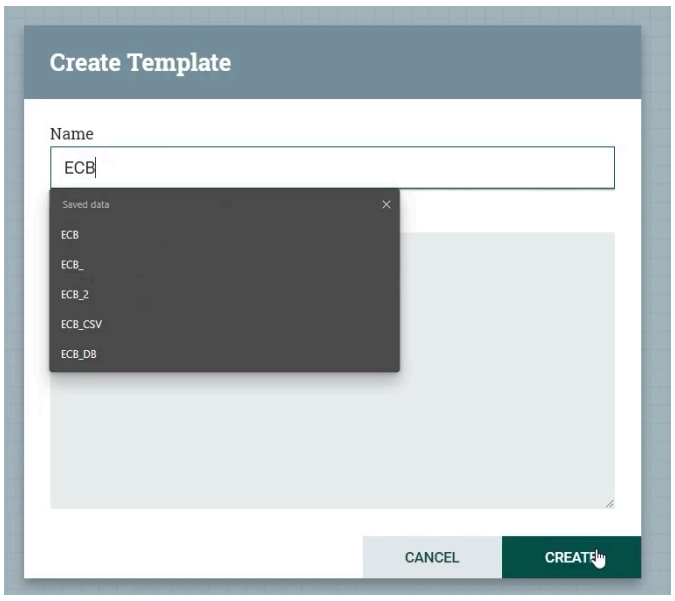
- Мы записали данные;
- Мы прочитали данные;
- Мы не использовали внешние программы.

Сохраним результаты нашего труда:

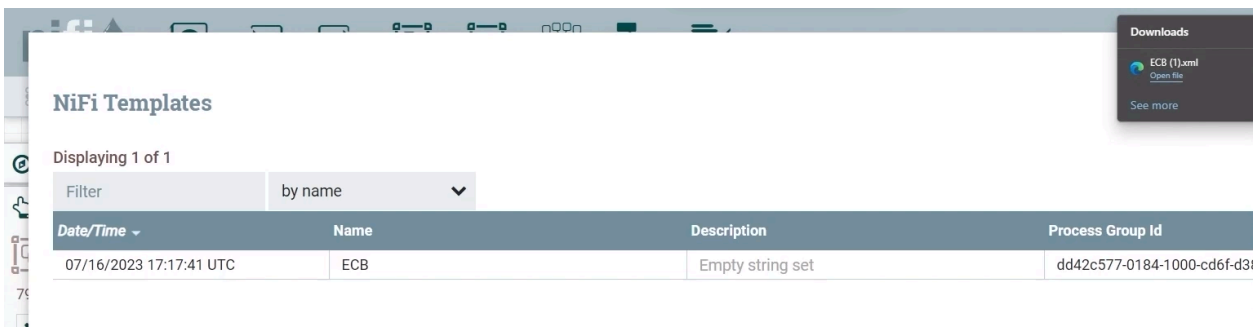
Выделяем группу, в которой мы создали поток:



Выделяем иконку в Operate. Далее нажимаем «Сохранить», пишем название:



Читаем, что Template был успешно создан. Мы можем найти его в разделе меню и сохранить его на компьютер:



Его можно использовать и передавать на другой компьютер.

ИТОГИ

Что мы сделали:

- Построили рабочий поток;
- Научились им управлять;
- Научились сохранять поток;
- Научились мониторингу ошибок;
- Научились контролю результатов;
- Отработали использование Data Provenance.

Как вам урок?

Изучил, далее