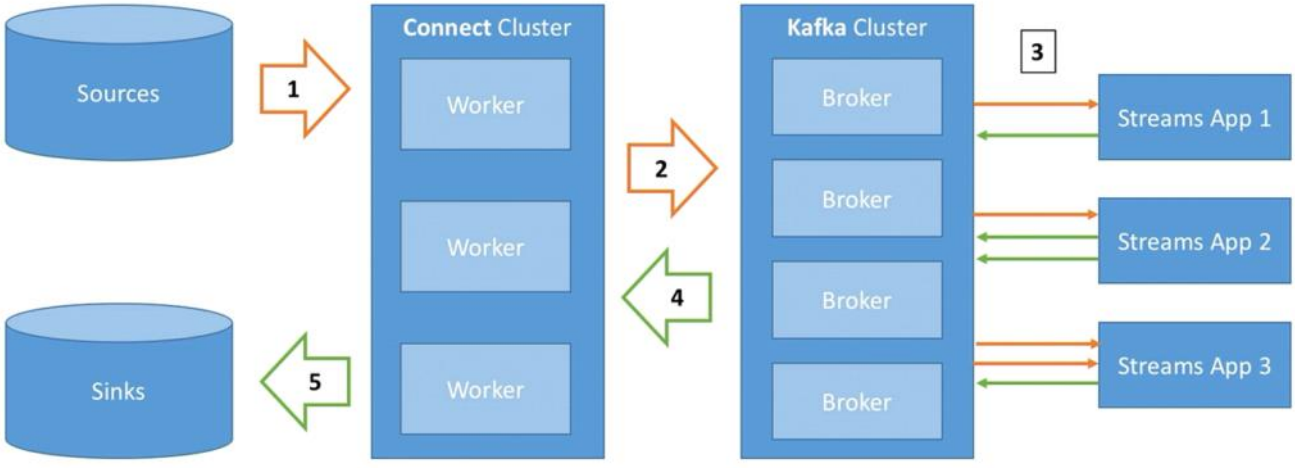
_общее

30 ноября 2020 г. 19:28

https://github.com/PacktPublishing/Apache-Kafka-Series---Kafka-Connect-Hands-on-Learning

Source =>	Kafka	Producer API	**Kafka Connect Source**
Kafka =>	Kafka	Consumer, Producer API	Kafka Streams
Kafka =>	Sink	Consumer API	**Kafka Connect Sink**
Kafka =>	App	Consumer API	

Kafka Connect and Streams Architecture Design



Kafka Connect - Concepts

- Kafka Connect Cluster has multiple loaded **Connectors**
  - Each connector is a re-usable piece of code (java jars)
  - Many connectors exist in the open source world, leverage them!
- Connectors + **User Configuration** => **Tasks**
  - A task is linked to a connector configuration
  - A job configuration may spawn multiple tasks

Kafka Connect - Concepts

- Kafka Connect Cluster has multiple loaded **Connectors**
  - Each connector is a re-usable piece of code (java jars)
  - Many connectors exist in the open source world, leverage them!
- Connectors + **User Configuration** => **Tasks**
  - A task is linked to a connector configuration
  - A job configuration may spawn multiple tasks
- Tasks are executed by Kafka Connect **Workers** (servers)
  - A worker is a single java process
  - A worker can be standalone or in a cluster

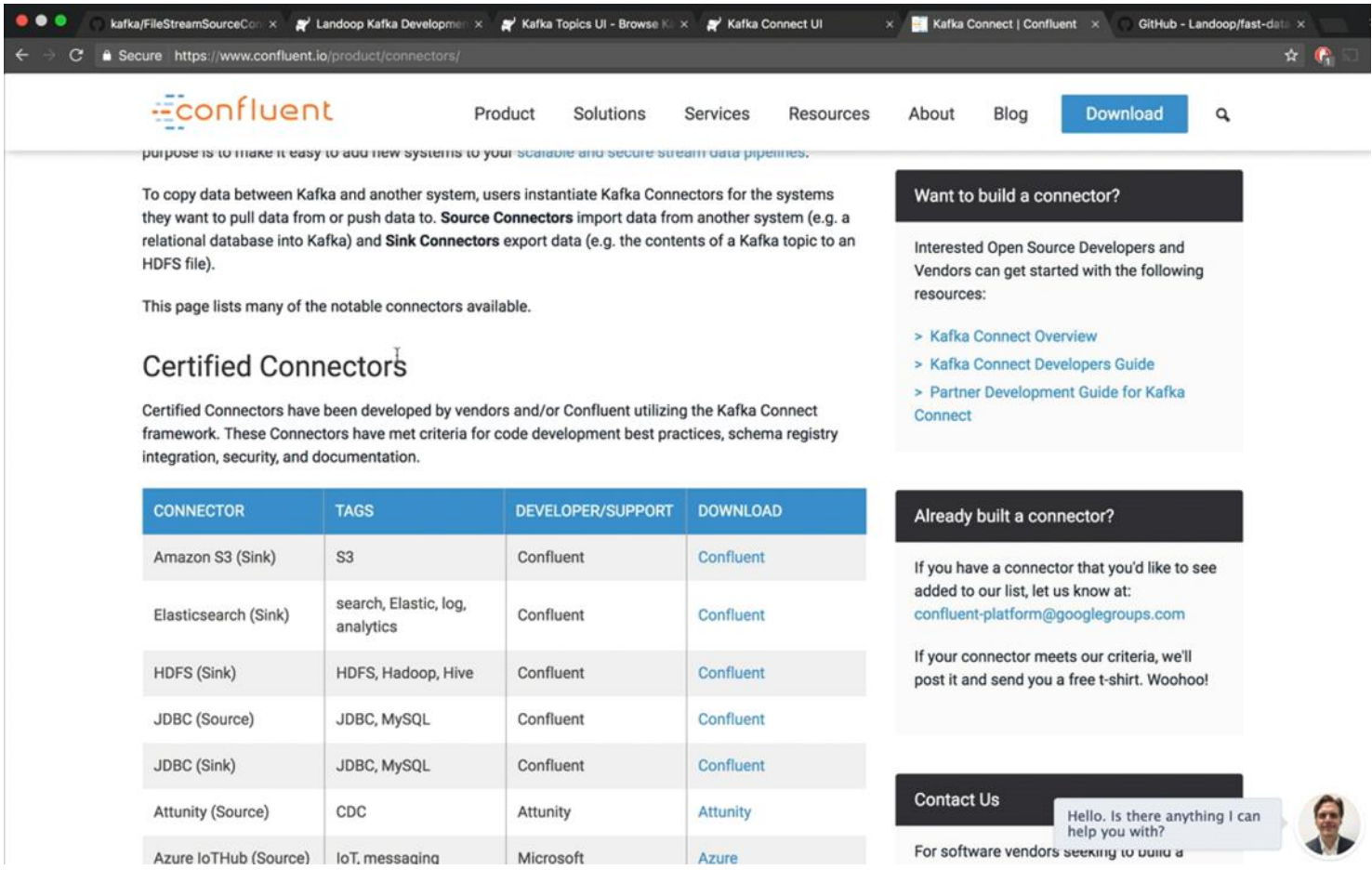
Standalone vs Distributed Mode

- Standalone:
  - A single process runs your connectors and tasks
  - Configuration is bundled with your process
  - Very easy to get started with, useful for development and testing
  - Not fault tolerant, no scalability, hard to monitor
- Distributed:
  - Multiple workers run your connectors and tasks
  - Configuration is submitted using a REST API
  - Easy to scale, and fault tolerant (rebalancing in case a worker dies)
  - Useful for production deployment of connectors

List of available connectors

- There are many source and sinks connectors
- The best place to find them is the Confluent website:

https://www.confluent.io/product/connectors/



confluent

Product Solutions Services Resources About Blog Download

purpose is to make it easy to add new systems to your scalable and secure stream data pipelines.

To copy data between Kafka and another system, users instantiate Kafka Connectors for the systems they want to pull data from or push data to. **Source Connectors** import data from another system (e.g. a relational database into Kafka) and **Sink Connectors** export data (e.g. the contents of a Kafka topic to an HDFS file).

This page lists many of the notable connectors available.

Certified Connectors

Certified Connectors have been developed by vendors and/or Confluent utilizing the Kafka Connect framework. These Connectors have met criteria for code development best practices, schema registry integration, security, and documentation.

CONNECTOR	TAGS	DEVELOPER/SUPPORT	DOWNLOAD
Amazon S3 (Sink)	S3	Confluent	Confluent
Elasticsearch (Sink)	search, Elastic, log, analytics	Confluent	Confluent
HDFS (Sink)	HDFS, Hadoop, Hive	Confluent	Confluent
JDBC (Source)	JDBC, MySQL	Confluent	Confluent
JDBC (Sink)	JDBC, MySQL	Confluent	Confluent
Attunity (Source)	CDC	Attunity	Attunity
Azure IoTHub (Source)	IoT, messaging	Microsoft	Azure

Want to build a connector?

Interested Open Source Developers and Vendors can get started with the following resources:

> Kafka Connect Overview
> Kafka Connect Developers Guide
> Partner Development Guide for Kafka Connect

Already built a connector?

If you have a connector that you'd like to see added to our list, let us know at: confluent-platform@googlegroups.com

If your connector meets our criteria, we'll post it and send you a free t-shirt. Woohoo!

Contact Us

Hello. Is there anything I can help you with?

For software vendors seeking to build a

внутренне коннектор/worker это обычный консьюмер/продьюсер

- Internally, Kafka Connect is just a Kafka client using the standard Producer
- Connect is a prebuilt framework which provides the basic client architecture (using the standard Kafka Client API). This framework is written to make Connect highly available and scalable. However, it relies on plugins called Connectors to provide the specific functionality to communicate with your chosen endpoints.
- Connect is restricted to data copying, with light data transformations
- Connect Converters are like wrappers around Serializers and Deserializers. Since a Worker can be both a Producer and a Consumer (depending on the Connector), it was decided that having one object to specify rather than two a more efficient way to configure the Worker.

- *Connectors* are logical jobs responsible for managing the copying of data between Kafka and another system
- Connector *Sources* read data *from* an external data system into Kafka
  - Internally, source connector is a Kafka Producer
- Connector *Sinks* write Kafka data *to* an external data system
  - Internally, sink connector is a Kafka Consumer Group

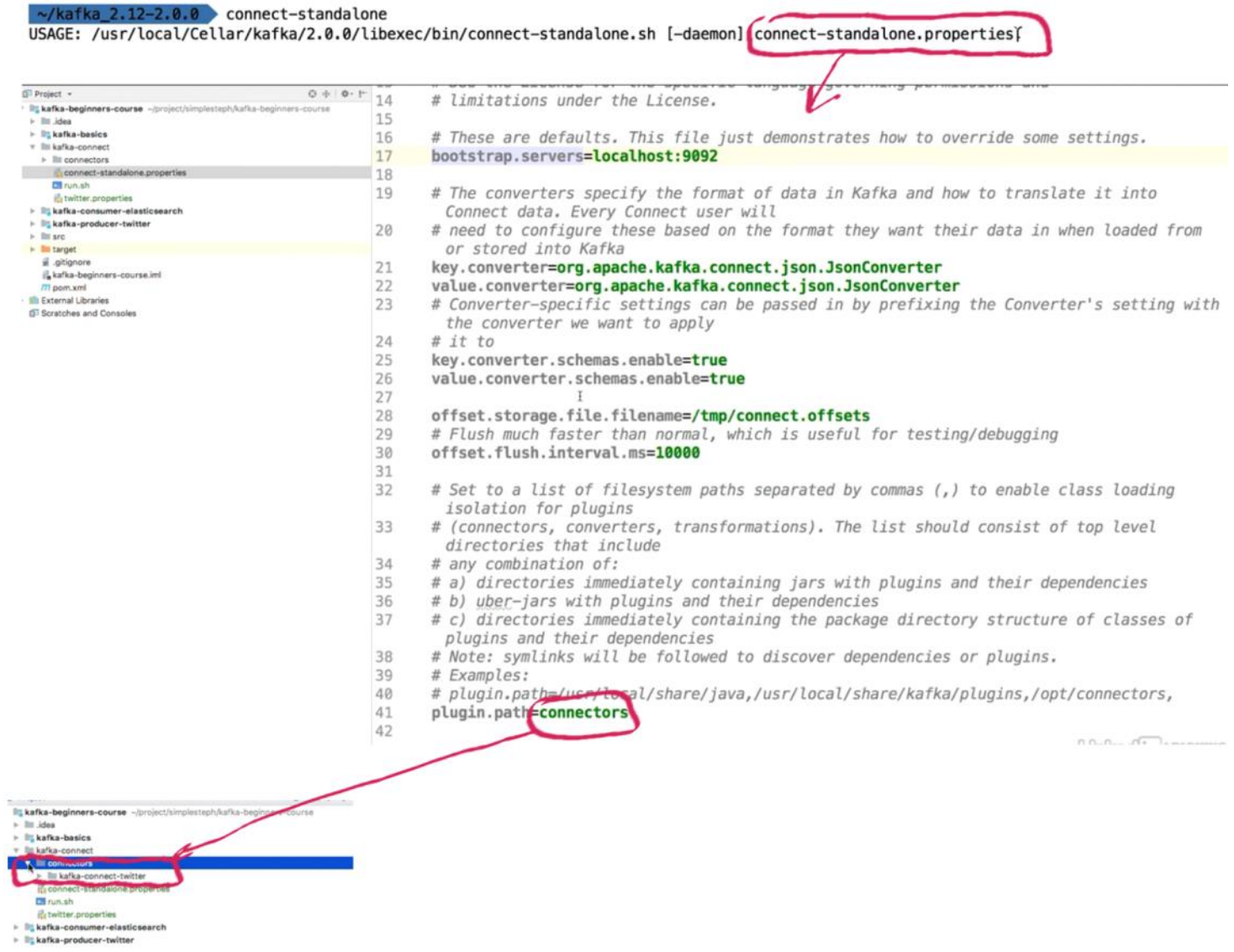
An instance of Connect (called a worker) can function as both a producer and consumer, depending on the type of connector you install. In fact, internally connectors are running the standard client code: Producer (source connectors) and Consumer (sink connectors).

Source connectors read from external data stores (e.g., databases). Sink Connectors pull data from a Kafka topic and write it out to an external application (e.g., HDFS, Elastisearch).

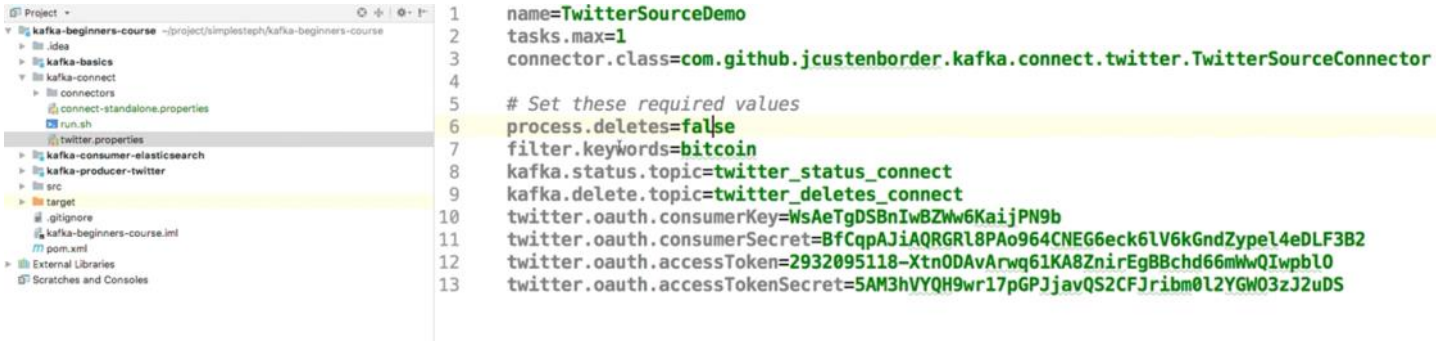
The term "Connector" can be used for either the type of plugin (e.g., the HDFS Connector) or a configured instance of a connector (e.g., the mycompany_HDFS connector).

CLI

30 ноября 2020 г. 19:45




```
connect-standalone.bat connect-standalone.properties twitter.properties
```



пример

2. Create a table `operators` in PostgreSQL:

```
postgres=# CREATE TABLE operators(
    id int not null primary key,
    name varchar(50) not null
);
```

```
$ docker container run --rm -d \
    -p 8080:80 \
    --net using-connect_confluent \
    -e PGADMIN_DEFAULT_EMAIL=student@confluent.io \
    -e PGADMIN_DEFAULT_PASSWORD=TopSecret \
    dpage/pgadmin4
```

1. Create a new Topic called `pg-operators` with one partition and one replica:

```
$ kafka-topics \
    --bootstrap-server kafka:9092 \
    --create \
    --topic pg-operators \
    --partitions 1 \
    --replication-factor 1
```

2. Add a JDBC source connector via command line and **REST API** of Connect:

```
$ curl -s -X POST \
        -H "Content-Type: application/json" \
        --data '{
            "name": "Operators-Connector",
            "config": {
                "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
                "connection.url": "jdbc:postgresql://postgres:5432/postgres",
                "connection.user": "postgres",
                "table.whitelist": "operators",
                "mode":"incrementing",
                "incrementing.column.name": "id",
                "table.types": "TABLE",
                "topic.prefix": "pg-",
                "numeric.mapping": "best_fit",
                "transforms": "createKey,extractInt",
                "transforms.createKey.type":
"org.apache.kafka.connect.transforms.ValueToKey",
                "transforms.createKey.fields": "id",
                "transforms.extractInt.type":
"org.apache.kafka.connect.transforms.ExtractField$Key",
                "transforms.extractInt.field": "id"
            }
        }' http://connect:8083/connectors | jq
```

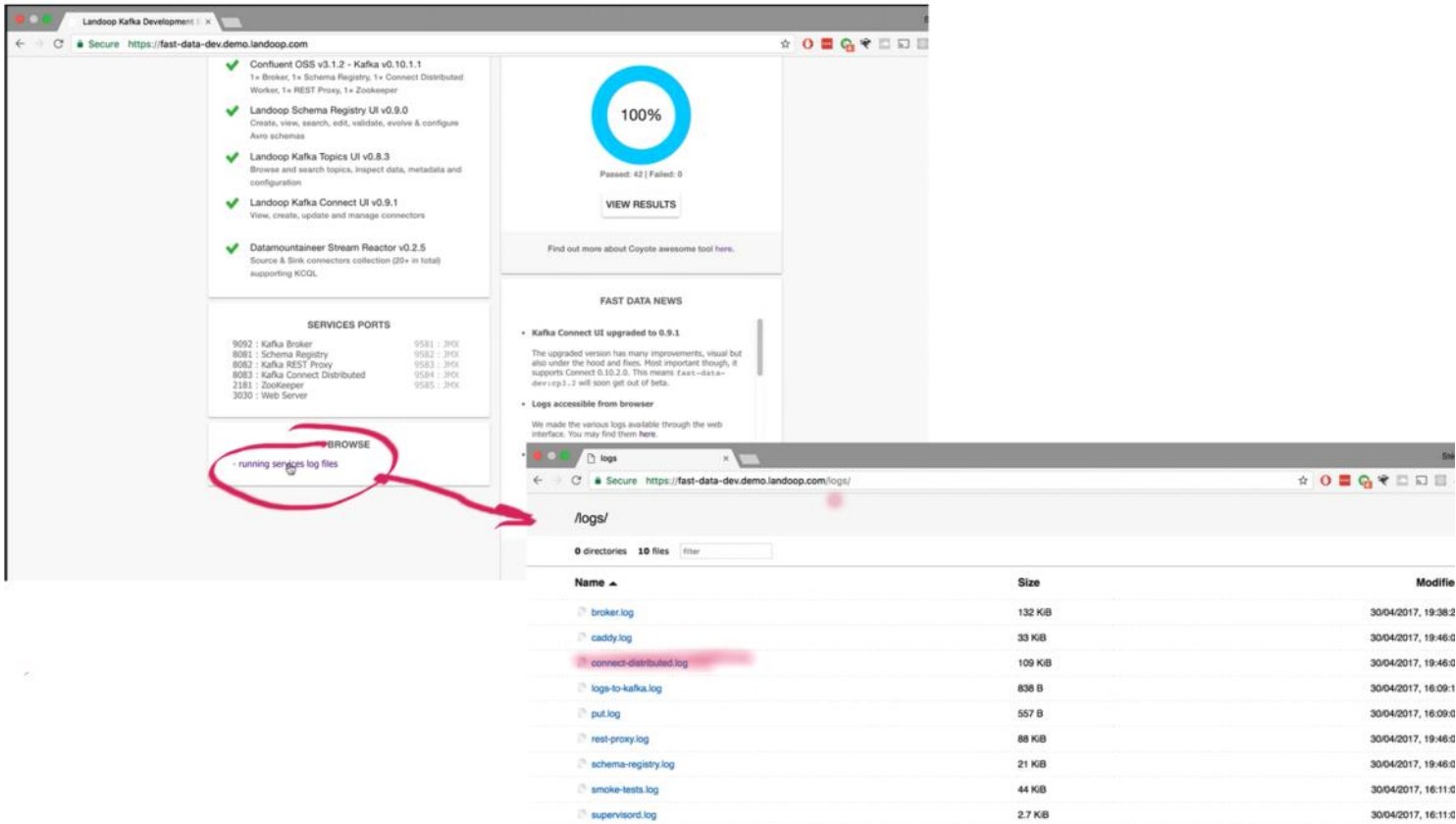
```
$ kafka-avro-console-consumer \
      --bootstrap-server kafka:9092 \
      --property schema.registry.url=http://schema-registry:8081 \
      --topic pg-operators \
      --from-beginning \
      --property print.key=true
```

LOGS

4 декабря 2020 г. 20:19

Viewing Kafka Connect Logs

- In standalone mode
  - They will directly be in your terminal
- In distributed mode
  - They will be at the URL http://127.0.0.1:3030/logs/
  - Take the file `connect-distributed.log`
- Look for "ERROR" in the log, that's a good indicator of what went wrong (an error message will come with the error)

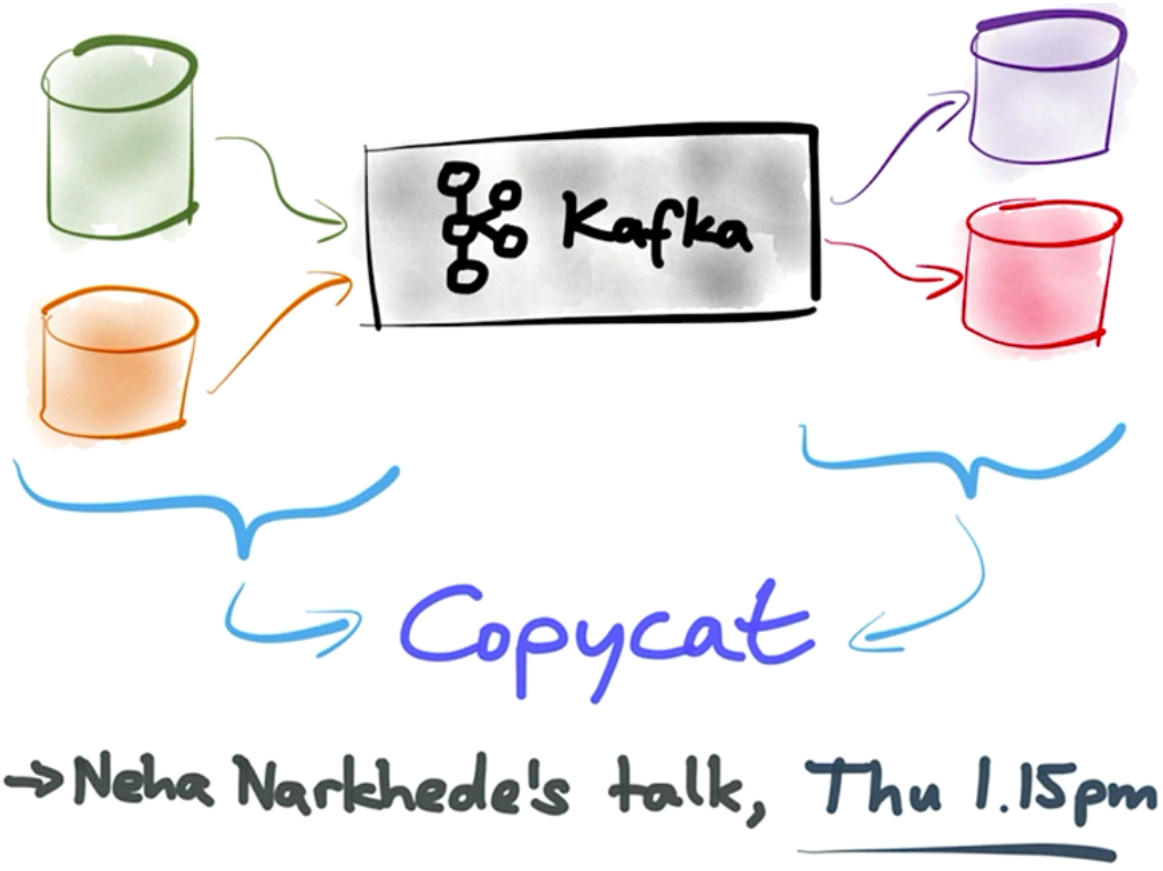


ETL with KAFKA

3 марта 2021 г. 12:53

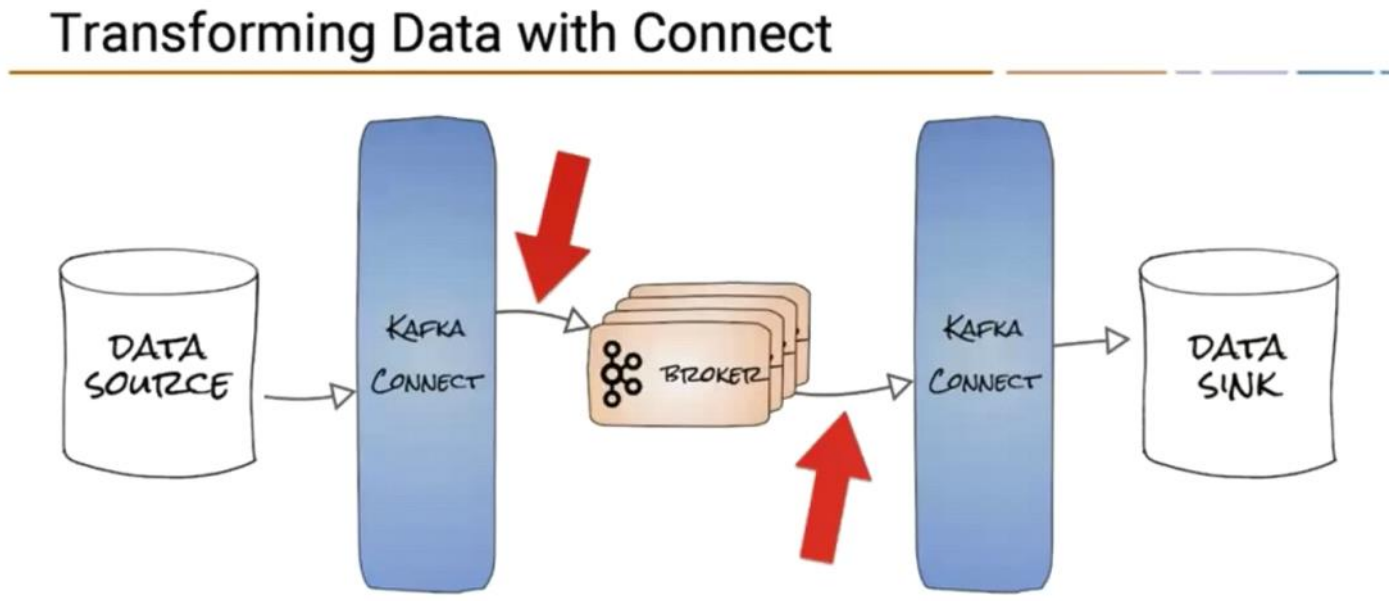
ETL with KAFKA

раньше тулза для этого называлась copycat
https://blog.codecentric.de/en/2018/03/etl-kafka/

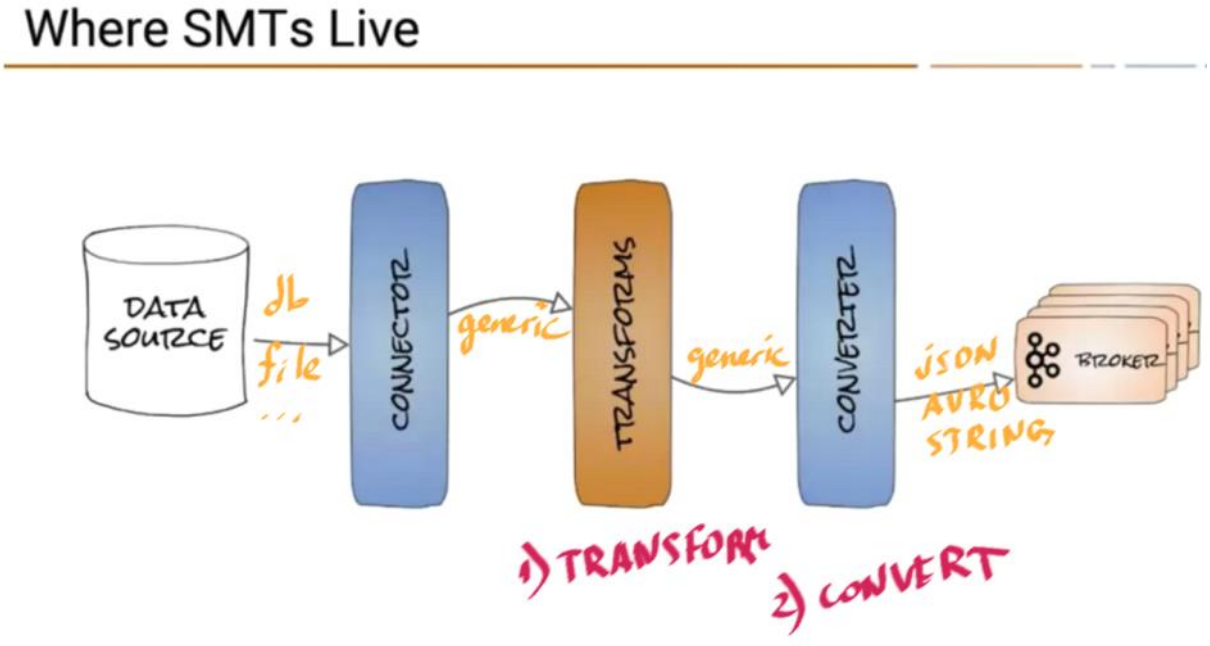


SMT single message transform

14 декабря 2020 г. 12:57



1) connector преобразует формат источника в общий формат generic format
2) generic->generic трансформация
3) generic->(avro,JSON,String) преобразует в итоговый формат



Single message transforms are a new stage in the Kafka Connect pipeline. Here we can see where they fit in for a source connector.

