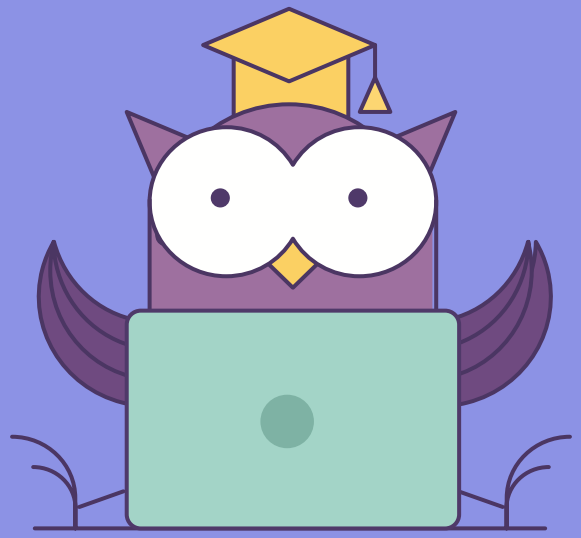
ОНЛАЙН-ОБРАЗОВАНИЕ

Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

OTUS

Загрузка данных

Источники, способы, инструменты

Егор Матешук

Цели сегодняшнего занятия

- Классифицировать разные источники
- Узнать об универсальных инструментах для загрузки данных
- Построить свой clickstream

01

Типы источников

Какие вы знаете примеры источников данных?

Примеры источников

- Очередь сообщений Kafka, RabbitMQ, MQTT
- TCP/UDP
- Syslog
- FTP-сервер
- Файловая система
- База данных
- HTTP API
- ...

Чем отличаются различные типы источников?

Свойства источников

- Активный / пассивный источник
- Структурированные / неструктурированные данные
- Плоские / вложенные структуры данных
- Бинарный / текстовый формат данных
- Наличие уникальных идентификаторов
- Наличие временных меток
- Append-only / updateable
- Полная / инкрементальная загрузка
- Наличие дубликатов
- Возможность получения истории
- Параллельная / однопоточная загрузка
- Партицирование параллельных потоков

Пассивные и активные источники

Пассивный источник

Источник, имеющий интерфейс, через который можно запросить данные.

- Требует постоянного опроса и активного элемента, который эти опросы будет осуществлять
- Позволяет повторить чтение в случае наличия проблем
- Позволяет контролировать скорость загрузки
- Примеры: веб-сервис с HTTP API, FTP-сервер, RDBMS.

Активный источник

Источник, который сам пишет данные в какое-либо хранилище (например, очередь сообщений).

- Более эффективен с точки зрения ресурсов, так как не требует постоянного опроса
- Сложно или невозможно повторить загрузку
- Скорость поступления данных напрямую зависит от скорости их генерации в источнике
- Примеры: веб-сервис, пишущий в Kafka, датчики, пишущие в MQTT, RDBMS.

Пассивные и активные источники

Пассивный источник

Источник, имеющий интерфейс, через который можно запросить данные.

- Требует постоянного опроса и активного элемента, который эти опросы будет осуществлять
- Позволяет повторить чтение в случае наличия проблем
- Позволяет контролировать скорость загрузки
- Примеры: веб-сервис с HTTP API, FTP-сервер, RDBMS.

Активный источник

Источник, который сам пишет данные в какое-либо хранилище (например, очередь сообщений).

- Более эффективен с точки зрения ресурсов, так как не требует постоянного опроса
- Сложно или невозможно повторить загрузку
- Скорость поступления данных напрямую зависит от скорости их генерации в источнике
- Примеры: веб-сервис, пишущий в Kafka, датчики, пишущие в MQTT, RDBMS.