Source connectors read data from another system and store the data in Kafka. The connector (or more accurately, the connector's tasks) read data from the data source. At this stage the data will be in whatever format the source system exposes, but the connector will translate that into Kafka Connect's generic data API. Once it is in this generic format, the Kafka Connect single message transformations can be applied. Because they use the generic data format, any transformation can be used with any connector, independent of the original format of the data. The transformation is applied and the resulting data is still in a generic data API format. Finally, the data is passed to a converter, which serializes it into a format like Avro or JSON before it is passed to Kafka for storage. The pipeline for sink connectors looks similar, just reversed.

As you can see, by breaking the processing down into separate steps and leveraging Connect's data API, we can make transformations very general and applicable regardless of which connector you're using, much like we make serialization formats reusable across all connectors.

SMT не годится для серьезных трансформаций, те только для простых преобразований (добавить поле, убрать поле, писать константу в поле для всех сообщений)

- для серьезных трансформаций лучше использовать kafka streams или KSQL

Single Message Transform (SMT)

Modify events before storing in Kafka:

- Mask sensitive information
- Add identifiers
- Tag events
- Lineage/provenance
- Remove unnecessary columns

Modify events going out of Kafka:

- Route high priority events to faster data stores
- Direct events to different Elasticsearch indices
- Cast data types to match destination
- Remove unnecessary columns

Single Message Transform - Details

`InsertField`	insert a field using attributes from the message metadata or from a configured static value
`ReplaceField`	rename fields, or apply a blacklist or whitelist to filter
`MaskField`	replace field with valid `null` value for the type (0, empty string, etc)
`ValueToKey`	replace the key with a new key formed from a subset of fields in the value payload
`HoistField`	wrap the entire event as a single field inside a `Struct` or a `Map`
`ExtractField`	extract a specific field from `Struct` and `Map` and include only this field in results
`SetSchemaMetadata`	modify the schema name or version
`TimestampRouter`	modify the Topic of a record based on the original Topic name and timestamp
`RegexRouter`	modify the topic of a record based on original topic, replacement string and a regular expression

Many customers need a way to perform simple filtering and transformations for which building a streams application is too much work. For example, a company might want to take order information and send that to a team for analysis but needed to redact sensitive information such as social security numbers or credit card numbers.

The above properties can be configured at the **connector level** and be applied to either **key** and/or **value**.

TimestampRouter - Modify the topic of a record based on original topic and timestamp. Useful when using a sink that needs to write to different tables or indexes based on timestamps

More info: `http://kafka.apache.org/documentation.html#connect_transforms`

SMT конфигурируется в настройках коннектора

Programming with Configuration

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=test.txt
topic=connect-test
transforms=MakeMap,InsertSource, InsertKey, ExtractStoreId, MessageTypeRouter
transforms.MakeMap.type=org.apache.kafka.connect.transforms.HoistField$Value
transforms.MakeMap.field=line
transforms.InsertSource.type=org.apache.kafka.connect.transforms.InsertField$Value
transforms.InsertSource.static.field=data_source
transforms.InsertSource.static.value=test-file-source
transforms.InsertKey.type=org.apache.kafka.connect.transforms.ValueToKey
transforms.InsertKey.fields=storeId
transforms.ExtractStoreId.type=org.apache.kafka.connect.transforms.ExtractField$Key
transforms.ExtractStoreId.field=storeId
transforms.MessageTypeRouter.type=org.apache.kafka.connect.transforms.RegexRouter
transforms.MessageTypeRouter.regex=(foo|bar|baz)-.*
transforms.MessageTypeRouter.replacement=$1-logs
```

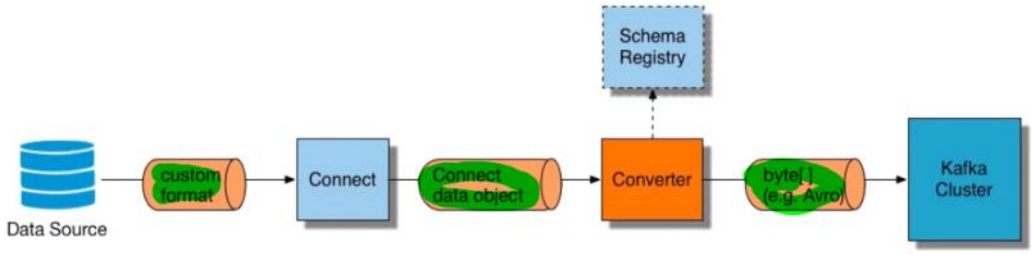
This starts to look kind of unpleasant!

1. Hoist Field: message sent `Foo` → transformed value `{"line":"Foo"}`
2. Insert Source: Add field `data_source` with value `test-file-source` to the imported records
3. Insert Key & ExtractStoreId: Take the value of the field `storeId` and use this as value for the Kafka record `key`
4. MessageTypeRouter: Update the record's topic using the configured regular expression and replacement string

- converter преобразует generic format уже в конкретный формат, который пойдет в кафку (те делает сериализацию и десериализацию)
- в конверторах можно использовать а можно и не использовать схему данных (коннектор интегрируется с реестром схем)

Converting Data

- *Converters* provide the data format written to or read from Kafka (like Serializers)
- Converters are decoupled from Connectors to enable reuse of converters
  - Allows any connector to work with any serialization format
- Example of format conversion in a source converter (sink converter is the reverse)



Serialization formats may seem like a minor detail, but **not** separating the details of data serialization in Kafka from the details of source or sink systems would result in a lot of inefficiency:

- First, a lot of code for doing simple data conversions are duplicated across a large number of ad hoc connector implementations.
- Second, each connector ultimately contains its own set of serialization options as it is used in more environments—JSON, Avro, Thrift, ProtoBuf, and more.

Much like the serializers in Kafka's producer and consumer, the **Converters** abstract away the details of serialization. Converters are different because they guarantee data is transformed to a common data API defined by Kafka Connect. This API supports both schema and schema-less data, common primitive data types, complex types like structs, and logical type extensions. By sharing this API, connectors write one set of translation code and Converters handle format-specific details. For example, the JDBC connector can easily be used to produce either JSON or Avro to Kafka, without any format-specific code in the connector.

Converter Data Formats

- Converters apply to both the key and value of the message
  - Key and value converters can be set independently
    - `key.converter`
    - `value.converter`
- Pre-defined data formats for Converter
  - Avro: `AvroConverter`
  - JSON: `JsonConverter`
  - String: `StringConverter`
  - Byte Array: `ByteArrayConverter`

The `ByteArrayConverter` (detailed in KIP-128, introduced in Kafka 0.11) provides an option for dealing with raw data, in contrast to structured data. It saves cost (CPU, memory, garbage collection overhead) of deserializing/converting data to Connect format.

For example, this is particularly useful with **Replicator**. This connector doesn't need to convert to/from byte array format and objects - it just copies the raw data as bytes for better performance.

как добавить в конфиге ссылку на реестр схем

Avro Converter as a Best Practice

Use an Avro Converter and Schema Registry!

```
key.converter=io.confluent.connect.avro.AvroConverter
key.converter.schema.registry.url=http://schemaregistry1:8081
value.converter=io.confluent.connect.avro.AvroConverter
value.converter.schema.registry.url=http://schemaregistry1:8081
```

пример

-

В этом примере мы воспользуемся двумя встроенными преобразованиями:
TimestampConverter и ReplaceField

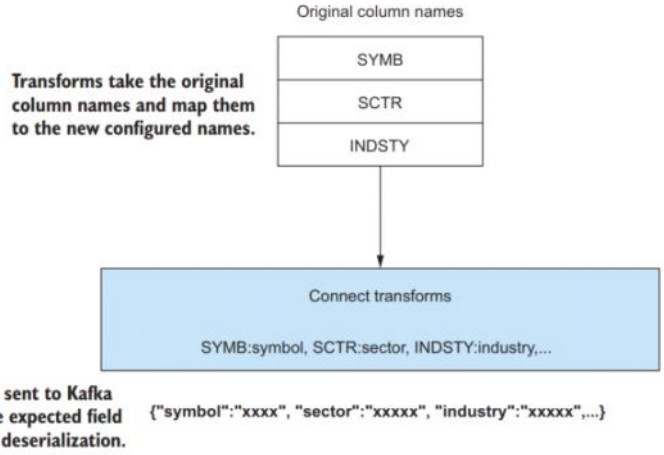


Figure 9.3 Transforming the names of the columns to match expected field names

Listing 9.1 JDBC connector properties

```
transforms=ConvertDate,Rename
transforms.ConvertDate.type=
➥ org.apache.kafka.connect.transforms.TimestampConverter$Value
transforms.ConvertDate.field=TXNTS
transforms.ConvertDate.target.type=string
transforms.ConvertDate.format=yyyy-MM-dd'T'HH:mm:ss.SSS-0400
transforms.Rename.type=
➥ org.apache.kafka.connect.transforms.ReplaceField$Value
transforms.Rename.renames=SMBL:symbol, SCTR:sector,....
```

Date field to convert
Aliases for the transformers
Type for the ConvertDate alias
Output type of the converted date field
Format of the date
Type for the Rename alias
List of column names (truncated for clarity) to replace. The format is Original:Replacement.

преобразования должны быть простыми.

- хотя может показаться заманчивым применить преобразования повсеместно, импортируя данные в Kafka через Kafka Connect, помните: преобразования должны быть простыми. Для любых преобразований, кроме самых простых, показанных в наших примерах, лучше извлечь данные в Kafka и воспользоваться Kafka Streams для вычислительно сложных преобразований.

standalone mode (dev)

4 декабря 2020 г. 18:31

Two Modes: Standalone and Distributed

Standalone mode

- 1 worker on 1 machine
- When to use:
  - testing and development
  - non distributable process

Distributed mode

- n workers on m machines
- When to use:
  - production environments
  - for **fault tolerance**
  - and **scalability**

Kafka Connect can be run in two modes

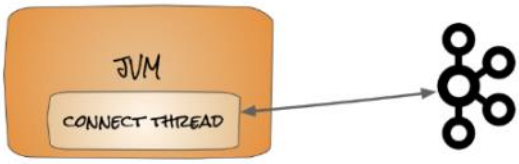
- **Standalone**
  - 1 worker process on single machine
  - Use cases:
    - testing and development, or
    - when a process should not be distributed (*e.g.* `tail` a log file)
- **Distributed mode**
  - Multiple worker processes on one or more machines
  - Use Case: requirements for fault tolerance and scalability

Running in Standalone Mode

To run in standalone mode, start a process by providing as arguments

- Standalone config properties file
- 1…n connector config files

Each connector instance runs in own thread




```
$ connect-standalone connect-standalone.properties \
    connector1.properties [connector2.properties connector3.properties ...]
```

A single worker can run as many connectors as you wish. Once the connectors are running, changes to their configuration files will only take effect on restart of the worker. Making changes to the connectors without a restart will be discussed later in the chapter.

в

Building and running a connector in standalone mode

- Building: *mvn* clean package
- Running (see ./run.sh – only for linux / mac users):
  1. Export classpath:
     - *export* CLASSPATH="$(find target/ -type f -name '*.jar'| grep '\-package' | tr '\n' ':')"
  2. Build the Dockerfile:
     - *docker* build . -t simplesteph/kafka-connect-source-github:1.0
  3. Run the docker image:
     - *docker* run -e CLASSPATH=$CLASSPATH \
       --net=host --rm -t \
       -v $*(pwd)*/offsets:/kafka-connect-source-github/offsets \
       simplesteph/kafka-connect-source-github:1.0

```
connect-standalone config/worker.properties config/GitHubSourceConnectorExample.properties
```

Configuring Workers: Standalone Mode

Parameter	Description
`offset.storage.file.filename`	The filename in which to store offset data for the Connectors (Default: ""). This enables a standalone process to be stopped and then resume where it left off.

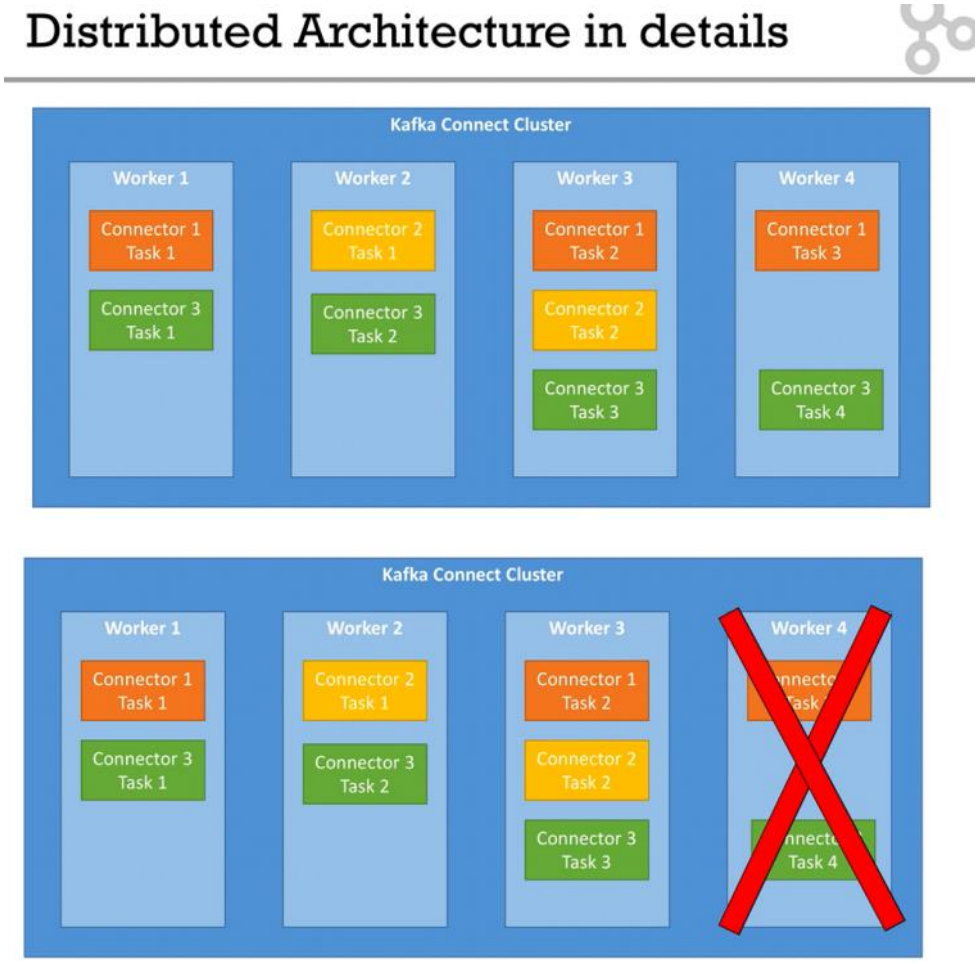
The provided file `/etc/kafka/connect-standalone.properties` sets this parameter to `/tmp/connect.offsets` which is not a very durable location on disk

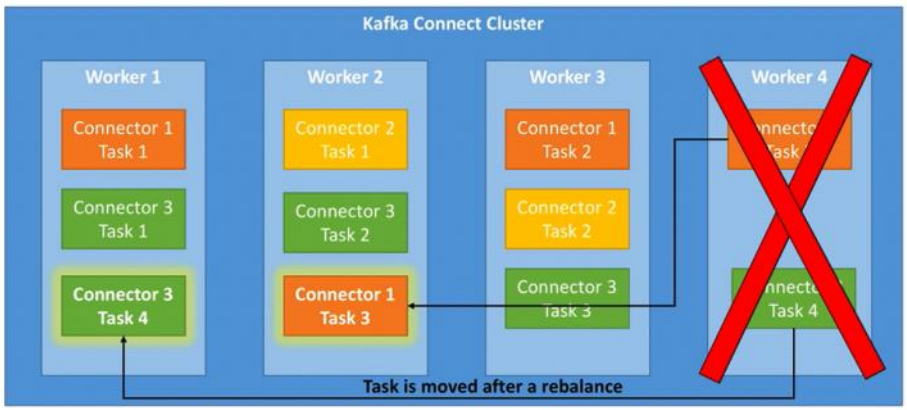
… distributed mode (prod)

4 декабря 2020 г. 18:31

tasks & workers

- таски автоматически распределяются между воркерами (и перераспределяются в случае ошибки)
- таски относящиеся к коннектору не обязательно должны запускаться вместе на одном и том же сервере
- As we can see, Kafka Connect uses a similar technique that we know from consumer groups to reschedule or reallocate workload evenly across the available workers. In our case the connector job B has been moved to worker 2 whilst the connector job C and task C1 have been moved to worker 3.
- Rebalancing/task assignment is performed by the Worker, not by the Connector instance thread. Much like consumer groups, there's a concept of a "leader" Worker which performs task reassignments whenever rebalancing is triggered, which can involve movement of connector threads in addition to tasks. See https://github.com/apache/kafka/blob/2.2/connect/runtime/src/main/java/org/apache/kafka/connect/runtime/distributed/WorkerCoordinator.java#L220-L258 for more detail





в отличие от самописных клиентов, стандартный коннектор обеспечивает load balancing + fault tolerance

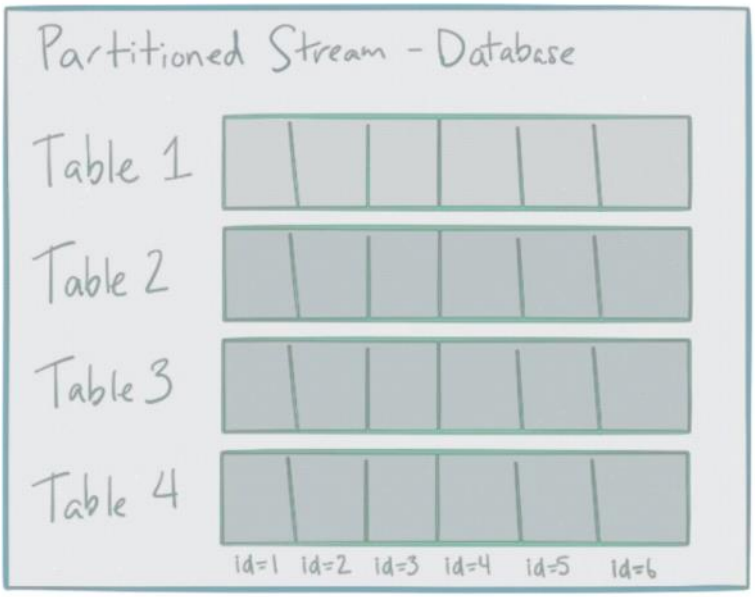
Kafka Connect Reliability and Scalability

- Elasticity and scalability
  - To add resources to Connect simply start more workers
  - Connect Workers discover each other and load-balance the work automatically
  - Allows running more connectors and more tasks
- Reliability and fault-tolerance
  - Failed connector will automatically restart
  - Tasks and connectors on failed worker will automatically (re)start on another worker
- ALL connectors support at-least-once semantics
- SOME connectors support exactly-once semantics
- MANY connectors support schema evolution
  - Data format changes in source system will reflect in Kafka and all target systems

To clarify the point "**Failed Connector will automatically restart**": If a worker fails, any connectors running on that worker will automatically be restarted. If a connector thread or task fails (while the worker continues to run), fault tolerance is not automatic!

connector=DB мы параллельно копируем каждую таблицу в отдельный топик
connector=FILE мы параллельно копируем каждый файл в отдельный топик

Providing Parallelism and Scalability



Kafka Connect also uses the concept of partitioning the input stream of data. Admittedly it is a bit **confusing** to use the same terminology for two different things...

In case of a relational DB as source, the **partitions** of the input stream of data would correspond to the **tables** of the DB

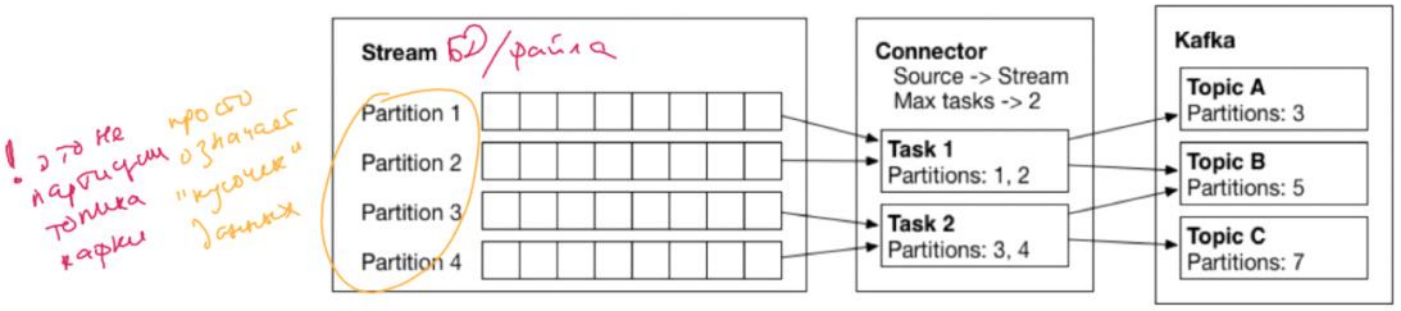
In case of importing CSV files, the **partitions** would correspond to **files**

в кафка коннект термин партиция обозначает не партицию_кафки а просто кусочек данных исходной системы (но идея похожа: например партиция-файл не может быть на значена на два таска)

- см выше: в БД-коннекторе партиция это таблица, а НЕ ее части ,
- в FILE-коннекторе партиция это файл. а НЕ куски одного и того же файла

Providing Parallelism and Scalability

- Partitioning → parallelism & scalability
- 1 Connector job → 1..n **tasks**
- 1 Worker → 1..n **tasks**
- 1 Task per Thread



- Splitting the workload into smaller pieces provides the parallelism and scalability
- Connector jobs are broken down into *tasks* that do the actual copying of the data
- *Workers* are processes running one or more tasks, each in a different thread

A Connector is a Connector class and a configuration. Each connector defines and updates a set of tasks that actually copy the data. Connect distributes these tasks across workers.

In the case of Connect, the term partition can mean any subset of data in the source or the sink. How a partition is represented depends on the type of Connector (e.g., tables are partitions for the JDBC connector, files are the partitions for the FileStream connector).

Worker processes are not managed by Kafka. Other tools can do this: Kubernetes, Mesos, YARN, Chef, Puppet, custom scripts, etc.

not all connectors support multiple tasks and parallelism. For example, the **syslog** source connector only supports one task.

коннектор отслеживает (и записывает себе в специальный топик) позицию прочетнных данных файла источника или таблицы БД

- для обеспечения EOS в sink-коннекторах требуется хранить позицию прочетнных данных не в кафке (например если sink=файл то и позция в отдельном файле)

Source and Sink Offsets

- Kafka Connect tracks the produced and consumed offsets so it can restart at the correct place, in case of failure
- What the offset corresponds to depends on the Connector, for example:
  - File input: offset → position in file
  - Database input: offset → timestamp or sequence id
- The method of tracking the offset depends on the specific Connector
  - Each Connector can determine its own way of doing this
- Examples of source and sink offset tracking methods:
  - JDBC source in distributed mode: a special Kafka Topic
  - HDFS sink: an HDFS file
  - FileStream source: a separate local file

As with Kafka topics, the position within the partitions used by Connect must be tracked as well to prevent the unexpected replay of data. As with the partitions, the object used as the offset varies from connector to connector, as does the method of tracking the offset. The most common way to track the offset is through the use of a Kafka topic, though the Connector developer can use whatever he or she wants.

The variation is primarily with source connectors. With sink connectors, the offsets are Kafka topic partition offsets, and they're usually stored in the external system where the data is being written.

connect UI

4 декабря 2020 г. 18:31

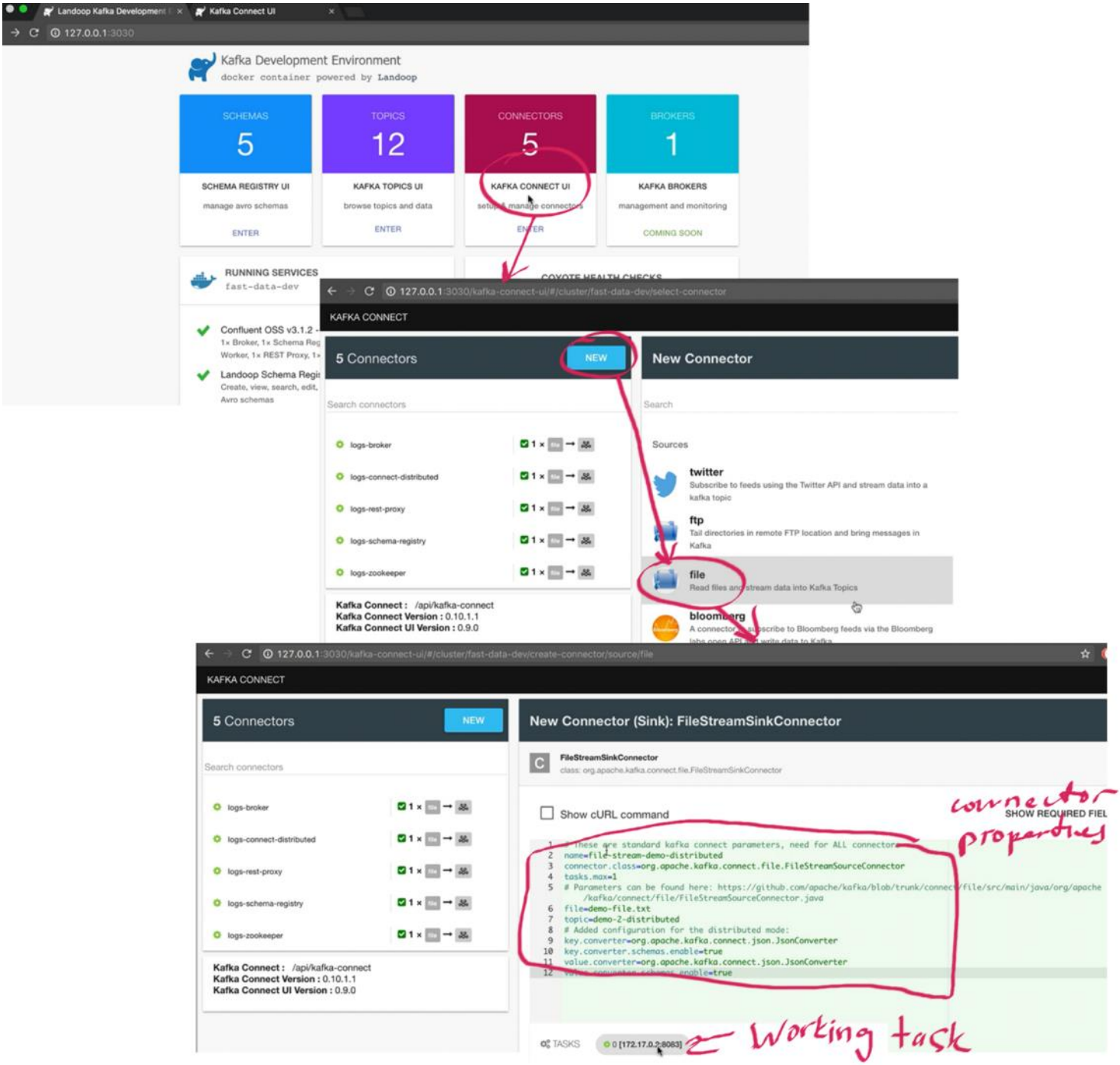
Configuring the Connector

Parameter	Description
`name`	Connector's unique name
`connector.class`	Connector's Java class
`tasks.max`	Maximum tasks to create. The Connector may create fewer if it cannot achieve this level of parallelism (Default: 1)
`key.converter`	(optional) Override the worker key converter
`value.converter`	(optional) Override the worker value converter
`topics` (Sink connectors only)	List of input Topics (to consume from)

используем landoop UI

```
version: '2'

services:
  # this is our kafka cluster.
  kafka-cluster:
    image: landoop/fast-data-dev:cp3.3.0
    environment:
      ADV_HOST: 127.0.0.1         # Change to 192.168.99.100 if using Docker Toolbox
      RUNTESTS: 0                 # Disable Running tests so the cluster starts faster
    ports:
      - 2181:2181                 # Zookeeper
      - 3030:3030                 # Landoop UI
      - 8081-8083:8081-8083       # REST Proxy, Schema Registry, Kafka Connect ports
      - 9581-9585:9581-9585       # JMX Ports
      - 9092:9092                 # Kafka Broker

  # we will use elasticsearch as one of our sinks.
  # This configuration allows you to start elasticsearch
  elasticsearch:
    image: itzg/elasticsearch:2.4.3
    environment:
      PLUGINS: appbaseio/dejavu
      OPTS: -Dindex.number_of_shards=1 -Dindex.number_of_replicas=0
    ports:
      - "9200:9200"

  # we will use postgres as one of our sinks.
  # This configuration allows you to start postgres
  postgres:
    image: postgres:9.5-alpine
    environment:
      POSTGRES_USER: postgres     # define credentials
      POSTGRES_PASSWORD: postgres # define credentials
      POSTGRES_DB: postgres       # define database
    ports:
      - 5432:5432                 # Postgres port
```

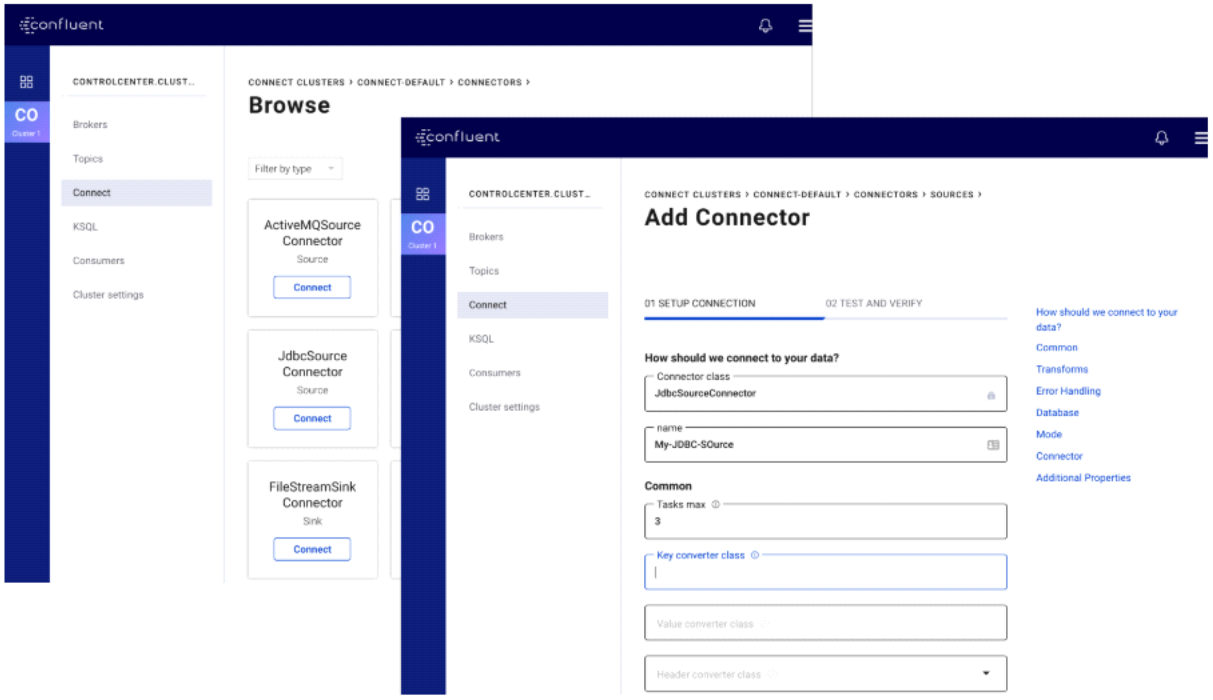
как создать distributed connector через UI


Landoop Kafka Development
Kafka Connect UI
127.0.0.1:3030
Kafka Development Environment
docker container powered by Landoop
SCHEMAS
5
SCHEMA REGISTRY UI
manage avro schemas
ENTER
TOPICS
12
KAFKA TOPICS UI
browse topics and data
ENTER
CONNECTORS
5
KAFKA CONNECT UI
ENTER
BROKERS
1
KAFKA BROKERS
management and monitoring
COMING SOON
RUNNING SERVICES
fast-data-dev
Confluent OSS v3.1.2
Landoop Schema Regi
Create, view, search, edit,
Avro schemas
127.0.0.1:3030/kafka-connect-ui/#/cluster/fast-data-dev/select-connector
KAFKA CONNECT
5 Connectors
NEW
Search connectors
logs-broker
logs-connect-distributed
logs-rest-proxy
logs-schema-registry
logs-zookeeper
Kafka Connect : /api/kafka-connect
Kafka Connect Version : 0.10.1.1
Kafka Connect UI Version : 0.9.0
New Connector
Search
Sources
twitter
Subscribe to feeds using the Twitter API and stream data into a kafka topic
ftp
Tail directories in remote FTP location and bring messages in Kafka
file
Read files and stream data into Kafka Topics
bloomberg
127.0.0.1:3030/kafka-connect-ui/#/cluster/fast-data-dev/create-connector/source/file
KAFKA CONNECT
5 Connectors
NEW
Search connectors
logs-broker
logs-connect-distributed
logs-rest-proxy
logs-schema-registry
logs-zookeeper
Kafka Connect : /api/kafka-connect
Kafka Connect Version : 0.10.1.1
Kafka Connect UI Version : 0.9.0
New Connector (Sink): FileStreamSinkConnector
FileStreamSinkConnector
class: org.apache.kafka.connect.file.FileStreamSinkConnector
Show cURL command
SHOW REQUIRED FIEL
connector properties
# These are standard kafka connect parameters, need for ALL connectors
name=file-stream-demo-distributed
connector.class=org.apache.kafka.connect.file.FileStreamSourceConnector
tasks.max=1
# Parameters can be found here: https://github.com/apache/kafka/blob/trunk/connect/file/src/main/java/org/apache/kafka/connect/file/FileStreamSourceConnector.java
file=demo-file.txt
topic=demo-2-distributed
# Added configuration for the distributed mode:
key.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=true
TASKS
0 [172.17.0.2:8083]
Working task

… Connectors with Confluent Control Center

18 декабря 2020 г. 11:40

Configuring Connectors with Confluent Control Center



Connectors can also be configured in Confluent Control Center. We can define source and sink connectors there.

Not all connectors may be fully configurable in the Control Center. In this case one has to use the Connect REST API to create and configure the connector. In the hands-on lab we give an example of this.

я могу динамически менять конфигурацию работающего коннектора

connect REST API

4 декабря 2020 г. 18:31

Using the REST API

Some important REST endpoints

Method	Path	Description
GET	/connectors	Get a list of active connectors
POST	/connectors	Create a new Connector
GET	/connectors/(string: name)/config	Get configuration information for a Connector
PUT	/connectors/(string: name)/config	Create a new Connector, or update the configuration of an existing Connector

More information on the REST API can be found at https://docs.confluent.io/current/connect/references/restapi.html

Configuring Connectors with the REST API

- Add, modify delete connectors
- Distributed Mode:
  - Config **only** via REST API
  - Config stored in Kafka topic
  - REST call to **any** worker
- Standalone Mode:
  - Config also via REST API
  - Changes **not persisted**!
- Control Center uses REST API

- Connectors can be added, modified, and deleted via a REST API on port 8083
- In distributed mode, configuration can be done only via this REST API
  - Changes made this way will persist after a worker process restart
  - Connector configuration data is stored in a special Kafka Topic
  - The REST requests can be made to any worker
- In standalone mode, configuration can also be done via a REST API
  - However, typically configuration is done via a properties file
    - Changes made via the REST API when running in standalone mode will not persist after worker restart
- Confluent Control Center leverages this REST API to let users configure and manage connectors through the GUI

To make changes to connectors, the Worker will also run a REST API as part of its code. This allows HTTP calls to be sent to any of the Workers that will then configure the connectors. Changes made via the REST API take effect immediately and without a reboot.

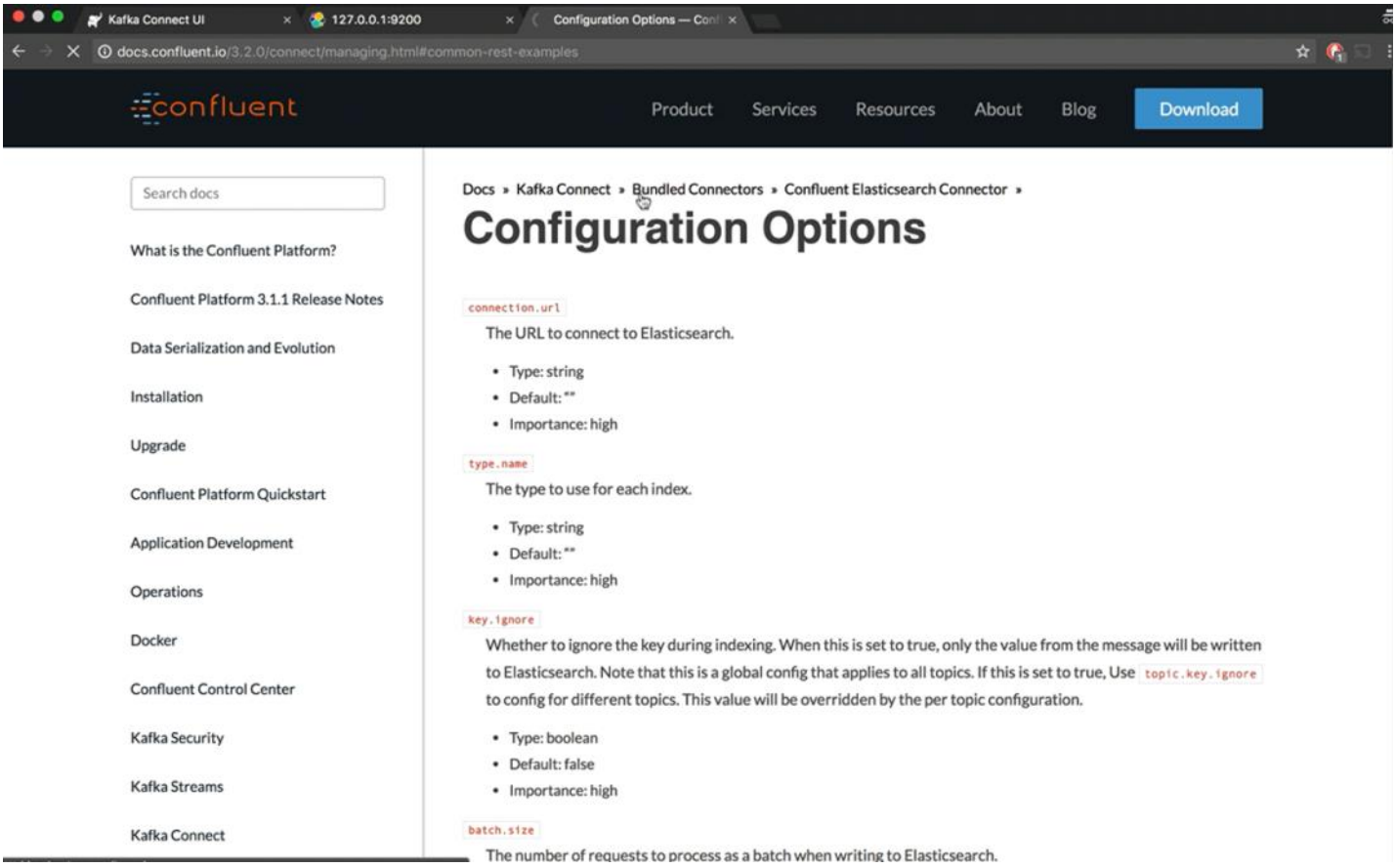
The REST API included with a Connect Worker is different from the Confluent REST Proxy.

kafka connect полностью работает только через REST API

Kafka Connect REST API

- All the actions performed by Landoop Kafka Connect UI are actually triggering REST API calls to Kafka Connect.
- Let's learn about those (http://docs.confluent.io/3.2.0/connect/managing.html#common-rest-examples)

1. Get Worker information
2. List Connectors available on a Worker
3. Ask about Active Connectors
4. Get information about a Connector Tasks and Config
5. Get Connector Status
6. Pause / Resume a Connector
7. Delete our Connector
8. Create a new Connector
9. Update Connector configuration
10. Get Connector Configuration



пример

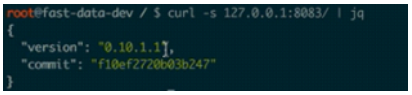
```
#!/bin/bash

# let's start a command line to all have linux commands
docker run --rm -it --net=host landoop/fast-data-dev bash
# Install jq to pretty print json
apk update && apk add jq

# Examples are taken from here: http://docs.confluent.io/3.2.0/connect/managing.html#common-rest-examples
# Replace 127.0.0.1 by 192.168.99.100 if you're using docker toolbox
```

1) Get Worker information

```
curl -s 127.0.0.1:8083/ | jq
```

2) List Connectors available on a Worker

```
curl -s 127.0.0.1:8083/connector-plugins | jq
```

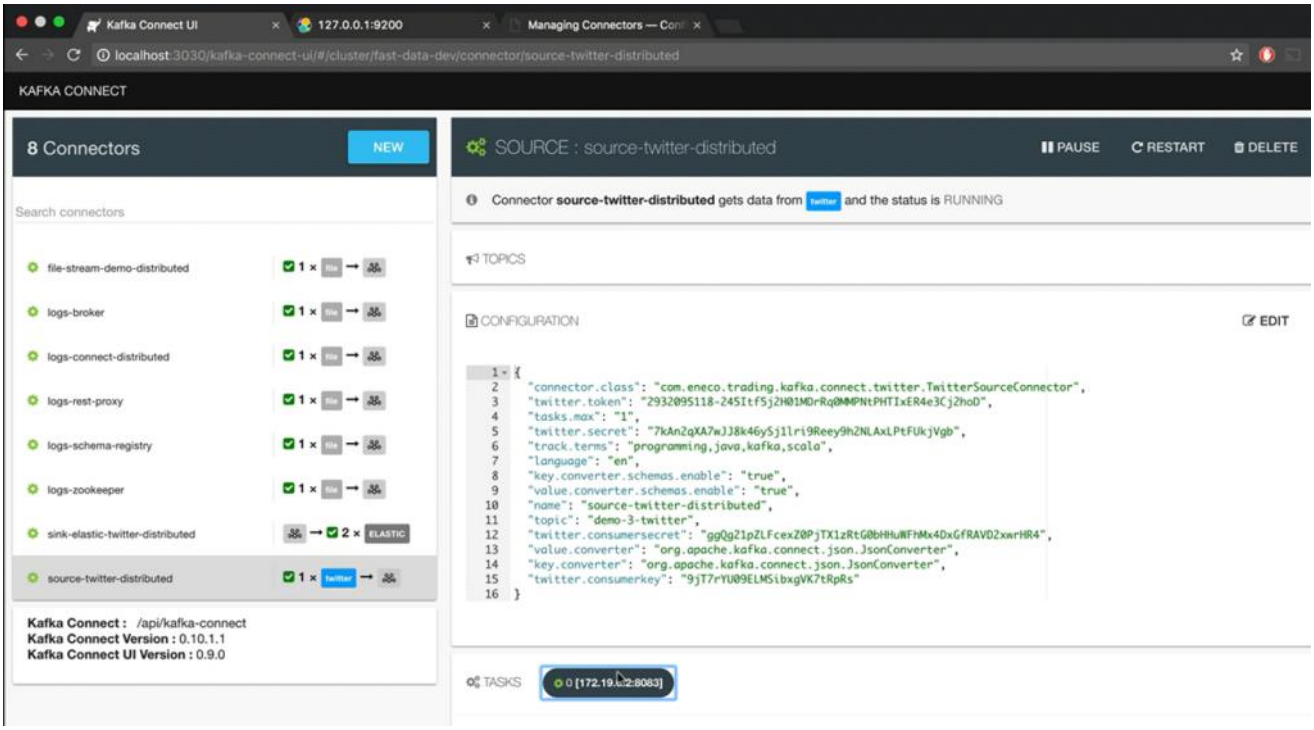


3) Ask about Active Connectors

```
curl -s 127.0.0.1:8083/connectors | jq
```

4) Get information about a Connector Tasks and Config

```
curl -s 127.0.0.1:8083/connectors/source-twitter-distributed/tasks | jq
```



5) Get Connector Status

curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed/status | jq

6) Pause / Resume a Connector (no response if the call is succesful)

curl -s -X PUT 127.0.0.1:8083/connectors/file-stream-demo-distributed/pause
curl -s -X PUT 127.0.0.1:8083/connectors/file-stream-demo-distributed/resume

7) Get Connector Configuration

curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed | jq

8) Delete our Connector

curl -s -X DELETE 127.0.0.1:8083/connectors/file-stream-demo-distributed

9) Create a new Connector

curl -s -X POST -H "Content-Type: application/json" --data '{"name": "file-stream-demo-distributed",
"config":{"connector.class":"org.apache.kafka.connect.file.FileStreamSourceConnector","key.converter.schemas.enable
":"true","file":"demo-file.txt","tasks.max":"1","value.converter.schemas.enable":"true","name":"file-stream-demo-
distributed","topic":"demo-2-
distributed","value.converter":"org.apache.kafka.connect.json.JsonConverter","key.converter":"org.apache.kafka.conn
ect.json.JsonConverter"}}' http://127.0.0.1:8083/connectors | jq

10) Update Connector configuration

```
curl -s -X PUT -H "Content-Type: application/json" --data
'{"connector.class":"org.apache.kafka.connect.file.FileStreamSourceConnector","key.converter.schemas.enable":"true"
,"file":"demo-file.txt","tasks.max":"2","value.converter.schemas.enable":"true","name":"file-stream-demo-
distributed","topic":"demo-2-
distributed","value.converter":"org.apache.kafka.connect.json.JsonConverter","key.converter":"org.apache.kafka.conn
ect.json.JsonConverter"}' 127.0.0.1:8083/connectors/file-stream-demo-distributed/config | jq
```

как создать свой коннектор

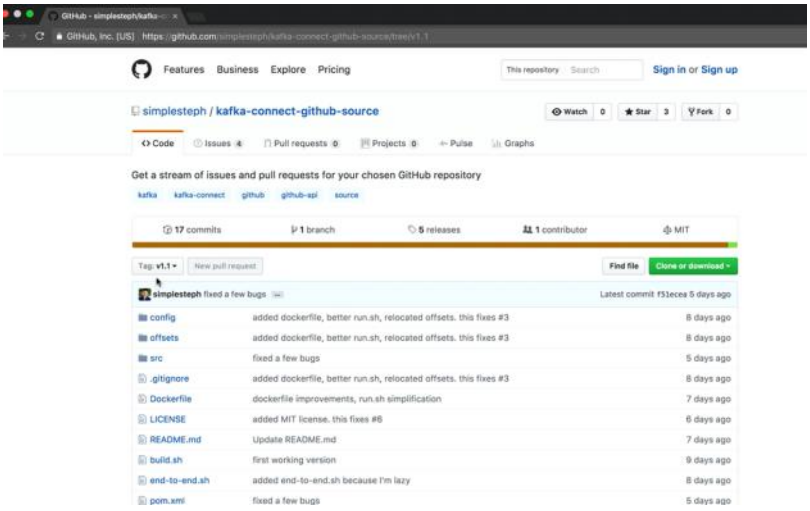
5 декабря 2020 г. 0:25

Section goals

- Understand how connectors are made, using the GitHubSourceConnector as an example
  - Dependencies
  - ConfigDef
  - Connector
  - Schema & Struct
  - Source Partition & Source Offsets
  - Task
- Learn how to deploy your connectors (or any connector)
  - Package Jars
  - Run jars in standalone mode
  - Deploy Jars

- The code can be found at https://github.com/simplesteph/kafka-connect-github-source/tree/v1.1
- It's my own code that I put together specially for this class!
- **Knowledge of Java is necessary for this section**
- I recommend you install IntelliJ Community Editions on your laptop to have the same environment as me: https://www.jetbrains.com/idea/download/
- You need to install Java JDK 8 (Java SE Development Kit) as well: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

- https://www.confluent.io/product/connectors/
  - List of open source connectors
  - Read the code!!!
- https://www.confluent.io/wp-content/uploads/Partner-Dev-Guide-for-Kafka-Connect.pdf?x18424
  - General guidelines
- http://docs.confluent.io/3.2.0/connect/devguide.html



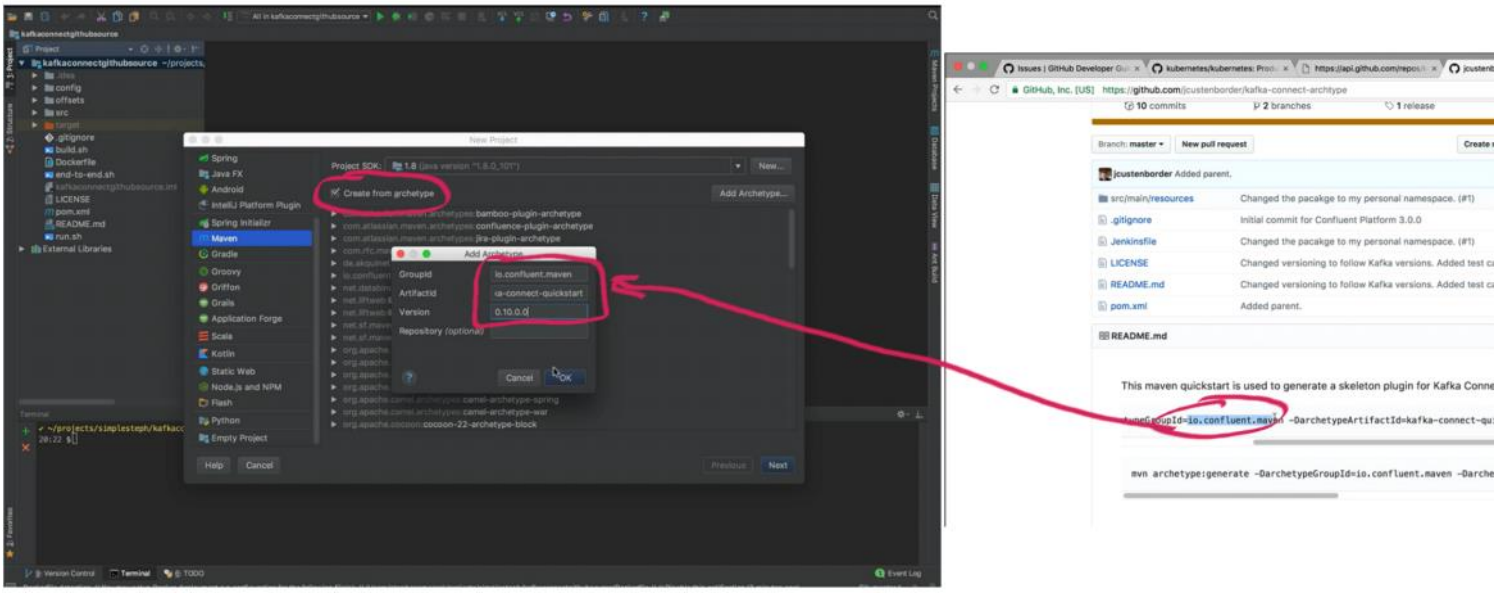
intelijIDEA проект будет создаваться из архетипа

Maven - Archetype

- The easiest way to get started on a Kafka Connect project is to use an archetype provided here: https://github.com/jcustenborder/kafka-connect-archtype

- We will do a walkthrough on IntelliJ to install and use that archetype

- We will change the Kafka dependency to be 0.10.2.0 (use the one matching the version from Landoop)



Connector

- That's the class of your Kafka Connector that is referenced from the configs. It should load the config, and create a few tasks.