Структурированные / неструктурированные

Структурированные данные

Данные, имеющие описание схемы в машиночитаемом формате

- Явно описывает типы данных
- Труднее начать читать, но проще в сопровождении
- Необходима передача схемы при её изменении
- Явно сломается при несовместимом изменении схемы
- Примеры: Thrift, Avro, MsgPack

Неструктурированные данные

Данные не имеющие описания схемы в машиночитаемом формате

- Требует описания типов данных на стороне получателя
- Проще начать читать, но труднее в сопровождении
- Может сломаться неявно при изменении схемы
- Примеры: CSV, JSON, XML

Пример: эволюция схемы

Допустим, у нас есть файл со структурой

```
timestamp,user,country,cookie_id
```

и мы добавляем поле city

```
timestamp,user,country,city,cookie_id
```

Что произойдет в случае CSV и в случае Avro?

Бинарный / текстовый формат данных

Бинарный формат

Формат, поля которого кодируются в бинарном представлении, соответствующим типу данных

- Занимает меньше места
- Поддерживает контекстные типы данных (date, datetime)
- Позволяет сохранять любые бинарные последовательности
- Сложно отлаживать
- Примеры: Thrift, Avro, MsgPack

Текстовый формат

Формат, поля которого кодируются в форматированных строках

- Занимает больше места
- Бинарные последовательности могут сломать формат
- Проще отлаживать
- Примеры: CSV, JSON, XML

Плоские / вложенные структуры данных

- Плоская структура: все поля расположены в корне, нет массивов и словарей
- Вложенная структура: поле может содержать несколько других полей внутри, словарь или массив

Пример: “уплощение” структуры

```
{
   “request_id”: 3306,
   “datetime”: “2019-05-03T21:34:00”,
   “results”: [
      result_record,
      result_record,
      ...
   ]
}
```

где result_record - объект с большим количеством полей

Наличие уникальных идентификаторов

- Наличие уникальных идентификаторов позволяет перезаливать данные и убирать дубликаты
- Идентификаторы могут быть
  - Простые: одно поле с уникальным содержимым
  - Составные: комбинация нескольких полей (например, пользователь - номер заказа - номер позиции)

Параллельная загрузка

- Загрузка может происходить параллельно в несколько партиций
- В случае параллельной загрузки необходимо учитывать соотношение содержимого данных и партиций
  - Для оптимизации производительности
  - Для сохранения порядка данных

Гарантии доставки

- At least once
- At most once
- Exactly once

Определяется всегда комбинацией

источник-читатель

Бизнес-требования

- Частота доступа
- Доступ к сырым данным
- Хранение истории
- Критичность потери
- Актуальность

Как поставить задачу на загрузку данных

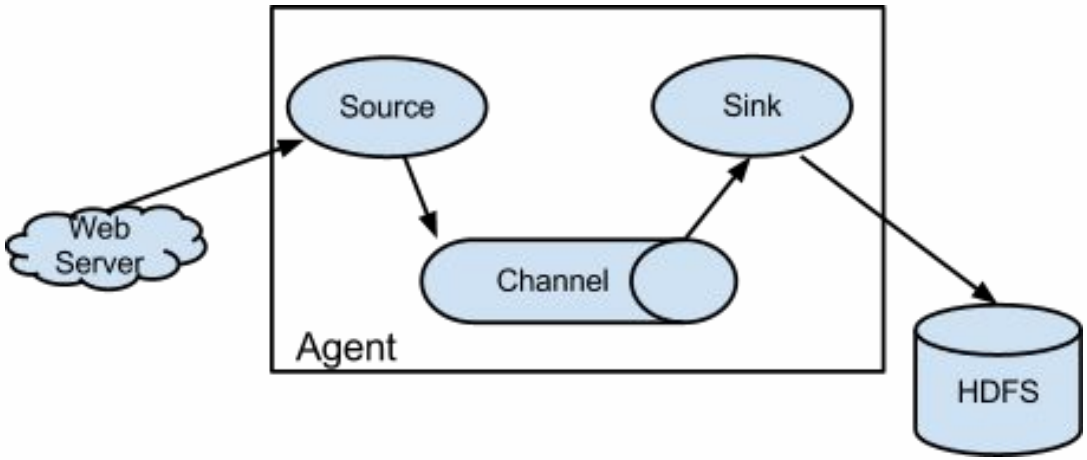
OTUS

- Указание владельца и пользователей
- Описание бизнес-требований
- Описание источника данных
- Резервирование места

02

Инструменты

Apache Flume - инструмент для транспортировки данных между различными источниками и хранилищами.