```
public String version()
public void start(Map<String, String> map)
public Class<? extends Task> taskClass()
public List<Map<String, String>> taskConfigs(int i)
public void stop()
public ConfigDef config()
```

GitHubSourceConnector

```
1  package com.simplesteph.kafka;
2
3  import ...
12
13 public class GitHubSourceConnector extends SourceConnector {
14     private static Logger log = LoggerFactory.getLogger(GitHubSourceConnector.class);
15     private GitHubSourceConnectorConfig config;
16
17     @Override
18     public String version() { return VersionUtil.getVersion(); }
21
22     @Override
23     public void start(Map<String, String> map) { config = new GitHubSourceConnectorConfig(map); }
26
27     @Override
28     public Class<? extends Task> taskClass() { return GitHubSourceTask.class; }
31
32     @Override
33     public List<Map<String, String>> taskConfigs(int i) {
34         // Define the individual task configurations that will be executed.
35         ArrayList<Map<String, String>> configs = new ArrayList<>( initialCapacity: 1);
36         configs.add(config.originalsStrings());
37         return configs;
38     }
39
40     @Override
41     public void stop() {
42         // Do things that are necessary to stop your connector.
43         // nothing is necessary to stop for this connector
```

… как задеплоить коннектор (distributed mode)

5 декабря 2020 г. 0:25

confluent-hub install как задеплоить стандартный коннектор, который есть на confluetn hub

From Confluent Hub

- Use `confluent-hub` client included with Confluent Platform

```
$ confluent-hub install debezium/debezium-connector-mysql:latest
```

как задеплоить коннектор из обычного джарника

1) plugin.path в конфиге нужно указать путь к джарнику коннектора на серваке

From other sources

- Package Connector in JAR file
- Install JAR file on **all** Kafka Connect worker machines
  - Connectors are installed as plugins
  - There is **library isolation** between plugins

```
plugin.path=/path/to/my/plugins
```

Prior to Kafka 0.11.0, JARs were placed in a path specified by `CLASSPATH` and there was no library isolation between connectors

```
$ export CLASSPATH="$CLASSPATH:/path/to/my/connectors/*"
```

QUESTION: Why is library isolation important?

Answer: *Connectors built by different developers might use different versions of the same library.*

2) у всех коннекторов должен быть единый group.id - общая очередь для хранения общего конфига

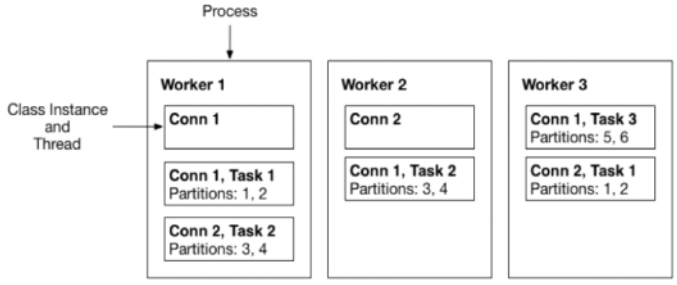
Running in Distributed Mode

Start Kafka Connect **on each** worker node

```
$ connect-distributed connect-distributed.properties
```

Group coordination

- Connect leverages Kafka's group membership protocol
  - Configure workers with the same `group.id`
- Workers distribute load within this Kafka Connect "cluster"



In distributed mode, the connector configurations **cannot** be kept on the Worker systems; a failure of a worker should not make the configurations unavailable. Instead, distributed workers keep their connector configurations in a special Kafka topic which is specified in the worker configuration file.

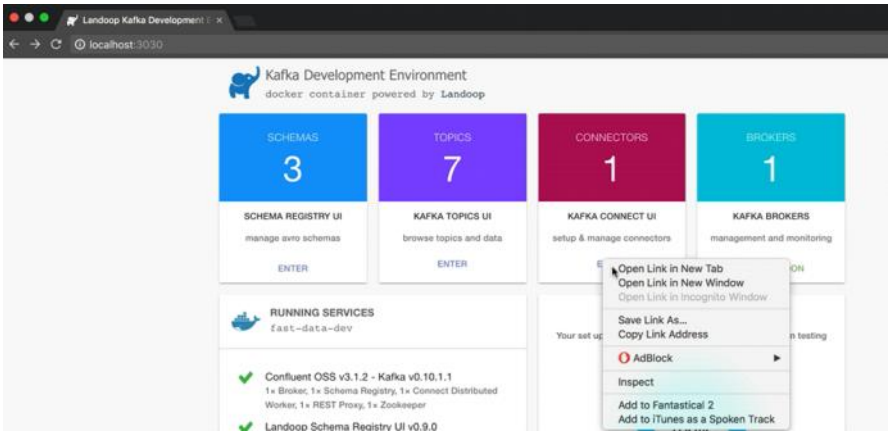
Workers coordinate their tasks to distribute the workload using the same mechanisms as Consumer Groups.

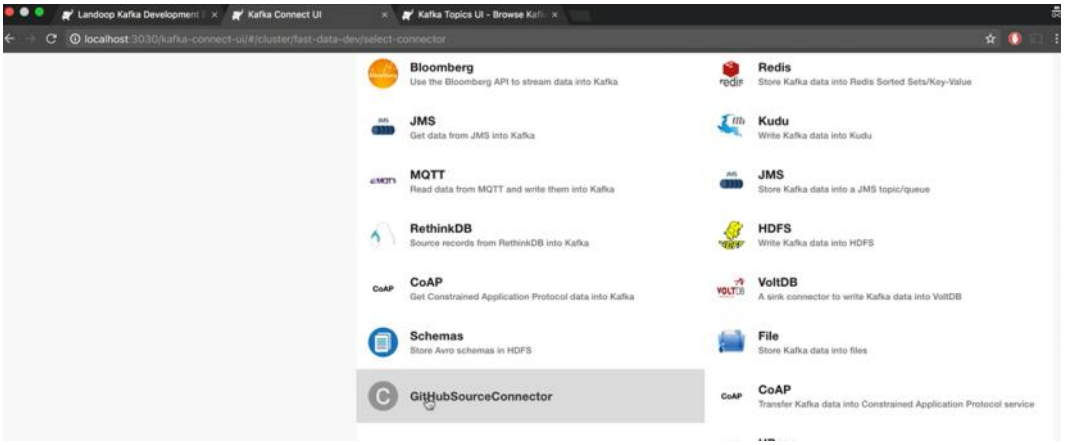
как конектор задеплоить в distrubuted через GUI

положить все jar моего коннектора в папку /connectors на серваке с кафкой

```
13:18 $ docker run -it --rm -p 2181:2181 -p 3030:3030 -p 8081:8081 -p 8082:8082 -p 8083:8083 -p 9092:9092 -e A
DV_HOST=127.0.0.1 -e RUNTESTS=0 -v ~/projects/simplesteph/kafkaconnectgithubsource/target/kafka-connect-github
-source-1.1-package/share/java/kafka-connect-github-source:/connectors landoop/fast-data-dev
```

коннектор автоматически подхватится и отобразится в UI





как задеплоить в продакшен

Setting up Kafka Connect in Production (2/2)

To setup Kafka Connect in production

1. Download Kafka Binaries
2. Set up the connect-distributed.properties as needed
3. Add your jars where needed (plugins.path or classpath)
4. And launch Kafka Connect!
5. (optional) Setup the Landoop Kafka Connect UI

You will be able to interact with Kafka Connect using the REST API, or the UI if you set it up

```
#!/bin/bash

# 0. Make sure Java is installed on your machine https://java.com/en/download/
# 1. Download Kafka at https://kafka.apache.org/downloads (>= 0.11.0.1)
# 2. Unzip Kafka to a directory of your choice (for example ~/kafka)

# 3. Open a terminal or text editor on the Kafka directory
# 4. Edit the file at config/connect-distributed.config
nano config/connect-distributed.properties
vi config/connect-distributed.properties
atom config/connect-distributed.properties

# 5. change bootstrap.servers to the correct ones
# 6. change rest.port=8083 to rest.port=8084 (or any available port)
# 7. Optional (if you want a separate cluster) -
  # 7a. Set offset.storage.topic=connect-offsets-2
  # 7b. Set config.storage.topic=connect-configs-2
  # 7c. Set status.storage.topic=connect-status-2
  # 7d. Set group.id=connect-cluster-2
# 8. Set plugin.path=/directory/of/your/choice
# 9. Adjust any other settings that matters to your setup (see doc)
# 10. Place any connector (jars) you need in the plugin.path sub-directory

# 11. In order to add connect workers, duplicate the connect-distributed file and change the rest.port if
running on the same host. The rest of the settings are the same

# 12. Optionally, setup Kafka Connect UI from Landoop
  # Instructions are at: https://github.com/Landoop/kafka-connect-ui
  # You can build from source or use docker for that (see their github)
```

/etc/kafka/ файл конфига всегда лежит в одной и той же папке

- You can modify Connect configuration settings
  - Distributed mode in /etc/kafka/connect-distributed.properties
  - Standalone mode in /etc/kafka/connect-standalone.properties
  - http://docs.confluent.io/current/connect/

config/connect-distributed.properties подправим настройки конфигов на всех коннектор-нодах кафки

```
~/kafka_2.11-0.11.0.1 ll config
total 128
-rw-r--r--@ 1 stephanemaarek  staff   906B  6 Sep 04:18 connect-console-sink.properties
-rw-r--r--@ 1 stephanemaarek  staff   909B  6 Sep 04:18 connect-console-source.properties
-rw-r--r--@ 1 stephanemaarek  staff   5.7K  6 Sep 04:18 connect-distributed.properties
-rw-r--r--@ 1 stephanemaarek  staff   883B  6 Sep 04:18 connect-file-sink.properties
-rw-r--r--@ 1 stephanemaarek  staff   881B  6 Sep 04:18 connect-file-source.properties
-rw-r--r--@ 1 stephanemaarek  staff   1.1K  6 Sep 04:18 connect-log4j.properties
-rw-r--r--@ 1 stephanemaarek  staff   2.7K  6 Sep 04:18 connect-standalone.properties
-rw-r--r--@ 1 stephanemaarek  staff   1.2K  6 Sep 04:18 consumer.properties
-rw-r--r--@ 1 stephanemaarek  staff   4.6K  6 Sep 04:18 log4j.properties
-rw-r--r--@ 1 stephanemaarek  staff   1.9K  6 Sep 04:18 producer.properties
-rw-r--r--@ 1 stephanemaarek  staff   6.8K  6 Sep 04:18 server.properties
-rw-r--r--@ 1 stephanemaarek  staff   1.0K  6 Sep 04:18 tools-log4j.properties
-rw-r--r--@ 1 stephanemaarek  staff   1.0K  6 Sep 04:18 zookeeper.properties
```

A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. указать бутстрап сервер кафки
bootstrap.servers=127.0.0.1:9092

# unique name for the cluster, used in forming the Connect cluster group. Note that this must not conflict with consumer grou p Ids лучше создать свой кластер коннекторов (у всех коннекторов едина группа)
group.id=connect-cluster-2

# The converters specify the format of data in Kafka and how to translate it into Connect data. Every Connect user will
# need to configure these based on the format they want their data in when loaded from or stored into Kafka
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
# Converter-specific settings can be passed in by prefixing the Converter's setting with the converter we want to apply
# it to
key.converter.schemas.enable=true
value.converter.schemas.enable=true

# The internal converter used for offsets, config, and status data is configurable and must be specified, but most users will
# always want to use the built-in default. Offset, config, and status data is never visible outside of Kafka Connect in this format.
internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter=org.apache.kafka.connect.json.JsonConverter
internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false

# Topic to use for storing offsets. This topic should have many partitions and be replicated and compacted.
# Kafka Connect will attempt to create the topic automatically when needed, but you can always manually create
# the topic before starting Kafka Connect if a specific topic configuration is needed.
# Most users will want to use the built-in default replication factor of 3 or in some cases even specify a larger value.
# Since this means there must be at least as many brokers as the maximum replication factor used, we'd like to be able
# to run this example on a single-broker cluster and so here we instead set the replication factor to 1. лучше создать свой топик для хранения офсетов
offset.storage.topic=connect-offsets-2
offset.storage.replication.factor=1
#offset.storage.partitions=25

# Topic to use for storing connector and task configurations; note that this should be a single partition, highly replicated,
# and compacted topic. Kafka Connect will attempt to create the topic automatically when needed, but you can always manually create
# the topic before starting Kafka Connect if a specific topic configuration is needed.
# Most users will want to use the built-in default replication factor of 3 or in some cases even specify a larger value.
# Since this means there must be at least as many brokers as the maximum replication factor used, we'd like to be able
# to run this example on a single-broker cluster and so here we instead set the replication factor to 1. лучше создать свой топик для хранения конфигов
config.storage.topic=connect-configs-2
config.storage.replication.factor=1

# Topic to use for storing statuses. This topic can have multiple partitions and should be replicated and compacted.
# Kafka Connect will attempt to create the topic automatically when needed, but you can always manually create
# the topic before starting Kafka Connect if a specific topic configuration is needed.
# Most users will want to use the built-in default replication factor of 3 or in some cases even specify a larger value.
# Since this means there must be at least as many brokers as the maximum replication factor used, we'd like to be able
# to run this example on a single-broker cluster and so here we instead set the replication factor to 1. лучше создать свой топик для хранения статусов
status.storage.topic=connect-status-2
status.storage.replication.factor=1
#status.storage.partitions=5

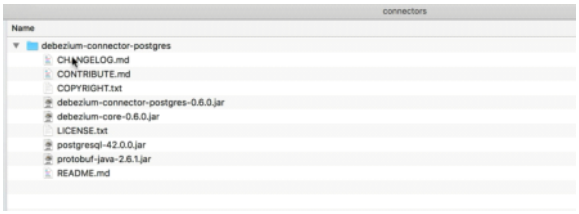
# Flush much faster than normal, which is useful for testing/debugging
offset.flush.interval.ms=10000

These are provided to inform the user about the presence of the REST host and port configs
Hostname & Port for the REST API to listen on. If this is set, it will bind to the interface used to listen to requests. на второй коннектор ноде нужно указать свой порт, например 8085
#rest.host.name=
rest.port=8084

The Hostname & Port that will be given out to other workers to connect to i.e. URLs that are routable from other servers.
#rest.advertised.host.name=
#rest.advertised.port=

# Set to a list of filesystem paths separated by commas (,) to enable class loading isolation for plugins
# (connectors, converters, transformations). The list should consist of top level directories that include
# any combination of:
# a) directories immediately containing jars with plugins and their dependencies
# b) uber-jars with plugins and their dependencies
# c) directories immediately containing the package directory structure of classes of plugins and their dependencies
# Examples:
# plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors, путь к моенму коннектору на серваке
plugin.path=/Users/stephanemaarek/kafka_2.11-0.11.0.1/connectors

поместим на коннектор-нода в папку /connectors с коннекторами jar-ник коннекторов



запустим на каждой коннек-ноде

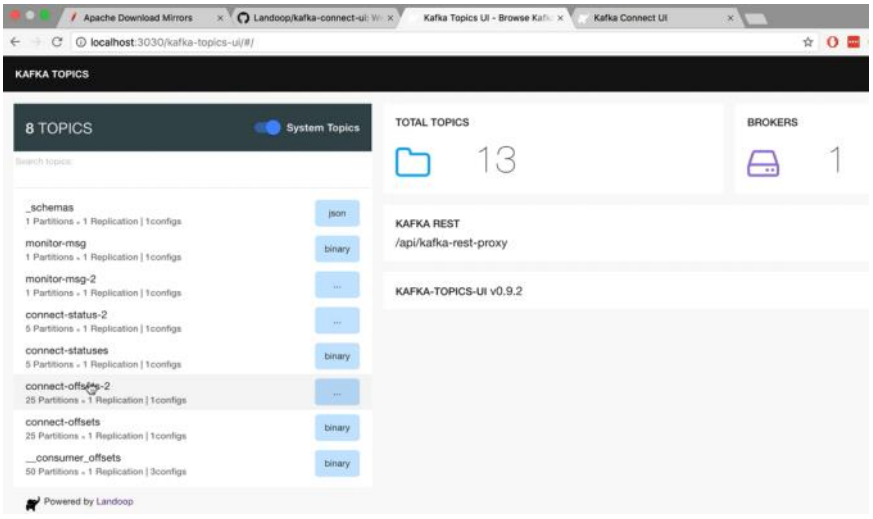
```
~/kafka_2.11-0.11.0.1 bin/connect-distributed.sh config/connect-distributed.properties
```

```
[2017-10-05 18:57:39,867] INFO Successfully joined group connect-cluster-2 with generation 1 (org.apache.kafka.clients.
consumer.internals.AbstractCoordinator:409)
[2017-10-05 18:57:39,867] INFO Joined group and got assignment: Assignment{error=0, leader='connect-1-4acb32d5-5f35-410
6-ad84-1d29bdb84f11', leaderUrl='http://192.168.1.5:8084/', offset=-1, connectorIds=[], taskIds=[]} (org.apache.kafka.c
onnect.runtime.distributed.DistributedHerder:1166)
[2017-10-05 18:57:39,868] INFO Starting connectors and tasks using config offset -1 (org.apache.kafka.connect.runtime.d
istributed.DistributedHerder:815)
[2017-10-05 18:57:39,868] INFO Finished starting connectors and tasks (org.apache.kafka.connect.runtime.distributed.Dis
tributedHerder:825)

consumer.internals.AbstractCoordinator:409)
[2017-10-05 19:06:52,433] INFO Joined group and got assignment: Assignment{error=0, leader='connect-1-71ac447d-878a-4d3
a-a660-44e741a4bef7', leaderUrl='http://192.168.1.5:8084/', offset=-1, connectorIds=[], taskIds=[]} (org.apache.kafka.c
onnect.runtime.distributed.DistributedHerder:1166)
```

проверим что обе ноды с коннекторами работают

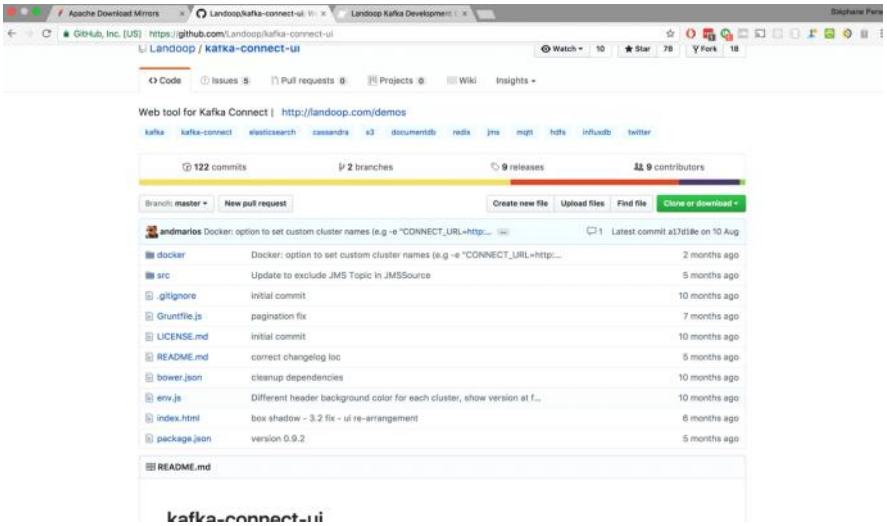
```
~/kafka_2.11-0.11.0.1 curl localhost:8085
{"version":"0.11.0.1","commit":"c2a0d5f9b1f45bf5"}%
~/kafka_2.11-0.11.0.1 curl localhost:8084
{"version":"0.11.0.1","commit":"c2a0d5f9b1f45bf5"}%
```

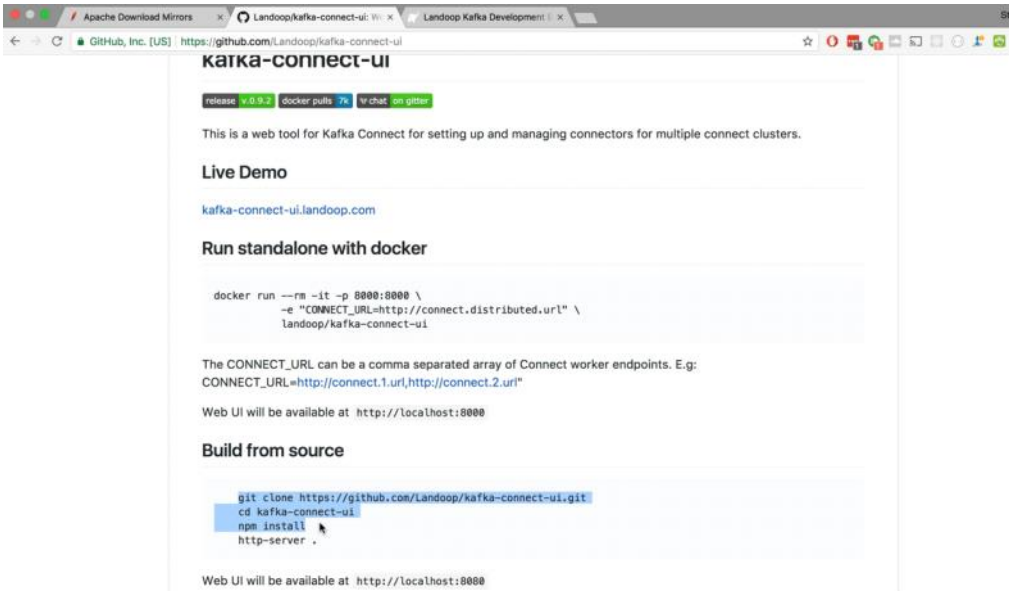
проверим что появились топики (connect-offset-2) для кластера-коннекторов

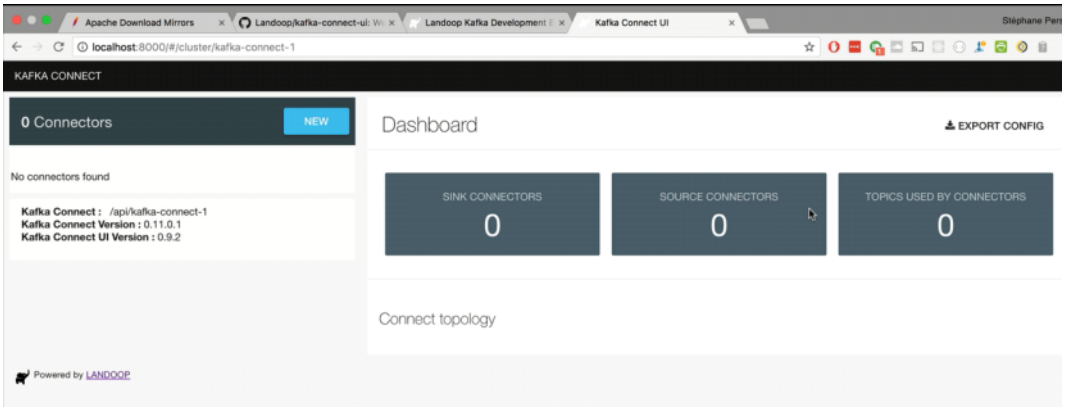


установим landoop connect UI (можно даже на отдельном докере и указать URL к коннект-нодам)

```
~/kafka_2.11-0.11.0.1 docker run --net=host --rm -it  \
           -e "CONNECT_URL=http://127.0.0.1:8084" \
           landoop/kafka-connect-ui
```


Landoop / kafka-connect-ui
Web tool for Kafka Connect | http://landoop.com/demos
122 commits
2 branches
9 releases
9 contributors
README.md
kafka-connect-ui


kafka-connect-ui
This is a web tool for Kafka Connect for setting up and managing connectors for multiple connect clusters.
Live Demo
kafka-connect-ui.landoop.com
Run standalone with docker
docker run --rm -it -p 8000:8000 \
-e "CONNECT_URL=http://connect.distributed.url" \
landoop/kafka-connect-ui
The CONNECT_URL can be a comma separated array of Connect worker endpoints. E.g:
CONNECT_URL=http://connect.1.url,http://connect.2.url"
Web UI will be available at http://localhost:8000
Build from source
git clone https://github.com/Landoop/kafka-connect-ui.git
cd kafka-connect-ui
npm install
http-server .
Web UI will be available at http://localhost:8000


KAFKA CONNECT
0 Connectors
NEW
No connectors found
Kafka Connect : /api/kafka-connect-1
Kafka Connect Version : 0.11.0.1
Kafka Connect UI Version : 0.9.2
Dashboard
EXPORT CONFIG
SINK CONNECTORS
0
SOURCE CONNECTORS
0
TOPICS USED BY CONNECTORS
0
Connect topology
Powered by LANDOOP

.... дополнительные настройки конектора

18 декабря 2020 г. 11:29

Configuring Workers: Distributed Mode

Parameter	Description
`group.id`	A unique string that identifies the Kafka Connect cluster group the worker belongs to
`session.timeout.ms`	Timeout used to detect failures when using Kafka's group management facilities
`heartbeat.interval.ms`	Expected time between heartbeats to the group coordinator when using Kafka's group management facilities. Must be smaller than `session.timeout.ms`
`config.storage.topic`	Topic in which to store Connector & task config. data
`offset.storage.topic`	Topic in which to store offset data for Connectors
`status.storage.topic`	Topic in which to store connector & task status

As with Consumer Groups, the `group.id` will determine which Workers will cooperate as a team.

Connect Worker Groups will use unique administrative topics to hold connector configurations, offsets for both source and sinks, and status (if necessary) of the external data sources/sinks. Different worker groups can use different administrative topics if you need to keep their configurations separate.

Make sure that all Workers with the same `group.id` are using the same names for the administrative topics or you will get unpredictable results.

конфиг-топик создается автоматически но
я могу вручную создать конфиг-топик для того чтобы явно задать replication-factor, partitions-count

- используется **компактед**-топик для хранения настроек

Creating Kafka Connect's Topics Manually

- The Topics used in Kafka Connect distributed mode will be **automatically created**
- They are created with **recommended values** for
  - replication factor
  - partition counts
  - cleanup policy

The following number of Partitions:

Topic	Partitions
`config.storage.topic`	1
`offset.storage.topic`	25
`status.storage.topic`	5

Notice that the recommendations for `offset.storage.topic` are similar to the `__consumer_offsets` Topic since they do similar jobs.

In general, it's usually ok to let Connect create these topics for us. Connect will create these topics with the recommended replication factor, partition counts, and cleanup policy. The only time you might want to create these topics manually would be when you need custom configuration, or Connect doesn't have appropriate ACL privileges to create them. See https://docs.confluent.io/current/connect/userguide.html#distributed-worker-configuration

Configuring the Connector

Parameter	Description
`name`	Connector's unique name
`connector.class`	Connector's Java class
`tasks.max`	Maximum tasks to create. The Connector may create fewer if it cannot achieve this level of parallelism (Default: 1)
`key.converter`	(optional) Override the worker key converter
`value.converter`	(optional) Override the worker value converter
`topics` (Sink connectors only)	List of input Topics (to consume from)

The number of tasks that a Connector should start will be dependent on the number of cores

The number of tasks that a Connector should start will be dependent on the number of cores and workers available.

The reason that the `topics` parameter only exists for sink connectors is that source connectors typically have logic built in that will create new topic names based on a specific naming convention.

Key and value converters were added to the connector configurations in AK 0.10.1. Prior to that, there was a single converter setting that was specified in the worker configurations. Additional converters can be installed via Confluent Hub (when available).

… connectors configuration

4 декабря 2020 г. 18:32

Config Definitions

- Config Def are the way to communicate to the user how you want your Configuration to be
- You can have mandatory or optional parameters (with default), and validate the types and constraints of your parameters. You should also document the parameters