Плюсы

- Есть примерно везде
- Давно используется
- Достаточно гибкий и расширяемый

Минусы

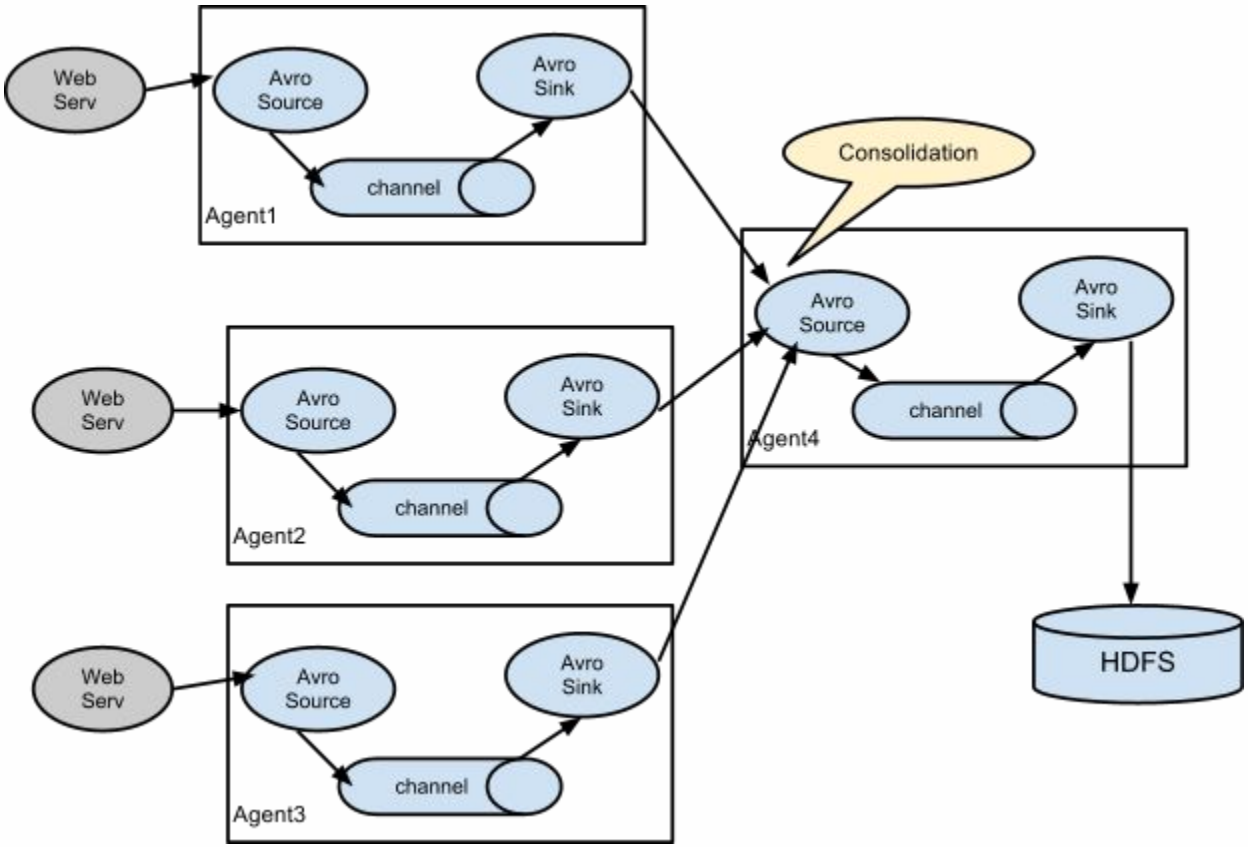
- Неудобная конфигурация
- Сложно мониторить

- **Agent** - один процесс Flume
- **Source** - элемент для получения данных
  - PollableSource - пассивный источник
  - EventDrivenSource - активный источник
- **Sink** - элемент, отвечающий за “слив” данных в хранилище или следующую систему
- **Channel** - буффер между source и sink, в котором хранятся данные до момента записи