```
public static final String OWNER_CONFIG = "github.owner";
private static final String OWNER_DOC = "Owner of the repository you'd like to follow";

public static ConfigDef conf() {
  return new ConfigDef()
      .define(OWNER_CONFIG, Type.STRING, Importance.HIGH, OWNER_DOC)
      .define(BATCH_SIZE_CONFIG, Type.INT, 100, new BatchSizeValidator(), Importance.LOW, BATCH_SIZE_DOC)
```

```
GitHubSourceConnectorConfig  TOPIC_DOC

import ...

public class GitHubSourceConnectorConfig extends AbstractConfig {

    public static final String TOPIC_CONFIG = "topic";
    private static final String TOPIC_DOC = "Topic to write to";

    public static final String OWNER_CONFIG = "github.owner";
    private static final String OWNER_DOC = "Owner of the repository you'd like to follow";

    public static final String REPO_CONFIG = "github.repo";
    private static final String REPO_DOC = "Repository you'd like to follow";

    public static final String SINCE_CONFIG = "since.timestamp";
    private static final String SINCE_DOC =
            "Only issues updated at or after this time are returned.\n"
                    + "This is a timestamp in ISO 8601 format: YYYY-MM-DDTHH:MM:SSZ.\n"
                    + "Defaults to a year from first launch.";

    public static final String BATCH_SIZE_CONFIG = "batch.size";
    private static final String BATCH_SIZE_DOC = "Number of data points to retrieve at a time
  Defaults to 100 (max value)";

    public static final String AUTH_USERNAME_CONFIG = "auth.username";
    private static final String AUTH_USERNAME_DOC = "Optional Username to authenticate calls'

    public static final String AUTH_PASSWORD_CONFIG = "auth.password";

    public GitHubSourceConnectorConfig(Map<String, String> parsedConfig) { this(conf(),
  parsedConfig); }

    public static ConfigDef conf() {
        return new ConfigDef()
                .define(TOPIC_CONFIG, Type.STRING, Importance.HIGH, TOPIC_DOC)
                .define(OWNER_CONFIG, Type.STRING, Importance.HIGH, OWNER_DOC)
                .define(REPO_CONFIG, Type.STRING, Importance.HIGH, REPO_DOC)
```

```java
    public GitHubSourceConnectorConfig(Map<String, String> parsedConfig) { this(conf(),
parsedConfig); }

    public static ConfigDef conf() {
        return new ConfigDef()
                .define(TOPIC_CONFIG, Type.STRING, Importance.HIGH, TOPIC_DOC)
                .define(OWNER_CONFIG, Type.STRING, Importance.HIGH, OWNER_DOC)
                .define(REPO_CONFIG, Type.STRING, Importance.HIGH, REPO_DOC)
                .define(BATCH_SIZE_CONFIG, Type.INT,  defaultValue: 100, new BatchSizeValidator(),
Importance.LOW, BATCH_SIZE_DOC)
                .define(SINCE_CONFIG, Type.STRING, ZonedDateTime.now().minusYears(1).toInstant()
.toString(),
                        new TimestampValidator(), Importance.HIGH, SINCE_DOC)
                .define(AUTH_USERNAME_CONFIG, Type.STRING,  defaultValue: "", Importance.HIGH,
AUTH_USERNAME_DOC)
```

```
name=GitHubSourceConnectorDemo
tasks.max=1
connector.class=com.simplesteph.kafka.GitHubSourceConnector
topic=github-issues
github.owner=kubernetes
github.repo=kubernetes
since.timestamp=2017-01-01T00:00:00Z
# I heavily recommend you set those two fields:
# auth.username=your_username
# auth.password=your_password
```

… schema (json для выходных данных)

5 декабря 2020 г. 10:39

Schemas

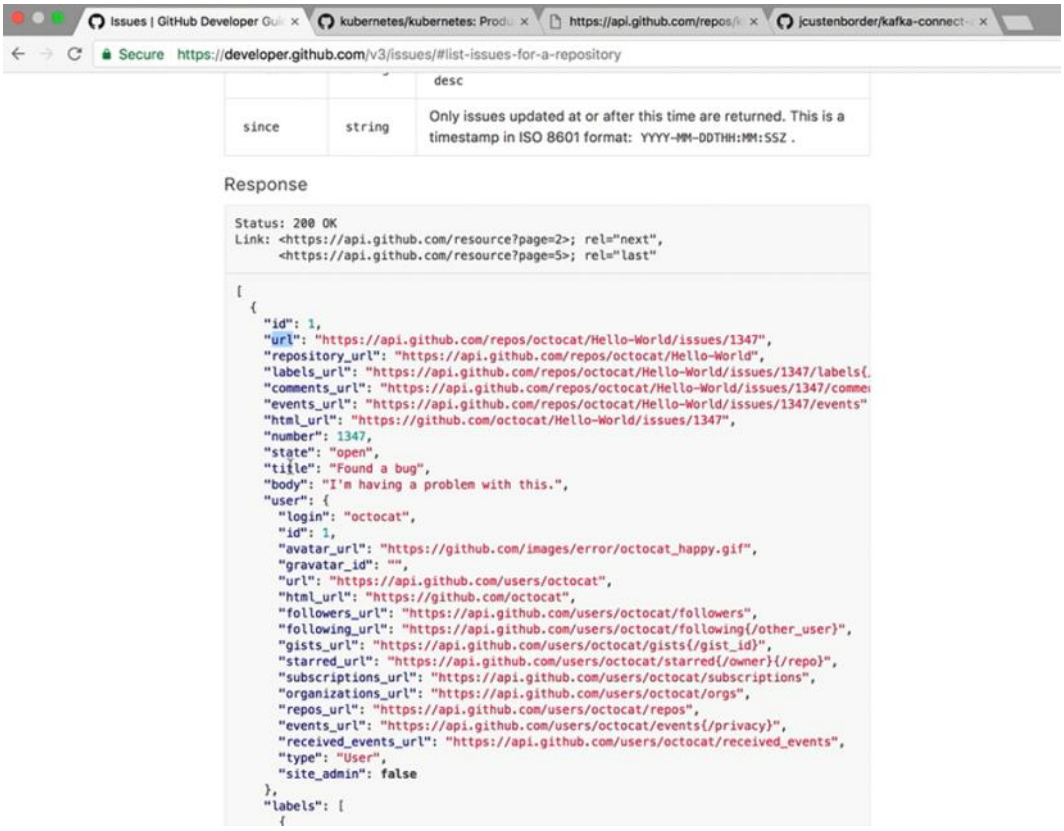
- Schema is an abstraction that allow you to define how your data structure will look like. It is using primitive types, and then the Kafka Connect framework will convert it into JSON or Avro as needed.
- It is necessary to correctly design your schema before you program

```
public static Schema USER_SCHEMA =
SchemaBuilder.struct().name(SCHEMA_VALUE_USER)
        .version(1)
        .field(USER_URL_FIELD, Schema.STRING_SCHEMA)
        .field(USER_ID_FIELD, Schema.INT32_SCHEMA)
        .field(USER_LOGIN_FIELD, Schema.STRING_SCHEMA)
        .build();
```

GitHubSchemas

```
1  package com.simplesteph.kafka;
2
3  import ...
5
6  public class GitHubSchemas {
7
8      public static String NEXT_PAGE_FIELD = "next_page";
9
10     // Issue fields
11     public static String FULL_REPO_FIELD = "owner/repository";
12     public static String OWNER_FIELD = "owner";
13     public static String REPOSITORY_FIELD = "repository";
14     public static String CREATED_AT_FIELD = "created_at";
15     public static String UPDATED_AT_FIELD = "updated_at";
16     public static String NUMBER_FIELD = "number";
17     public static String URL_FIELD = "url";
18     public static String HTML_URL_FIELD = "html_url";
19     public static String TITLE_FIELD = "title";
20     public static String STATE_FIELD = "state";
21
22     // User fields
23     public static String USER_FIELD = "user";
24     public static String USER_URL_FIELD = "url";
25     public static String USER_HTML_URL_FIELD = "html_url";
26     public static String USER_ID_FIELD = "id";
27     public static String USER_LOGIN_FIELD = "login";
28
29     // PR fields
30     public static String PR_FIELD = "pull_request";
```

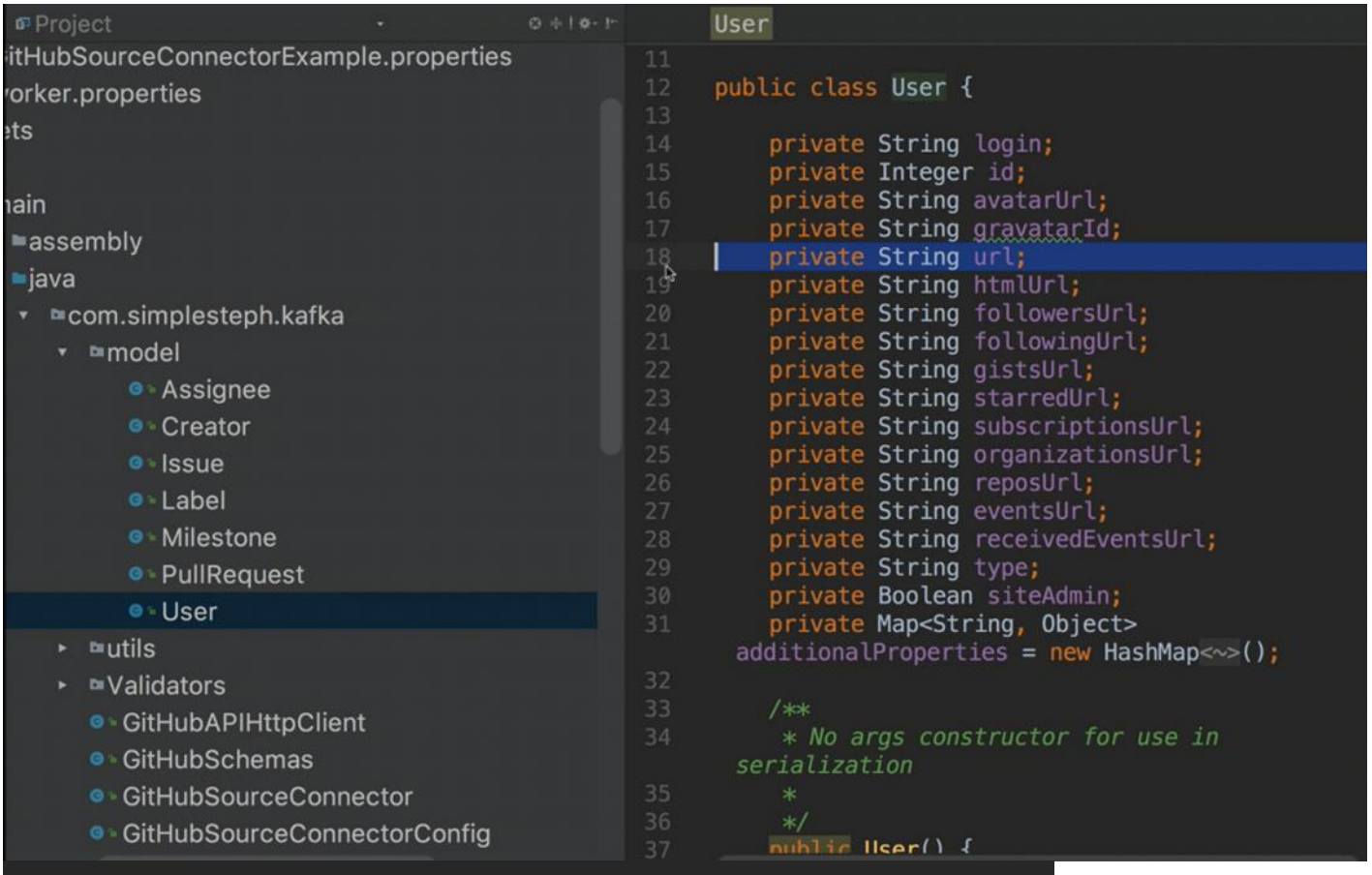
```java
47
48      // Value Schema
49      public static Schema USER_SCHEMA = SchemaBuilder.struct().name(SCHEMA_VALUE_USER)
50              .version(1)
51              .field(USER_URL_FIELD, Schema.STRING_SCHEMA)
52              .field(USER_ID_FIELD, Schema.INT32_SCHEMA)
53              .field(USER_LOGIN_FIELD, Schema.STRING_SCHEMA)
54              .build();
55
56      // optional schema
57      public static Schema PR_SCHEMA = SchemaBuilder.struct().name(SCHEMA_VALUE_PR)
58              .version(1)
59              .field(PR_URL_FIELD, Schema.STRING_SCHEMA)
60              .field(PR_HTML_URL_FIELD, Schema.STRING_SCHEMA)
61              .optional()
62              .build();
63
64      public static Schema VALUE_SCHEMA = SchemaBuilder.struct().name(SCHEMA_VALUE_ISSUE)
65              .version(1)
66              .field(URL_FIELD, Schema.STRING_SCHEMA)
67              .field(TITLE_FIELD, Schema.STRING_SCHEMA)
68              .field(CREATED_AT_FIELD, Schema.INT64_SCHEMA)
69              .field(UPDATED_AT_FIELD, Schema.INT64_SCHEMA)
70              .field(NUMBER_FIELD, Schema.INT32_SCHEMA)
71              .field(STATE_FIELD, Schema.STRING_SCHEMA)
72              .field(USER_FIELD, USER_SCHEMA) // mandatory
73              .field(PR_FIELD, PR_SCHEMA)     // optional
74              .build();
75  }
```



Data Model

- It is always really good to have your data exist in classes in a "model" package.
- I modelled Issue, User and Pull Request as POJOs

```
public class PullRequest {

   private String url;
   private String htmlUrl;
}
```




```
    public static User fromJson(JSONObject jsonObject) {
        User user = new User();
        user.setUrl(jsonObject.getString(USER_URL_FIELD));
        user.setHtmlUrl(jsonObject.getString(USER_HTML_URL_FIELD));
        user.setId(jsonObject.getInt(USER_ID_FIELD));
        user.setLogin(jsonObject.getString(USER_LOGIN_FIELD));
        return user;
    }
}
```

… source partition + source offset

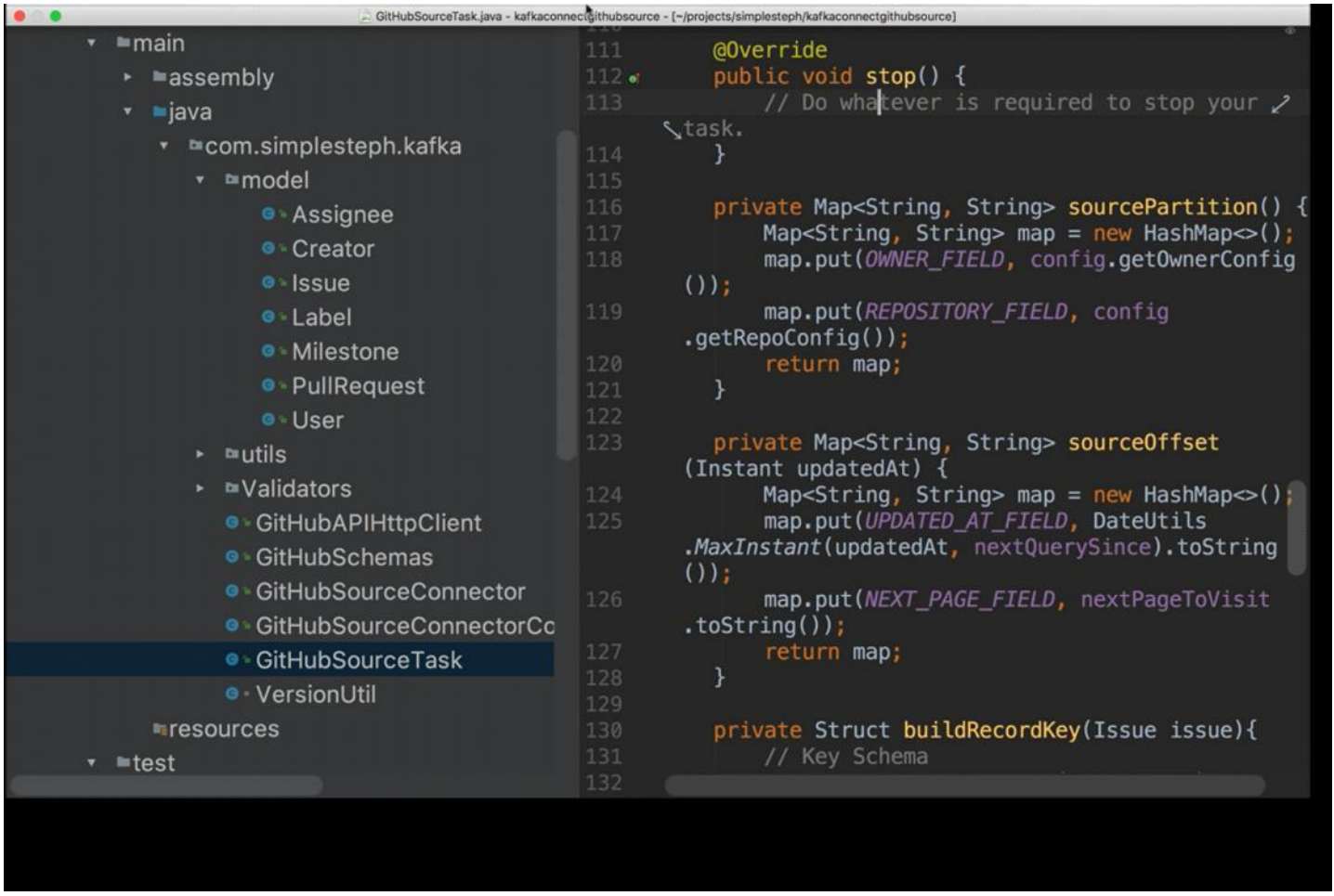
5 декабря 2020 г. 12:14

это НЕ partition и offset в кафке

Source partition, Source Offsets

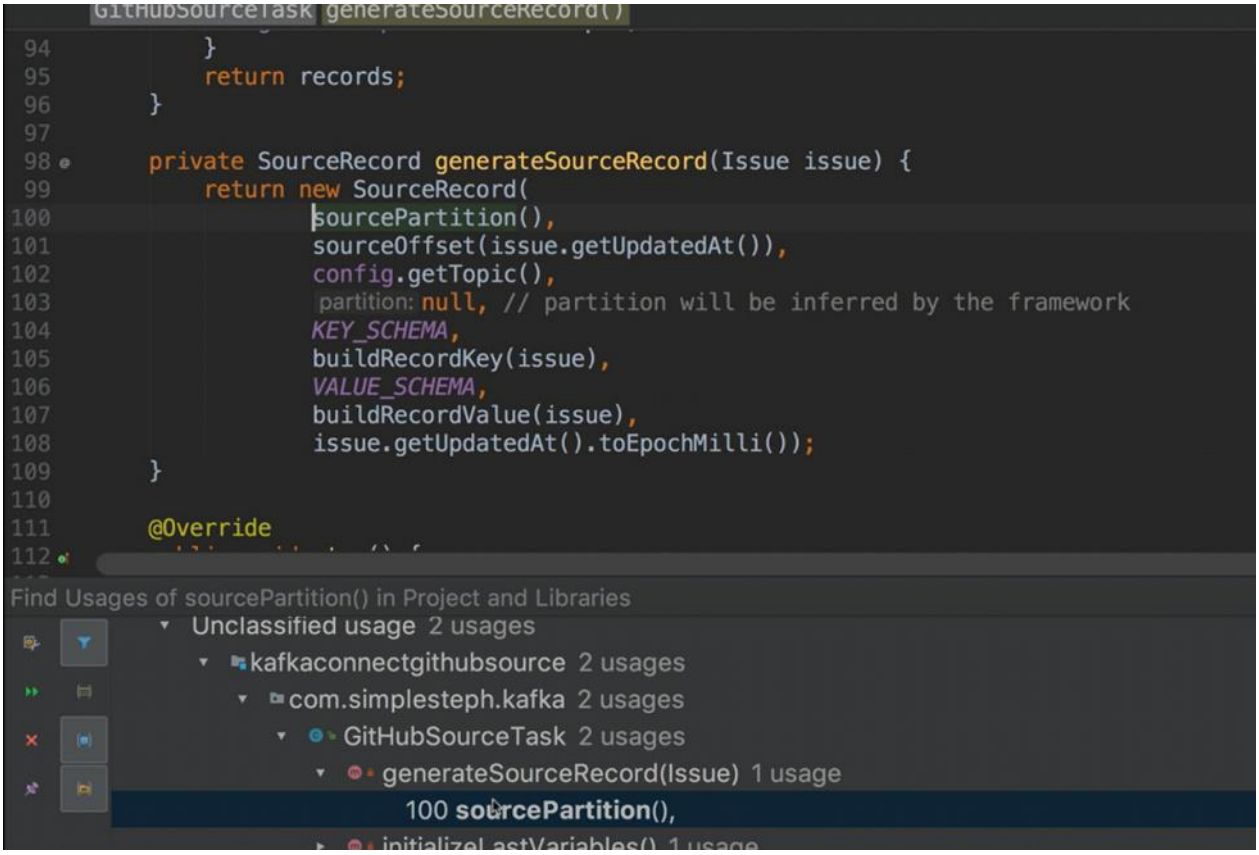
- Source Partition allows Kafka Connect to know which source you've been reading
- Source Offsets allow Kafka Connect to Track until when you've been reading for the Source Partition you chose
- **They are different than partition and offsets for Kafka.**
- Source Partition and Source Offsets are for Kafka Connect Source

```
private Map<String, String> sourcePartition() {
   Map<String, String> map = new HashMap<>();
   map.put(OWNER_FIELD, config.getOwnerConfig());
   map.put(REPOSITORY_FIELD, config.getRepoConfig());
   return map;
}
```

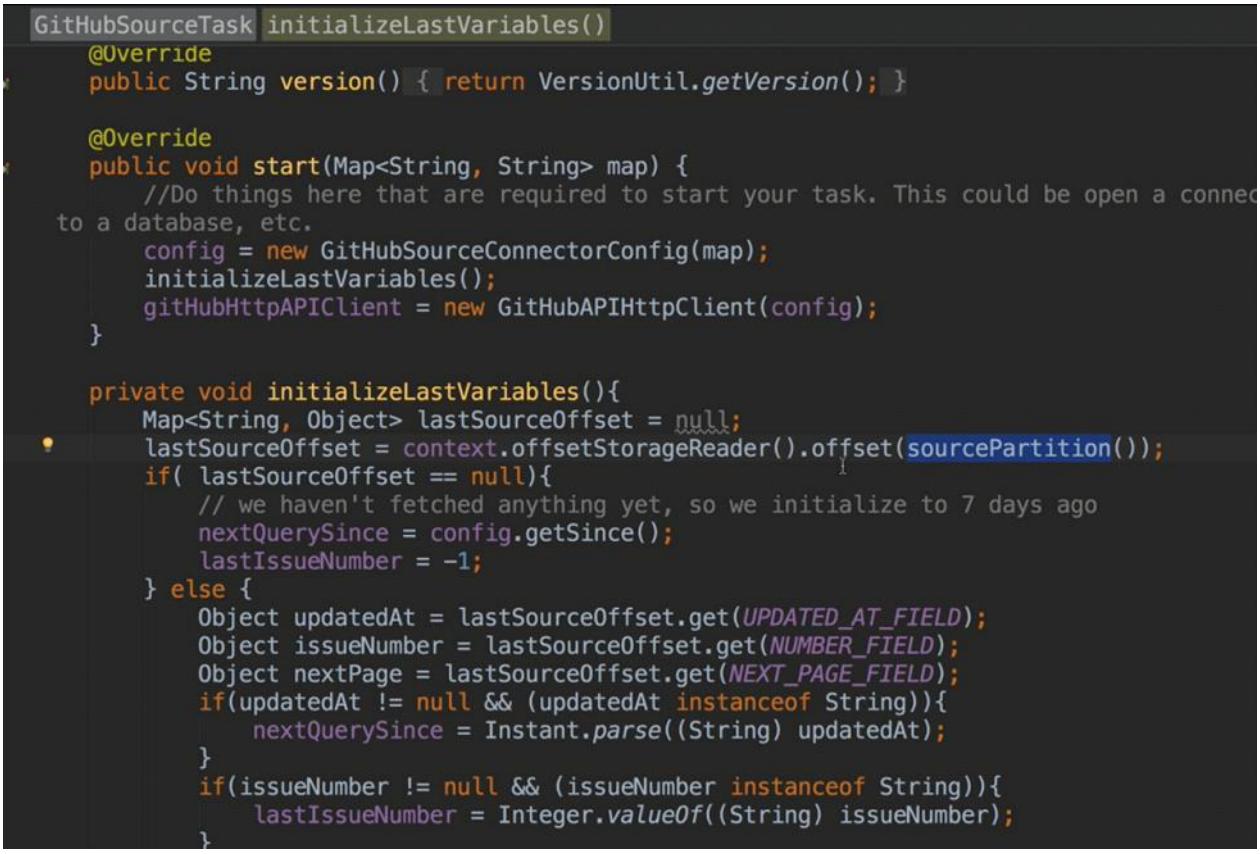


при каждом чтении из источника я буду сдвигать source offset

```
private Map<String, String> sourcePartition() {
    Map<String, String> map = new HashMap<>();
    map.put(OWNER_FIELD, config.getOwnerConfig());
    map.put(REPOSITORY_FIELD, config.getRepoConfig());
    return map;
}

private Map<String, String> sourceOffset(Instant updatedAt) {
    Map<String, String> map = new HashMap<>();
    map.put(UPDATED_AT_FIELD, DateUtils.MaxInstant(updatedAt, nextQuerySince).toString());
    map.put(NEXT_PAGE_FIELD, nextPageToVisit.toString());
    return map;
}

private Struct buildRecordKey(Issue issue){
    // Key Schema
    Struct key = new Struct(KEY_SCHEMA)
            .put(OWNER_FIELD, config.getOwnerConfig())
            .put(REPOSITORY_FIELD, config.getRepoConfig())
            .put(NUMBER_FIELD, issue.getNumber());
```



при инициализации таски она будет начинать с последнего source offset



GitHubSourceTask poll()

```java
@Override
public List<SourceRecord> poll() throws InterruptedException {
    gitHubHttpAPIClient.sleepIfNeed();

    // fetch data
    final ArrayList<SourceRecord> records = new ArrayList<>();
    JSONArray issues = gitHubHttpAPIClient.getNextIssues(nextPageToVisit, nextQuerySinc
    // we'll count how many results we get with i
    int i = 0;
    for (Object obj : issues) {
        Issue issue = Issue.fromJson((JSONObject) obj);
        SourceRecord sourceRecord = generateSourceRecord(issue);
        records.add(sourceRecord);
        i += 1;
        lastUpdatedAt = issue.getUpdatedAt();
    }
    if (i > 0) log.info(String.format("Fetched %s record(s)", i));
    if (i == 100){
        // we have reached a full batch, we need to get the next one
        nextPageToVisit += 1;
    }
    else {
        nextQuerySince = lastUpdatedAt.plusSeconds(1);
        nextPageToVisit = 1;
        gitHubHttpAPIClient.sleep();
    }
    return records;
}
```

GitHubSourceTask generateSourceRecord()

```java
        }
        else {
            nextQuerySince = lastUpdatedAt.plusSeconds(1);
            nextPageToVisit = 1;
            gitHubHttpAPIClient.sleep();
        }
        return records;
    }

    private SourceRecord generateSourceRecord(Issue issue) {
        return new SourceRecord(
                sourcePartition(),
                sourceOffset(issue.getUpdatedAt()),
                config.getTopic(),
                 partition: null, // partition will be inferred by the framework
                KEY_SCHEMA,
                buildRecordKey(issue),
                VALUE_SCHEMA,
                buildRecordValue(issue),
                issue.getUpdatedAt().toEpochMilli());
    }

    @Override
    public void stop() {
        // Do whatever is required to stop your task.
    }

    private Map<String, String> sourcePartition() {
```

… task

5 декабря 2020 г. 12:20

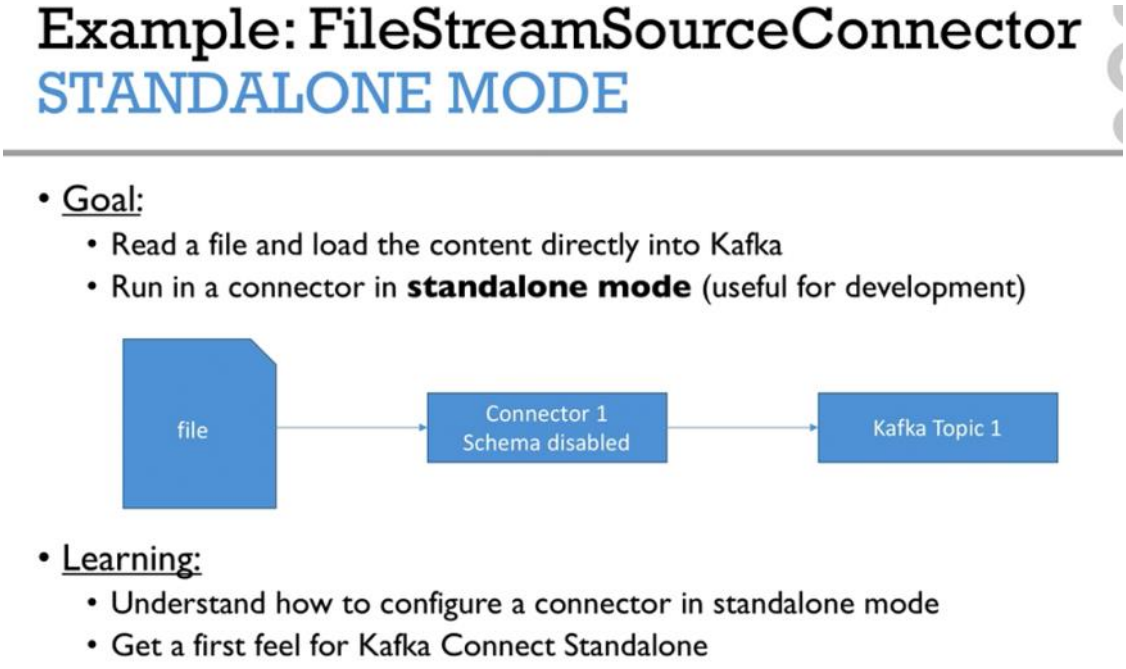
Task

- Our Task is what does the actual job
- It's supposed to initialize, then find where to resume from, and finally poll the source for records

```
public class GitHubSourceTask extends SourceTask {
    public String version()
    public void start(Map<String, String> map)
    public List<SourceRecord> poll() throws
    InterruptedException
    public void stop()
    }
```

* file source connector (standalone)

4 декабря 2020 г. 18:33



we launch the kafka connector in standalone mode:
cd /tutorial/source/demo-1

```
root@fast-data-dev / $ cd tutorial/source/demo-1/
root@fast-data-dev demo-1 $ ls
demo-file.txt  file-stream-demo-standalone.properties  worker.properties
```

create the topic we write to with 3 partitions
kafka-topics --create --topic demo-1-standalone --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181

Usage is connect-standalone worker.properties connector1.properties [connector2.properties connector3.properties]
connect-standalone worker.properties file-stream-demo-standalone.properties

write some data to the demo-file.txt !
shut down the terminal when you're done.
################

worker.properties -worker infrastructure config for standalone mode

# from more information, visit: http://docs.confluent.io/3.2.0/connect/userguide.html#common-worker-configs
bootstrap.servers=127.0.0.1:9092
key.converter=org.apache.kafka.connect.json.**Json**Converter
key.converter**.schemas.enable=false**
value.converter=org.apache.kafka.connect.json.**Json**Converter
value.converter**.schemas.enable=false**

# we always leave the internal key to JsonConverter
internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.key.converter.schemas.enable=false
internal.value.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter.schemas.enable=false

```
# Rest API
rest.port=8086
rest.host.name=127.0.0.1

# this config is only for standalone workers
offset.storage.file.filename=standalone.offsets
offset.flush.interval.ms=10000
```

file-stream-demo-standalone.properties file connector config

```
# These are standard kafka connect parameters, need for ALL connectors
name=file-stream-demo-standalone
connector.class=org.apache.kafka.connect.file.FileStreamSourceConnector
tasks.max=1  # число задач которые можно запускать параллельно

# Parameters can be found here:
https://github.com/apache/kafka/blob/trunk/connect/file/src/main/java/org/apache/kafka/connect/file/FileStreamSourceConnector.java
file=demo-file.txt   # файл откуда будем читать
topic=demo-1-standalone    # топик в который будем писать
```

при реалтайм добавлении строки в файл она будет автоматически подхватываться и передаваться в топик

* file source connector (distributed)

4 декабря 2020 г. 21:04

Example: FileStreamSourceConnector
DISTRIBUTED MODE

- Goal:
  - Read a file and load the content directly into Kafka
  - Run in **distributed mode** on our already set-up Kafka Connect Cluster

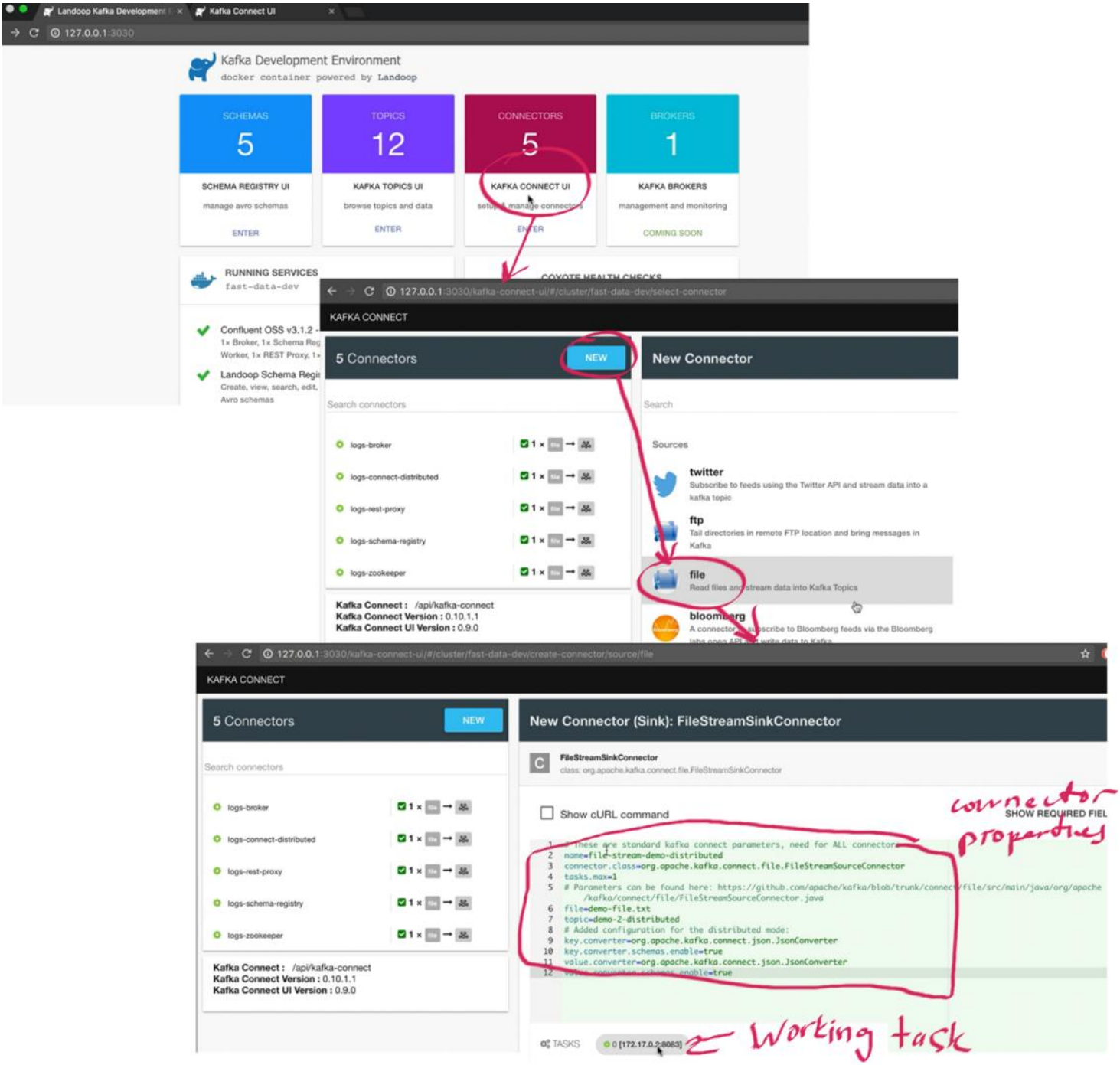
- Learning:
  - Understand how to configure a connector in distributed mode
  - Get a first feel for Kafka Connect Cluster
  - Understand the schema configuration option

```
###############
# B) FileStreamSourceConnector in distributed mode:
# create the topic we're going to write to
docker run --rm -it --net=host landoop/fast-data-dev bash
kafka-topics --create --topic demo-2-distributed --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181
# you can now close the new shell

# head over to 127.0.0.1:3030 -> Connect UI
# Create a new connector -> File Source
# Paste the configuration at source/demo-2/file-stream-demo-distributed.properties

# Now that the configuration is launched, we need to create the file demo-file.txt
docker ps
docker exec -it <containerId> bash
touch demo-file.txt
echo "hi" >> demo-file.txt
echo "hello" >> demo-file.txt
echo "from the other side" >> demo-file.txt
```

законектимся на докер и создадим файл


```
# Read the topic data
docker run --rm -it --net=host landoop/fast-data-dev bash
kafka-console-consumer --topic demo-2-distributed --from-beginning --bootstrap-server 127.0.0.1:9092
# observe we now have json as an output, even though the input was text!
################
```

как создать distributed connector через UI




```
# These are standard kafka connect parameters, need for ALL connectors
name=file-stream-demo-distributed
connector.class=org.apache.kafka.connect.file.FileStreamSourceConnector
tasks.max=1

# Parameters can be found here:
https://github.com/apache/kafka/blob/trunk/connect/file/src/main/java/org/apache/kafka/connect/file/FileStreamSourceConnector.java
file=demo-file.txt  # файл откуда будем читать
topic=demo-2-distributed # топик в который будем писать

# Added configuration for the distributed mode:   так как здесь нету worker.properties  файла  то все настройки worker прям здесь
key.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=true
```

используем landoop UI

```
version: '2'

services:
  # this is our kafka cluster.
```

```
kafka-cluster:
  image: landoop/fast-data-dev:cp3.3.0
  environment:
    ADV_HOST: 127.0.0.1         # Change to 192.168.99.100 if using Docker Toolbox
    RUNTESTS: 0                 # Disable Running tests so the cluster starts faster
  ports:
    - 2181:2181                 # Zookeeper
    - 3030:3030                 # Landoop UI
    - 8081-8083:8081-8083       # REST Proxy, Schema Registry, Kafka Connect ports
    - 9581-9585:9581-9585       # JMX Ports
    - 9092:9092                 # Kafka Broker

# we will use elasticsearch as one of our sinks.
# This configuration allows you to start elasticsearch
elasticsearch:
  image: itzg/elasticsearch:2.4.3
  environment:
    PLUGINS: appbaseio/dejavu
    OPTS: -Dindex.number_of_shards=1 -Dindex.number_of_replicas=0
  ports:
    - "9200:9200"

# we will use postgres as one of our sinks.
# This configuration allows you to start postgres
postgres:
  image: postgres:9.5-alpine
  environment:
    POSTGRES_USER: postgres     # define credentials
    POSTGRES_PASSWORD: postgres # define credentials
    POSTGRES_DB: postgres       # define database
  ports:
    - 5432:5432                 # Postgres port
```

* local file connector (only for dev)

14 декабря 2020 г. 13:33

коннектор отслеживает только новопоявившиеся строки в конце файла и не может определить изменения уже закачаенных строк

FileStream Connector

можно накрутить этот коннектор на линке, куда будут поступать внешние данные

- FileStream Connector acts on a local file
  - Local file Source Connector: tails local file and sends each line as a Kafka message
  - Local file Sink Connector: Appends Kafka messages to a local file

The FileSource Connector reads data from a file and sends it to Kafka. This connector will read only one file and send the data within that file to Kafka. It will then watch the file for appended updates only. Any modification of file lines already sent to Kafka will not be reprocessed.

The FileSink Connector reads data from Kafka and outputs it to a local file. Multiple topics may be specified as with any other sink connector. As messages are added to the topics specified in the configuration, they are produced to a local file as specified in the configuration.

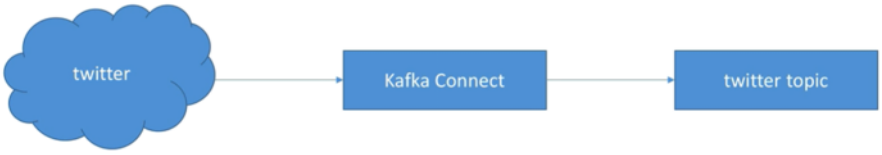
The FileStream Connector examples on docs.confluent.io are intended to show how a simple connector runs for those first getting started with Kafka Connect as either a user or developer. It is not recommended for production use. Instead, we encourage users to use them to learn in a local environment. The examples include both a file source and a file sink to demonstrate an end-to-end data flow implemented through Kafka Connect. The FileStream Connector examples are also detailed in the developer guide as a demonstration of how a custom connector can be implemented.

* twitter source connector (distributed)

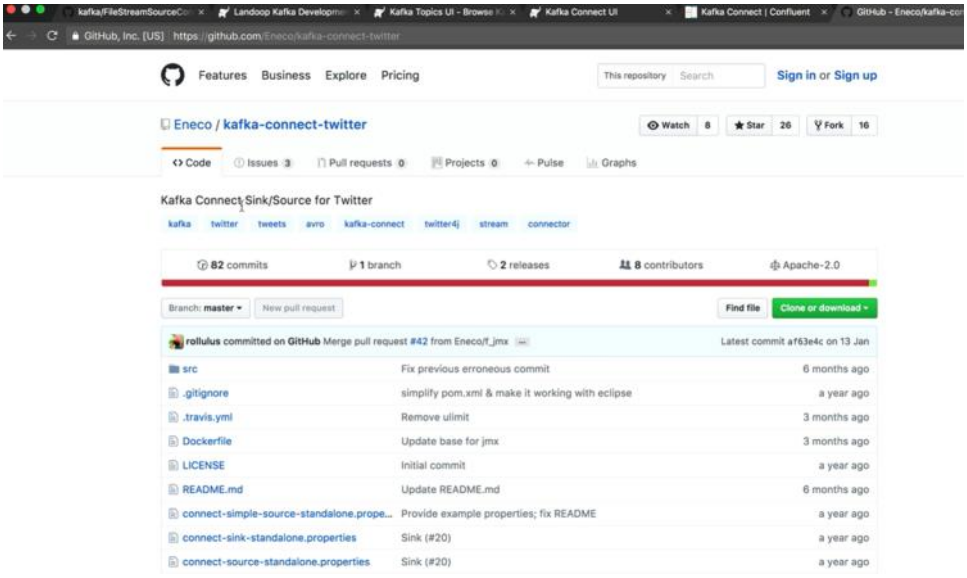
4 декабря 2020 г. 18:33

TwitterSourceConnector

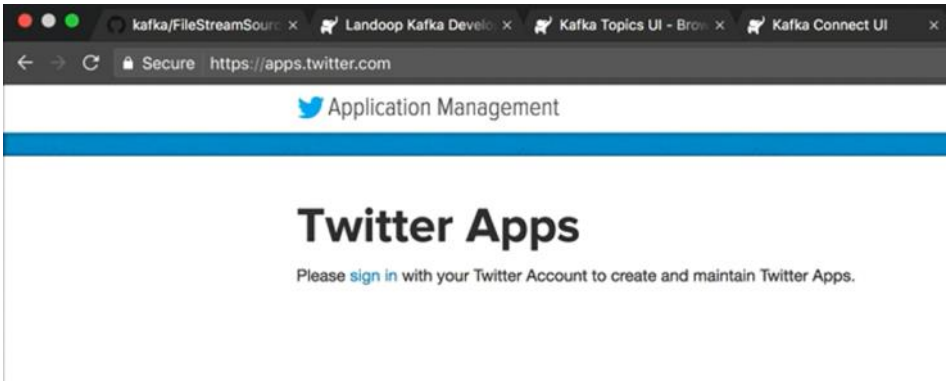
- Goal:
  - Gather data from Twitter in Kafka Connect Distributed mode

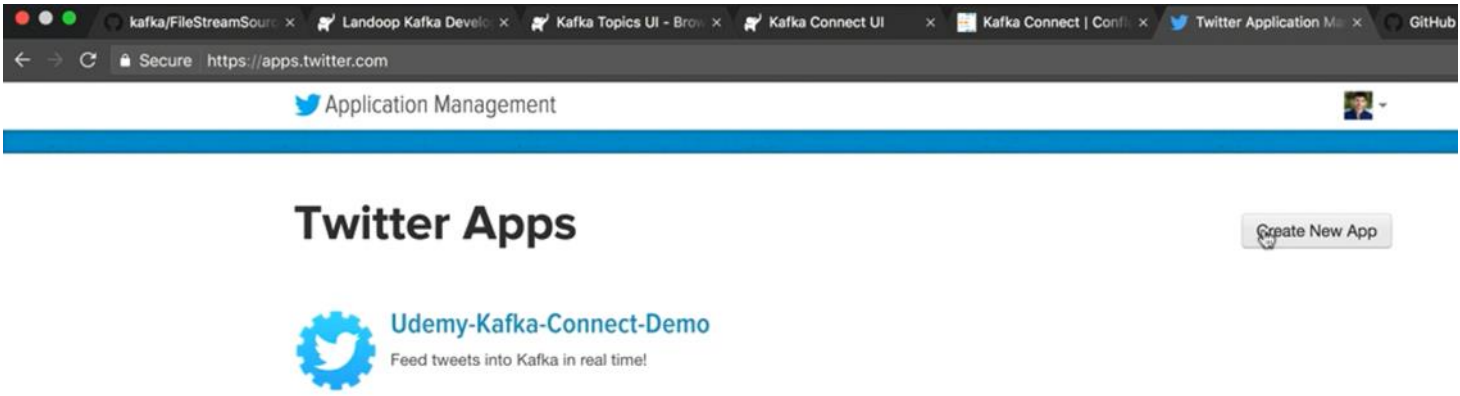


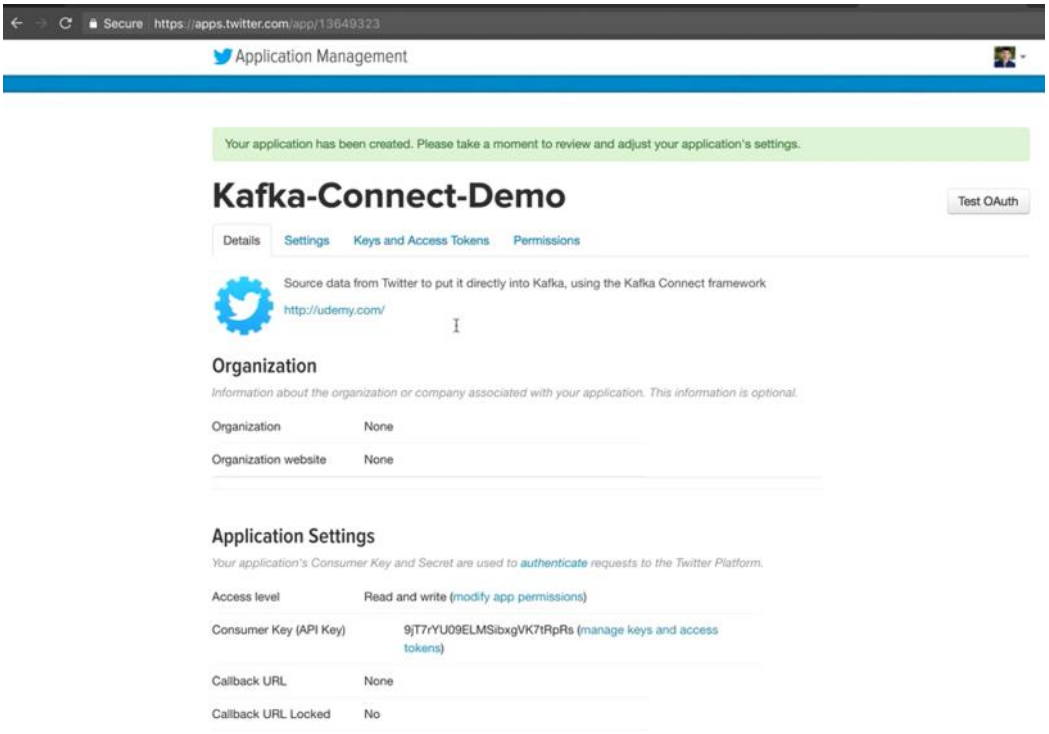
- Learning:
  - Gather real data using https://github.com/Eneco/kafka-connect-twitter



для того чтобы использовать API нужен twitter DEVELOPER account

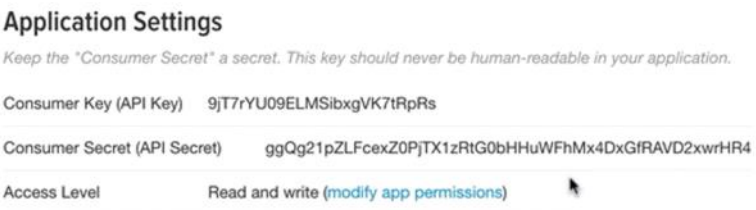


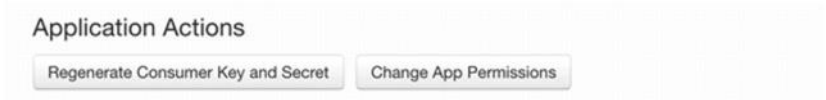

Secure https://apps.twitter.com
Application Management
Twitter Apps
Create New App
Udemy-Kafka-Connect-Demo
Feed tweets into Kafka in real time!

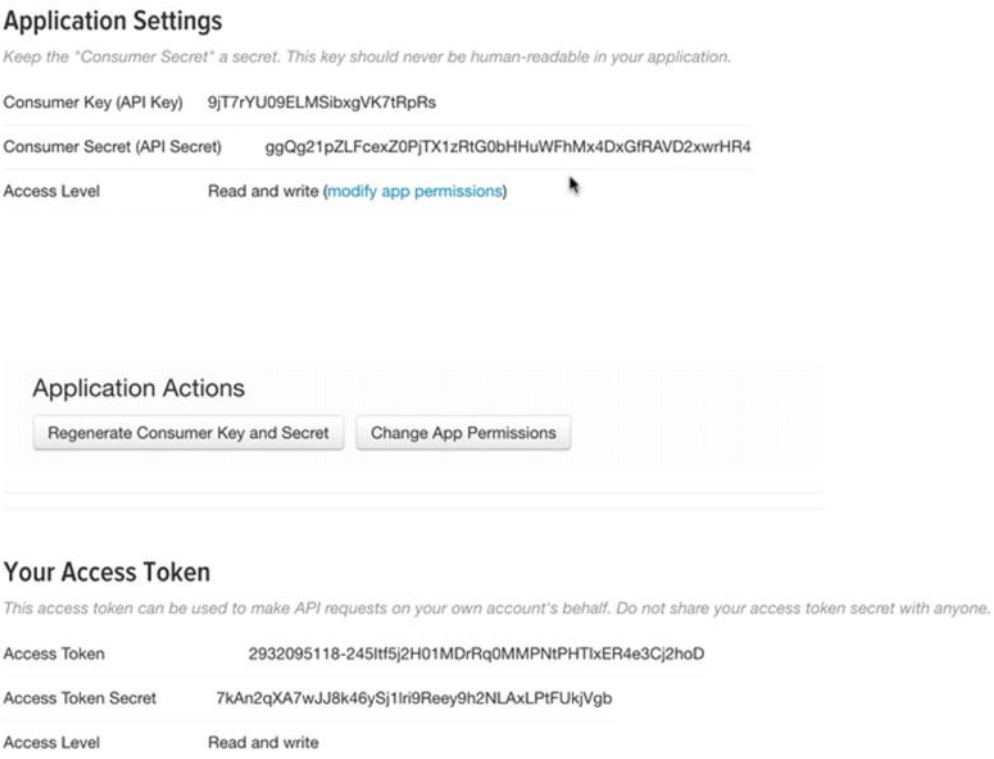

Secure https://apps.twitter.com/app/13649323
Application Management
Your application has been created. Please take a moment to review and adjust your application's settings.
Kafka-Connect-Demo
Test OAuth
Details
Settings
Keys and Access Tokens
Permissions
Source data from Twitter to put it directly into Kafka, using the Kafka Connect framework
http://udemy.com/
Organization
Information about the organization or company associated with your application. This information is optional.
Organization None
Organization website None
Application Settings
Your application's Consumer Key and Secret are used to authenticate requests to the Twitter Platform.
Access level Read and write (modify app permissions)
Consumer Key (API Key) 9jT7rYU09ELMSibxgVK7tRpRs (manage keys and access tokens)
Callback URL None
Callback URL Locked No


Keys and Access Tokens

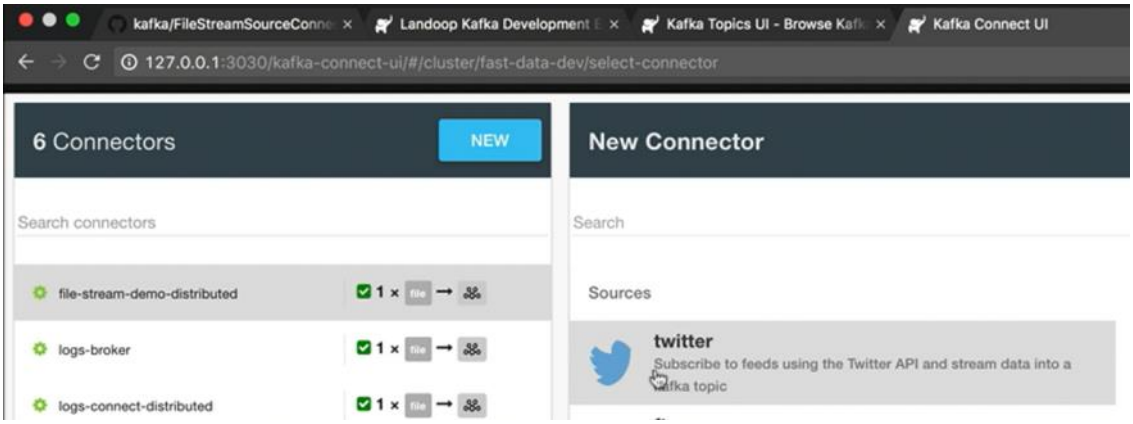

Token Actions
Create my access token


Application Settings
Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.
Consumer Key (API Key) 9jT7rYU09ELMSibxgVK7tRpRs
Consumer Secret (API Secret) ggQg21pZLFcexZ0PjTX1zRtG0bHHuWFhMx4DxGfRAVD2xwrHR4
Access Level Read and write (modify app permissions)


Application Actions
Regenerate Consumer Key and Secret
Change App Permissions



как создать distributed connector через UI



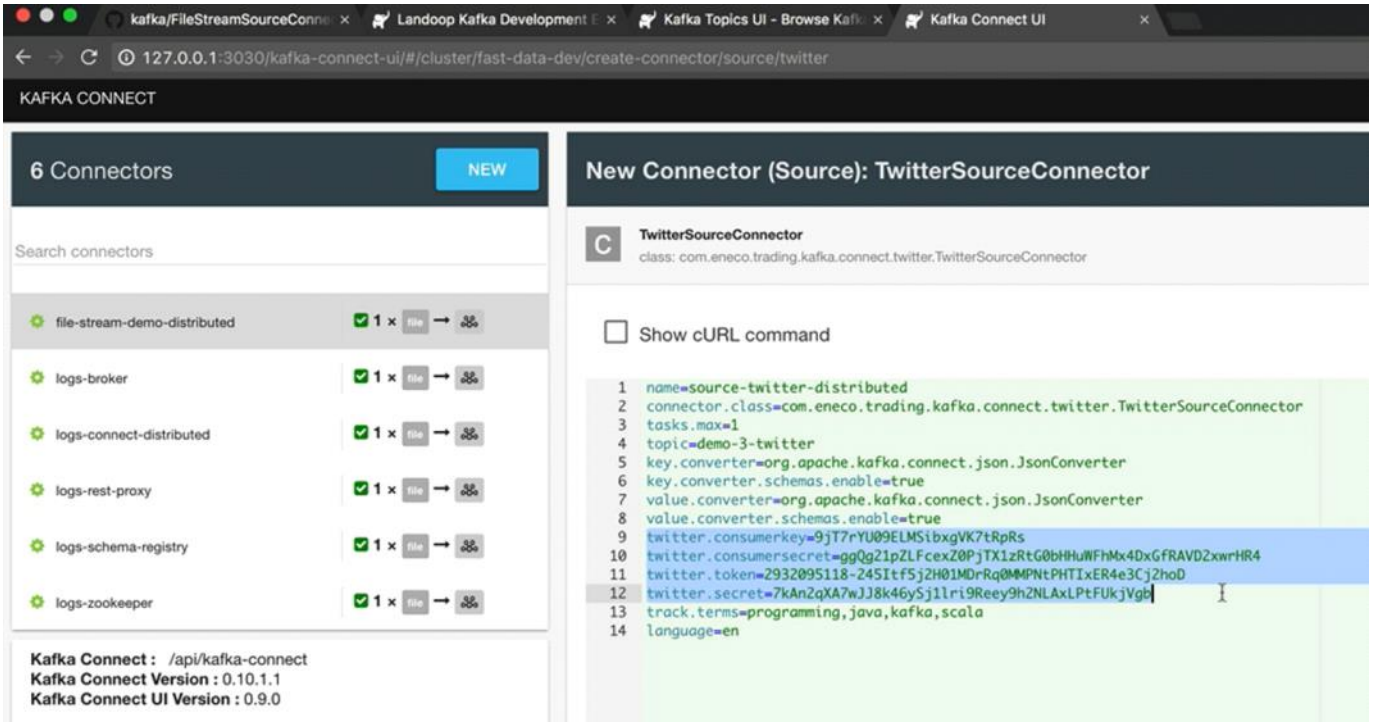
################
C) TwitterSourceConnector in distributed mode:
create the topic we're going to write to
docker run --rm -it --net=host landoop/fast-data-dev bash
kafka-topics --create --topic demo-3-twitter --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181

Start a console consumer on that topic
kafka-console-consumer --topic demo-3-twitter --bootstrap-server 127.0.0.1:9092

Follow the instructions at: https://github.com/Eneco/kafka-connect-twitter#creating-a-twitter-application
To obtain the required keys, visit https://apps.twitter.com/ and Create a New App. Fill in an application name & description & web site and accept the developer aggreement. Click on Create my access token and populate a file twitter-source.properties with consumer key & secret and the access token & token secret using the example file to begin with.

Setup instructions for the connector are at: https://github.com/Eneco/kafka-connect-twitter#setup

fill in the required information at demo-3/source-twitter-distributed.properties
Launch the connector and start seeing the data flowing in!



# Basic configuration for our connector
name=source-twitter-distributed
connector.class=com.eneco.trading.kafka.connect.twitter.TwitterSourceConnector
tasks.max=1
topic=demo-3-twitter
key.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=true

# Twitter connector specific configuration настройки с twitter API
twitter.consumerkey=9jT7rYU09ELMSibxgVK7tRpRs
twitter.consumersecret=ggQg21pZLFcexZ0PjTX1zRtG0bHHuWFhMx4DxGfRAVD2xwrHR4
twitter.token=2932095118-245Itf5j2H01MDrRq0MMPNtPHTIxER4e3Cj2hoD
twitter.secret=7kAn2qXA7wJJ8k46ySj1lri9Reey9h2NLAxLPtFUkjVgb

track.terms=programming,java,kafka,scala
language=en

* Elastic sink connector (distributed)

4 декабря 2020 г. 18:33

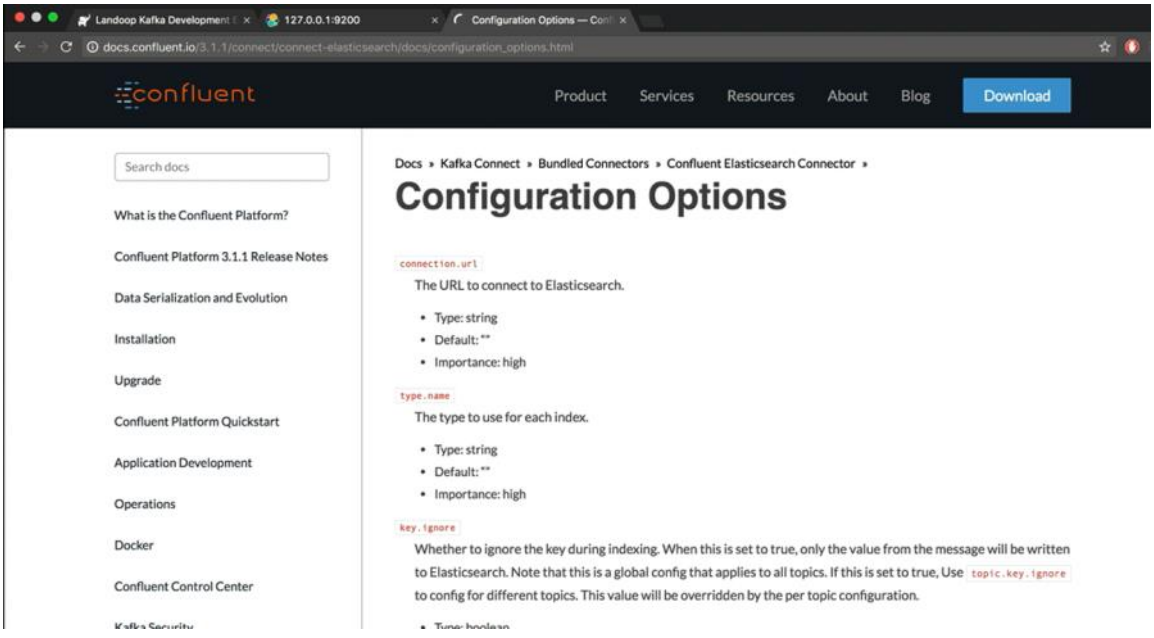
ElasticSearch

- We're going to start an ElasticSearch instance to Sink the data to
- We will use Docker to start our ElasticSearch instance
- This will serve as our Sink for our first Sink Connector

- ElasticSearch is an easy way to store json data and search across it
- Used by many companies around to world to power search engines and advanced analytics capabilities
- No knowledge of ElasticSearch is required for this course

- Goal:
  - Start an ElasticSearch instance using Docker
  - Sink a topic with multiple partitions to ElasticSearch
  - Run in distributed mode with multiple tasks

- Learning:
  - Learn about the tasks.max parameter
  - Understand how Sink Connectors work

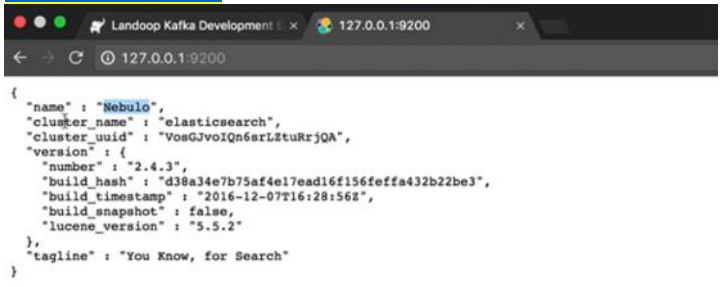


используем landoop UI для управления кафкой