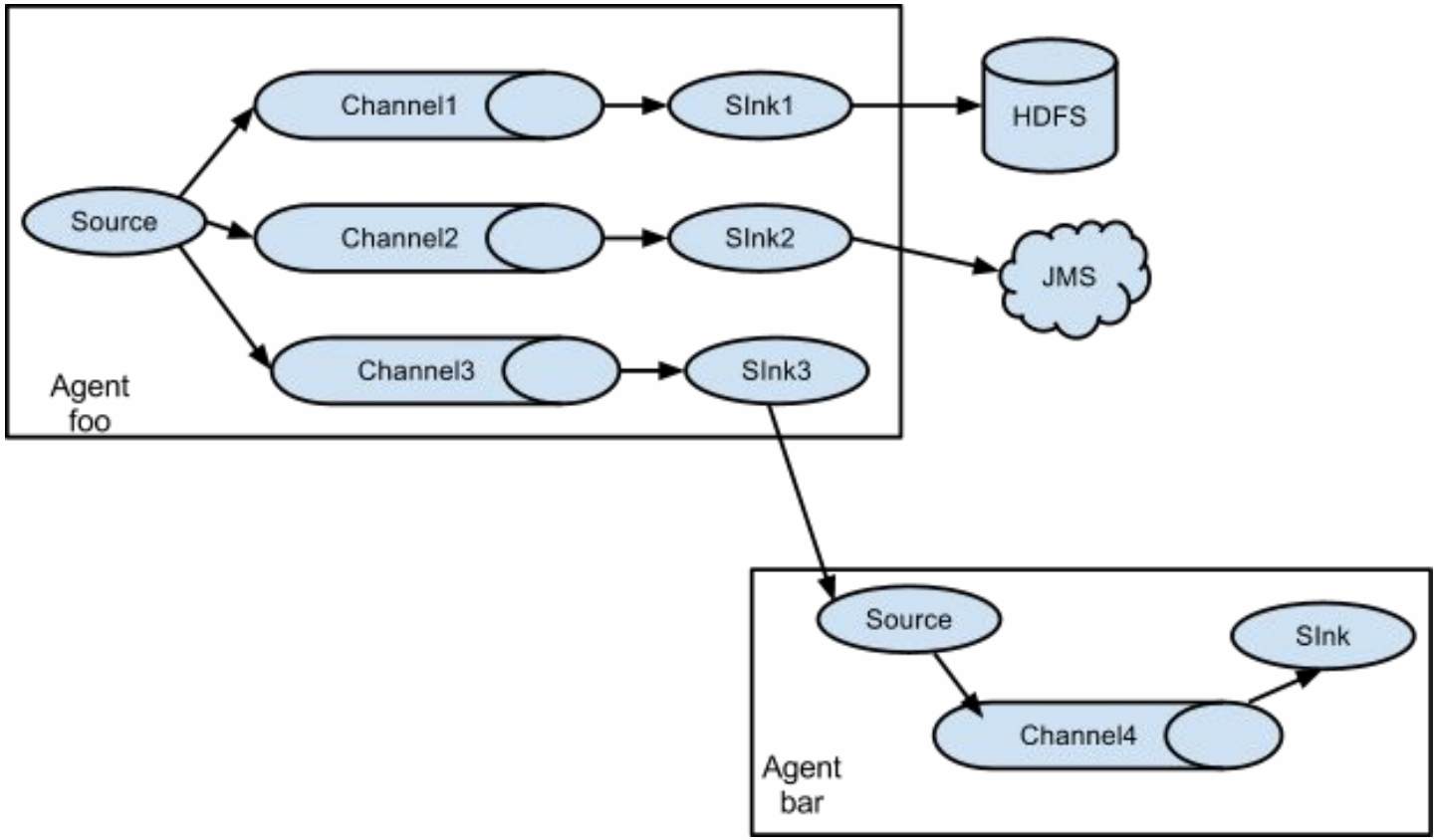
Каналы бываю нескольких типов и выбираются исходя из требований скорости/надежности

- **Memory Channel** - наиболее быстрый, наименее надежный
- **JDBC Channel**
- **Kafka Channel**
- **File Channel**

Можно строить цепочки агентов. Пример “сбора”



Пример одновременной заливки в несколько sink’ов



Пример одновременной заливки в несколько sink’ов

```
# List the sources, sinks and channels for the agent

<Agent>.sources = <Source1>

<Agent>.sinks = <Sink1> <Sink2>

<Agent>.channels = <Channel1> <Channel2>

# set list of channels for source (separated by space)

<Agent>.sources.<Source1>.channels = <Channel1> <Channel2>

# set channel for sinks

<Agent>.sinks.<Sink1>.channel = <Channel1>

<Agent>.sinks.<Sink2>.channel = <Channel2>


 <Agent>.sources.<Source1>.selector.type = replicating
```

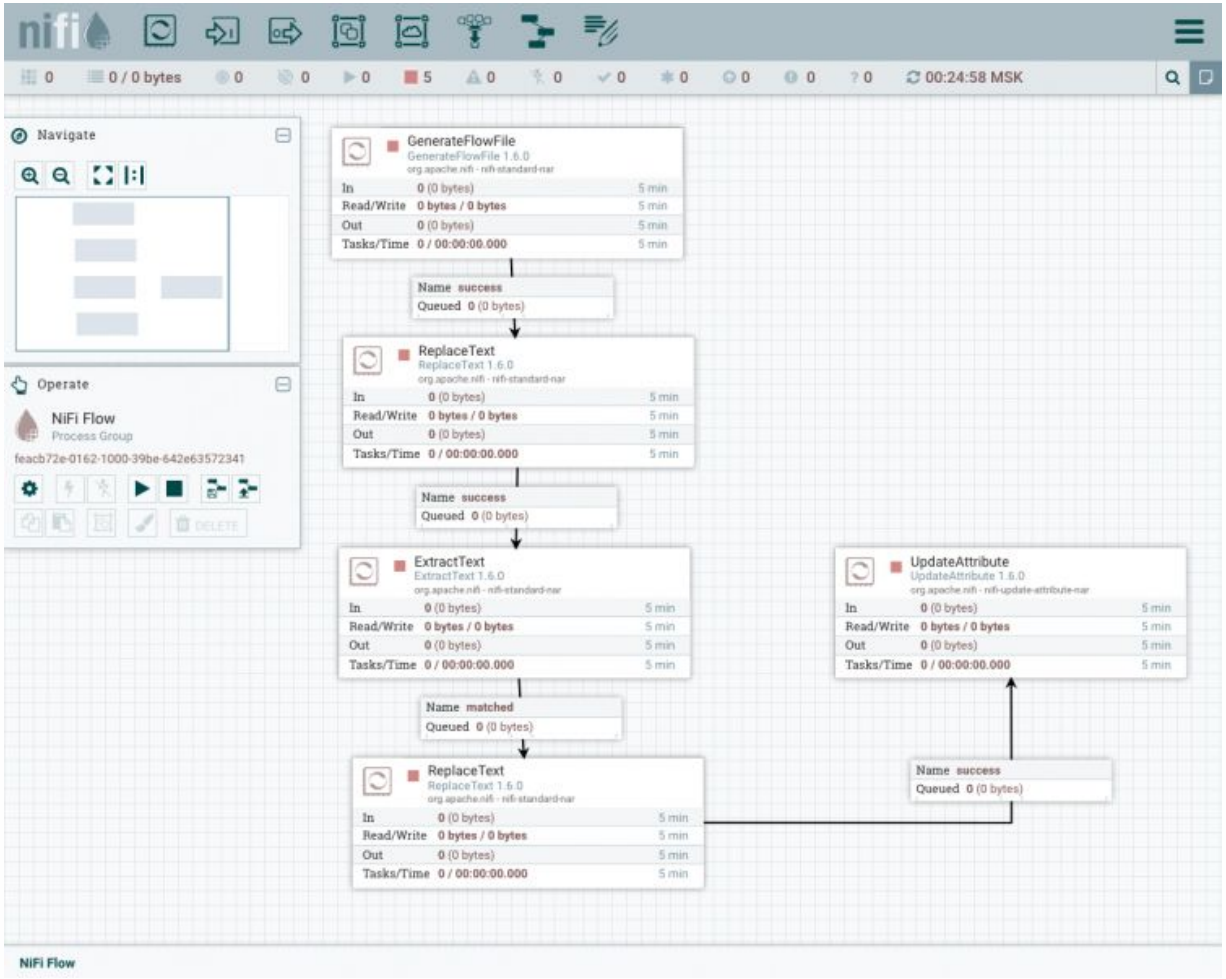

```
# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# Describe/configure the source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444

# Describe the sink
a1.sinks.k1.type = logger

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

Apache NiFi

OTUS

- Система управления потоком данных с визуальным интерфейсом
- Изначально разработана в АНБ
- Сейчас поддерживается и развивается Hortonworks
- Входит в состав HDF от Hortonworks
- Имеет особую версию MiNiFi для сбора данных с устройств

Выглядит примерно так



- **Dataflow** - граф, описывающий обработку данных, состоит из процессоров
- **Processor** - узел вычислений в графе
- **Process Group** - объединение процессоров для более удобного представления в UI
- **FlowFile** - “кусок” данных, движущийся по графу
- **Controller Service** - описание сервиса (базы данных, например) для переиспользования в разных графах

На случай, если нужного процессора не нашлось, есть несколько вариантов для специфических вычислений

- Свой внутренний язык выражений
- Запуск пользовательских задач
- Выполнение SQL-скриптов

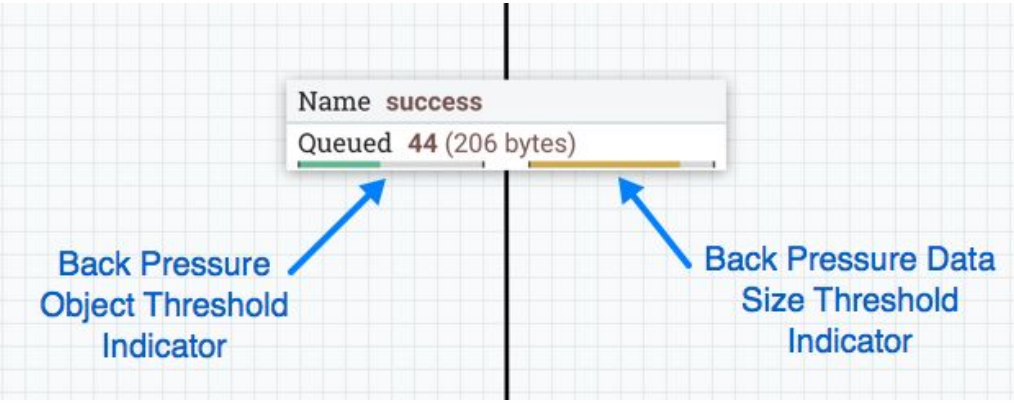
Процессоры могут запускаться по различным условиям

- Периодический запуск по таймеру
- Запуск по событию
- Запуск по расписанию

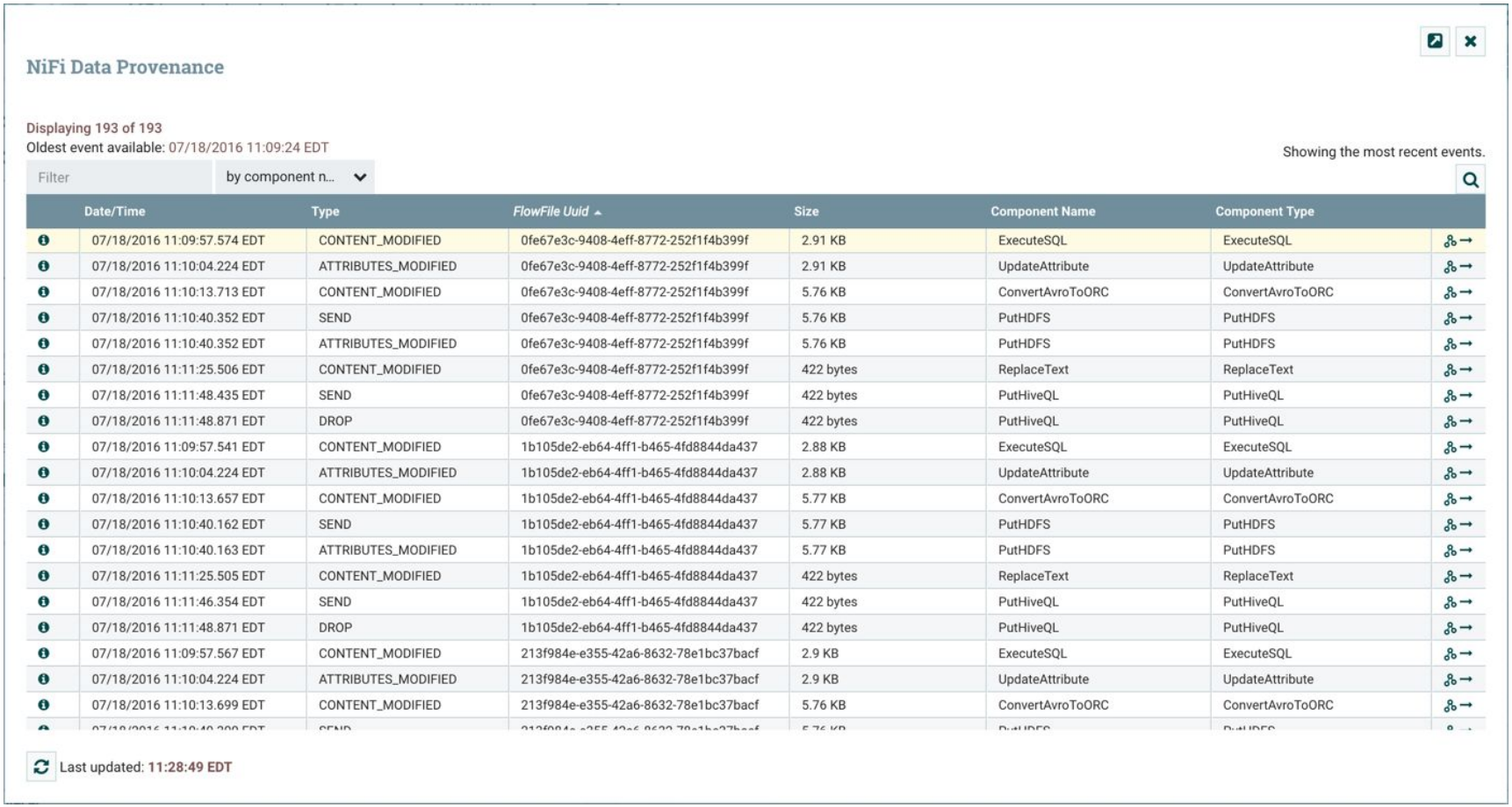
Одна из полезных фичей - контроль back pressure

Пример:

- Вы читаете данные из Kafka (быстро)
- Пишете в базу (медленно)
- В какой-то момент накапливается большая разница между полученными и записанными данными (возникает back pressure)
- NiFi останавливает источник данных до момента спада нагрузки



Механизм Data Provenance



NiFi Data Provenance

Displaying 193 of 193
Oldest event available: 07/18/2016 11:09:24 EDT

Showing the most recent events.

Filter | by component n...

Date/Time	Type	FlowFile Uuid	Size	Component Name	Component Type
07/18/2016 11:09:57.574 EDT	CONTENT_MODIFIED	0fe67e3c-9408-4eff-8772-252f1f4b399f	2.91 KB	ExecuteSQL	ExecuteSQL
07/18/2016 11:10:04.224 EDT	ATTRIBUTES_MODIFIED	0fe67e3c-9408-4eff-8772-252f1f4b399f	2.91 KB	UpdateAttribute	UpdateAttribute
07/18/2016 11:10:13.713 EDT	CONTENT_MODIFIED	0fe67e3c-9408-4eff-8772-252f1f4b399f	5.76 KB	ConvertAvroToORC	ConvertAvroToORC
07/18/2016 11:10:40.352 EDT	SEND	0fe67e3c-9408-4eff-8772-252f1f4b399f	5.76 KB	PutHDFS	PutHDFS
07/18/2016 11:10:40.352 EDT	ATTRIBUTES_MODIFIED	0fe67e3c-9408-4eff-8772-252f1f4b399f	5.76 KB	PutHDFS	PutHDFS
07/18/2016 11:11:25.506 EDT	CONTENT_MODIFIED	0fe67e3c-9408-4eff-8772-252f1f4b399f	422 bytes	ReplaceText	ReplaceText
07/18/2016 11:11:48.435 EDT	SEND	0fe67e3c-9408-4eff-8772-252f1f4b399f	422 bytes	PutHiveQL	PutHiveQL
07/18/2016 11:11:48.871 EDT	DROP	0fe67e3c-9408-4eff-8772-252f1f4b399f	422 bytes	PutHiveQL	PutHiveQL
07/18/2016 11:09:57.541 EDT	CONTENT_MODIFIED	1b105de2-eb64-4ff1-b465-4fd8844da437	2.88 KB	ExecuteSQL	ExecuteSQL
07/18/2016 11:10:04.224 EDT	ATTRIBUTES_MODIFIED	1b105de2-eb64-4ff1-b465-4fd8844da437	2.88 KB	UpdateAttribute	UpdateAttribute
07/18/2016 11:10:13.657 EDT	CONTENT_MODIFIED	1b105de2-eb64-4ff1-b465-4fd8844da437	5.77 KB	ConvertAvroToORC	ConvertAvroToORC
07/18/2016 11:10:40.162 EDT	SEND	1b105de2-eb64-4ff1-b465-4fd8844da437	5.77 KB	PutHDFS	PutHDFS
07/18/2016 11:10:40.163 EDT	ATTRIBUTES_MODIFIED	1b105de2-eb64-4ff1-b465-4fd8844da437	5.77 KB	PutHDFS	PutHDFS
07/18/2016 11:11:25.505 EDT	CONTENT_MODIFIED	1b105de2-eb64-4ff1-b465-4fd8844da437	422 bytes	ReplaceText	ReplaceText
07/18/2016 11:11:46.354 EDT	SEND	1b105de2-eb64-4ff1-b465-4fd8844da437	422 bytes	PutHiveQL	PutHiveQL
07/18/2016 11:11:48.871 EDT	DROP	1b105de2-eb64-4ff1-b465-4fd8844da437	422 bytes	PutHiveQL	PutHiveQL
07/18/2016 11:09:57.567 EDT	CONTENT_MODIFIED	213f984e-e355-42a6-8632-78e1bc37bacf	2.9 KB	ExecuteSQL	ExecuteSQL
07/18/2016 11:10:04.224 EDT	ATTRIBUTES_MODIFIED	213f984e-e355-42a6-8632-78e1bc37bacf	2.9 KB	UpdateAttribute	UpdateAttribute
07/18/2016 11:10:13.699 EDT	CONTENT_MODIFIED	213f984e-e355-42a6-8632-78e1bc37bacf	5.76 KB	ConvertAvroToORC	ConvertAvroToORC

Last updated: 11:28:49 EDT

StreamSets

- Система управления потоком данных с визуальным интерфейсом
- Разработана выходцами из Cloudera
- Легко устанавливается в виде Parcel на CDH
- Имеет особую версию SDC Edge для сбора данных с устройств
- Состоит из двух компонент
  - SDC - система выполняющая непосредственно обработку данных (free)
  - StreamSets Controller - центр управления несколькими SDC с дополнительными возможностями по разработке пайплайнов (paid)

С этой системой мы познакомимся на практике

Что плохого было в старом подходе?

На одном из предыдущих занятий вы создали пайплайн, который выгружал данные сервиса в BigQuery

Почему этот пайплайн был неидеален и как его можно улучшить?

Заполните, пожалуйста, опрос в личном кабинете!

Полезные ссылки

- https://flume.apache.org/FlumeUserGuide.html
- https://statsbot.co/blog/open-source-etl/

Егор Матешук

egor@mateshuk.com

Спасибо
за внимание!