```
version: '2'

services:
 # this is our kafka cluster.
 kafka-cluster:
  image: landoop/fast-data-dev:cp3.3.0
  environment:
   ADV_HOST: 127.0.0.1         # Change to 192.168.99.100 if using Docker Toolbox
   RUNTESTS: 0                 # Disable Running tests so the cluster starts faster
  ports:
   - 2181:2181                 # Zookeeper
   - 3030:3030                 # Landoop UI
   - 8081-8083:8081-8083       # REST Proxy, Schema Registry, Kafka Connect ports
   - 9581-9585:9581-9585       # JMX Ports
   - 9092:9092                 # Kafka Broker

 # we will use elasticsearch as one of our sinks.
 # This configuration allows you to start elasticsearch
 elasticsearch:
  image: itzg/elasticsearch:2.4.3
  environment:
   PLUGINS: appbaseio/dejavu
   OPTS: -Dindex.number_of_shards=1 -Dindex.number_of_replicas=0
  ports:
   - "9200:9200"

 # we will use postgres as one of our sinks.
 # This configuration allows you to start postgres
 postgres:
  image: postgres:9.5-alpine
  environment:
   POSTGRES_USER: postgres     # define credentials
   POSTGRES_PASSWORD: postgres # define credentials
   POSTGRES_DB: postgres       # define database
  ports:
   - 5432:5432                 # Postgres port
```

запустим докер

```
#!/bin/bash

# Make sure you change the ADV_HOST variable in docker-compose.yml
# if you are using docker Toolbox

# 1) Source connectors
# Start our kafka cluster
docker-compose up kafka-cluster elasticsearch postgres

# Wait 2 minutes for the kafka cluster to be started

################
# A) ElasticSearch Sink
# Info here: http://docs.confluent.io/3.2.0/connect/connect-elasticsearch/docs/elasticsearch_connector.html
# We make sure elasticsearch is working. Replace 127.0.0.1 by 192.168.99.100 if needed
http://127.0.0.1:9200/
```



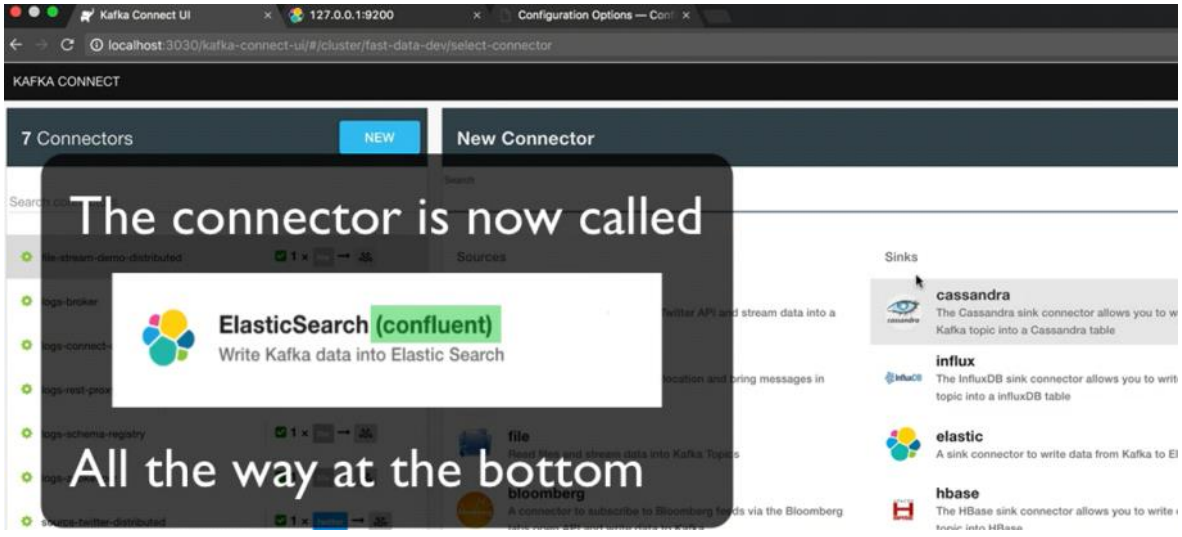
Go to the connect UI and apply the configuration at :
sink/demo-elastic/sink-elastic-twitter-distributed.properties

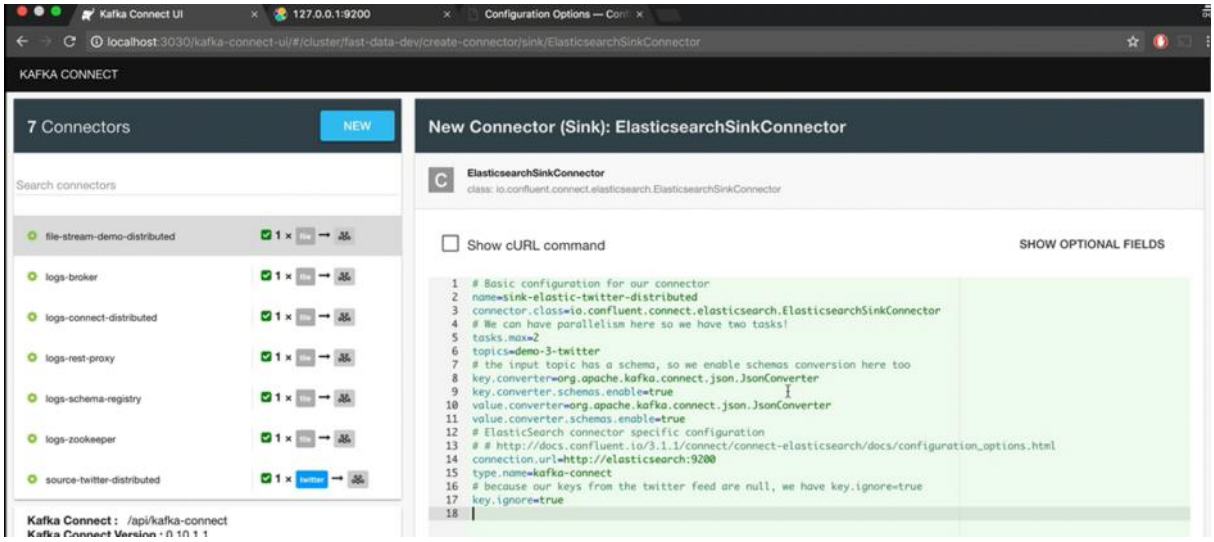
Visualize the data at:
http://127.0.0.1:9200/_plugin/dejavu

http://docs.confluent.io/3.1.1/connect/connect-elasticsearch/docs/configuration_options.html
Counting the number of tweets:
http://127.0.0.1:9200/demo-3-twitter/_count

You can download the data from the UI to see what it looks like
We can query elasticsearch for users who have a lot of friends, see query-high-friends.json
################

настроим коннектор через UI

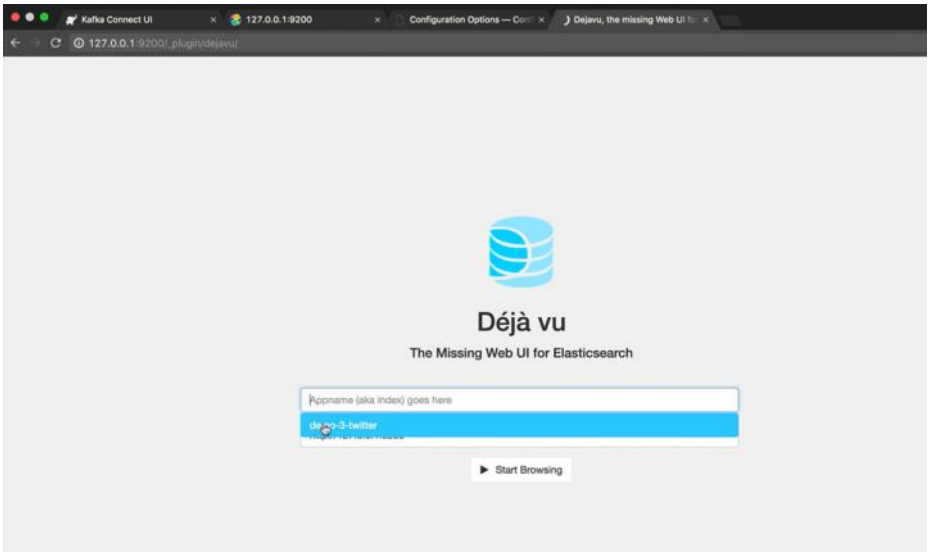


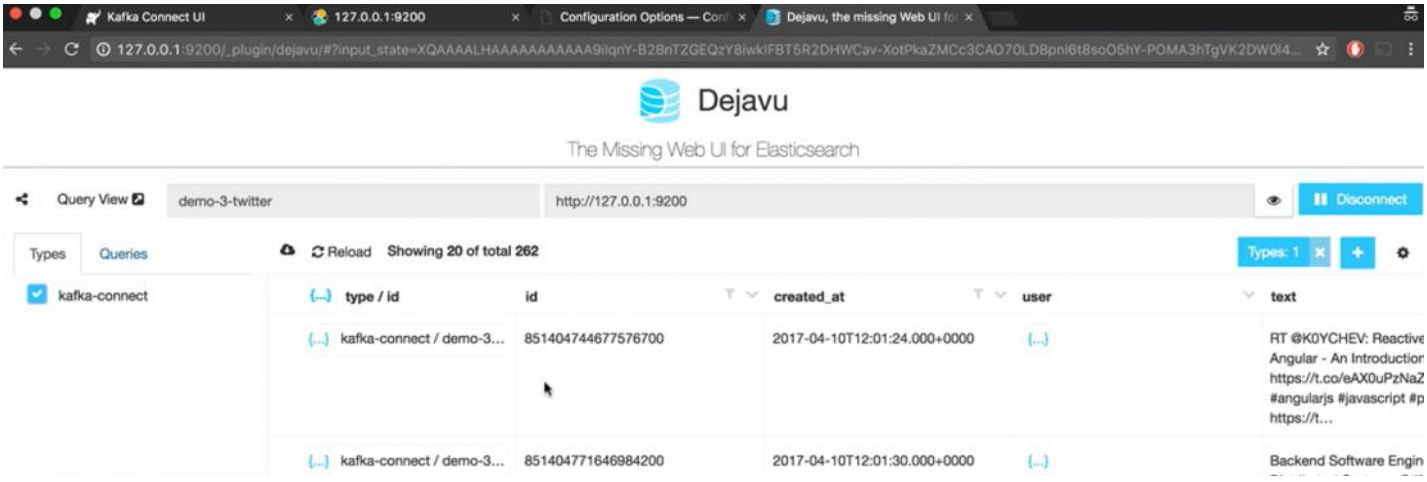


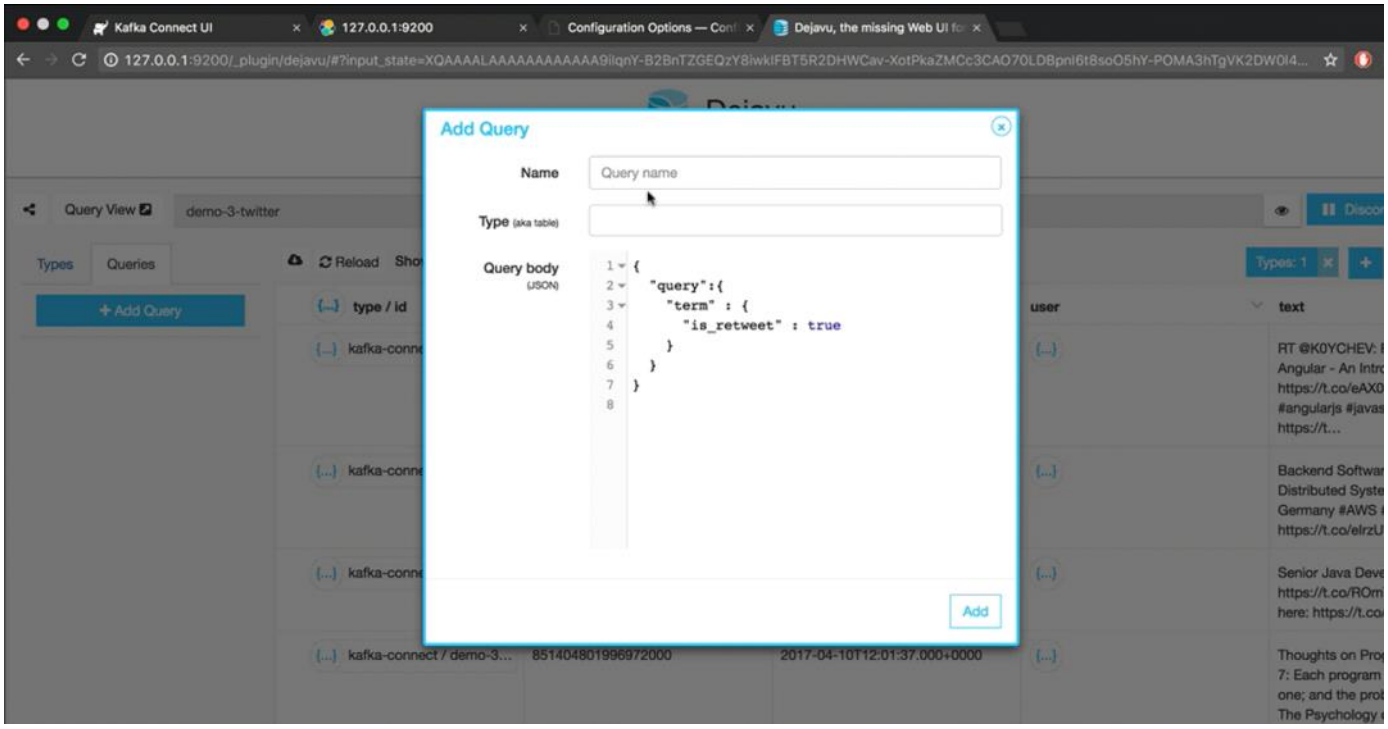
# Basic configuration for our connector
name=sink-elastic-twitter-distributed

```
connector.class=io.confluent.connect.elasticsearch.ElasticsearchSinkConnector

# We can have parallelism here so we have two tasks!
tasks.max=2
topics=demo-3-twitter

# the input topic has a schema, so we enable schemas conversion here too
key.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=true

# ElasticSearch connector specific configuration
# # http://docs.confluent.io/3.3.0/connect-elasticsearch/docs/configuration_options.html
connection.url=http://elasticsearch:9200
type.name=kafka-connect
# because our keys from the twitter feed are null, we have key.ignore=true
key.ignore=true
# some (dummy) settings we need to add to ensure the configuration is accepted (see https://www.udemy.com/apache-kafka-series-kafka-connect-hands-on-learning/learn/v4/questions/2250394). The bug is tracked at https://github.com/Landoop/fast-data-dev/issues/42
topic.index.map="demo-3-twitter:index1"
topic.key.ignore=true
topic.schema.ignore=true
```

посмотрим как работает elasticsearch






Kafka Connect UI
127.0.0.1:9200
Configuration Options — Conf
Dejavu, the missing Web UI fo
127.0.0.1:9200/_plugin/dejavu/#?input_state=XQAAAALAAAAAAAAAAAA9ilqnY-B2BnTZGEQzY8iwkIFBT5R2DHWCav-XotPkaZMCc3CAO70LDBpni6t8soO5hY-POMA3hTgVK2DW0I4...
Add Query
Name
Query name
Type (aka table)
Query body
(JSON)
{
"query":{
"term" : {
"is_retweet" : true
}
}
}
Add
Query View
demo-3-twitter
Types
Queries
Reload
+ Add Query
type / id
user
text
kafka-connect / demo-3...
851404801996972000
2017-04-10T12:01:37.000+0000

* jdbc source conncector

4 декабря 2020 г. 22:42

на каждую таблицу БД используется свой топик

- ?? новые таблицы отслеживаются автоматически -> и автоматом создается топик??

JDBC Source Connector: Overview

- JDBC Source periodically polls a relational database for new or recently modified rows
  - Creates a record for each row, and Produces that record as a Kafka message
- Records from each table are Produced to their own Kafka Topic
- New and deleted tables are handled automatically

The JDBC source connector allows you to import data from any relational database with a JDBC driver into Kafka topics. By using JDBC, this connector can support a wide variety of databases without requiring custom code for each one.

Data is loaded by periodically executing a SQL query and creating an output record for each row in the result set. By default, all tables in a database are copied, each to its own output topic. The database is monitored for new or deleted tables and adapts automatically. When copying data from a table, the connector can load only new or modified rows by specifying which columns should be used to detect new or modified data.

Here is a good place to discuss CDC versus query based source connectors (Pros and Cons of each)!

query based CDC connector - в отличие от log-based CDC коннекторов, этот коннектор не может определять deleted rows (а также проблемы с инсертами и апдейтами)

- log-based означает что коннектору надо влазить в кишки/log самой БД
- для определения апдейта только таймстампа недостаточно, тк могут быть записи с одинаковым таймстампом (те должна быть колонка с автоинкрементом и если такой нет то проблема, или должна быть возможность изменить схему таблицы источника)
-

Detecting New and Updated Rows

Incremental query mode	Description
Incrementing column определяет только insert не может определить update	Check a single column where newer rows have a larger, auto-incremented ID. Does not support updated rows
Timestamp column	Checks a single 'last modified' column. Can't guarantee reading all updates
Timestamp and incrementing column определяет апдейты	Combination of the two methods above. Guarantees that all updates are read
Custom query	Used in conjunction with the options above for custom filtering

`bulk` mode for one-time load, not incremental, unfiltered

The Connector can detect new and updated rows in several ways as described on the slide.

For the reasons stated on the slides, many environments will use both the timestamp and the incrementing column to capture all updates.

Note: Because timestamps are not necessarily unique, the timestamp column mode cannot guarantee all updated data will be delivered. If two rows share the same timestamp and are returned by an incremental query, but only one has been processed before a crash, the second update will be missed when the system recovers.

The custom query option can only be used in conjunction with one of the other incremental modes as long as the necessary WHERE clause can be appended to the query. In some cases, the custom query may handle all filtering itself.

в bulk-mode мы каждый раз будем запрашивать содержимое таблицы целиком

- апдейты 100% определяются
- годится только для маленьких таблиц
- для миграции: годится для первоначальной-одноразовой загрузки содержимого БД

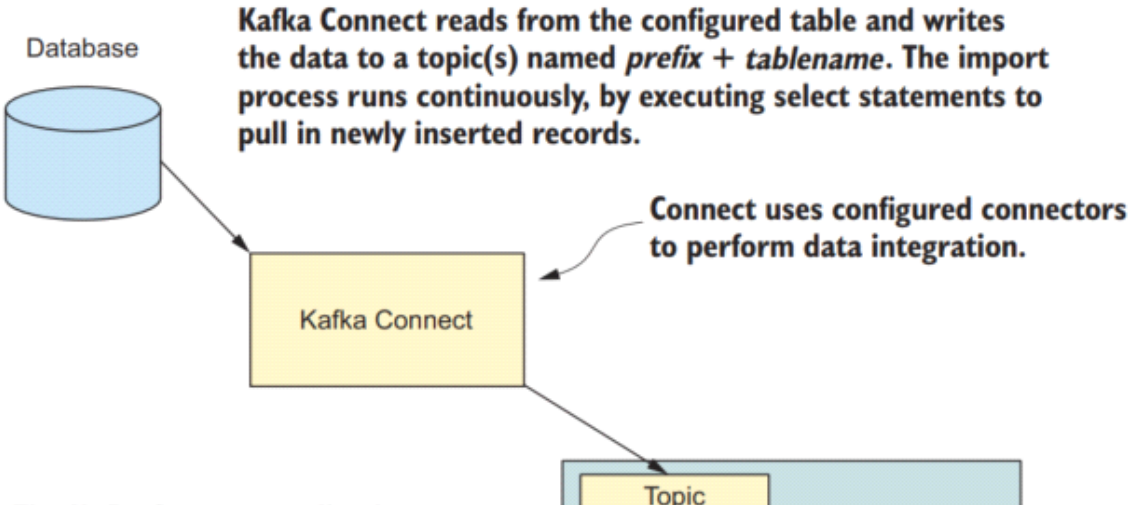
как настроить

JDBC Source Connector: Configuration

Parameter	Description
`connection.url`	The JDBC connection URL for the database
`topic.prefix`	The prefix to prepend to table names to generate the Kafka Topic name
`mode`	The mode for detecting table changes. Options are `bulk`, `incrementing`, `timestamp`, `timestamp+incrementing`
`query`	The custom query to run, if specified
`poll.interval.ms`	The frequency in milliseconds to poll for new data in each table
`table.blacklist`	A list of tables to ignore and not import. If specified, `tables.whitelist` cannot be specified
`table.whitelist`	A list of tables to import. If specified, `tables.blacklist` cannot be specified

See `https://docs.confluent.io/current/connect/connect-jdbc/docs/index.html` for a complete list

Setting both `table.whitelist` and `table.blacklist` does not fail any upfront configuration validation checks but will fail when starting the connector at runtime.



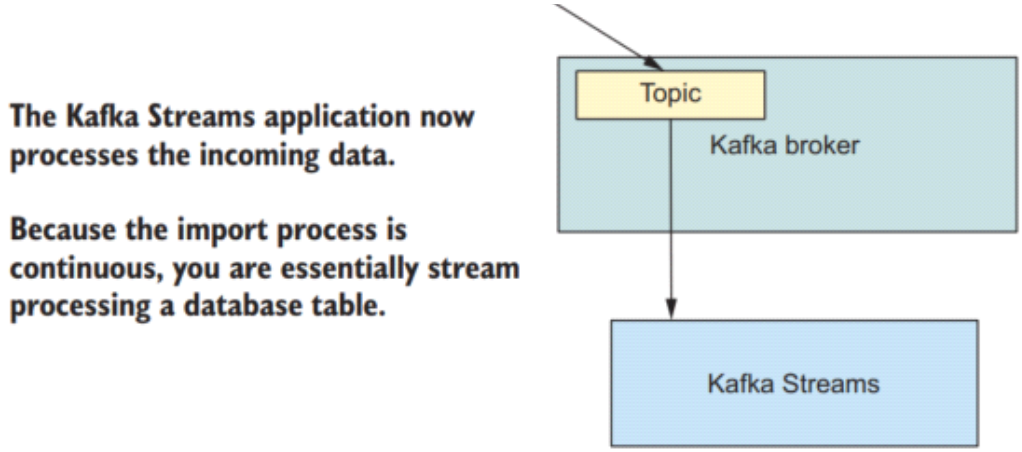


Figure 9.1 Kafka Connect integrating a database table and Kafka Streams

- `bootstrap.servers` — разделенный запятыми список используемых Connect брокеров Kafka;
- `key.converter` — класс преобразователя, отвечающего за сериализацию ключей из формата Connect в формат, в котором данные записываются в Kafka;
- `value.converter` — класс преобразователя, отвечающего за сериализацию значений из формата Connect в формат, в котором данные записываются в Kafka. В нашем примере будет применяться встроенный класс `org.apache.kafka.connect.json.JsonConverter`;
- `value.converter.schemas.enable` — может быть равен `true` или `false`. Указывает, должен ли Connect включать схему для значения. В этом примере мы установим его равным `false`. В следующем разделе я объясню почему;
- `plugin.path` — содержит местоположение коннектора для Connect и его зависимостей. Можно указать один каталог верхнего уровня, содержащий один или несколько файлов JAR. Можно также указать несколько путей в виде разделенного запятыми списка;
- `offset.storage.file.filename` — файл, содержащий сохраненные потребителем Connect смещения.

Вам придется также указать несколько параметров конфигурации для JDBC-коннектора. Вот их список:

- `name` — название коннектора;
- `connector.class` — класс коннектора;
- `tasks.max` — максимальное число задач этого коннектора;
- `connection.url` — URL подключения к базе данных;
- `mode` — метод отслеживания изменений JDBC-коннектором для источника;
- `incrementing.column.name` — название столбца, по которому отслеживаются изменения;
- `topic.prefix` — Kafka Connect записывает все таблицы в топики с названиями `топик.префикс + название_таблицы`.

* log-based CDC source connector (debezium)

14 декабря 2020 г. 13:12

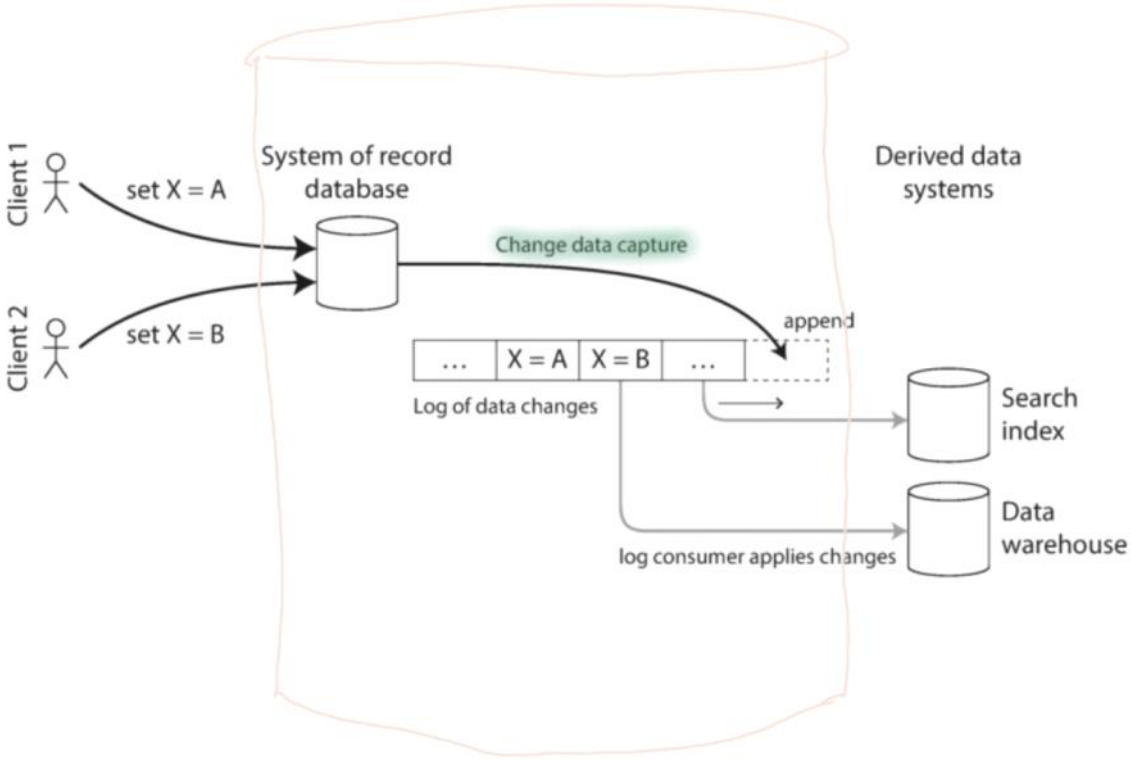
log based CDC connector может даже лезть в журналы логов БД (те тесно интегрирован с БД)

- log-based означает что коннектору надо влазить в кишки/log самой БД

  - так как у каждой БД свой подход к transaction logs and change logs
  - Debezium подходит для нескольких механизмов БД

например, query CDC connector - в отличие от log-based CDC коннекторов, этот коннектор не может определять deleted rows (а также проблемы с инсертами и апдейтами)

- CDC использует собственный «событийный» интерфейс,
- (а) очень эффективен, поскольку коннектор отслеживает файл или запускается непосредственно при возникновении изменений, а не отправляет запросы через основной API базы данных, и
- (б) очень точен, поскольку выдача запросов через основной API базы данных часто создает возможность пропуска операций, если несколько операций поступают для одной и той же строки в течение периода опроса.



Как и брокеры сообщений, перехват данных обычно является асинхронным:

- система записи данных в базу не ждет, пока изменения будут применены у потребителей, прежде чем завершить операцию. Эксплуатационные преимущества такой архитектуры заключаются в том, что появление медленного потребителя не очень сильно влияет на систему записи, но имеет такой недостаток: ей свойственны все проблемы, связанные с задержкой репликации

- Kafka Connect [41] — это попытка объединить Kafka с инструментами сбора информации об изменениях данных для широкого спектра баз

  - те позволяет создать стрим изменений из БД

Представление баз данных в виде потоков открывает широкие возможности для интеграции систем

- Мы также увидели, что полезно осуществлять запись в базу данных в виде потока: можно фиксировать историю всех изменений, сделанных в базе, — либо неявно, путем захвата данных об изменениях, либо явно, через источники событий. Уплотнение журнала позволяет потоку сохранять полную копию содержимого БД.
- Можно постоянно поддерживать актуальность производных информационных систем, таких как индексы поиска, кэши и аналитические системы, потребляя журнал изменений и применяя их к производной системе. Можно даже создавать новые представления для уже существующих данных, потребляя журнал изменений с самого начала вплоть до настоящего времени

кто использует

- Эта идея, будучи масштабированной, используется в
- LinkedIn Databus [25],
- Facebook Wormhole [26] и
- Yahoo! Sherpa [27].
- В Bottled Water реализован CDC для PostgreSQL, где специальный API декодирует журнал с упреждающей записью [28].
- В Maxwell и Debezium реализовано нечто похожее для MySQL, с синтаксическим анализом двоичного журнала [29–31].
- Mongoriver читает операционный журнал MongoDB [32, 33],
- а GoldenGate предоставляет аналогичные возможности для Oracle [34, 35]

* jdbc sink conncector (distributed)

4 декабря 2020 г. 18:35

PostgreSQL

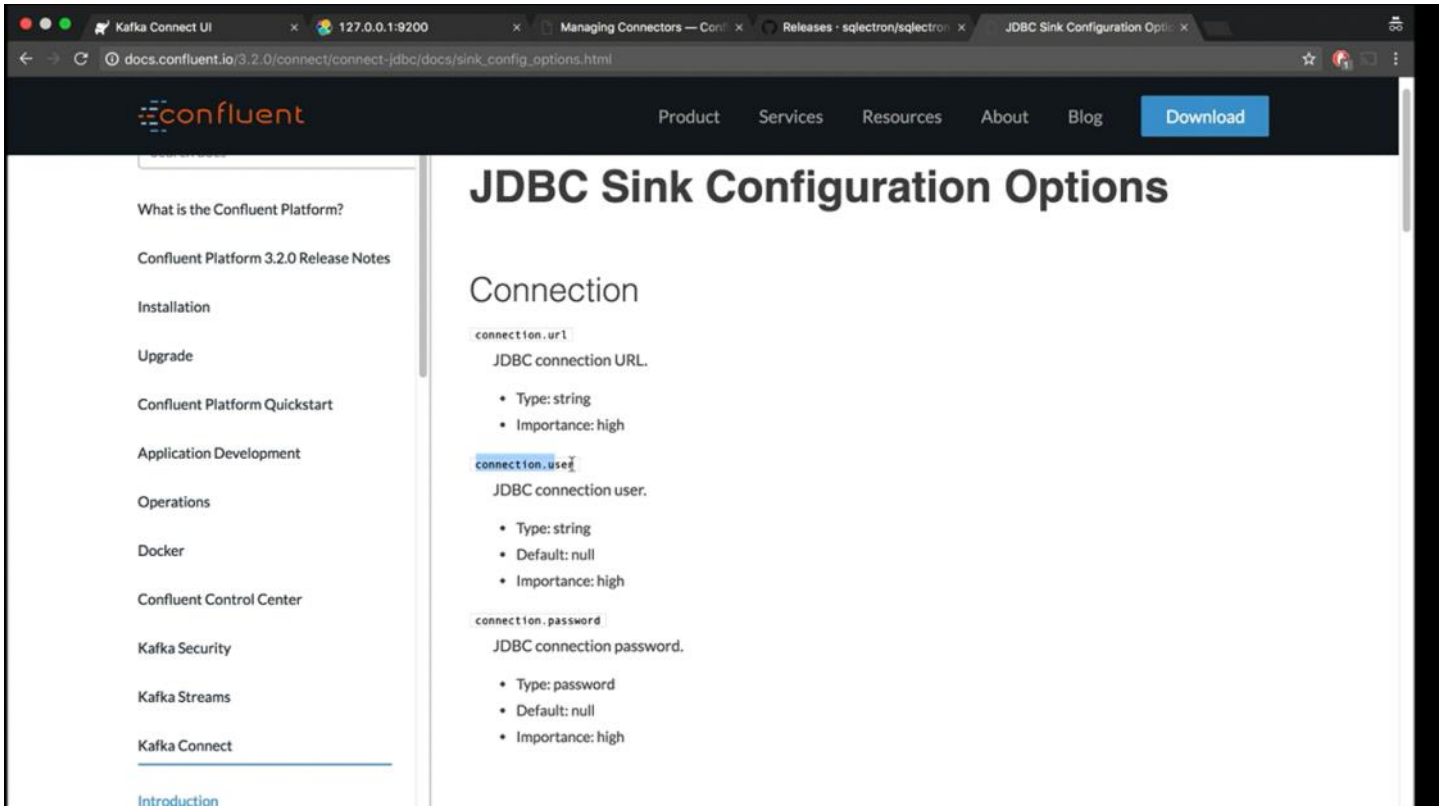
- We're going to start an PostgreSQL instance to Sink the data to
- We will use Docker to start our PostgreSQL instance
- This will serve as our Sink for our second Sink Connector

- PostgreSQL is a very popular relational database
- Used by many companies around to world to back their website or run BI reports onto
- No knowledge of PostgreSQL is required for this course

Example: JDBCSinkConnector DISTRIBUTED MODE

- Goal:
  - Start an PostgreSQL instance using Docker
  - Run in distributed mode with multiple tasks

- Learning:
  - Learn about the JDBC Sink Connector



################

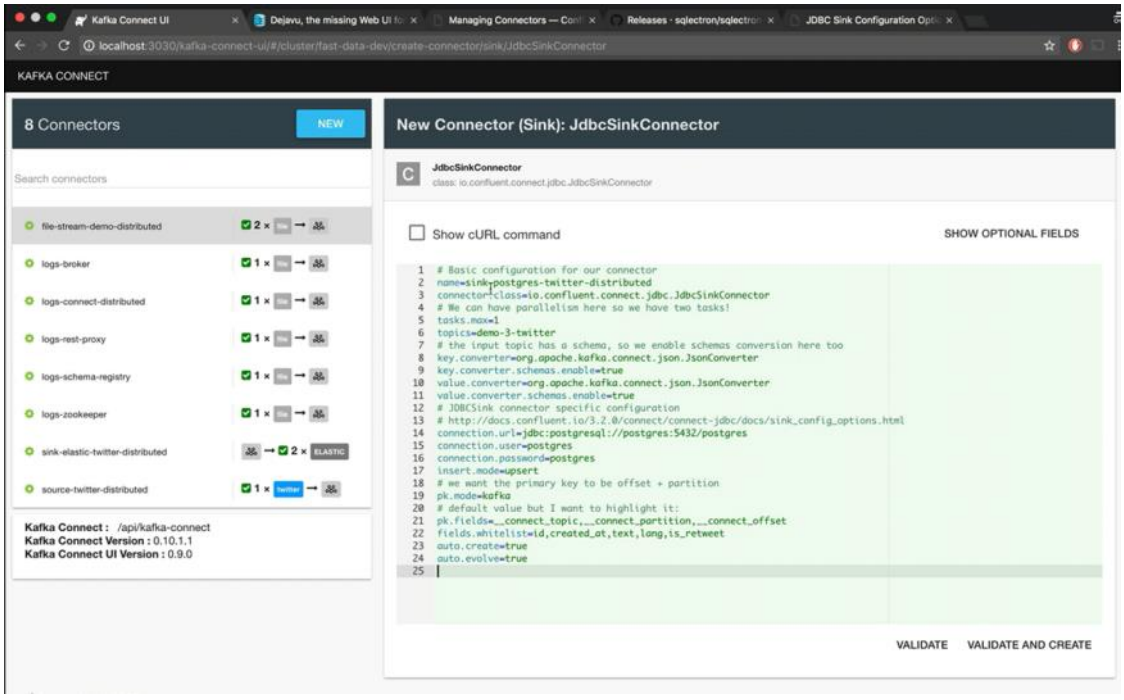
################
C) PostgresSQL demo
Examples are taken from here: http://docs.confluent.io/3.2.0/connect/connect-jdbc/docs/sink_connector.html#quickstart

используем landoop UI

```
version: '2'

services:
  # this is our kafka cluster.
  kafka-cluster:
    image: landoop/fast-data-dev:cp3.3.0
    environment:
      ADV_HOST: 127.0.0.1         # Change to 192.168.99.100 if using Docker Toolbox
      RUNTESTS: 0                 # Disable Running tests so the cluster starts faster
    ports:
      - 2181:2181                 # Zookeeper
      - 3030:3030                 # Landoop UI
      - 8081-8083:8081-8083       # REST Proxy, Schema Registry, Kafka Connect ports
      - 9581-9585:9581-9585       # JMX Ports
      - 9092:9092                 # Kafka Broker

  # we will use elasticsearch as one of our sinks.
  # This configuration allows you to start elasticsearch
  elasticsearch:
    image: itzg/elasticsearch:2.4.3
    environment:
      PLUGINS: appbaseio/dejavu
      OPTS: -Dindex.number_of_shards=1 -Dindex.number_of_replicas=0
    ports:
```

```
    - "9200:9200"

# we will use postgres as one of our sinks.
# This configuration allows you to start postgres
postgres:
  image: postgres:9.5-alpine
  environment:
    POSTGRES_USER: postgres     # define credentials
    POSTGRES_PASSWORD: postgres # define credentials
    POSTGRES_DB: postgres       # define database
  ports:
    - 5432:5432             # Postgres port
```

настроим коннектор через UI




```
# Basic configuration for our connector
name=sink-postgres-twitter-distributed
connector.class=io.confluent.connect.jdbc.JdbcSinkConnector
# We can have parallelism here so we have two tasks!
tasks.max=1
topics=demo-3-twitter

# the input topic has a schema, so we enable schemas conversion here too
key.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=true

# JDBCSink connector specific configuration
# http://docs.confluent.io/3.2.0/connect/connect-jdbc/docs/sink_config_options.html
connection.url=jdbc:postgresql://postgres:5432/postgres
connection.user=postgres
connection.password=postgres
insert.mode=upsert
# we want the primary key to be offset + partition
pk.mode=kafka
# default value but I want to highlight it:
pk.fields=__connect_topic,__connect_partition,__connect_offset
fields.whitelist=id,created_at,text,lang,is_retweet
auto.create=true
auto.evolve=true
```

? нужно сделать первоначальную настройку

```
# Load the recent and long term stats into Postgres using Kafka Connect Sink!
confluent load SinkTopics -d kafka-connectors/SinkTopicsInPostgres.properties
```

? auto table creation

* HDFS sink connector

14 декабря 2020 г. 13:26

HDFS Sink Connector: Overview

- Continuously polls from Kafka and writes to HDFS (Hadoop Distributed File System)
- Integrates with Hive
  - Auto table creation
  - Schema evolution with Avro
- Works with secure HDFS and the Hive Metastore, using Kerberos
- Provides exactly once delivery
- Data format is extensible
  - Avro, Parquet, custom formats
- Pluggable Partitioner, supporting:
  - Kafka Partitioner (default)
  - Field Partitioner
  - Time Partitioner
  - Custom Partitioners

The HDFS connector allows you to export data from Kafka topics to HDFS files in a variety of formats and integrates with Hive to make data immediately available for querying with HiveQL.

The connector periodically polls data from Kafka and writes them to HDFS. The data from each Kafka topic is partitioned by the provided partitioner and divided into chunks. Each chunk of data is represented as an HDFS file with topic, kafka partition, start and end offsets of this data chunk in the filename. If no partitioner is specified in the configuration, the default partitioner which preserves the Kafka partitioning is used. The size of each data chunk is determined by the number of records written to HDFS, the time written to HDFS and schema compatibility.

The HDFS connector integrates with Hive and when it is enabled, the connector automatically creates an external Hive partitioned table for each Kafka topic and updates the table according to the available data in HDFS.

kafka connect security

18 декабря 2020 г. 11:39

Enabling Security with Connect

- If you have security enabled in the Kafka cluster (*e.g.* SSL or SASL), you need to enable it in Connect as well
  - Configure security on a per-worker basis
  - Override the default Producer or Consumer configurations that the worker uses
  - Configurations apply to *all* connectors running on the worker

You need the configuration at the worker top level because Connect leverages Kafka's group membership protocol to coordinate the workers and distribute work between them (as it does for Consumer Groups). The configuration at the producer/consumer level is required to interact with the Kafka cluster as a client.

If using SASL for authentication, you must use the same principal for workers and connectors as only a single JAAS is currently supported on the client side.

Securing passwords used for individual connectors is handled differently and will be discussed later in the module.

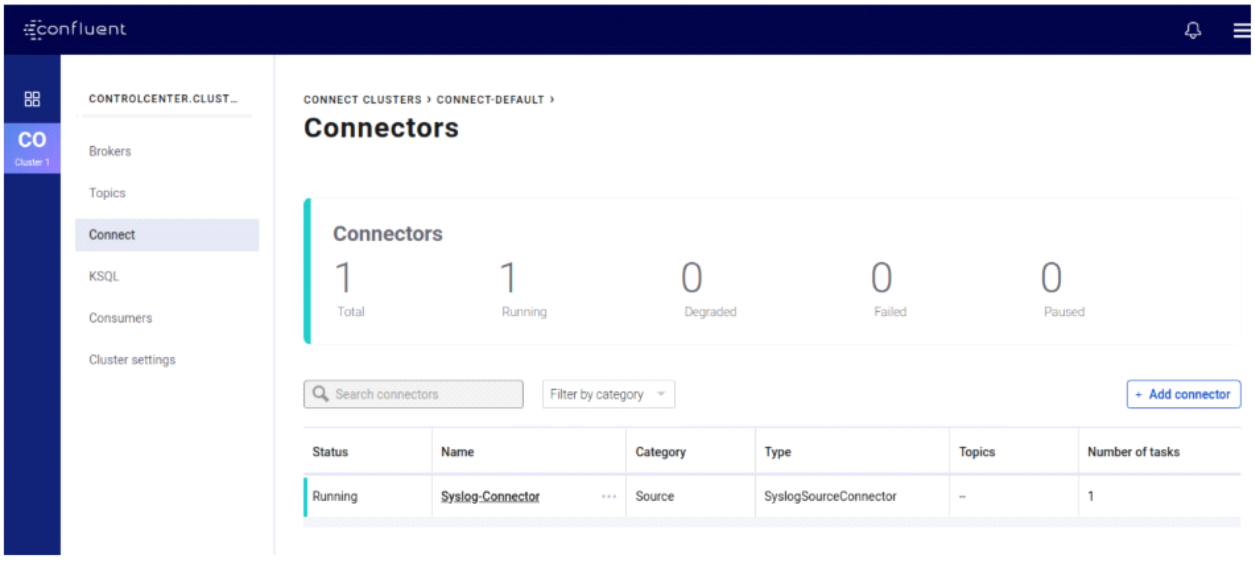
* syslog connector

18 декабря 2020 г. 14:40

3. Once the `connect` service is ready, run the following command to configure the Syslog source connector in Kafka Connect:

```
$ curl -s -X POST \
        -H "Content-Type: application/json" \
        --data '{
            "name": "Syslog-Connector",
            "config": {
                "connector.class": "io.confluent.connect.syslog.SyslogSourceConnector",
                "tasks.max": "1",
                "syslog.port": "5454",
                "syslog.listener": "TCP",
                "confluent.topic.bootstrap.servers": "kafka:9092",
                "confluent.topic.replication.factor": "1",
                "confluent.license":""
            }
        }' http://connect:8083/connectors | jq
```

4. In a new browser tab navigate to http://localhost:9021 to access Confluent Control Center:
   - Select cluster **Cluster 1**
   - then select the **Connect** tab. You will be shown the list of Connect clusters associated with your environment. There should only be a single one such cluster called `connect-default`
   - select the `connect-default` cluster and verify that there is indeed a source connector `Syslog-Connector` defined and that it is in status `running`:



7. Now let's test the connector by simulating a (syslog) message on port `5454` using the `netcat` (or `nc`) tool:
   - Output a test log entry to port 5454:

```
$ echo "<34>1 2003-10-11T22:14:15.003Z host1.acme.com su - ID47 - Engine started" \
    | nc -w 1 connect 5454
```

8. Let's verify that the entry has been written to the `syslog` topic:

```
$ kafka-avro-console-consumer \
    --bootstrap-server kafka:9092 \
    --property schema.registry.url=http://schema-registry:8081 \
    --topic syslog \
    --from-beginning | jq '.'
```

You should see this: