Kafka Connect Notes

By

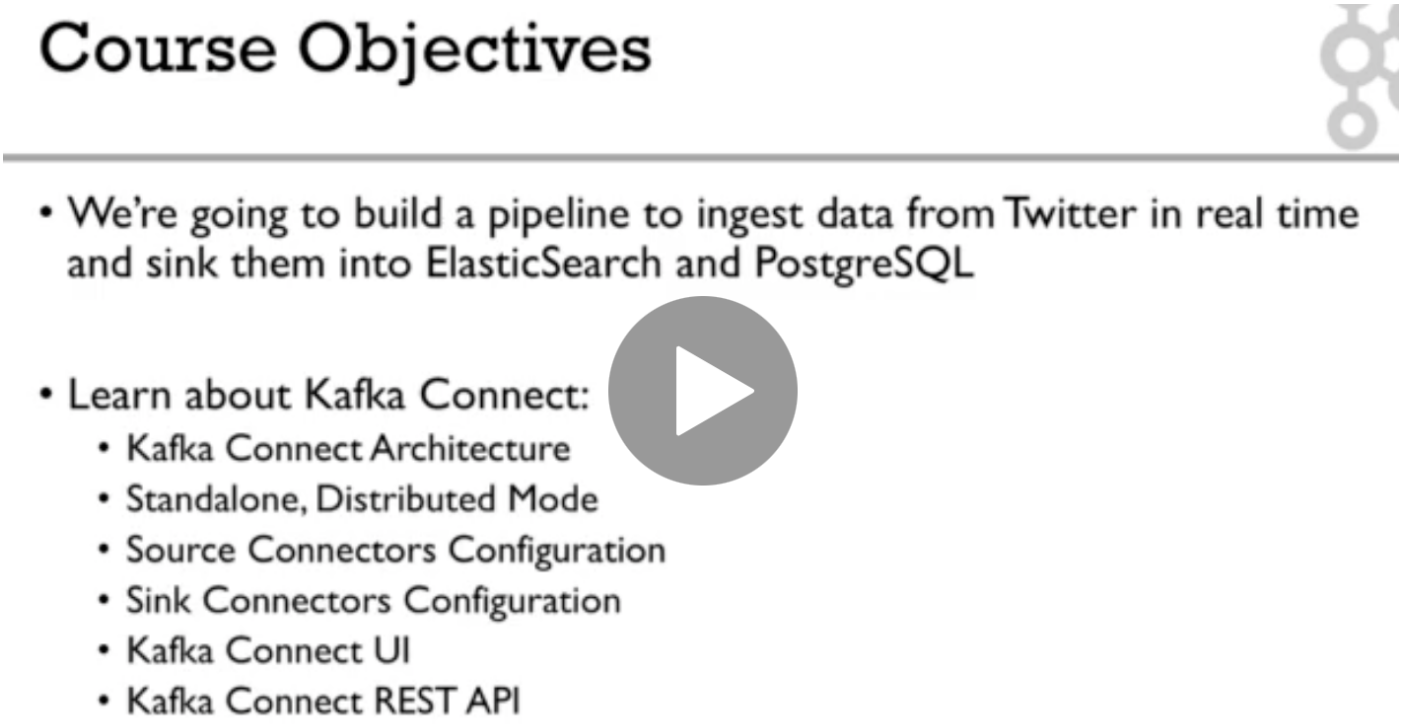
Naveen Kumar M

https://www.linkedin.com/in/naveen-kumar-m-5a6960119/

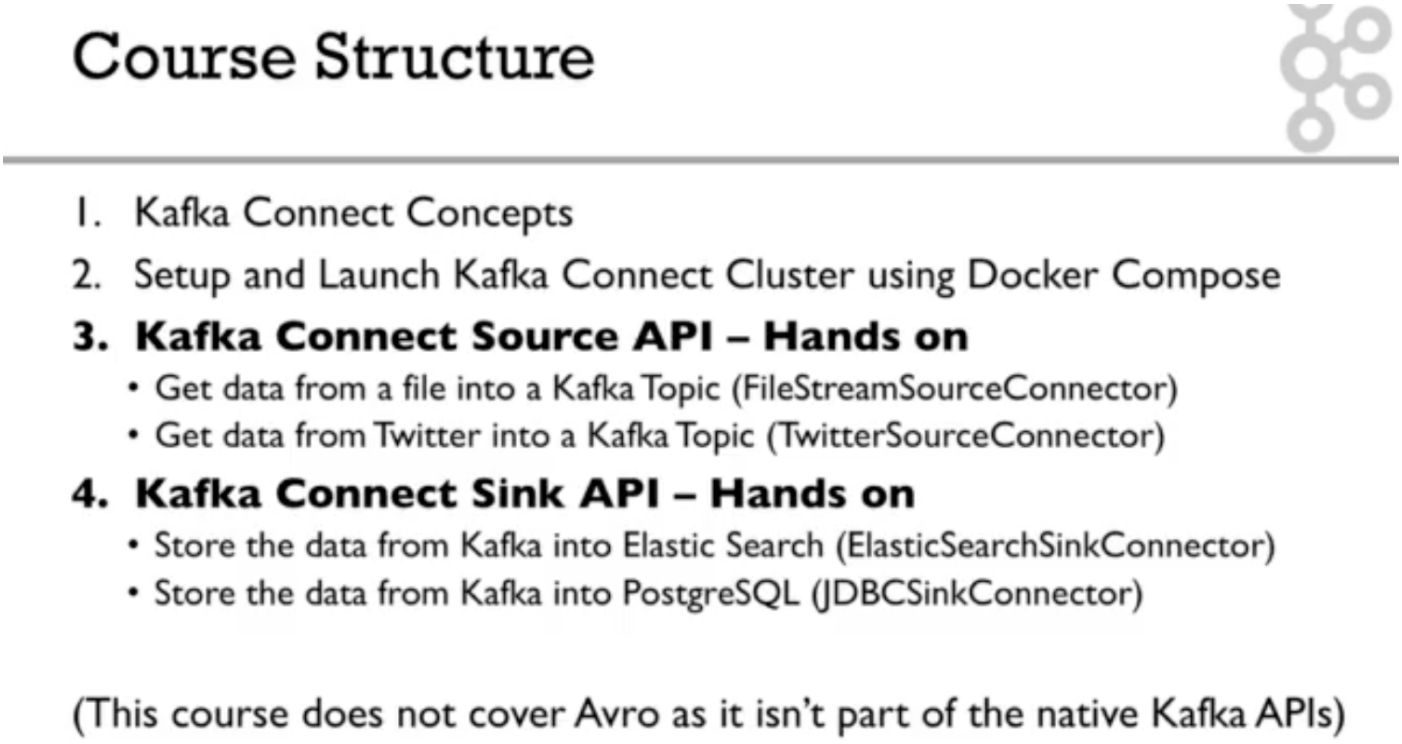
Table of Contents

About the course

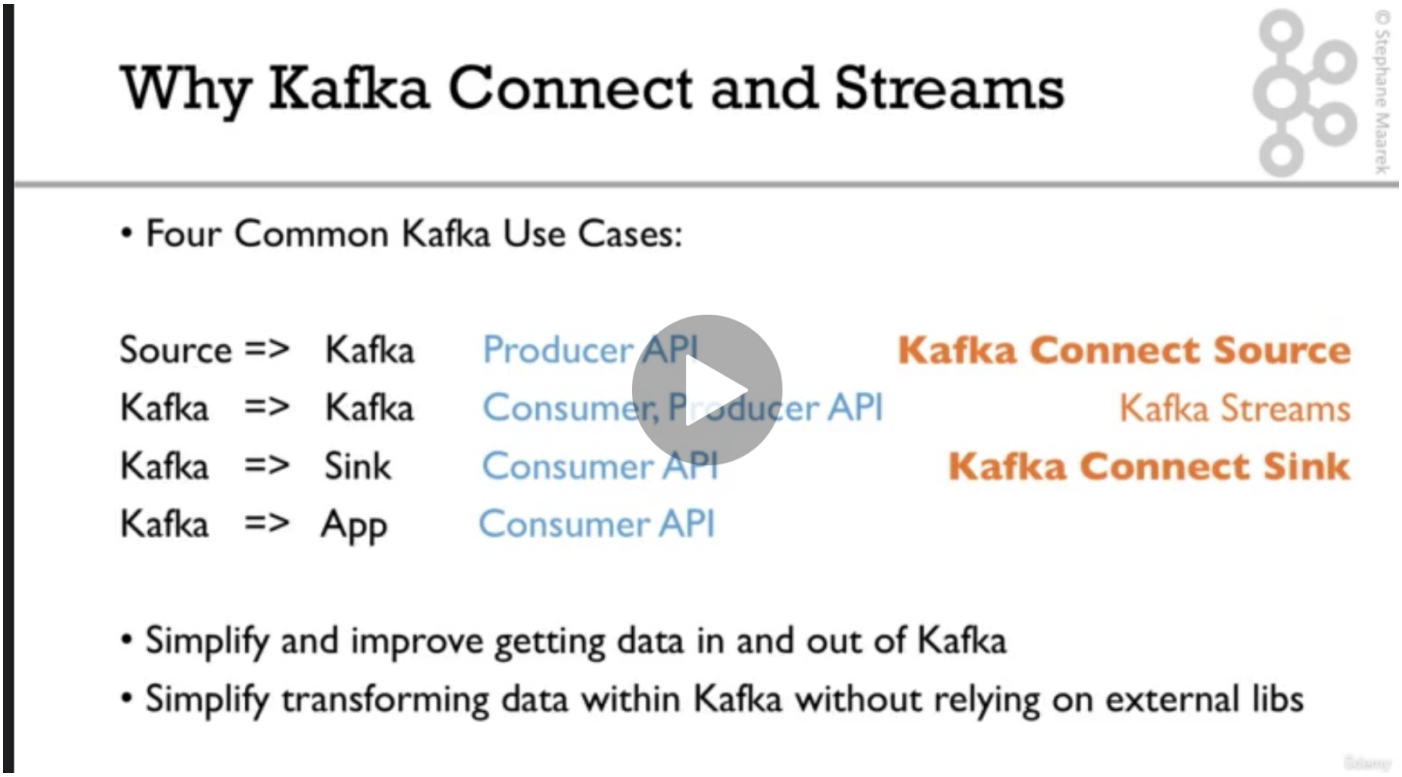
Course Objectives



Course Structure



Kafka Connect Overview



Kafka Connect Source

- Kafka Connect API, put all the data from source to Kafka.

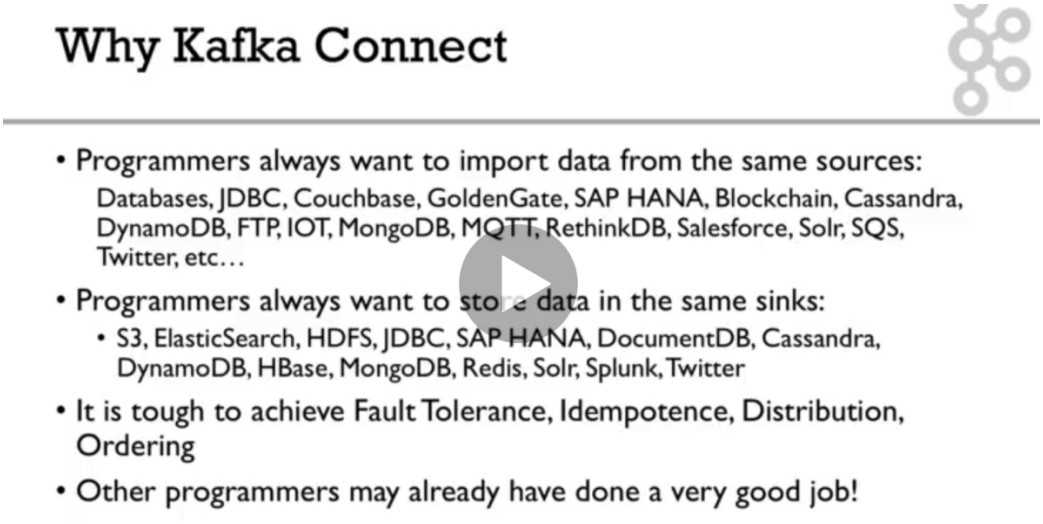
Kafka Streams

- Does transformation on Kafka topic.

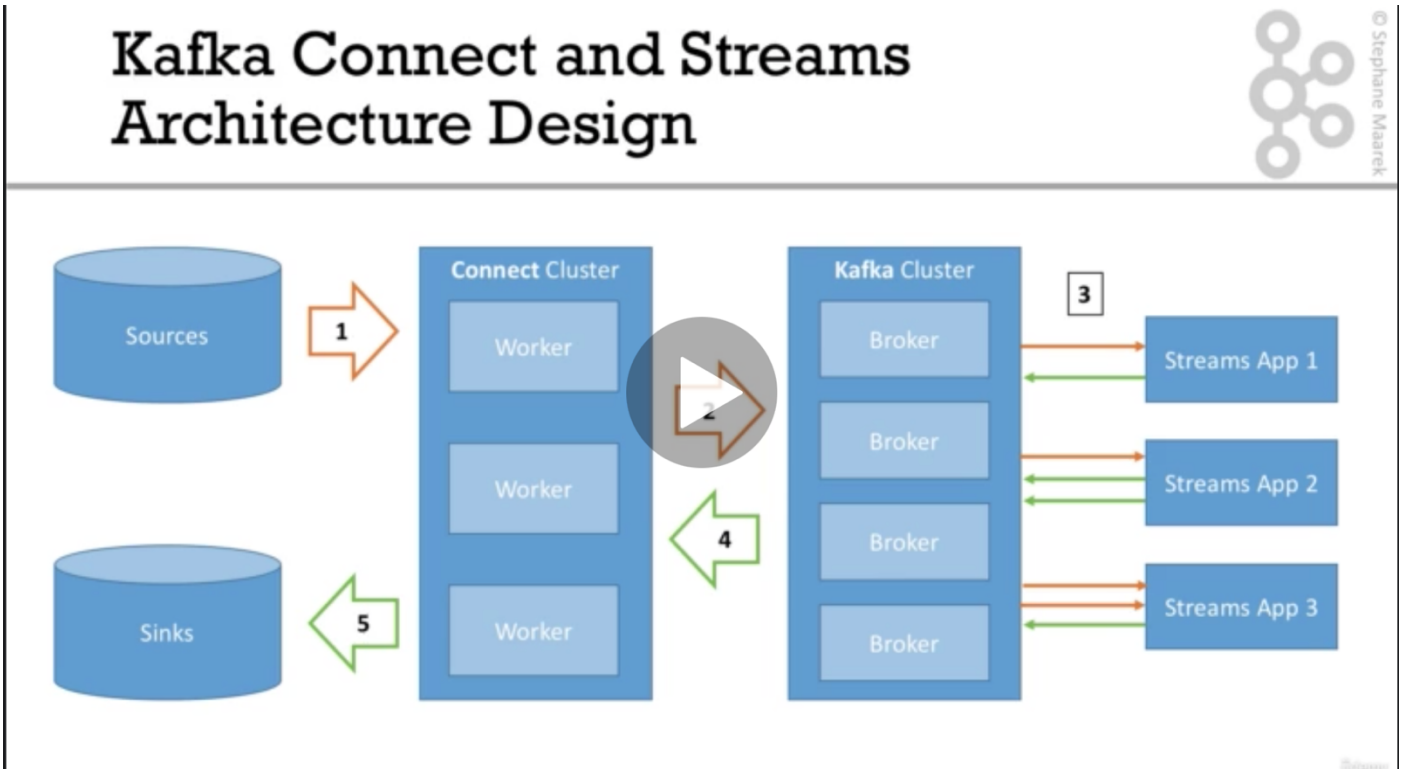
Kafka Connect Sink

- Getting data out of Kafka and put it into target.

Why Kafka Connect



Kafka Connect and Streams Architecture Design



Steps

- Connect cluster made of workers. Pull the data from source and push it into Kafka cluster.
- Stream pull the data and does the transformation and put it back to Kafka cluster.

Kafka Connect Concepts

High Level

Kafka Connect – High level

- Source Connectors to get data from Common Data Sources
- Sink Connectors to publish that data in Common Data Stores
- Make it easy for non-experienced dev to quickly get their data reliably into Kafka
- Part of your ETL pipeline
- Scaling made easy from small pipelines to company-wide pipelines
- Re-usable code!

Kafka Connect Concepts

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

Key Notes:

- Connectors are re-usable piece of code.
- Connectors + User configuration gives us tasks.
- Task linked to user configuration.
- Job is a combination of multiple tasks.
- Task are executed by Kafka connect workers.

Standalone vs Distributed Mode

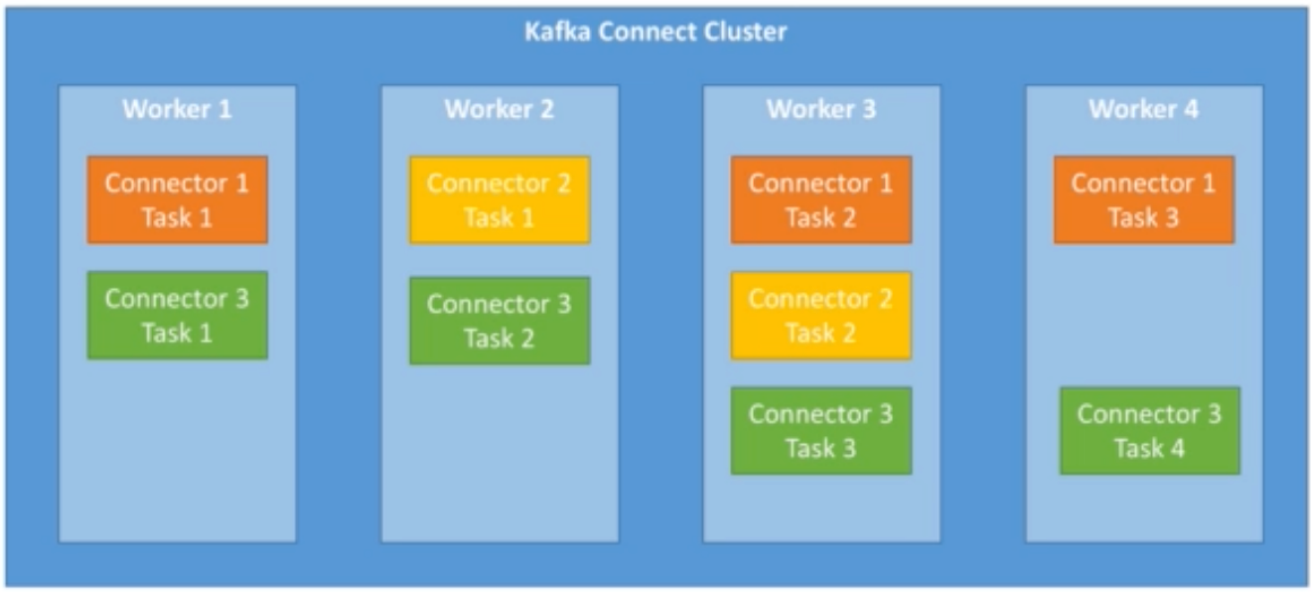
Kafka Connect Workers
Standalone vs Distributed Mode

- 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

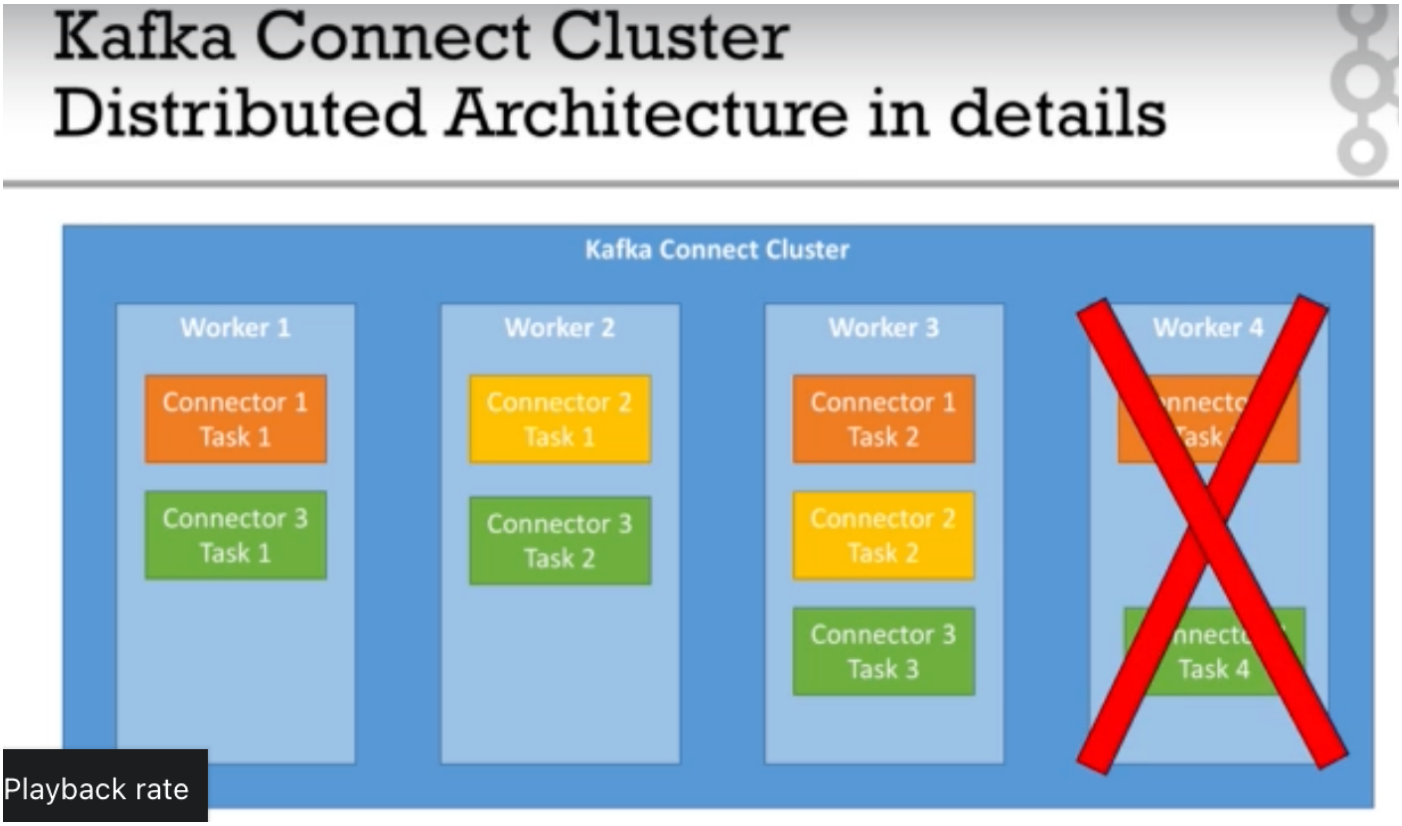
Distributed Mode in Details

We have connector's tasks spread across in workers.

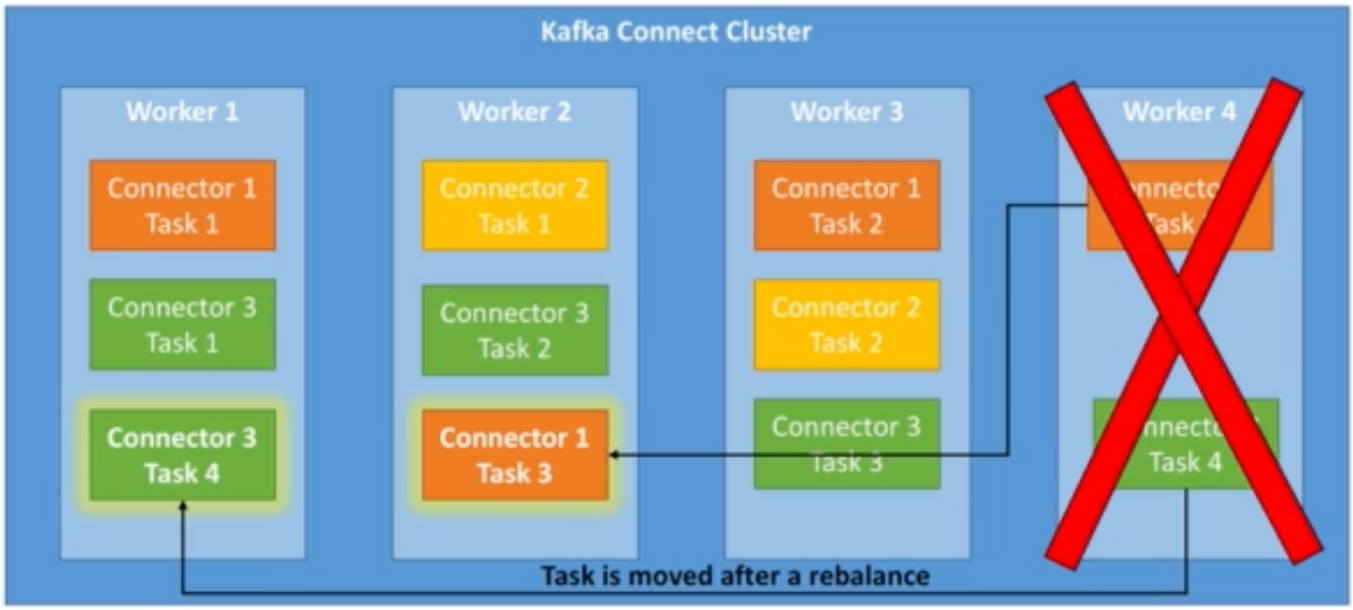
Kafka Connect Cluster
Distributed Architecture in details



Unfortunately, one worker dies due to network error.



Now Kafka Connect Cluster does the automatic rebalancing. So, tasks exist in failed worker node moved to working worked node.

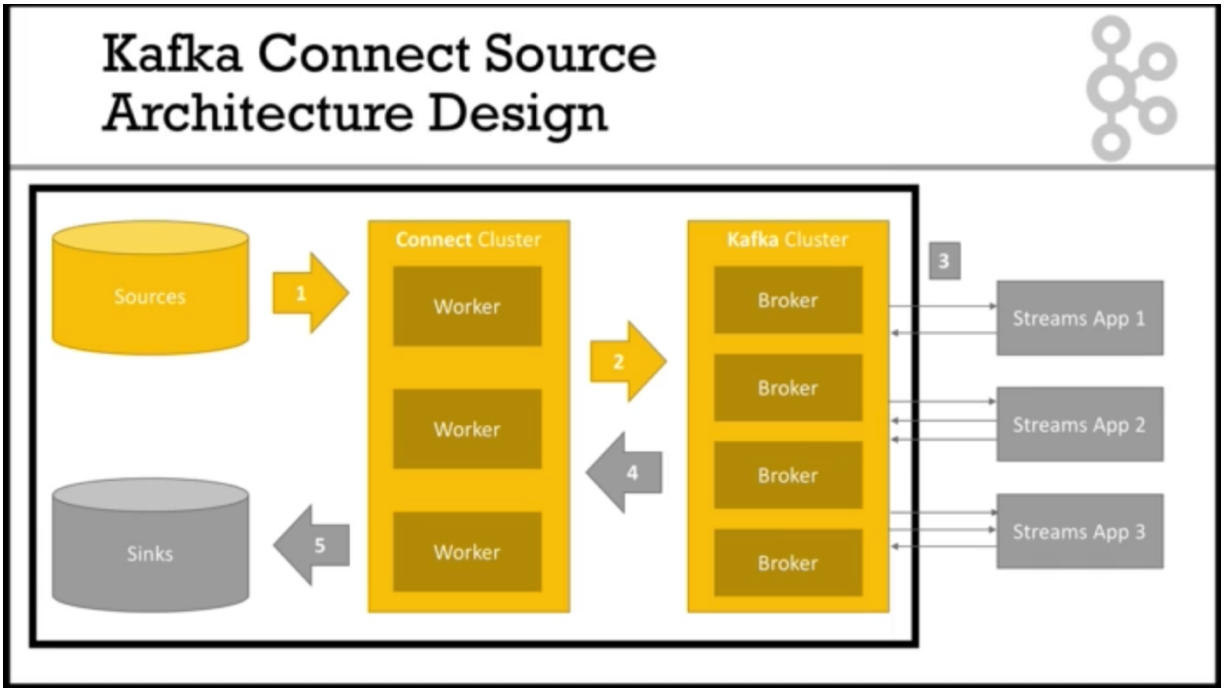


Docker

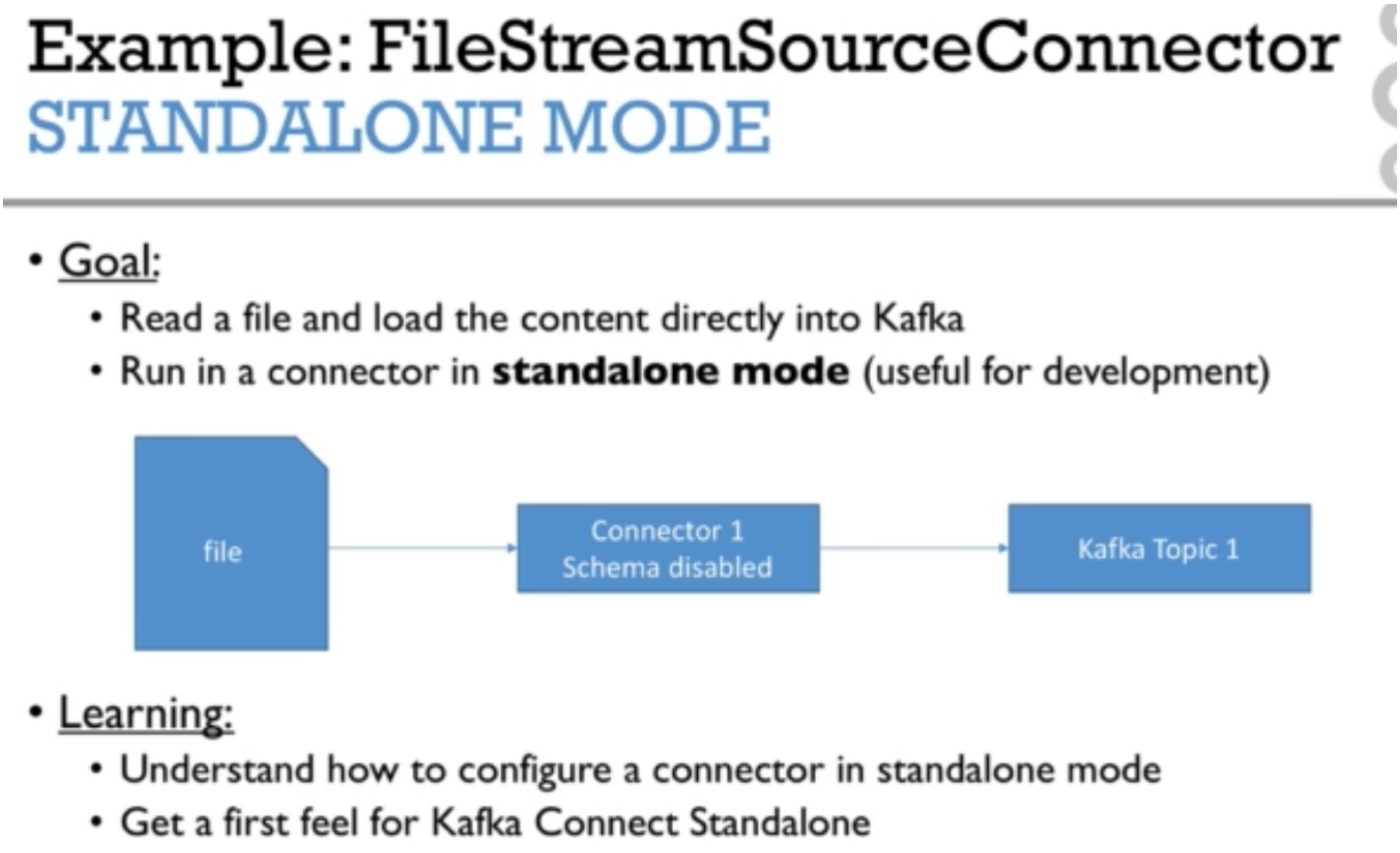
- Docker allows you to run containers.
- Containers are basically applications that have already been configured for you and that require no additional setup.

- It's the only way to ensure that an application will run the same way on every computer, and that's why we use it in this course.

Kafka Connect Source

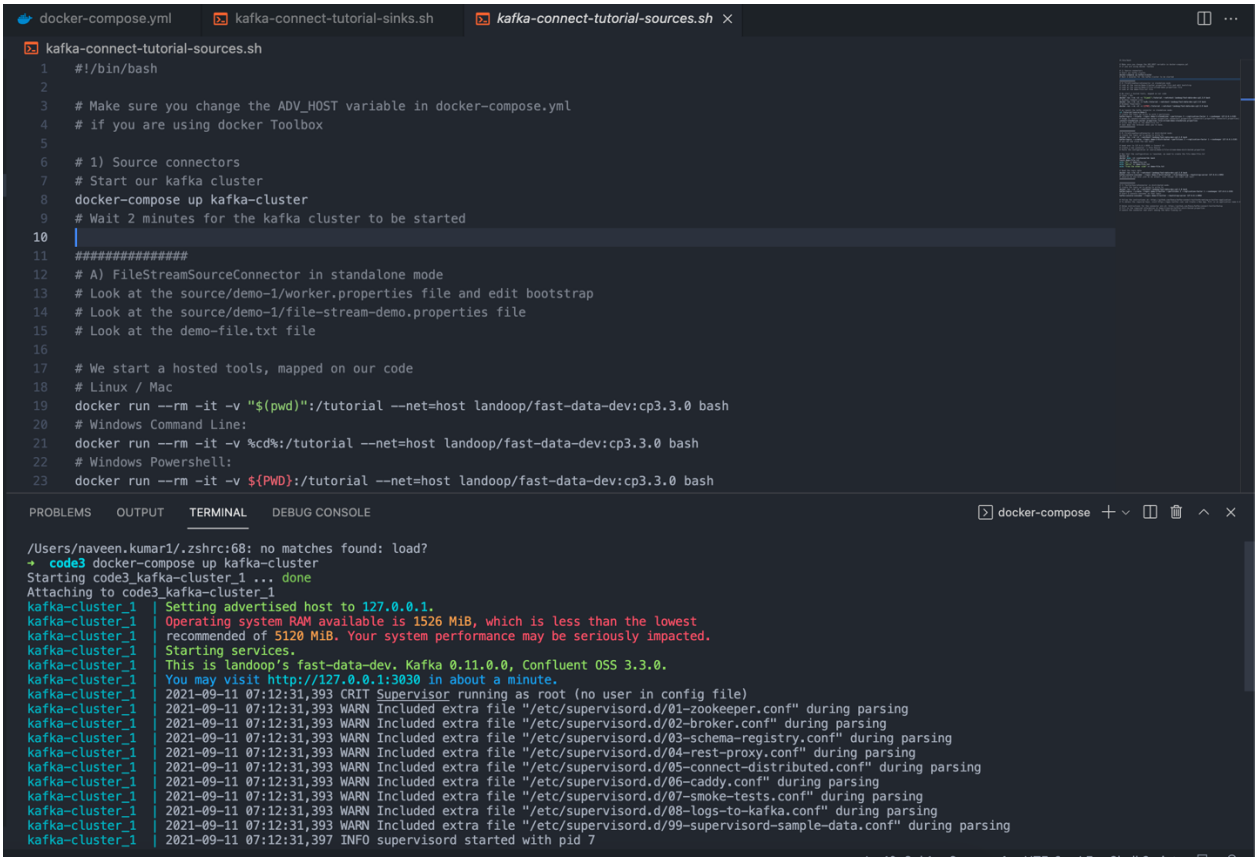


Standalone Mode



Standalone Mode Demo

Step 1: docker-compose up kafka-cluster



Step 2: Configuring property files.

Set up for infrastructure.

source > demo-1 > worker.properties

```
# from more information, visit: http://docs.confluent.io/3.2.0/connect
bootstrap.servers=127.0.0.1:9092
key.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=false
value.converter=org.apache.kafka.connect.json.JsonConverter
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
```

It is required only for standalone mode. For distributed mode worker.properties is not required.

Set up for connector

source > demo-1 > file-stream-demo-standalone.properties

```
# 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
# Parameters can be found here: https://github.com/apache/kafka/blob/trunk
file=demo-file.txt
topic=demo-1-standalone

```

Step 3: Run the container.

docker run --rm -it -v "$(pwd)":/tutorial --net=host landoop/fast-data-dev:cp3.3.0 bash

--rm – is used when you need the container to be deleted after the task for it is complete.

-it – Starts the container in the interactive mode (hence -it flag) that allows you to interact with /bin/bash of the container. That means now you will have bash session inside the container, so you can ls, mkdir, or do any bash command inside the container.

```
→  code3 docker run --rm -it -v "$(pwd)":/tutorial --net=host landoop/fast-data-dev:cp3.3.0 bash
root@fast-data-dev / $ ls /tutrial
ls: cannot access '/tutrial': No such file or directory
root@fast-data-dev / $ ls /tutorial
docker-compose.yml  kafka-connect-tutorial-sinks.sh  kafka-connect-tutorial-sources.sh  setup  sink  source
root@fast-data-dev / $
```

Step 4: Create a Kafka topic.

kafka-topics --create --topic demo-1-standalone --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181

```
root@fast-data-dev / $ kafka-topics --create --topic demo-1-standalone --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181

Created topic "demo-1-standalone".
```

Under the demo-1 folder we have a property file required for connector standalone.

```
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
root@fast-data-dev demo-1 $
```

Step 5: Run standalone connector.

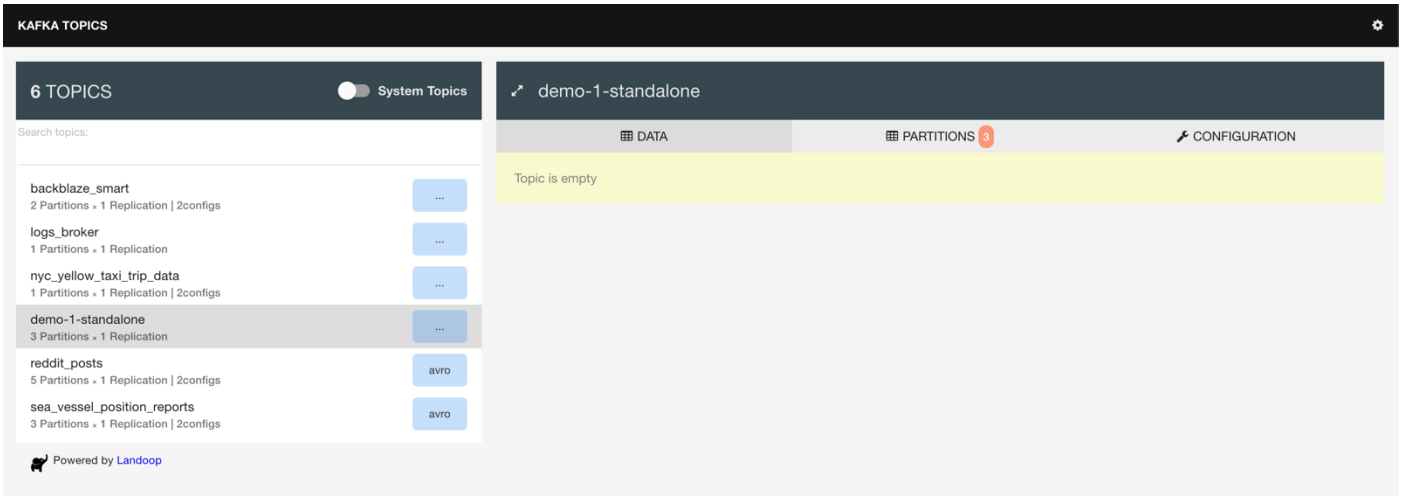
connect-standalone worker.properties file-stream-demo-standalone.properties

```
root@fast-data-dev demo-1 $ connect-standalone worker.properties file-stream-demo-standalone.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/confluent/share/java/kafka/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/confluent/share/java/kafka-connect-hdfs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/confluent/share/java/kafka-connect-twitter/kafka-connect-twitter-0.1-master-af63e4c-cp3.3.0-jar-with-dependencies.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
```

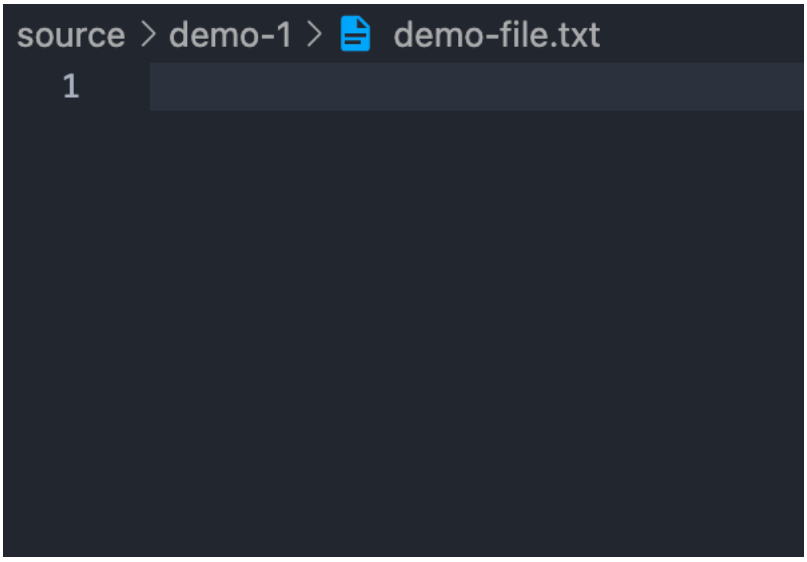
Since we redirected to the property files folder. We could able to pass the property file directly without giving the folder path.

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

        ssl.keymanager.algorithm = SunX509
        ssl.keystore.location = null
        ssl.keystore.password = null
        ssl.keystore.type = JKS
        ssl.protocol = TLS
        ssl.provider = null
        ssl.secure.random.implementation = null
        ssl.trustmanager.algorithm = PKIX
        ssl.truststore.location = null
        ssl.truststore.password = null
        ssl.truststore.type = JKS
        transaction.timeout.ms = 60000
        transactional.id = null
        value.serializer = class org.apache.kafka.common.serialization.ByteArraySerializer
 (org.apache.kafka.clients.producer.ProducerConfig:223)
[2021-09-11 09:40:18,952] INFO Kafka version : 0.11.0.0-cp1 (org.apache.kafka.common.utils.AppInfoParser:83)
[2021-09-11 09:40:18,969] INFO Kafka commitId : 5cadaa94d0a69e0d (org.apache.kafka.common.utils.AppInfoParser:84)
[2021-09-11 09:40:20,565] INFO Source task WorkerSourceTask{id=file-stream-demo-standalone-0} finished initialization and start (org.apache.kafka.connect.runtime.WorkerSourceTask:143)
[2021-09-11 09:40:21,067] INFO Created connector file-stream-demo-standalone (org.apache.kafka.connect.cli.ConnectStandalone:91)
```

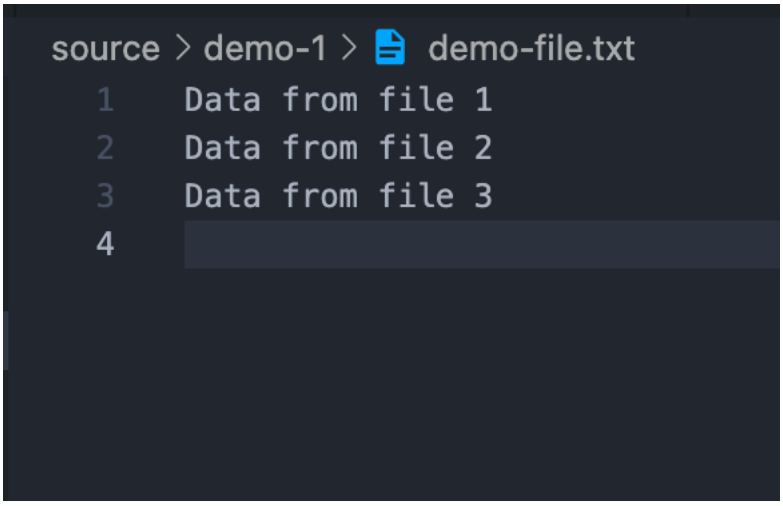
Step 6: Checking the output in UI

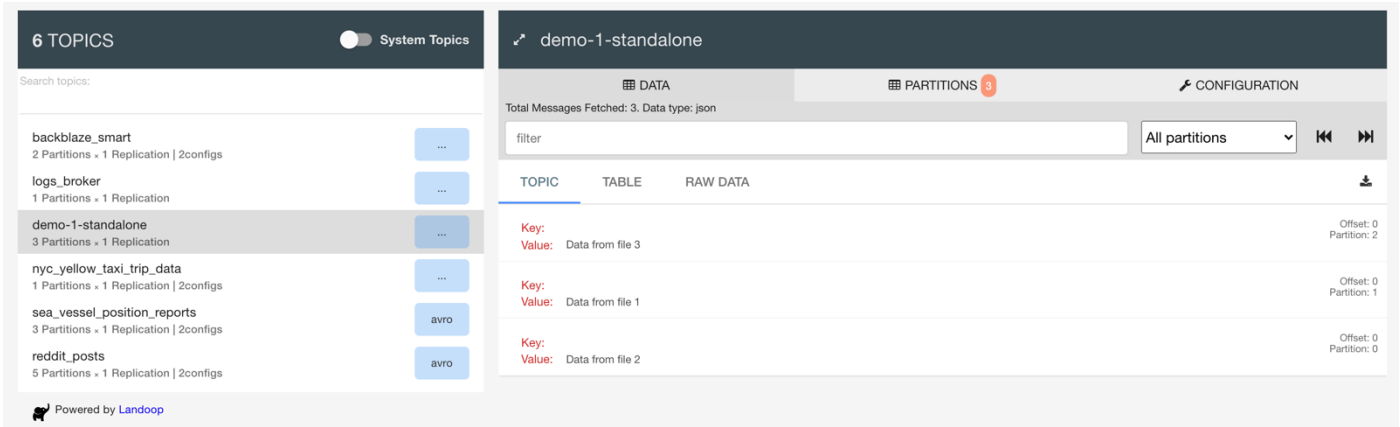


Topic is empty because our file is empty.

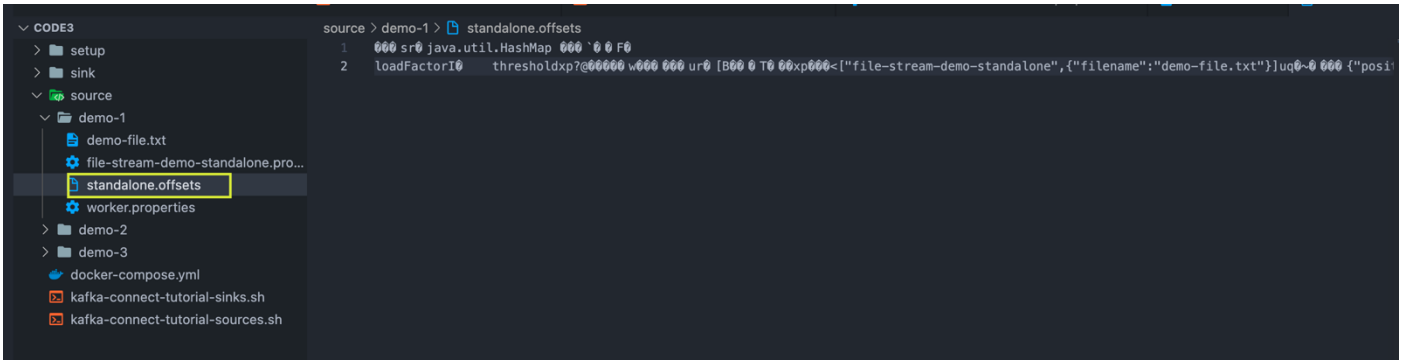


Let’s put some data in file check the topic data.

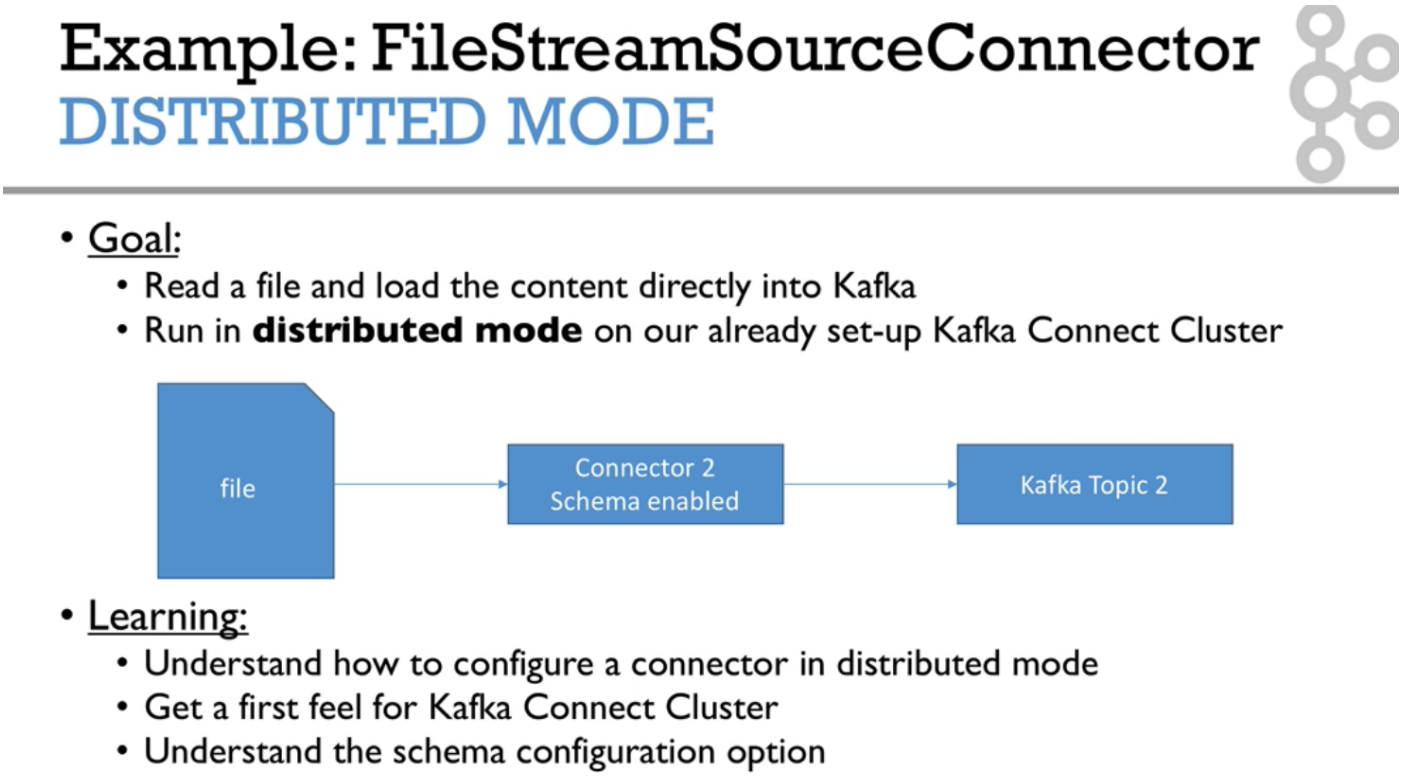




It created the standalone.offsets file to store the offset.



Distributed Mode



Step 1: Run the landoop tool and create a topic.

docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash

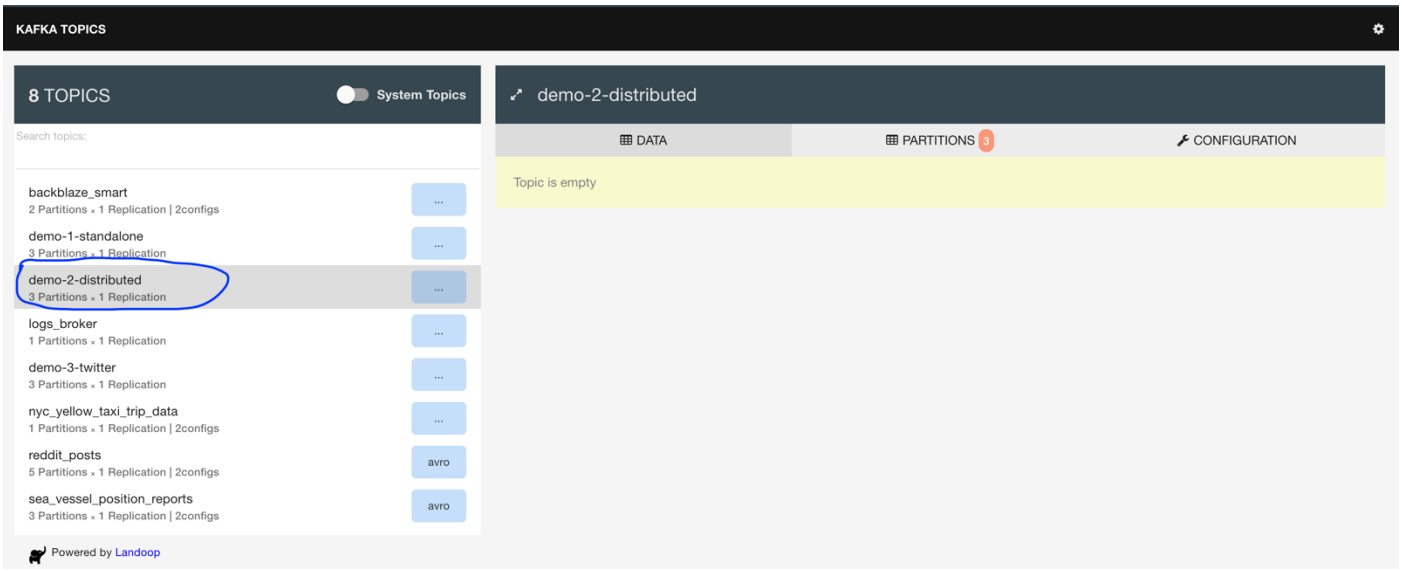
```
➜  code3 docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
root@fast-data-dev / $
```

Create a topic named demo-2-distributed.

kafka-topics --create --topic demo-2-distributed --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181

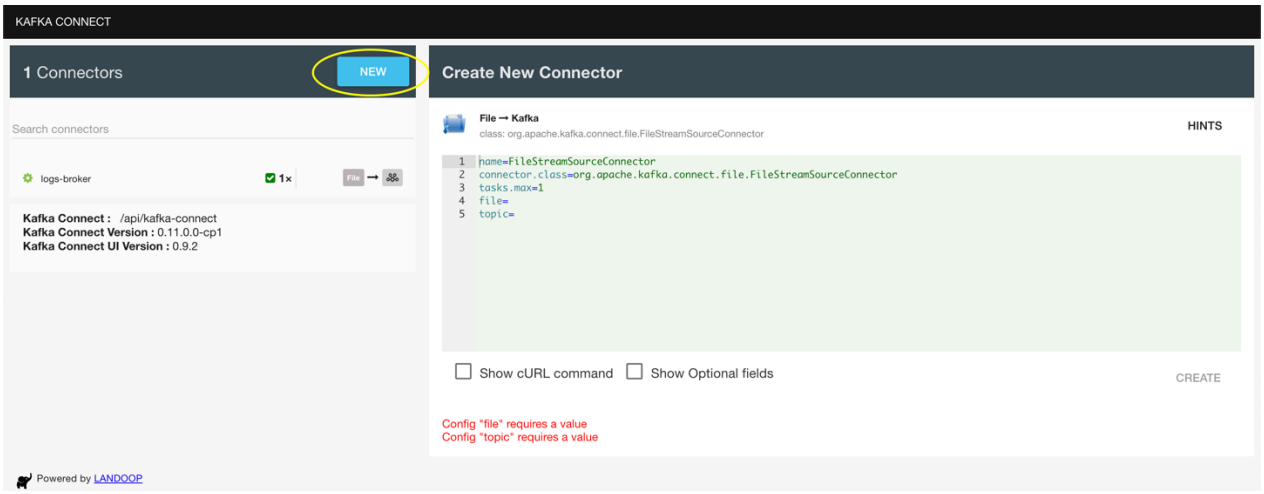
```
root@fast-data-dev / $ kafka-topics --create --topic demo-2-distributed --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181
Created topic "demo-2-distributed".
root@fast-data-dev / $
```

As we can see that topic we had created, reflected in the UI.

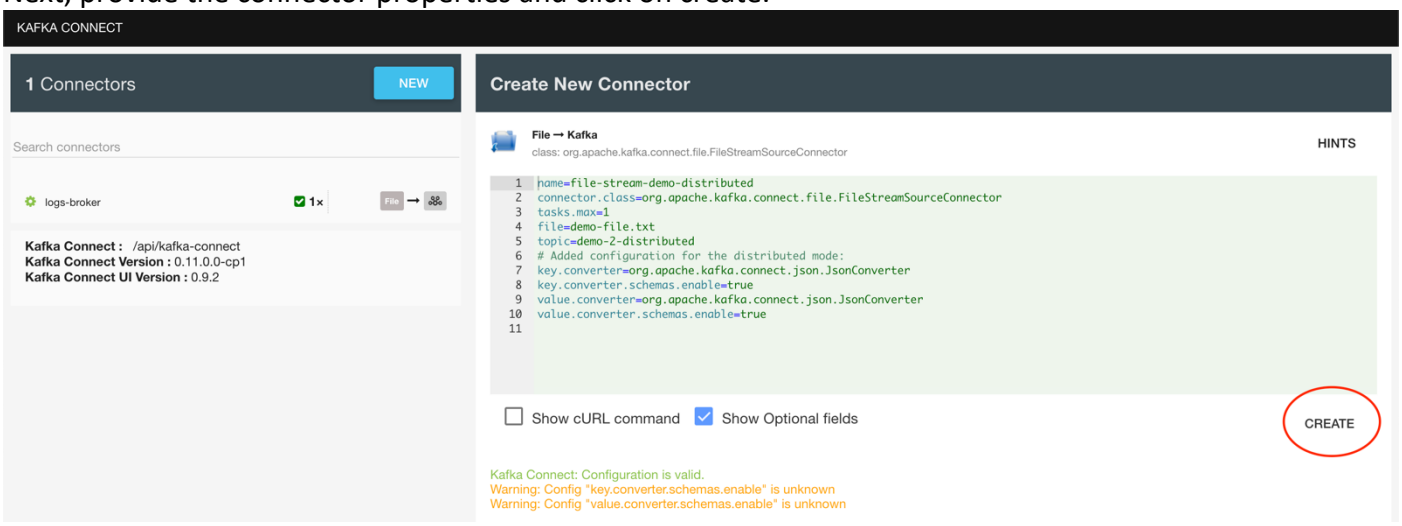


Step 2: Create a connector cluster.

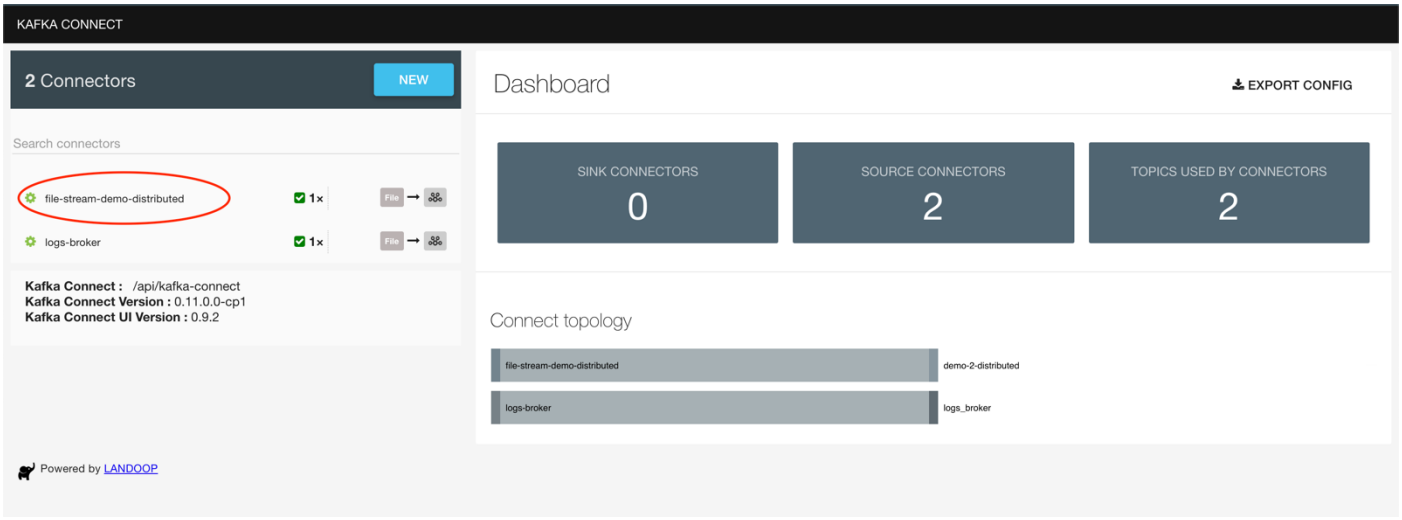
From the dashboard, go to connector UI and click ‘NEW’ button for creating a new connector.



Next, provide the connector properties and click on create.



Now our connector created successfully.



Step 3: Creating a source file inside the Kafka Connect Cluster.

Identifying docker container id.

docker ps

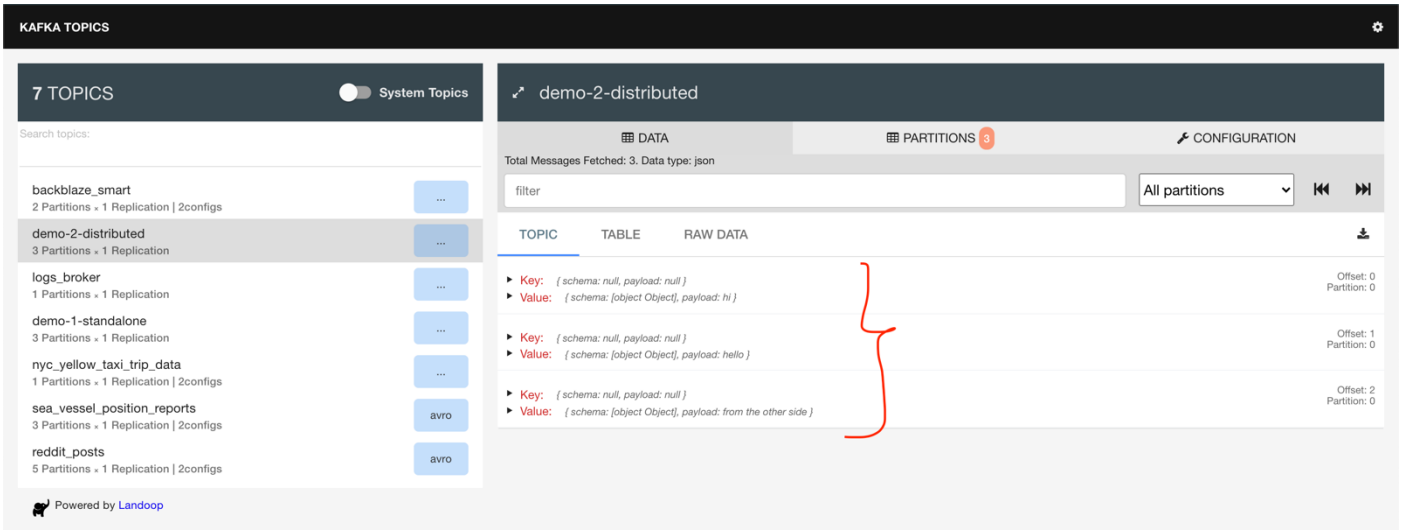
```
→  code3 docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED        STATUS         PORTS
                                                                                                     NAMES
545ee6112c13   landoop/fast-data-dev:cp3.3.0   "/usr/local/bin/dumb…"   44 hours ago   Up 29 minutes   0.0.0.0:2181->2181/tcp, :::2181->2181/tcp, 0.
0.0.0:3030->3030/tcp, :::3030->3030/tcp, 0.0.0.0:8081-8083->8081-8083/tcp, :::8081-8083->8081-8083/tcp, 0.0.0.0:9092->9092/tcp, :::9092->9092/tcp, 0
.0.0.0:9581-9585->9581-9585/tcp, :::9581-9585->9581-9585/tcp, 3031/tcp   code3_kafka-cluster_1
→  code3
```

Next, we are creating the file inside the container and feeding the data.

docker exec -it 545ee6112c13 bash
touch demo-file.txt
echo "hi" >> demo-file.txt
echo "hello" >> demo-file.txt
echo "from the other side" >> demo-file.txt

```
→  code3 docker exec -it 545ee6112c13 bash

root@fast-data-dev / $ touch demo-file.txt
root@fast-data-dev / $ ls
bin         connectors     dev  extra-connect-jars  lib    mnt  proc  run   srv  tmp  var
build.info  demo-file.txt  etc  home                media  opt  root  sbin  sys  usr
root@fast-data-dev / $ echo "hi" >> demo-file.txt
root@fast-data-dev / $ echo "hello" >> demo-file.txt
root@fast-data-dev / $ echo "from the other side" >> demo-file.txt
root@fast-data-dev / $
```

Step 3: Checking the topic to see the feed data.



Step 4: Checking output for validating the schema configuration.

As per our connector's configuration, we can see the data in json format.

docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
kafka-console-consumer --topic demo-2-distributed --from-beginning --bootstrap-server 127.0.0.1:9092

```
root@fast-data-dev / $ exit
exit
➜  code3 docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
root@fast-data-dev / $ kafka-console-consumer --topic demo-2-distributed --from-beginning --bootstrap-server 127.0.0.1:9092
{"schema":{"type":"string","optional":false},"payload":"hi"}
{"schema":{"type":"string","optional":false},"payload":"hello"}
{"schema":{"type":"string","optional":false},"payload":"from the other side"}
```

List of Available 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/hub/

- The second best place to find them is Google ☺

Connectors in Landoop distribution

- Many connectors are bundled with the Landoop Docker image so you can use them right away
- If you want to add your own, you need to map your volumes to your Landoop Docker image on start-up.
- More information:
  https://github.com/Landoop/fast-data-dev#enable-additional-connectors

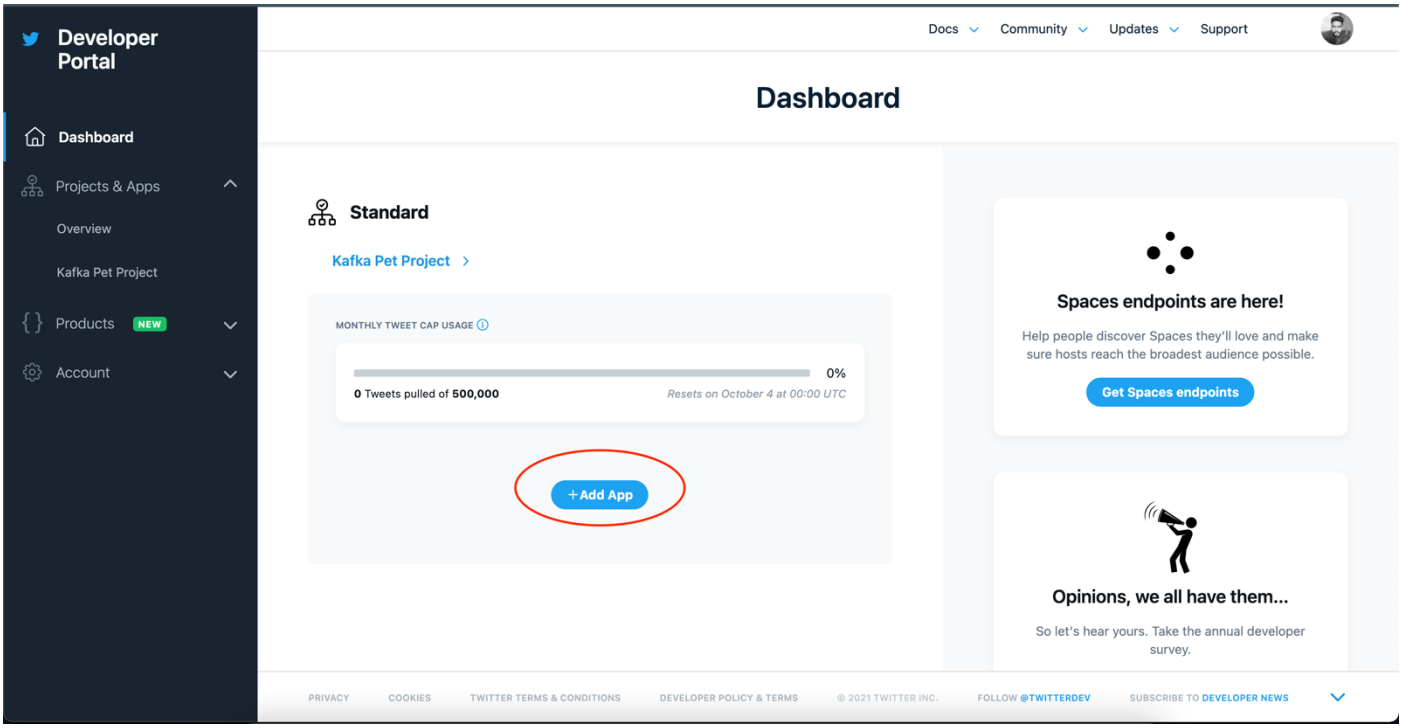
Twitter Source Connector in Distributed Mode

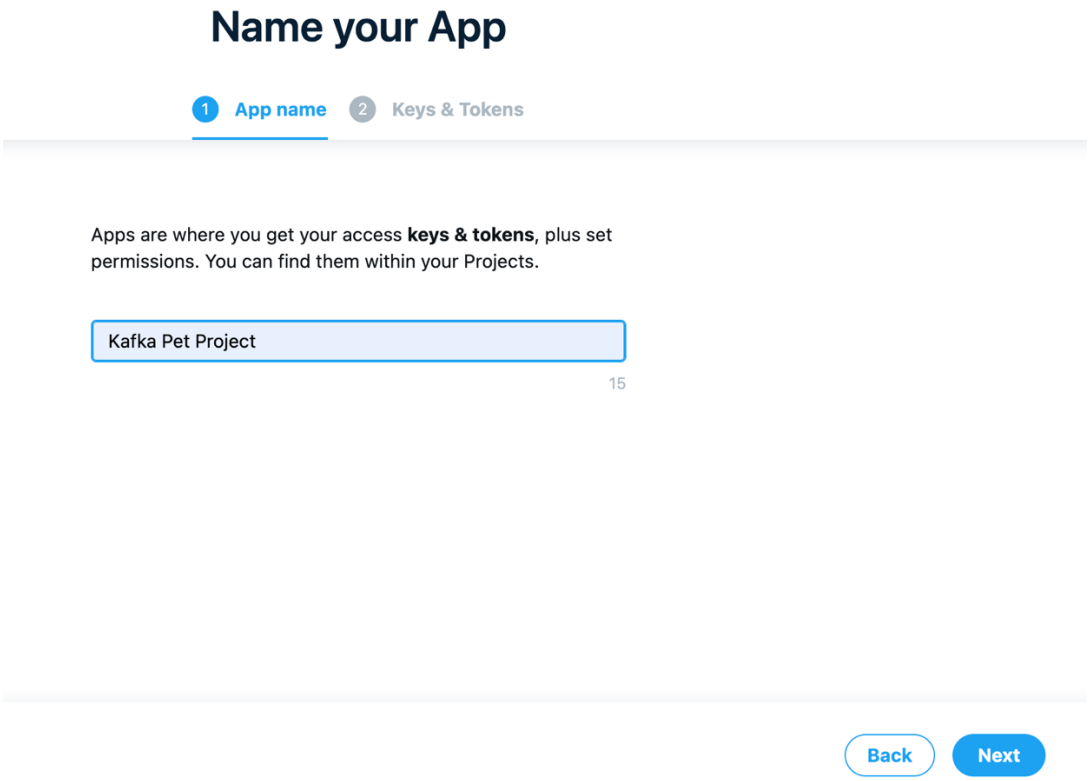
TwitterSourceConnector

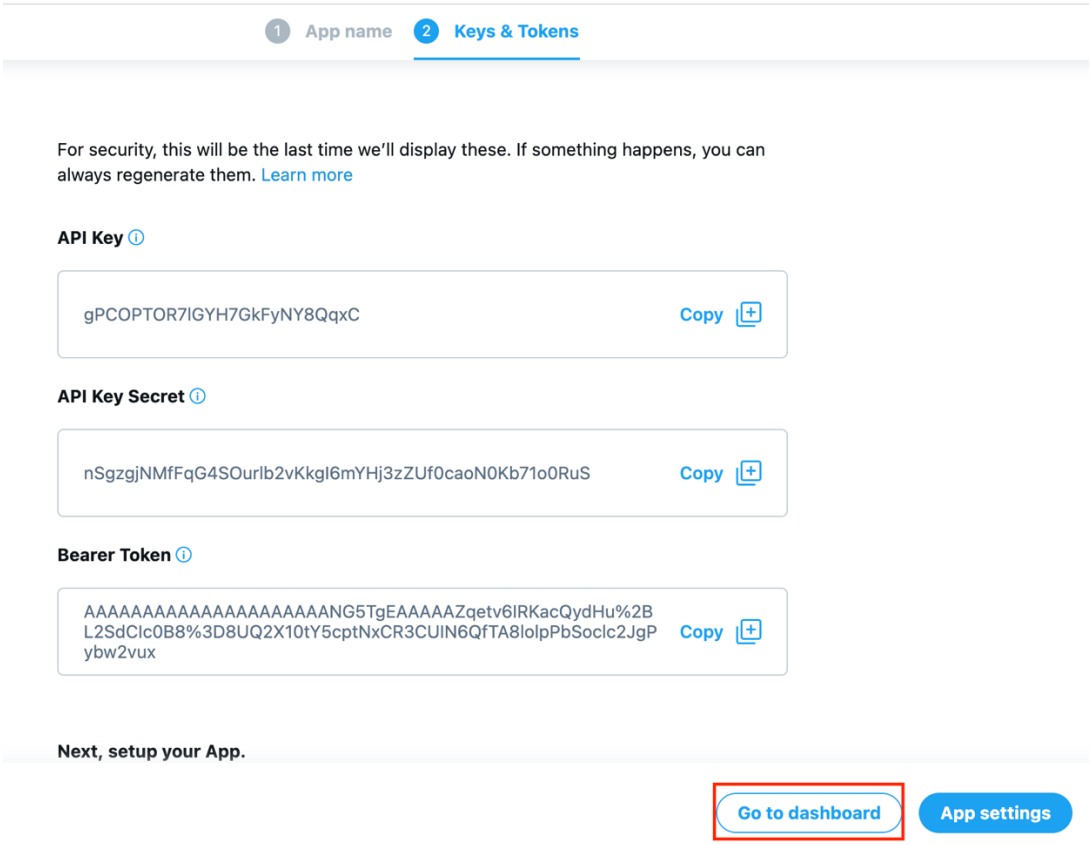
- Goal:
  - Gather data from Twitter in Kafka Connect Distributed mode

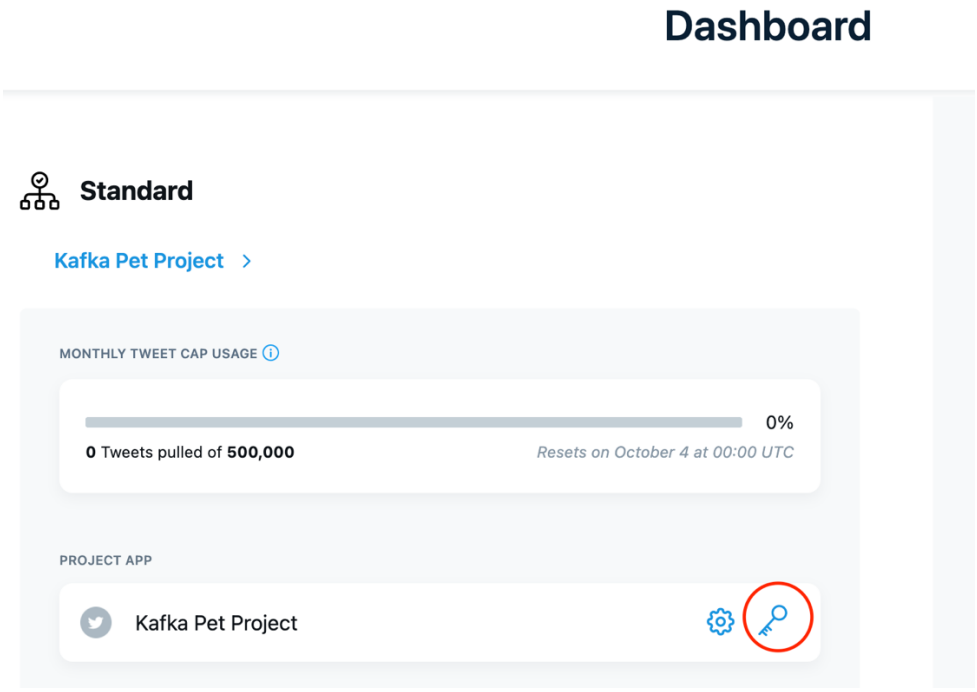
- Learning:
  - Gather real data using https://github.com/Eneco/kafka-connect-twitter

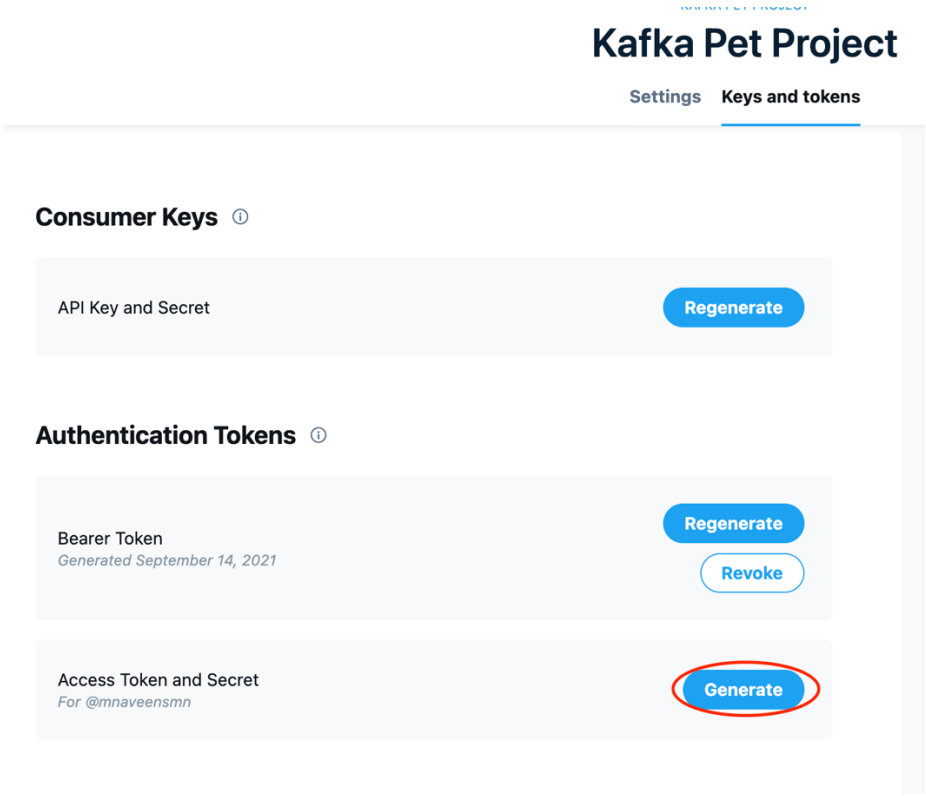
Step 1: Create a twitter developer account and create an app.

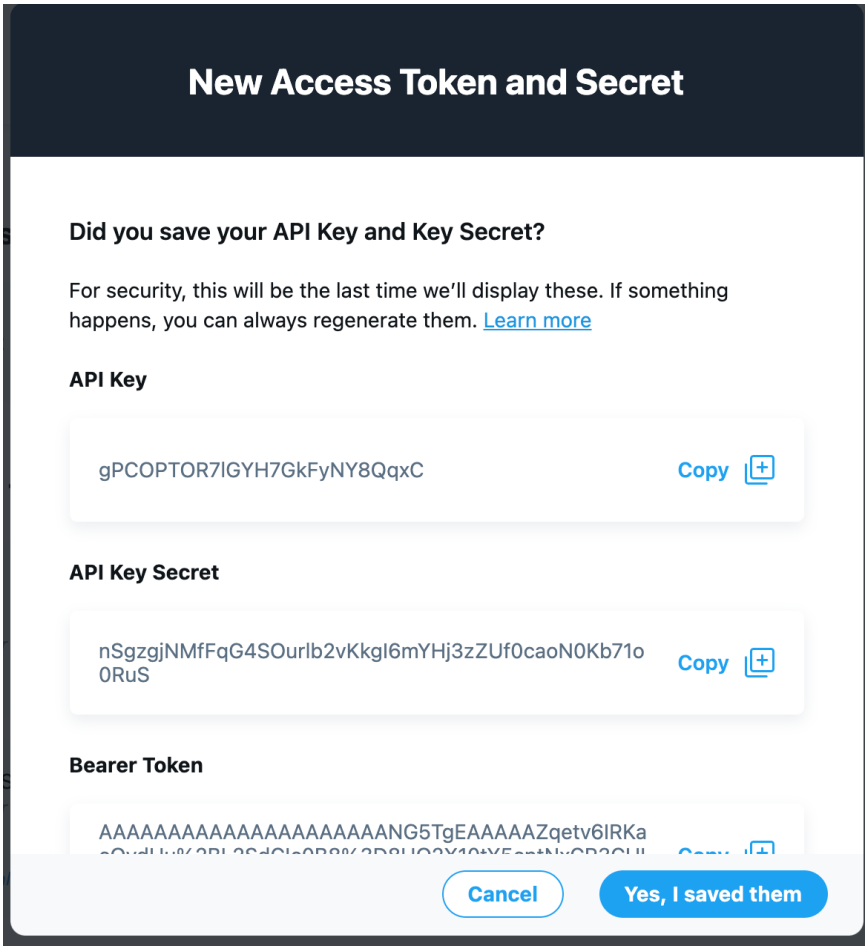

Developer Portal
Dashboard
Projects & Apps
Overview
Kafka Pet Project
Products
NEW
Account
Docs
Community
Updates
Support
Dashboard
Standard
Kafka Pet Project
MONTHLY TWEET CAP USAGE
0%
0 Tweets pulled of 500,000
Resets on October 4 at 00:00 UTC
+Add App
Spaces endpoints are here!
Help people discover Spaces they'll love and make sure hosts reach the broadest audience possible.
Get Spaces endpoints
Opinions, we all have them...
So let's hear yours. Take the annual developer survey.
PRIVACY
COOKIES
TWITTER TERMS & CONDITIONS
DEVELOPER POLICY & TERMS
© 2021 TWITTER INC.
FOLLOW @TWITTERDEV
SUBSCRIBE TO DEVELOPER NEWS


Name your App
1 App name
2 Keys & Tokens
Apps are where you get your access keys & tokens, plus set permissions. You can find them within your Projects.
Kafka Pet Project
15
Back
Next


1 App name
2 Keys & Tokens
For security, this will be the last time we'll display these. If something happens, you can always regenerate them. Learn more
API Key
gPCOPTOR7lGYH7GkFyNY8QqxC
Copy
API Key Secret
nSgzgjNMfFqG4SOurlb2vKkgI6mYHj3zZUf0caoN0Kb71o0RuS
Copy
Bearer Token
AAAAAAAAAAAAAAAAAAAAANG5TgEAAAAAZqetv6lRKacQydHu%2BL2SdClc0B8%3D8UQ2X10tY5cptNxCR3CUlN6QfTA8lolpPbSoclc2JgPybw2vux
Copy
Next, setup your App.
Go to dashboard
App settings


Dashboard
Standard
Kafka Pet Project
MONTHLY TWEET CAP USAGE
0%
0 Tweets pulled of 500,000
Resets on October 4 at 00:00 UTC
PROJECT APP
Kafka Pet Project





Step 2: Configure a property for 'TwitterSourceConnector' configuration.

source > demo-3 > source-twitter-distributed.properties

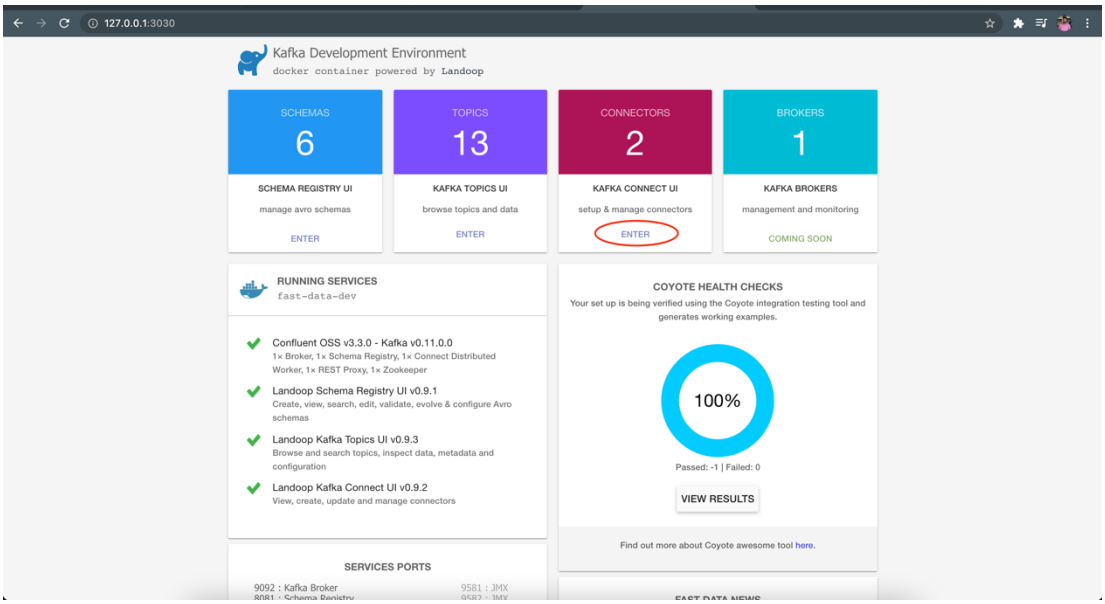
```
# Basic configuration for our connector
name=source-twitter-distributed
connector.class=com.eneco.trading.kafka.connect.twitter.TwitterSourceConnector
tasks.max=1
topic=demo-3-twitter
key.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=true
# Twitter connector specific configuration
twitter.consumerkey=gPCOPTOR7lGYH7GkFyNY8QqxC
twitter.consumersecret=nSgzgjNMfFqG4SOurlb2vKkgI6mYHj3zZUf0caoN0Kb71o0RuS
twitter.token=1434035052732583938-y8gN6T6fqHTN1Yg8XGnl8GsOUVKgLV
twitter.secret=MqcWZnKcvYojiSadpH6J8JQGLLBnHIKnoVxnneJmSgBNu
track.terms=programming,java,kafka,scala
language=en
```

Step 3: Create a topic named “demo-3-twitter”

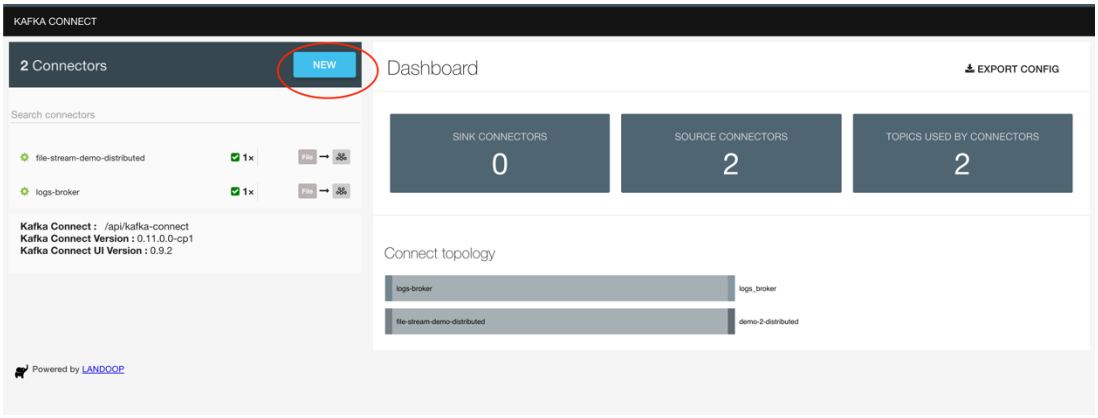
```
→  code3 docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash

root@fast-data-dev / $ kafka-topics --create --topic demo-3-twitter --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181
Created topic "demo-3-twitter".
```

Step 4: Create a new Twitter connector in Landoop UI.

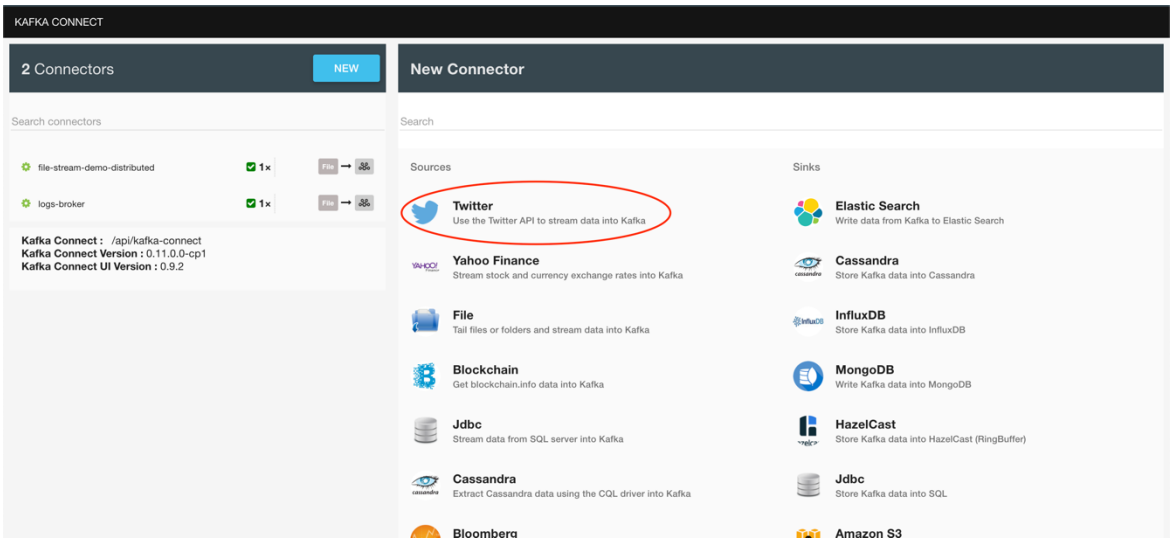
In Landoop main page, click ‘ENTER’ in the Connector card.



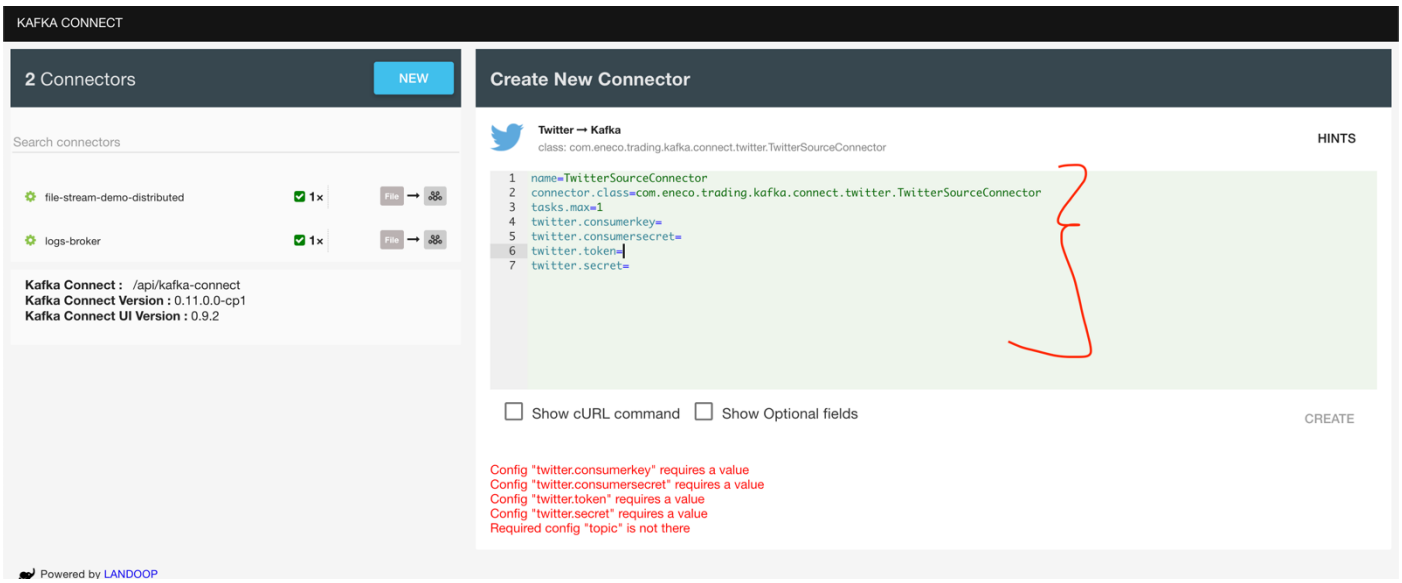
Click on ‘NEW’.



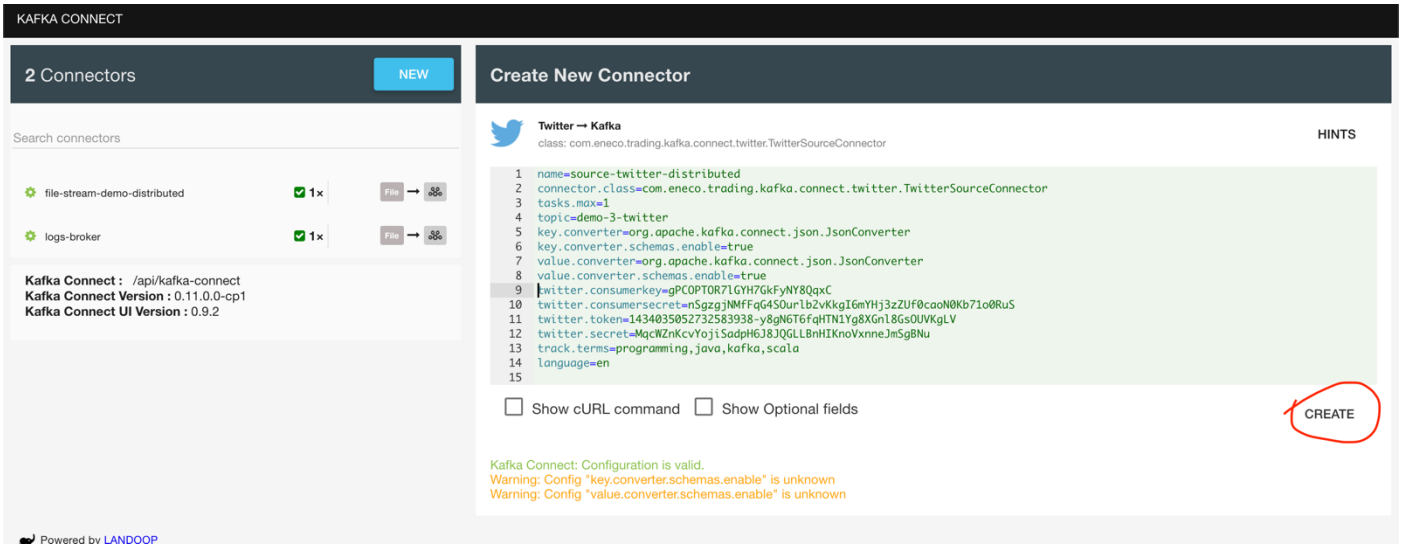
Click on ‘Twitter’.



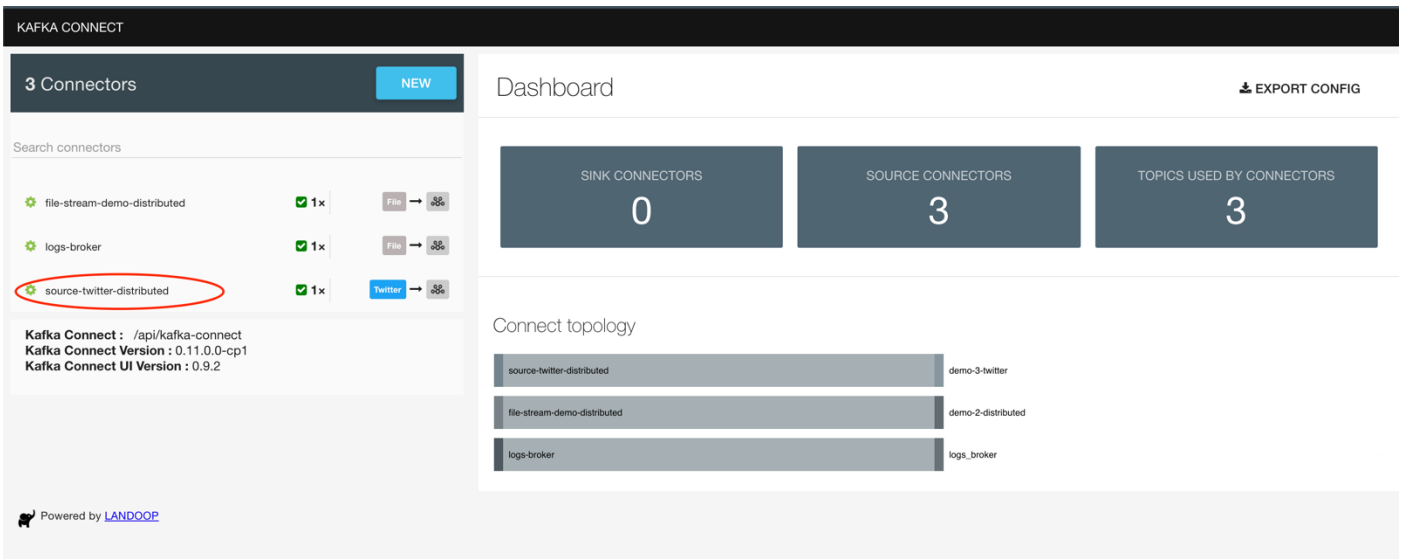
Paste the properties created in the step 2.



Click on ‘CREATE’.

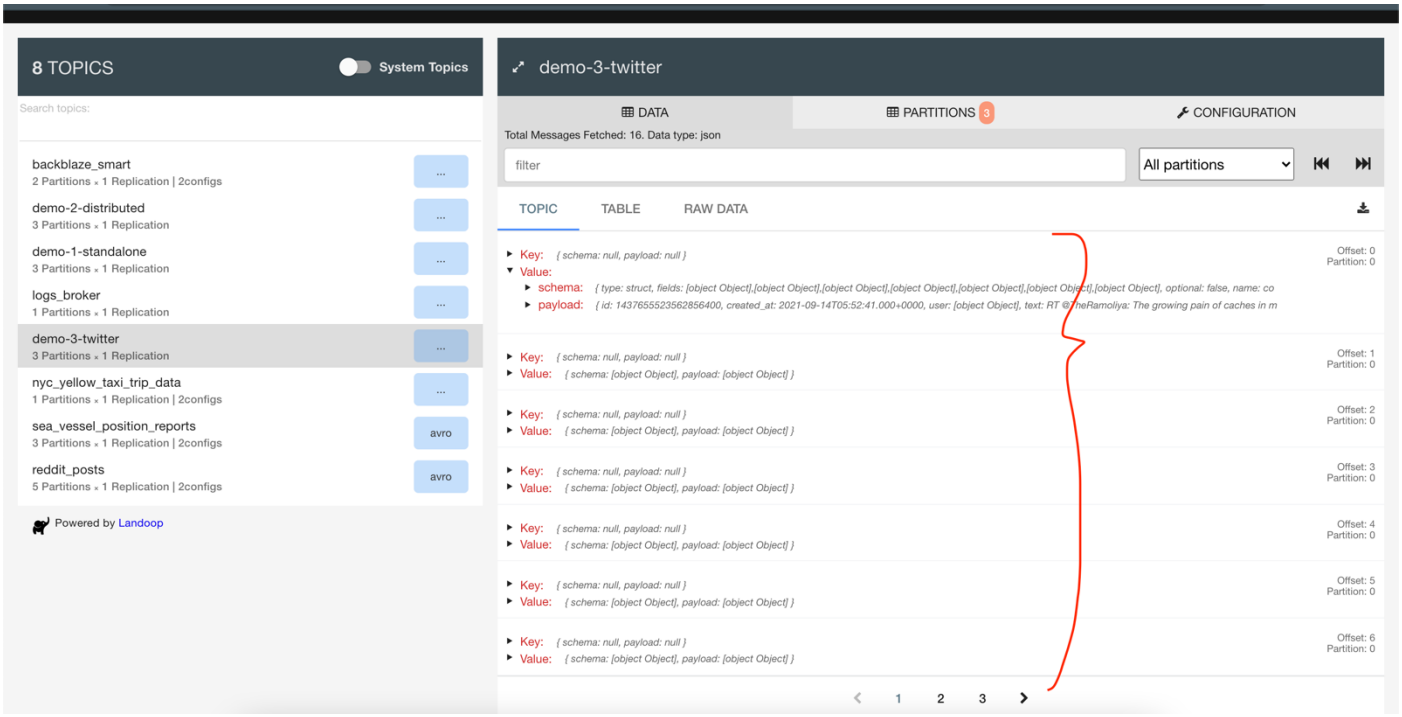


Now connector is created.



Step 5: Checking for topic in UI.

As we can see, we are getting the data from twitter to the specified topic.



What we learned

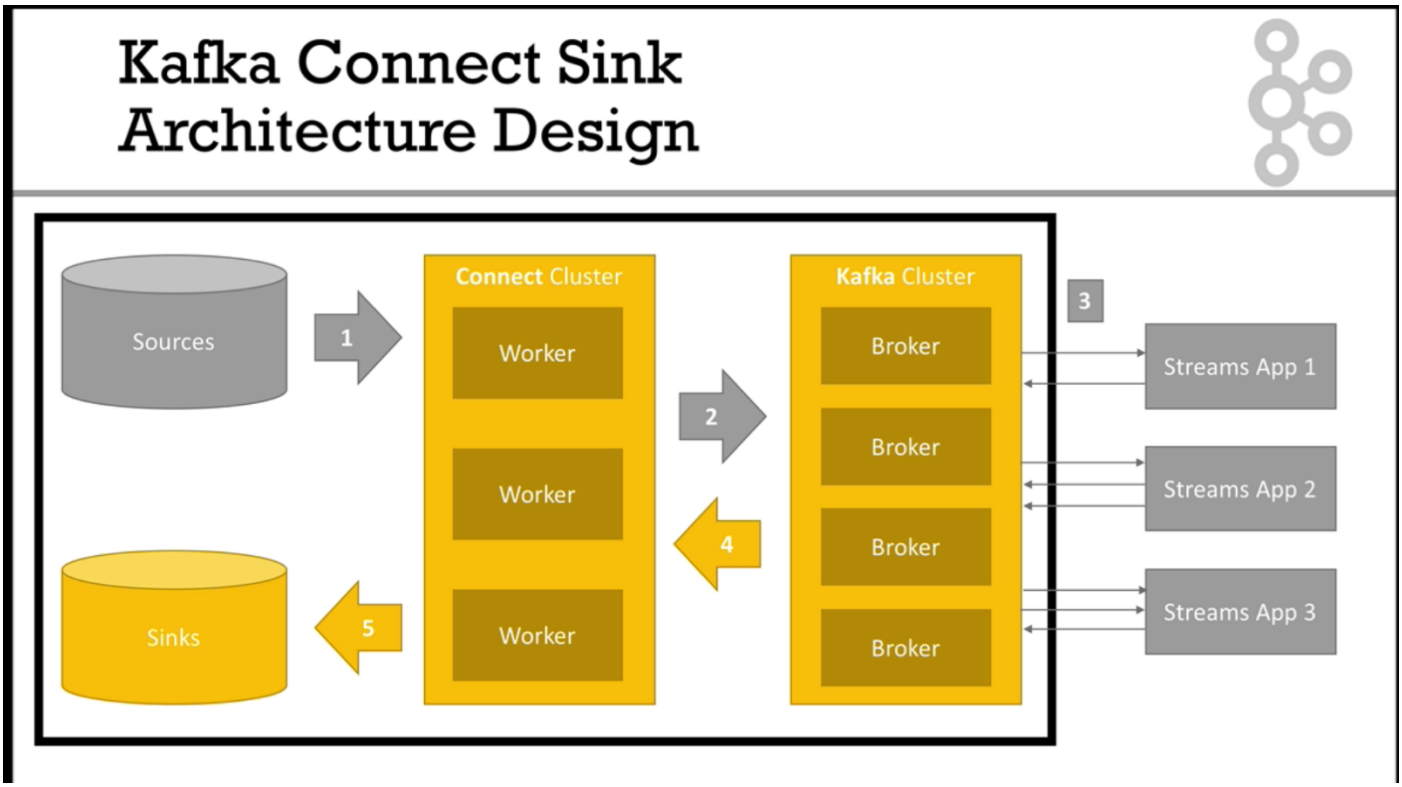
What we learned

- How to run Kafka connectors in
  - Standalone mode (good for development)
  - Distributed mode (good for production)
- Where to find Kafka connectors
- How to pull data from Twitter directly into a Kafka topic

- Now, you can go online and learn how to use other connectors!

Kafka Connector Sink

Architecture



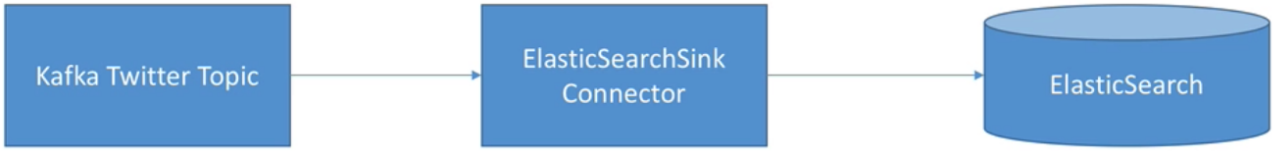
Elastic Search Sink Connector

- 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

Example: ElasticSearchSink
DISTRIBUTED MODE

- 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

Step 1: Running Docker File

docker-compose.yml

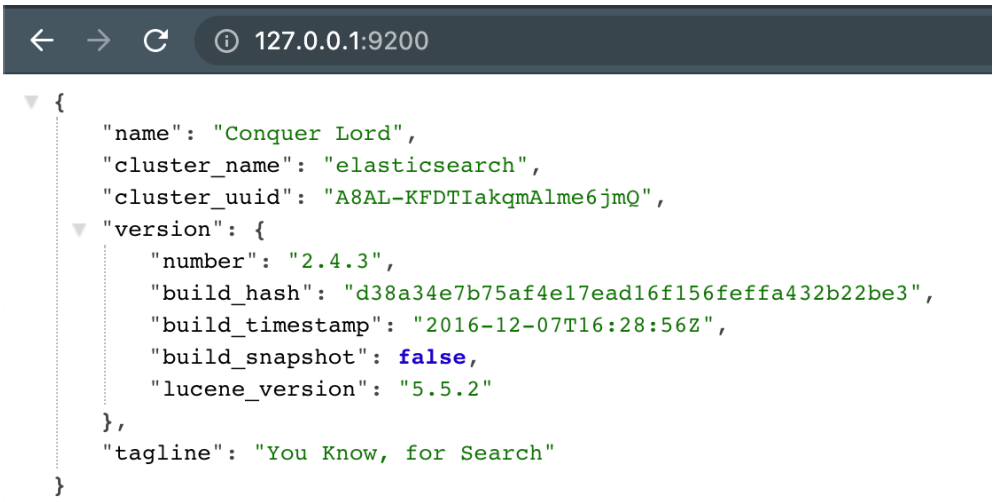
```yaml
version: '2'

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

```

docker-compose up kafka-cluster elasticsearch postgres

```
➜  code3 docker-compose up kafka-cluster elasticsearch postgres
Pulling elasticsearch (itzg/elasticsearch:2.4.3)...
2.4.3: Pulling from itzg/elasticsearch
b7f33cc0b48e: Pull complete
43a564ae36a3: Pull complete
efb75a810eee: Downloading [==========>                                        ]  8.936MB/39.67MB
b0aeda70fb4d: Downloading [=======>                                           ]  4.172MB/27.35MB
d99b5af0dd89: Downloading [==>                                                ]  1.407MB/27.34MB
8c3b2ac50d38: Waiting
8e45dd41801a: Waiting
```

Step 2: Checking whether ElasticSearch working fine or not.

http://127.0.0.1:9200/

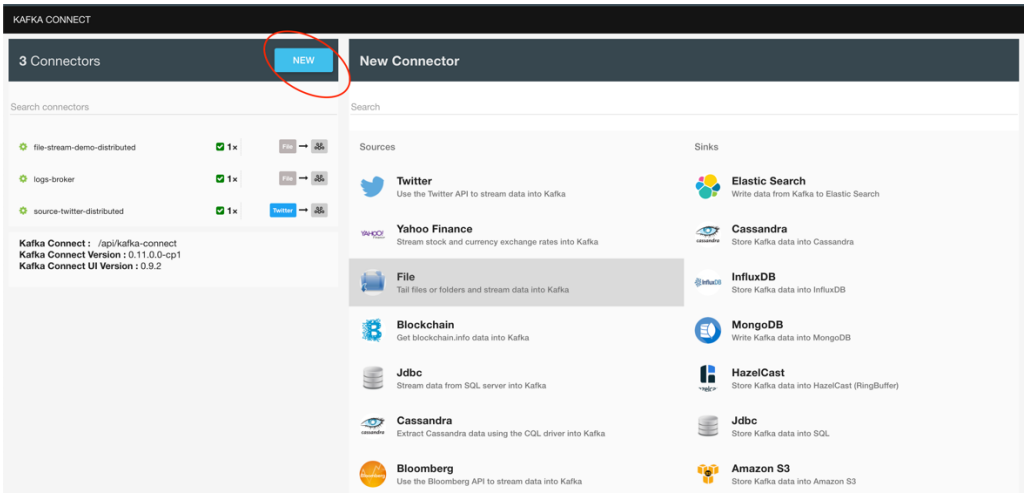


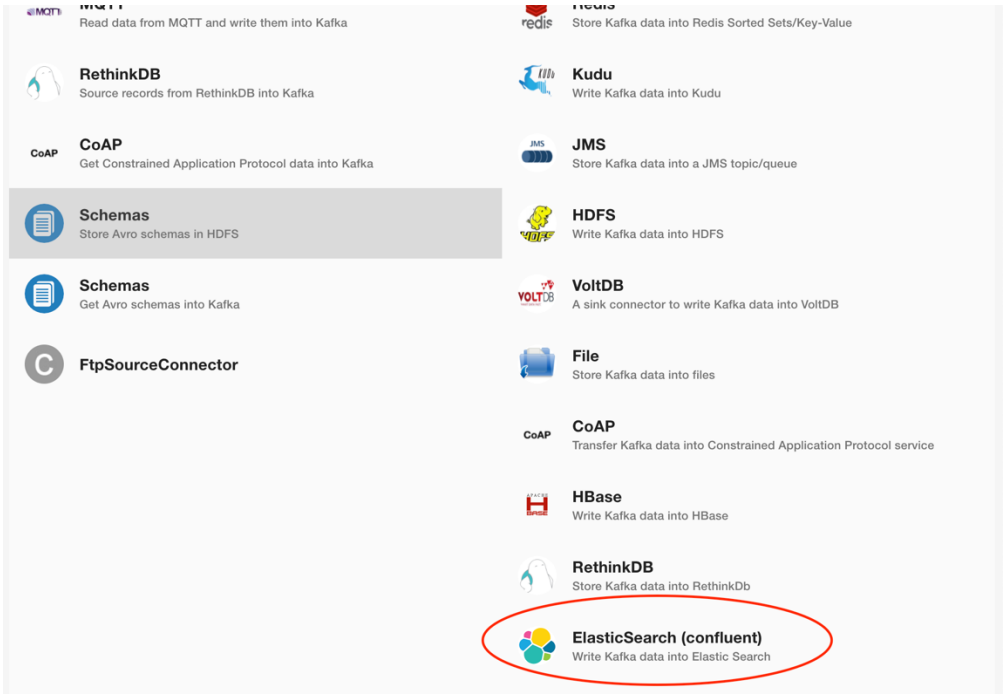
Step 3: Prepare properties for Elastic Search Sink Connector

sink > demo-elasticsearch > sink-elastic-twitter-distributed.properties

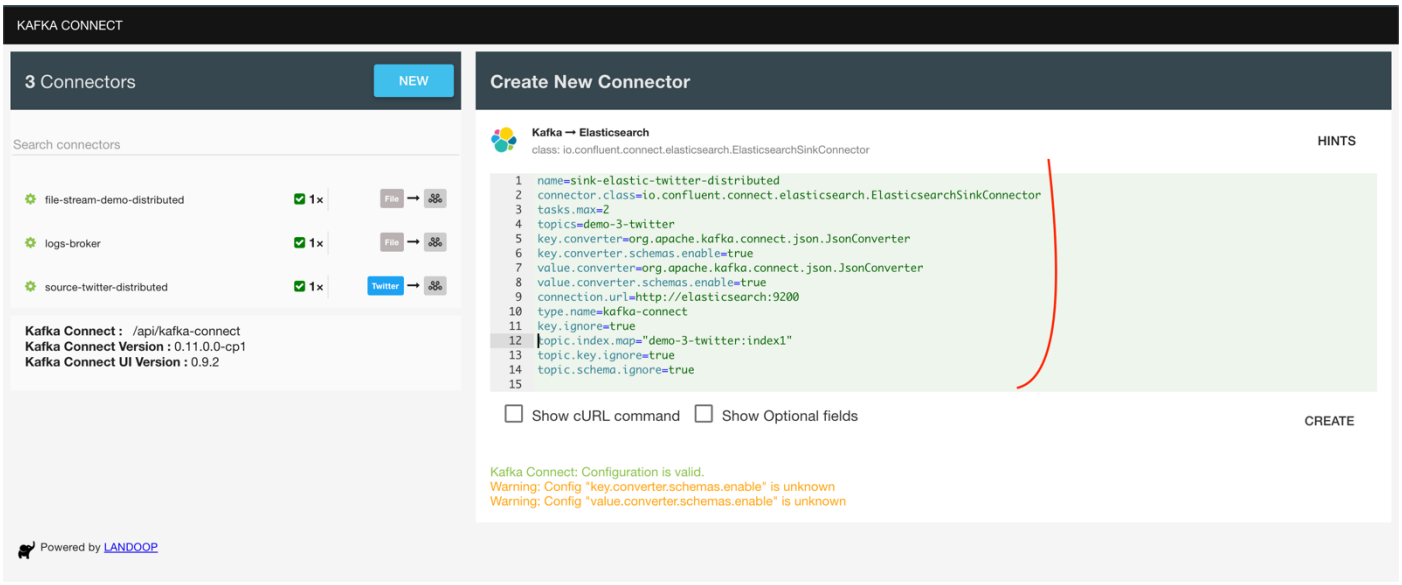
```
# Basic configuration for our connector
name=sink-elastic-twitter-distributed
connector.class=io.confluent.connect.elasticsearch.ElasticsearchSinkConnector
# We can have parallelism here so we have two tasks!
tasks.max=2
topics=demo-3-twitter
# the input topic has a schema, so we enable schemas conversion 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/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
# 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
```

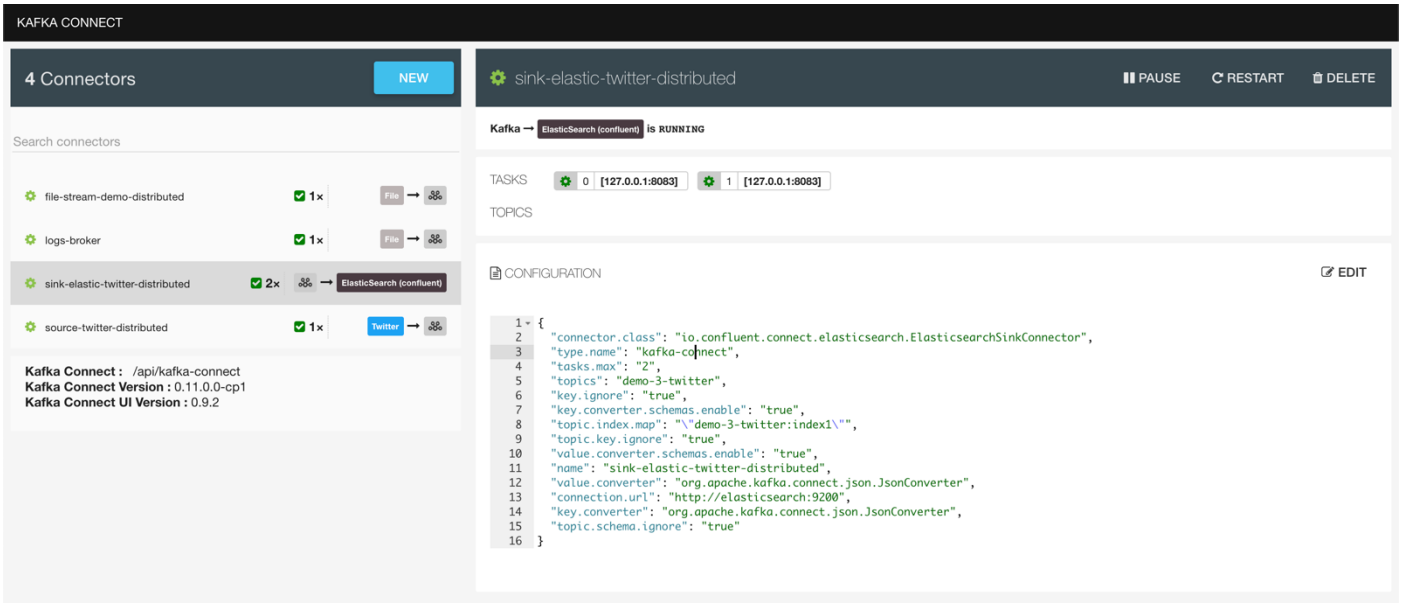
Step 4: Create a new Elastic Search Sink Connector in landoop UI.





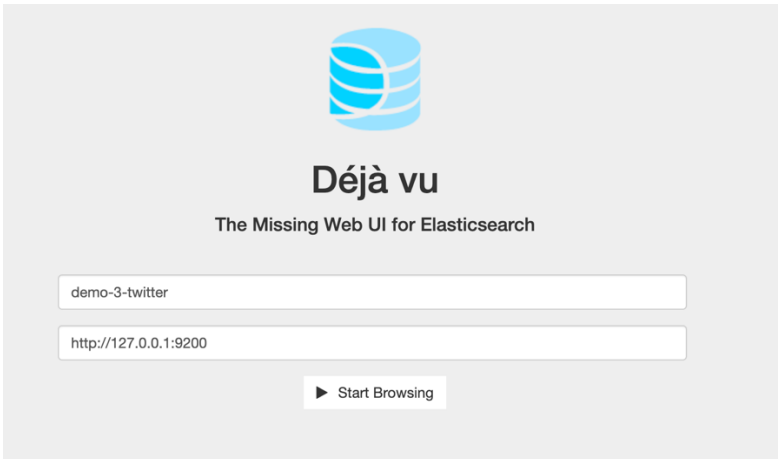
Paste the properties created in step 3. And click on click on create.

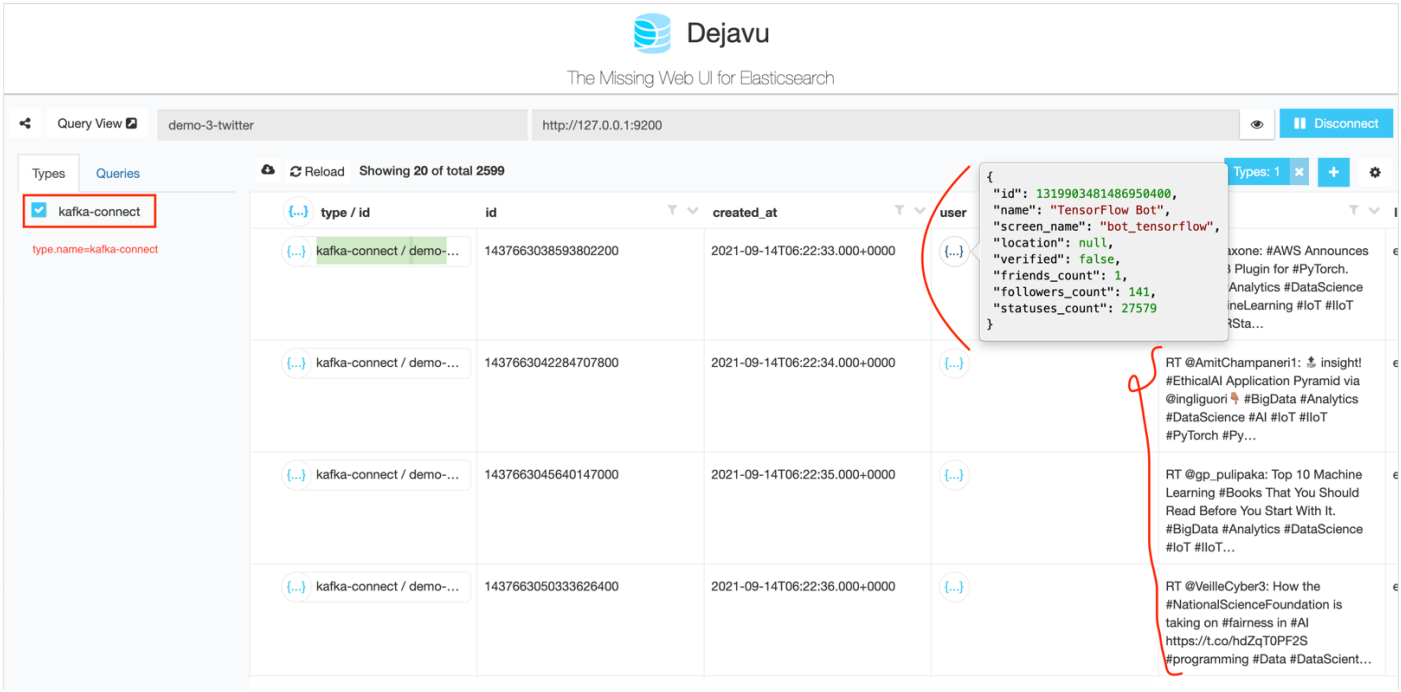




Step 5: Checking twitter data in Elasticsearch UI.

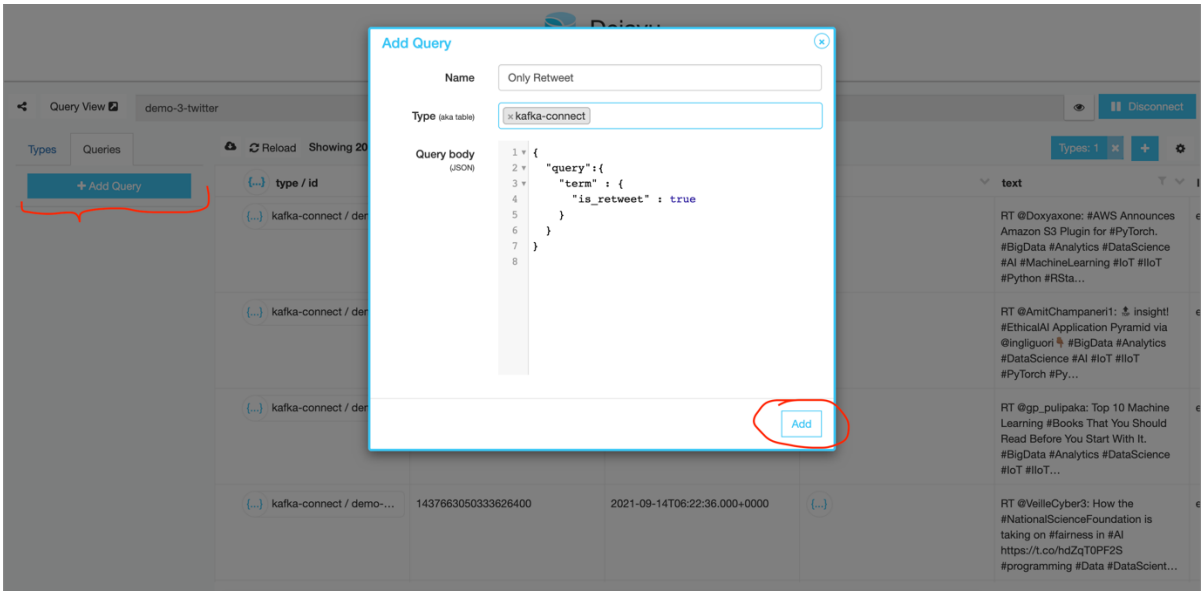
http://127.0.0.1:9200/_plugin/dejavu/

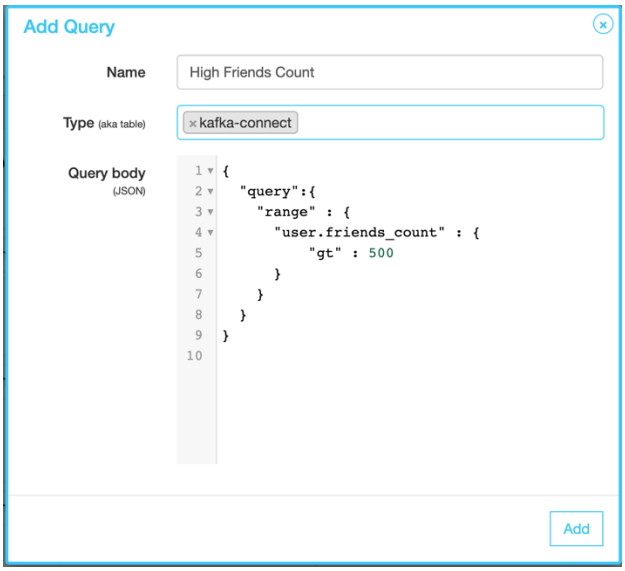


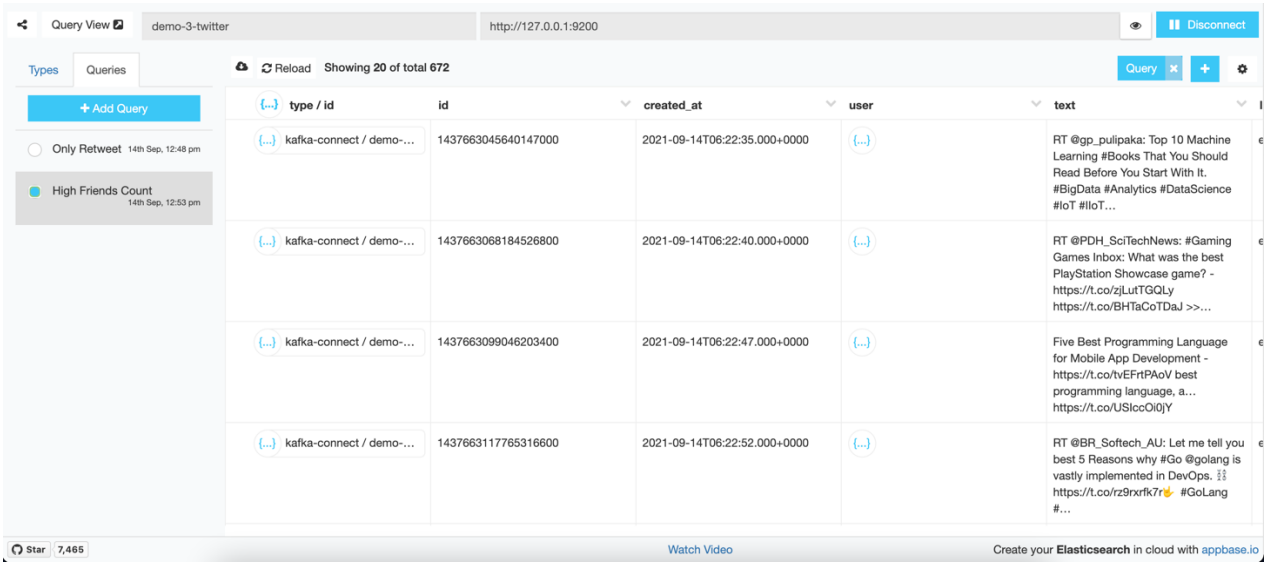


Step 6: Querying from Dejavu UI.

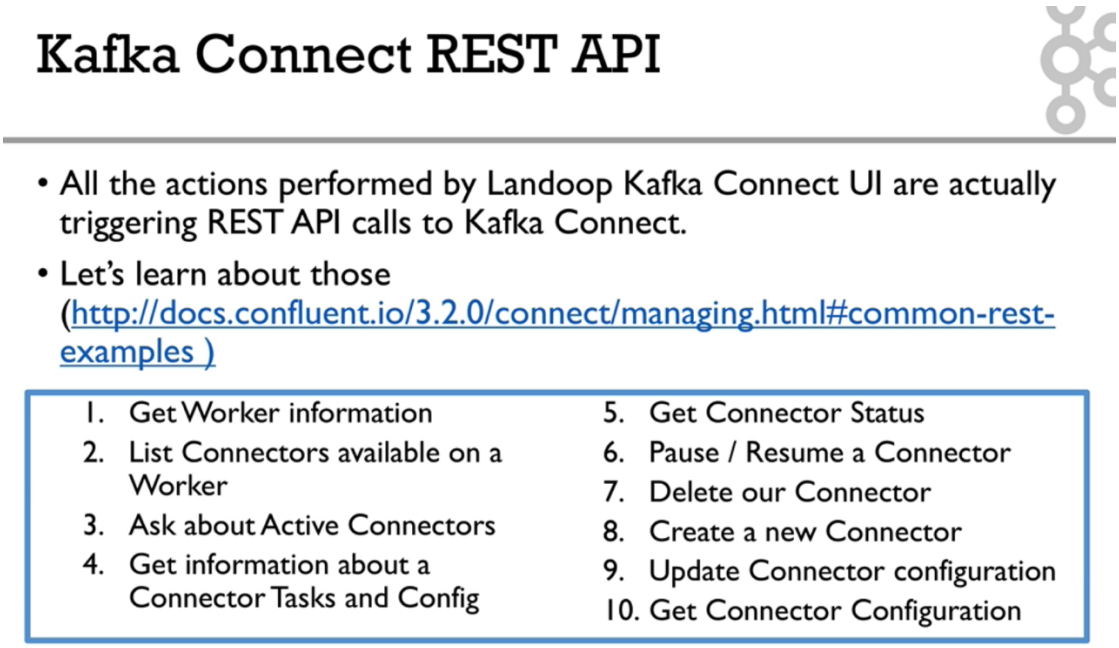
Creating a query.







Kafka Connect RESP API



Setting up the command line

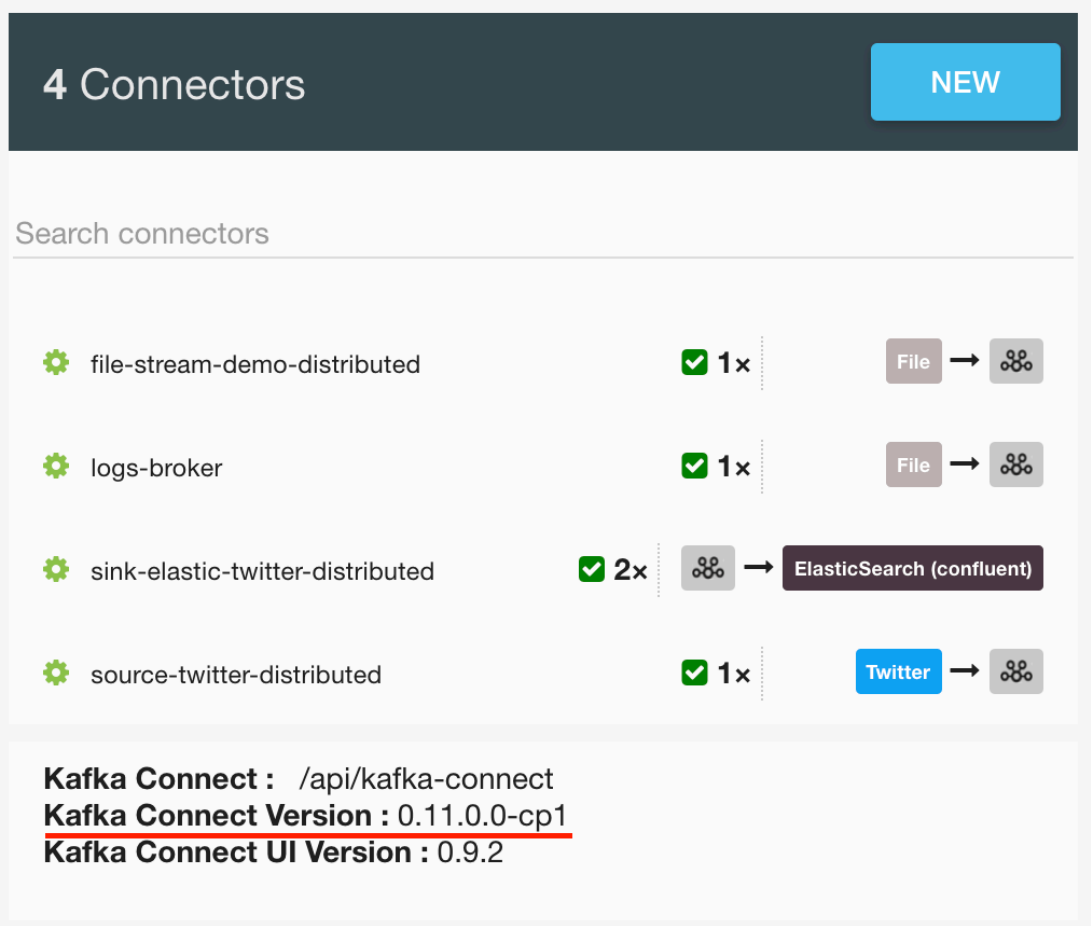
let's start a command line to all have linux commands
docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
Install jq to pretty print json
apk update && apk add jq

```
→  code3 docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
root@fast-data-dev / $ apk update && apk add jq
fetch http://dl-cdn.alpinelinux.org/alpine/v3.6/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.6/community/x86_64/APKINDEX.tar.gz
v3.6.5-44-gda55e27396 [http://dl-cdn.alpinelinux.org/alpine/v3.6/main]
v3.6.5-34-gf0ba0b43d5 [http://dl-cdn.alpinelinux.org/alpine/v3.6/community]
OK: 8468 distinct packages available
(1/2) Installing oniguruma (6.3.0-r0)
(2/2) Installing jq (1.5-r4)
Executing busybox-1.26.2-r7.trigger
OK: 140 MiB in 63 packages
root@fast-data-dev / $
```

Get Worker information

curl -s 127.0.0.1:8083/ | jq

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/ | jq
{
  "version": "0.11.0.0-cp1",
  "commit": "5cadaa94d0a69e0d"
}
root@fast-data-dev / $
```

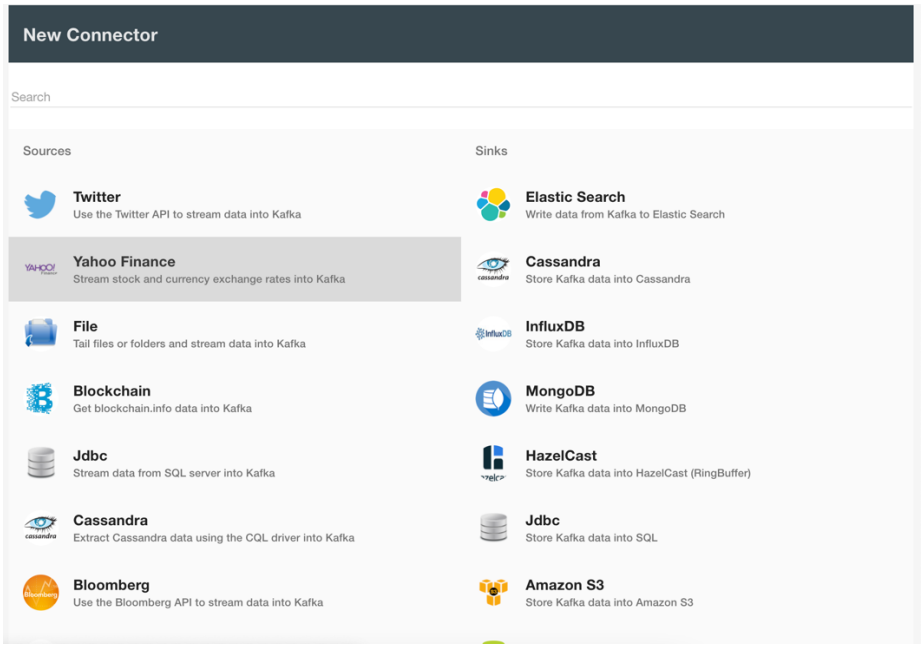


List Connectors available on a Worker

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

It will list all the connectors.

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connector-plugins | jq
[
  {
    "class": "com.datamountaineer.streamreactor.connect.azure.documentdb.sink.DocumentDbSinkConnector",
    "type": "unknown",
    "version": "null"
  },
  {
    "class": "com.datamountaineer.streamreactor.connect.blockchain.source.BlockchainSourceConnector",
    "type": "source",
    "version": "null"
  },
  {
    "class": "com.datamountaineer.streamreactor.connect.bloomberg.BloombergSourceConnector",
    "type": "source",
    "version": "null"
  },
  {
    "class": "com.datamountaineer.streamreactor.connect.cassandra.sink.CassandraSinkConnector",
    "type": "unknown",
    "version": "1"
  },
  {
    "class": "com.datamountaineer.streamreactor.connect.cassandra.source.CassandraSourceConnector",
    "type": "source",
    "version": "null"
  },
  {
    "class": "com.datamountaineer.streamreactor.connect.coap.sink.CoapSinkConnector",
    "type": "sink",
    "version": "null"
  },
  {
    "class": "com.datamountaineer.streamreactor.connect.coap.source.CoapSourceConnector",
    "type": "source",
    "version": "null"
  },
  {
    "class": "com.datamountaineer.streamreactor.connect.druid.DruidSinkConnector",
    "type": "sink",
    "version": "null"
  },
  {
```



Ask about Active Connectors

curl -s 127.0.0.1:8083/connectors | jq

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors | jq
[
  "sink-elastic-twitter-distributed",
  "source-twitter-distributed",
  "logs-broker",
  "file-stream-demo-distributed"
]
```

Get information about a Connector Tasks and Config

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

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/source-twitter-distributed/tasks | jq
[
  {
    "id": {
      "connector": "source-twitter-distributed",
      "task": 0
    },
    "config": {
      "connector.class": "com.eneco.trading.kafka.connect.twitter.TwitterSourceConnector",
      "twitter.token": "1434035052732583938-y8gN6T6fqHTN1Yg8XGnl8GsOUVKgLV",
      "tasks.max": "1",
      "twitter.secret": "MqcWZnKcvYojiSadpH6J8JQGLLBnHIKnoVxnneJmSgBNu",
      "track.terms": "programming,java,kafka,scala",
      "language": "en",
      "key.converter.schemas.enable": "true",
      "task.class": "com.eneco.trading.kafka.connect.twitter.TwitterSourceTask",
      "value.converter.schemas.enable": "true",
      "name": "source-twitter-distributed",
      "topic": "demo-3-twitter",
      "twitter.consumersecret": "nSgzgjNMfFqG4SOurlb2vKkgI6mYHj3zZUf0caoN0Kb71o0RuS",
      "value.converter": "org.apache.kafka.connect.json.JsonConverter",
      "key.converter": "org.apache.kafka.connect.json.JsonConverter",
      "twitter.consumerkey": "gPCOPTOR7lGYH7GkFyNY8QqxC"
    }
  }
]
```

Get Connector Status

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

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed/status | jq
{
  "name": "file-stream-demo-distributed",
  "connector": {
    "state": "RUNNING",
    "worker_id": "127.0.0.1:8083"
  },
  "tasks": [
    {
      "state": "RUNNING",
      "id": 0,
      "worker_id": "127.0.0.1:8083"
    }
  ]
}
root@fast-data-dev / $
```

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

```
root@fast-data-dev / $ curl -s -X PUT 127.0.0.1:8083/connectors/file-stream-demo-distributed/pause
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed/status | jq
{
  "name": "file-stream-demo-distributed",
  "connector": {
    "state": "PAUSED",
    "worker_id": "127.0.0.1:8083"
  },
  "tasks": [
    {
      "state": "PAUSED",
      "id": 0,
      "worker_id": "127.0.0.1:8083"
    }
  ]
}
root@fast-data-dev / $
```

```
root@fast-data-dev / $ curl -s -X PUT 127.0.0.1:8083/connectors/file-stream-demo-distributed/resume
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed/status | jq
{
  "name": "file-stream-demo-distributed",
  "connector": {
    "state": "RUNNING",
    "worker_id": "127.0.0.1:8083"
  },
  "tasks": [
    {
      "state": "RUNNING",
      "id": 0,
      "worker_id": "127.0.0.1:8083"
    }
  ]
}
root@fast-data-dev / $
```

Get Connector Configuration

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

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed | jq
{
  "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.connect.json.JsonConverter"
  },
  "tasks": [
    {
      "connector": "file-stream-demo-distributed",
      "task": 0
    }
  ]
}
root@fast-data-dev / $
```

Delete our Connector

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

```
root@fast-data-dev / $ curl -s -X DELETE 127.0.0.1:8083/connectors/file-stream-demo-distributed
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed | jq
{
  "error_code": 404,
  "message": "Connector file-stream-demo-distributed not found"
}
root@fast-data-dev / $
```

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.connect.json.JsonConverter"}}' http://127.0.0.1:8083/connectors | jq

```
root@fast-data-dev / $ 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.connect.json.JsonConverter"}}' http://127.0.0.1:8083/connectors | jq
{
  "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.connect.json.JsonConverter"
  },
  "tasks": [
    {
      "connector": "file-stream-demo-distributed",
      "task": 0
    }
  ]
}
```

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.connect.json.JsonConverter"}' 127.0.0.1:8083/connectors/file-stream-demo-distributed/config | jq

```
root@fast-data-dev / $ 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.connect.json.JsonConverter"}' 127.0.0.1:8083/connectors/file-stream-demo-distributed/config | jq
{
  "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": "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.connect.json.JsonConverter"
  },
  "tasks": [
    {
      "connector": "file-stream-demo-distributed",
      "task": 0
    }
  ]
}
root@fast-data-dev / $
```

JDBC Sink Connector

PostgreSQL

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

Distributed Mode

Example: JDBCSinkConnector
DISTRIBUTED MODE

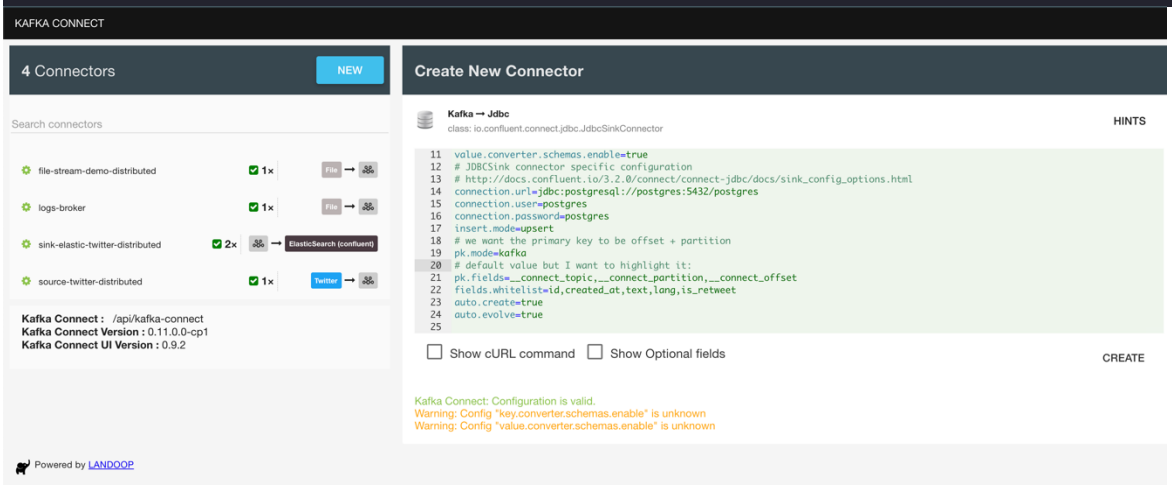
- Goal:
  - Start an PostgreSQL instance using Docker
  - Run in distributed mode with multiple tasks

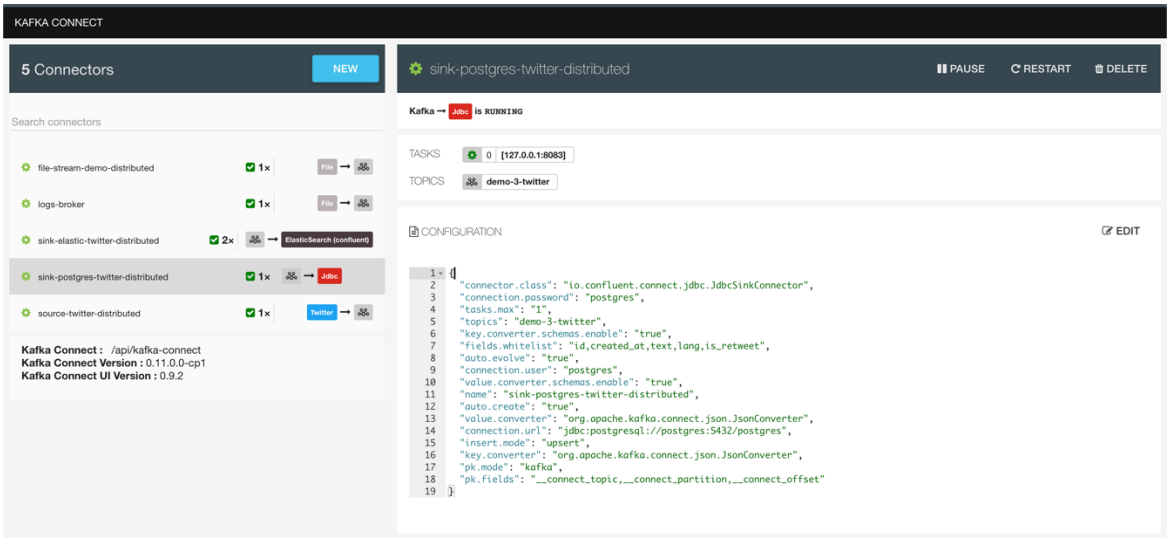
- Learning:
  - Learn about the JDBC Sink Connector

Step 1: Creating a JDBCSinkConnector in landoop UI.

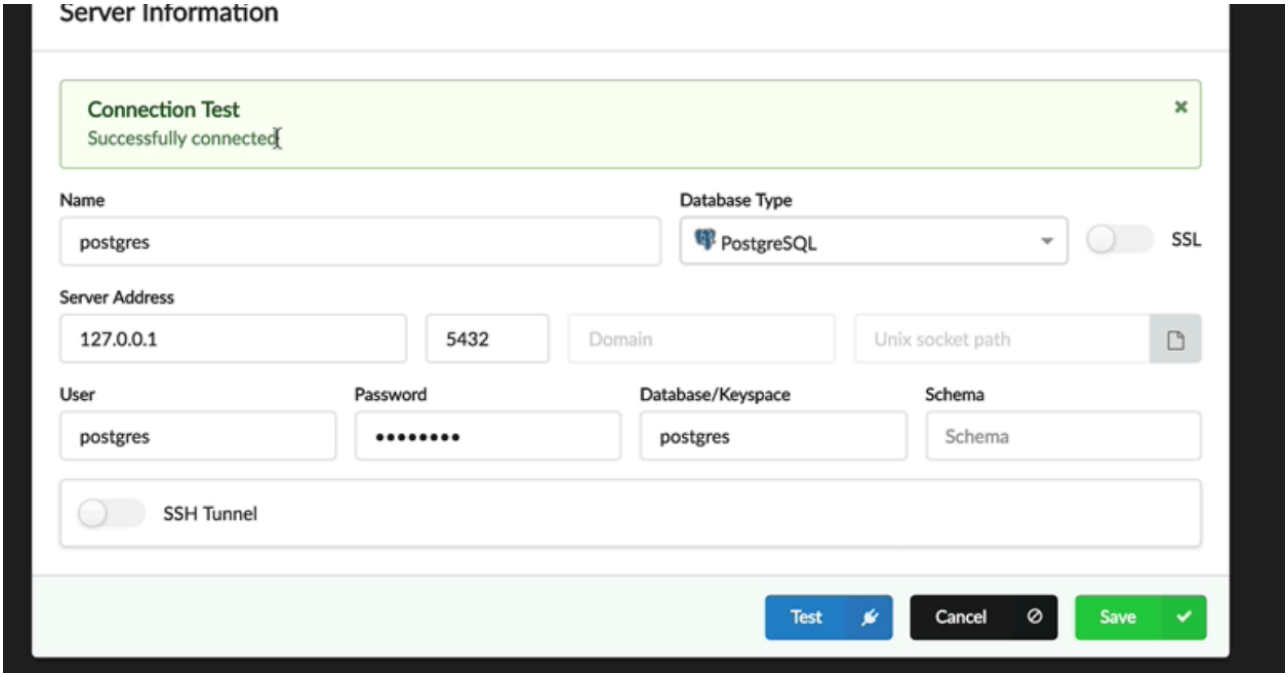
sink > demo-postgres > sink-postgres-twitter-distributed.properties

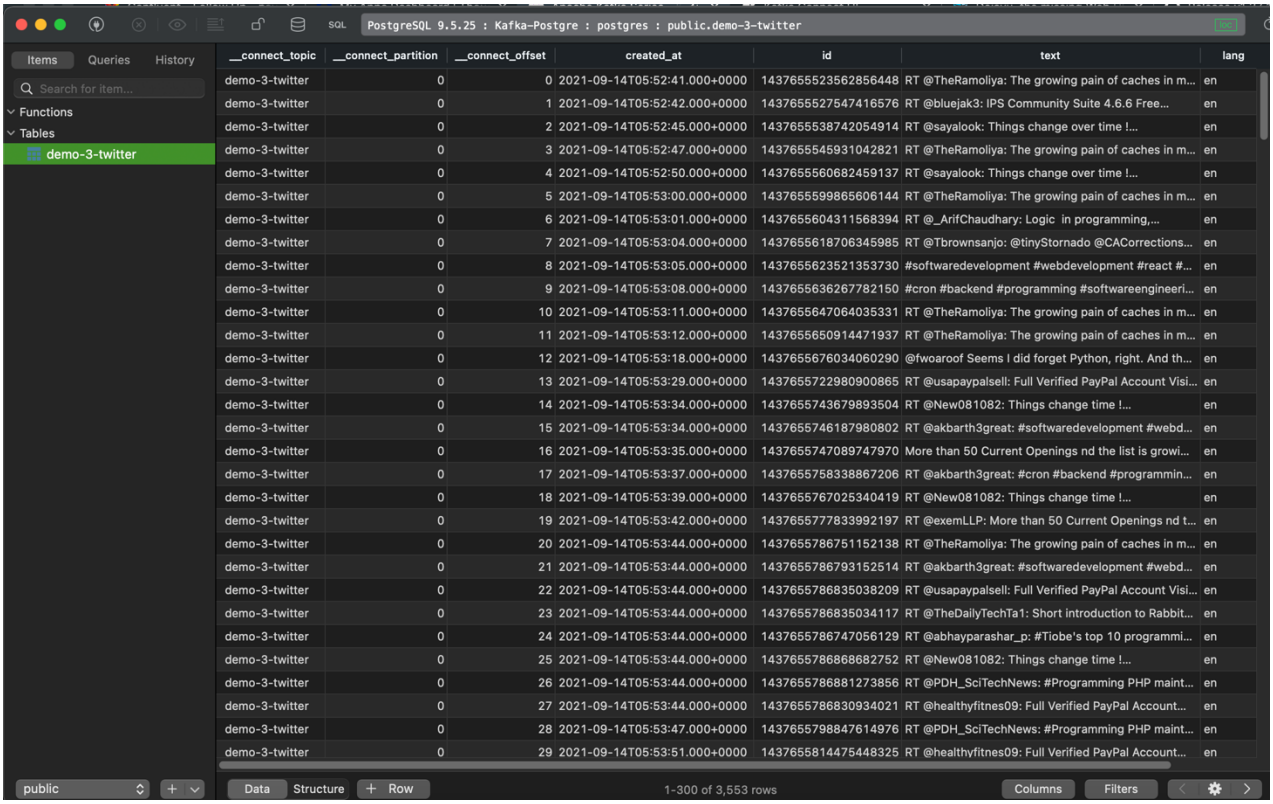
```
# Basic configuration for our connector
name=sink-postgres-twitter-distributed
connector.class=io.confluent.connect.jdbc.JdbcSinkConnector
# We can have parallelism here so we have two tasks!
tasks.max=1
topics=demo-3-twitter
# the input topic has a schema, so we enable schemas conversion here too
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
```





Step 2: Checking the twitter data in PostgreSQL.





__connect_topic	__connect_partition	__connect_offset	created_at	id	text	lang
demo-3-twitter	0	0	2021-09-14T05:52:41.000+0000	1437655523562856448	RT @TheRamoliya: The growing pain of caches in m...	en
demo-3-twitter	0	1	2021-09-14T05:52:42.000+0000	1437655527547416576	RT @bluejak3: IPS Community Suite 4.6.6 Free...	en
demo-3-twitter	0	2	2021-09-14T05:52:45.000+0000	1437655538742054914	RT @sayalook: Things change over time !...	en
demo-3-twitter	0	3	2021-09-14T05:52:47.000+0000	1437655545931042821	RT @TheRamoliya: The growing pain of caches in m...	en
demo-3-twitter	0	4	2021-09-14T05:52:50.000+0000	1437655560682459137	RT @sayalook: Things change over time !...	en
demo-3-twitter	0	5	2021-09-14T05:53:00.000+0000	1437655599865606144	RT @TheRamoliya: The growing pain of caches in m...	en
demo-3-twitter	0	6	2021-09-14T05:53:01.000+0000	1437655604311568394	RT @_ArifChaudhary: Logic in programming,...	en
demo-3-twitter	0	7	2021-09-14T05:53:04.000+0000	1437655618706345985	RT @Tbrownsanjo: @tinyStornado @CACorrections...	en
demo-3-twitter	0	8	2021-09-14T05:53:05.000+0000	1437655623521353730	#softwaredevelopment #webdevelopment #react #...	en
demo-3-twitter	0	9	2021-09-14T05:53:08.000+0000	1437655636267782150	#cron #backend #programming #softwareengineeri...	en
demo-3-twitter	0	10	2021-09-14T05:53:11.000+0000	1437655647064035331	RT @TheRamoliya: The growing pain of caches in m...	en
demo-3-twitter	0	11	2021-09-14T05:53:12.000+0000	1437655650914471937	RT @TheRamoliya: The growing pain of caches in m...	en
demo-3-twitter	0	12	2021-09-14T05:53:18.000+0000	1437655676034060290	@fwoaroof Seems I did forget Python, right. And th...	en
demo-3-twitter	0	13	2021-09-14T05:53:29.000+0000	1437655722980900865	RT @usapaypalsell: Full Verified PayPal Account Visi...	en
demo-3-twitter	0	14	2021-09-14T05:53:34.000+0000	1437655743679893504	RT @New081082: Things change time !...	en
demo-3-twitter	0	15	2021-09-14T05:53:34.000+0000	1437655746187980802	RT @akbarth3great: #softwaredevelopment #webd...	en
demo-3-twitter	0	16	2021-09-14T05:53:35.000+0000	1437655747089747970	More than 50 Current Openings nd the list is growi...	en
demo-3-twitter	0	17	2021-09-14T05:53:37.000+0000	1437655758338867206	RT @akbarth3great: #cron #backend #programmin...	en
demo-3-twitter	0	18	2021-09-14T05:53:39.000+0000	1437655767025340419	RT @New081082: Things change time !...	en
demo-3-twitter	0	19	2021-09-14T05:53:42.000+0000	1437655777833992197	RT @exemLLP: More than 50 Current Openings nd t...	en
demo-3-twitter	0	20	2021-09-14T05:53:44.000+0000	1437655786751152138	RT @TheRamoliya: The growing pain of caches in m...	en
demo-3-twitter	0	21	2021-09-14T05:53:44.000+0000	1437655786793152514	RT @akbarth3great: #softwaredevelopment #webd...	en
demo-3-twitter	0	22	2021-09-14T05:53:44.000+0000	1437655786835038209	RT @usapaypalsell: Full Verified PayPal Account Visi...	en
demo-3-twitter	0	23	2021-09-14T05:53:44.000+0000	1437655786835034117	RT @TheDailyTechTa1: Short introduction to Rabbit...	en
demo-3-twitter	0	24	2021-09-14T05:53:44.000+0000	1437655786747056129	RT @abhayparashar_p: #Tiobe's top 10 programmi...	en
demo-3-twitter	0	25	2021-09-14T05:53:44.000+0000	1437655786686682752	RT @New081082: Things change time !...	en
demo-3-twitter	0	26	2021-09-14T05:53:44.000+0000	1437655786881273856	RT @PDH_SciTechNews: #Programming PHP maint...	en
demo-3-twitter	0	27	2021-09-14T05:53:44.000+0000	1437655786830934021	RT @healthyfitnes09: Full Verified PayPal Account...	en
demo-3-twitter	0	28	2021-09-14T05:53:47.000+0000	1437655798847614976	RT @PDH_SciTechNews: #Programming PHP maint...	en
demo-3-twitter	0	29	2021-09-14T05:53:51.000+0000	1437655814475448325	RT @healthyfitnes09: Full Verified PayPal Account...	en

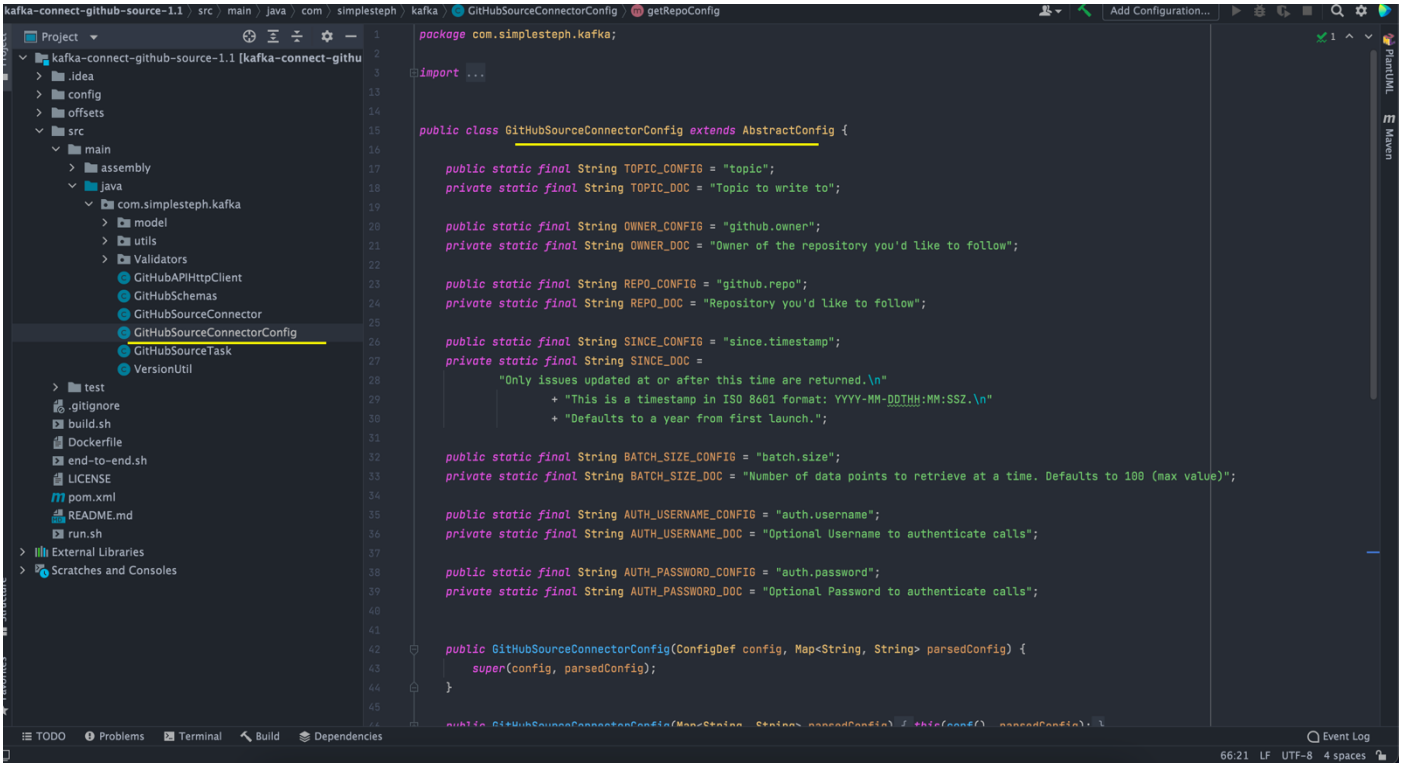
Section goals

- 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

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)
```

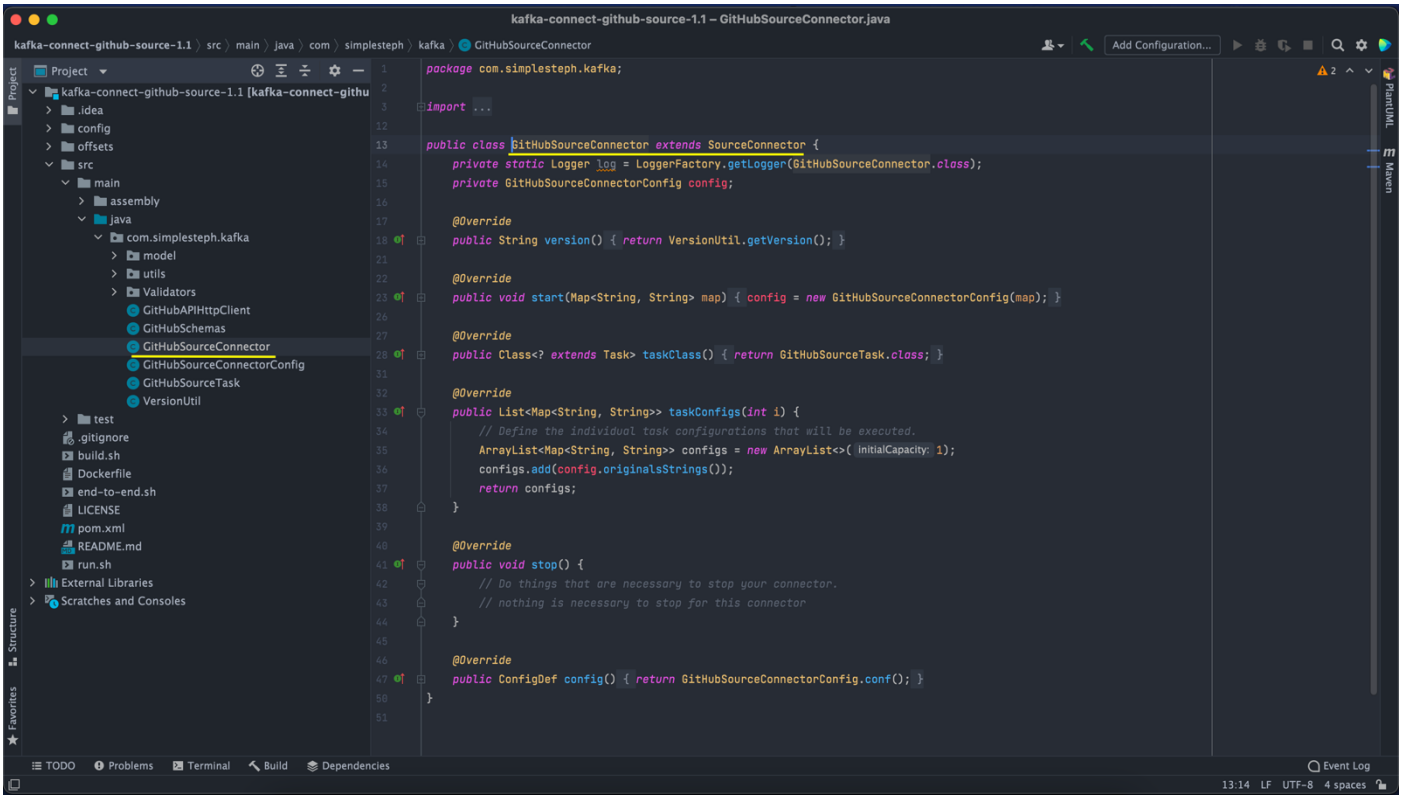


Connector

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()
```



Start() function map argument values

```
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
```

Writing a Schema

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

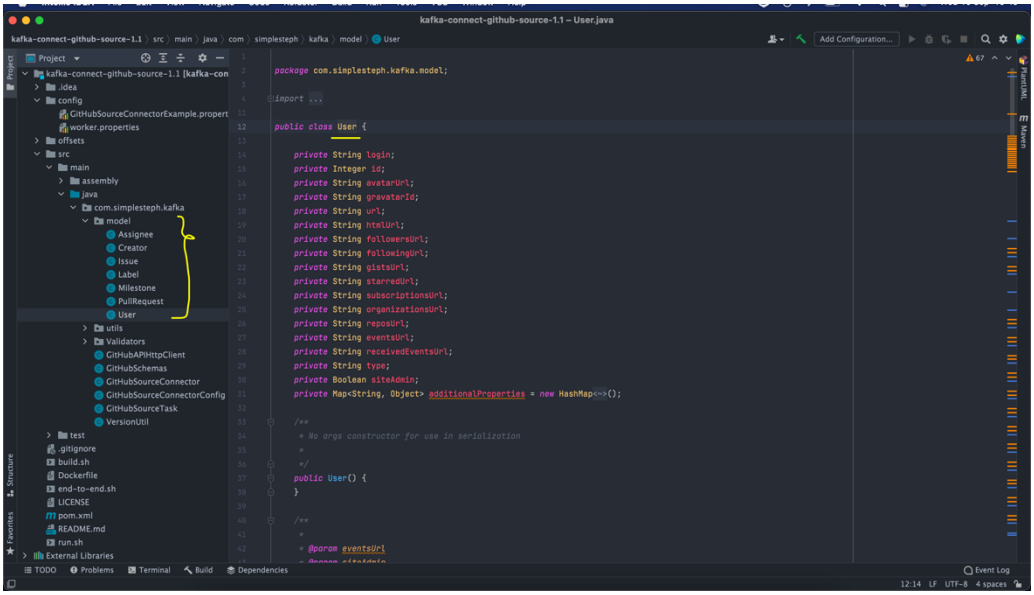
```
public static Schema USER_SCHEMA =
SchemaBuilder.struct().name(SCHEMA_VALUE_USER)
        .version(1)
        .field(USER_URL_FIELD, Schema.STRING_SCHEMA)
        .field(USER_ID_FIELD, Schema.INT32_SCHEMA)
        .field(USER_LOGIN_FIELD, Schema.STRING_SCHEMA)
        .build();
```

```
// Key Schema
public static Schema KEY_SCHEMA = SchemaBuilder.struct().name(SCHEMA_KEY)
        .version(1)
        .field(OWNER_FIELD, Schema.STRING_SCHEMA)
        .field(REPOSITORY_FIELD, Schema.STRING_SCHEMA)
        .field(NUMBER_FIELD, Schema.INT32_SCHEMA)
        .build();

// Value Schema
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();

// optional schema
public static Schema PR_SCHEMA = SchemaBuilder.struct().name(SCHEMA_VALUE_PR)
        .version(1)
        .field(PR_URL_FIELD, Schema.STRING_SCHEMA)
        .field(PR_HTML_URL_FIELD, Schema.STRING_SCHEMA)
        .optional()
        .build();

public static Schema VALUE_SCHEMA = SchemaBuilder.struct().name(SCHEMA_VALUE_ISSUE)
        .version(1)
        .field(URL_FIELD, Schema.STRING_SCHEMA)
        .field(TITLE_FIELD, Schema.STRING_SCHEMA)
        .field(CREATED_AT_FIELD, Schema.INT64_SCHEMA)
        .field(UPDATED_AT_FIELD, Schema.INT64_SCHEMA)
        .field(NUMBER_FIELD, Schema.INT32_SCHEMA)
        .field(STATE_FIELD, Schema.STRING_SCHEMA)
        .field(USER_FIELD, USER_SCHEMA) // mandatory
        .field(PR_FIELD, PR_SCHEMA)     // optional
        .build();
```

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;
}
```

UserTest.java

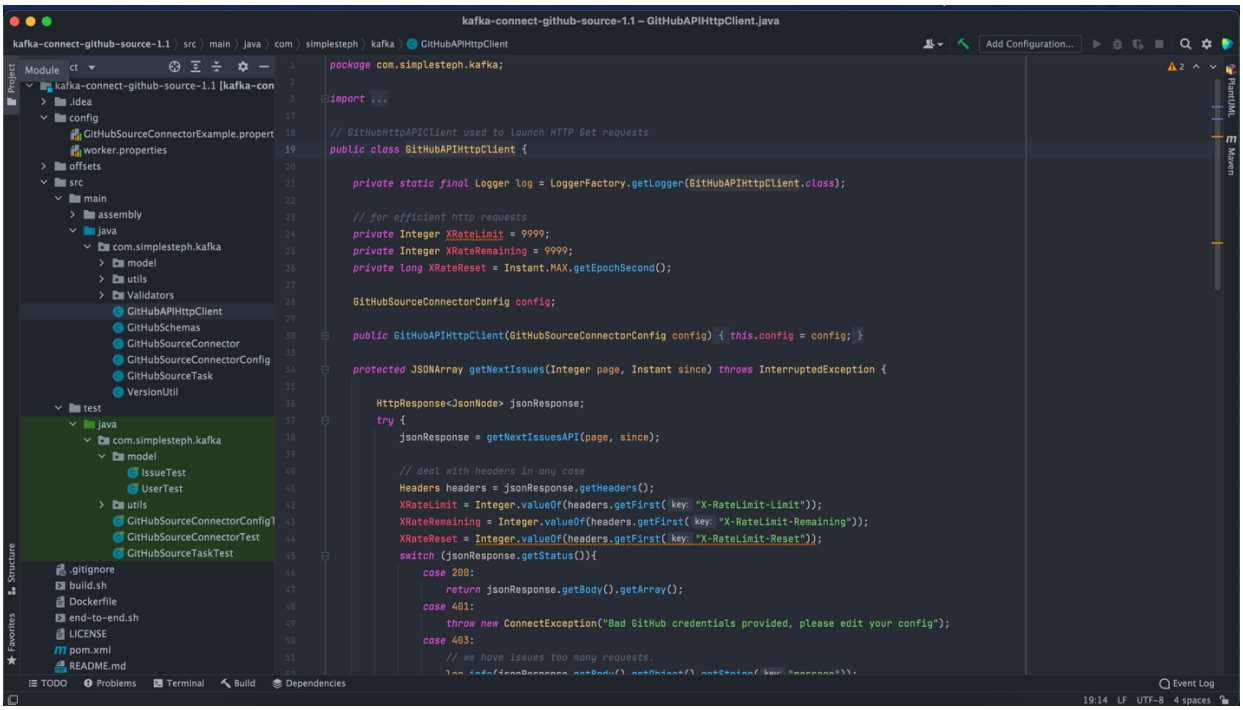
```java
@Test
public void canParseJson(){
    User user = User.fromJson(userJson);
    assertEquals(user.getId(),(Integer) 20851561);
    assertEquals(user.getUrl(), actual: "https://api.github.com/users/simplesteph");
    assertEquals(user.getLogin(), actual: "simplesteph");
}
```

Getting Data from Github API

Getting Data from GitHub API

- I query the GitHub API using the Unirest Library, that we wrap in our own client
- I implemented the concepts of
  - Dynamic URL
  - Pagination
  - Rate Throttling
  - Error Handling

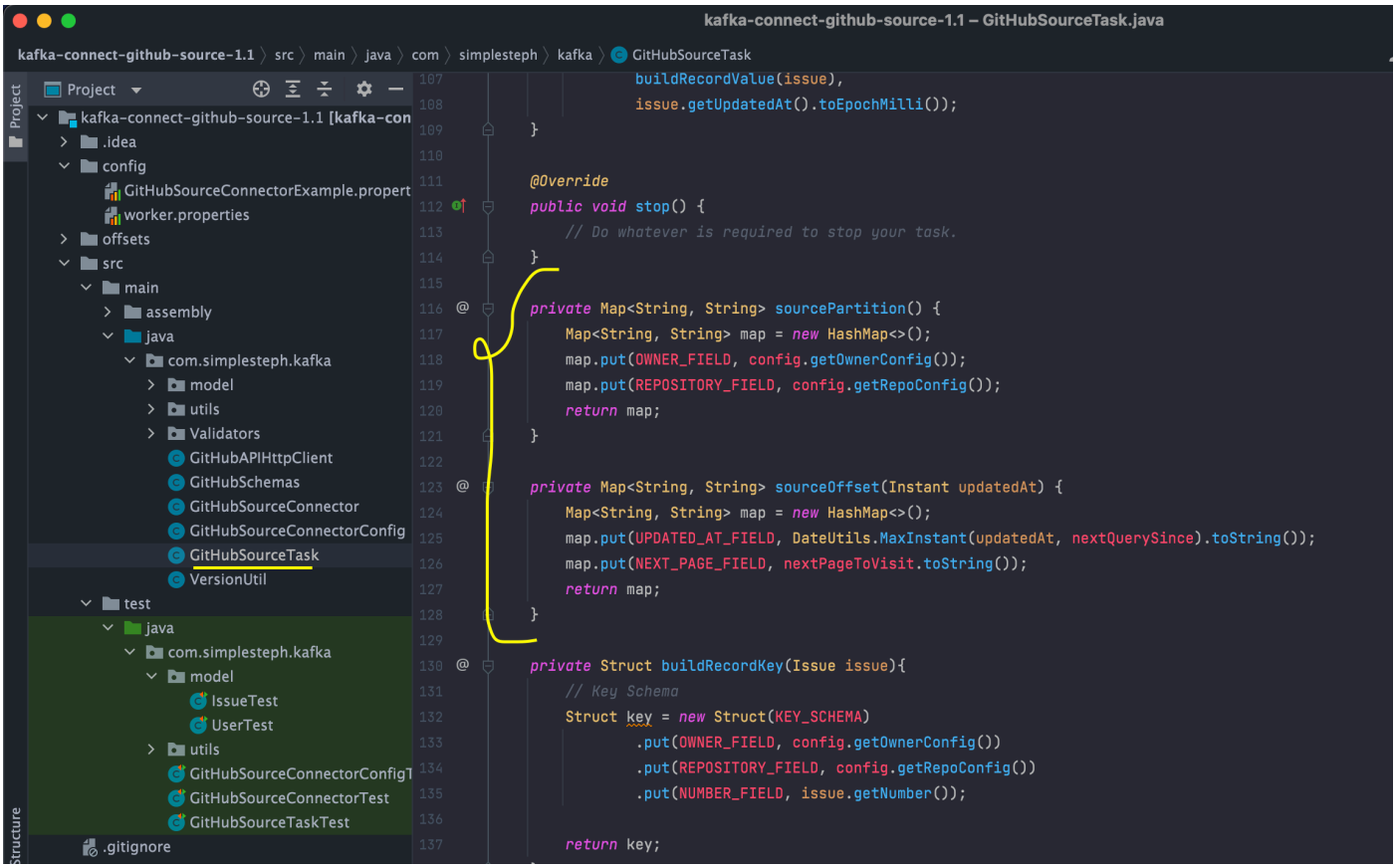
```
public class GitHubAPIHttpClient{
   protected JSONArray getNextIssues() throws InterruptedException
   protected HttpResponse<JsonNode> getNextIssuesAPI() throws UnirestException
   protected String constructUrl()
}
```



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

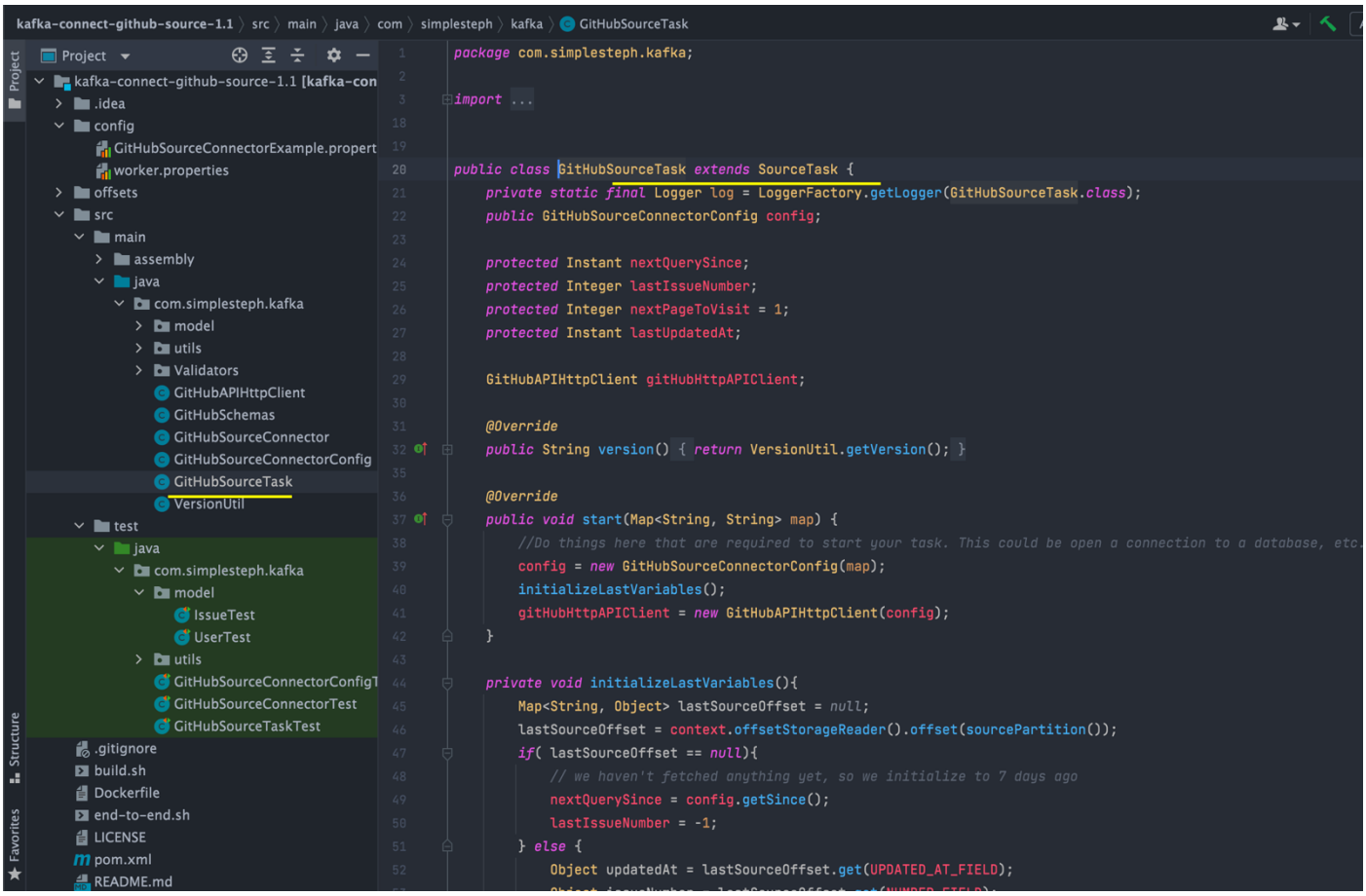
```
private Map<String, String> sourcePartition() {
    Map<String, String> map = new HashMap<>();
    map.put(OWNER_FIELD, config.getOwnerConfig());
    map.put(REPOSITORY_FIELD, config.getRepoConfig());
    return map;
```



Task

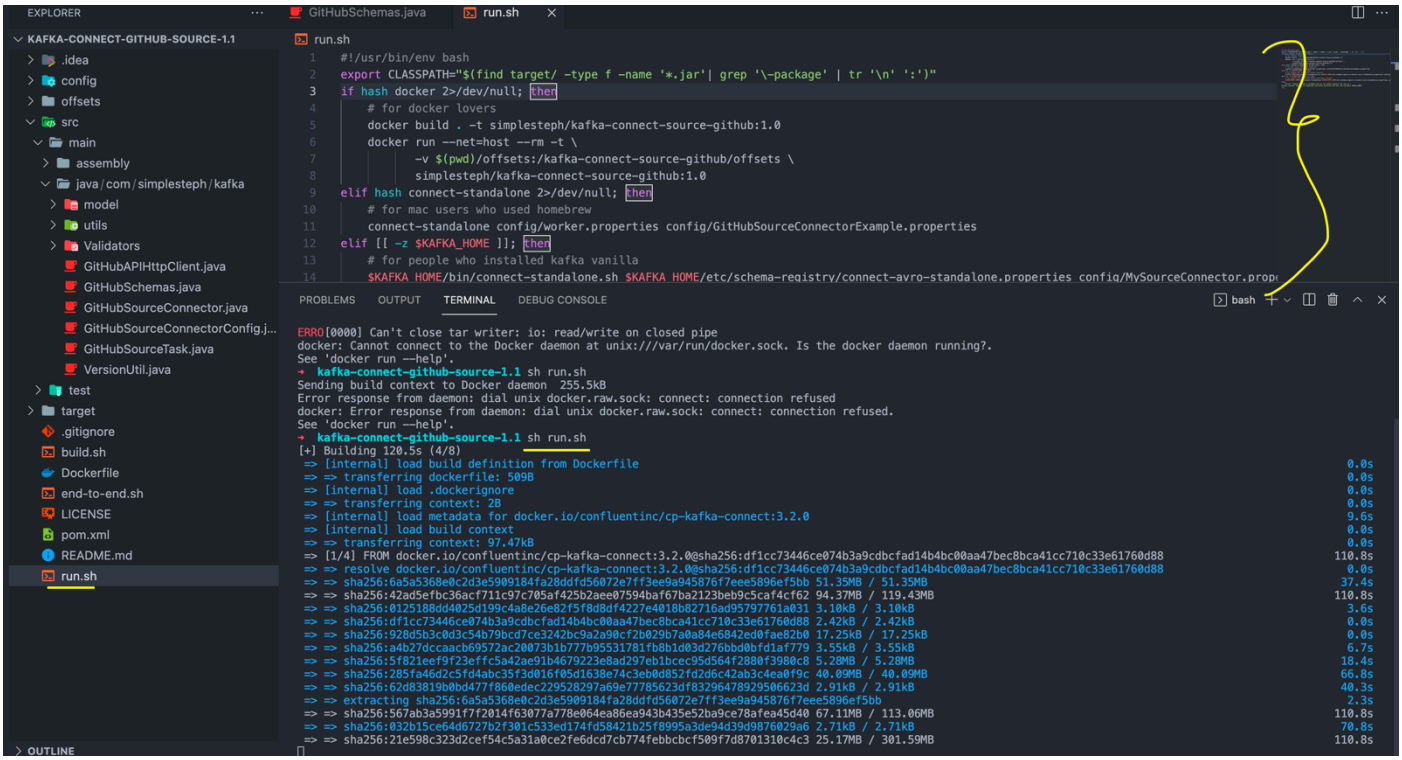
- Our Task is what does the actual job
- It's supposed to initialize, then find where to resume from, and finally poll the source for records

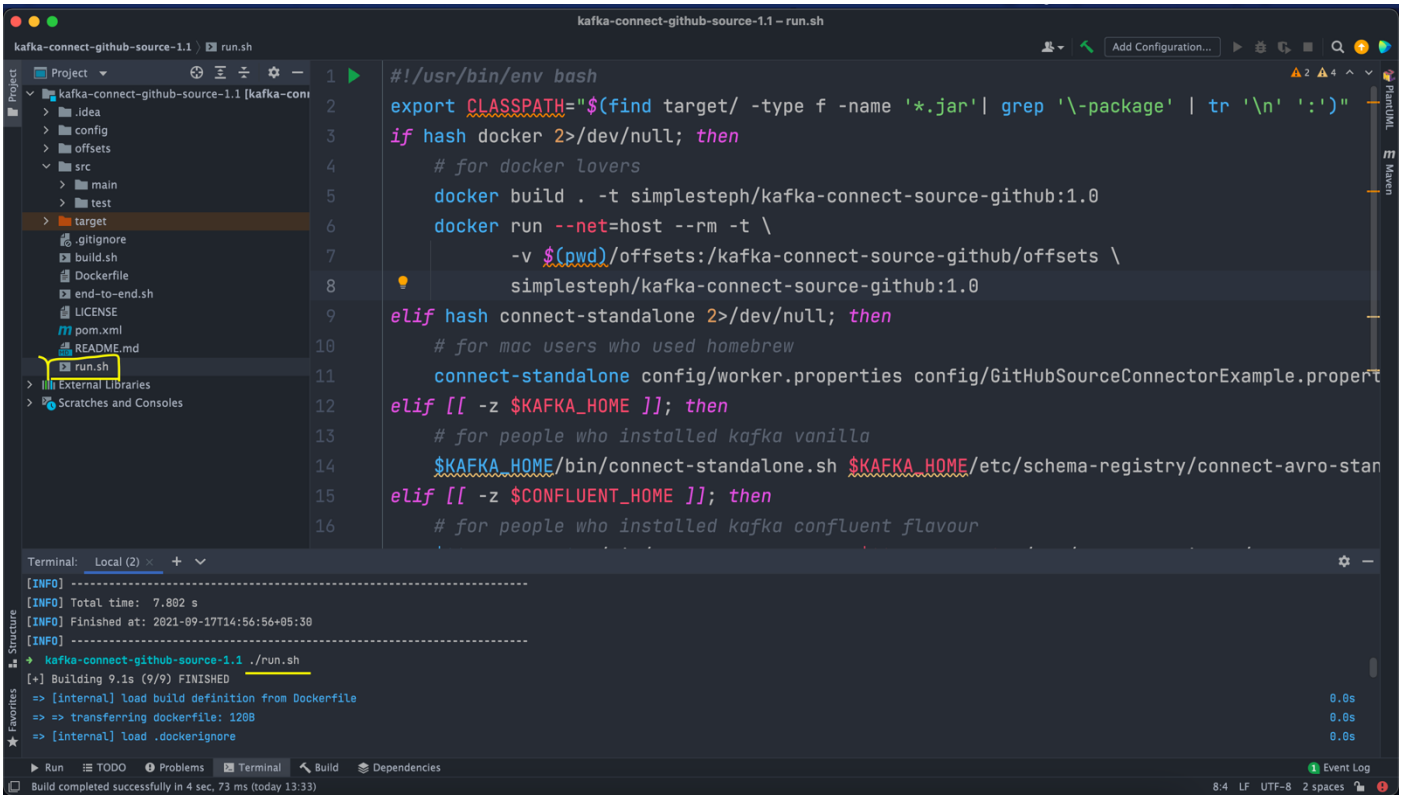
```
public class GitHubSourceTask extends SourceTask {
    public String version()
    public void start(Map<String, String> map)
    public List<SourceRecord> poll() throws
    InterruptedException
    public void stop()
    }
```



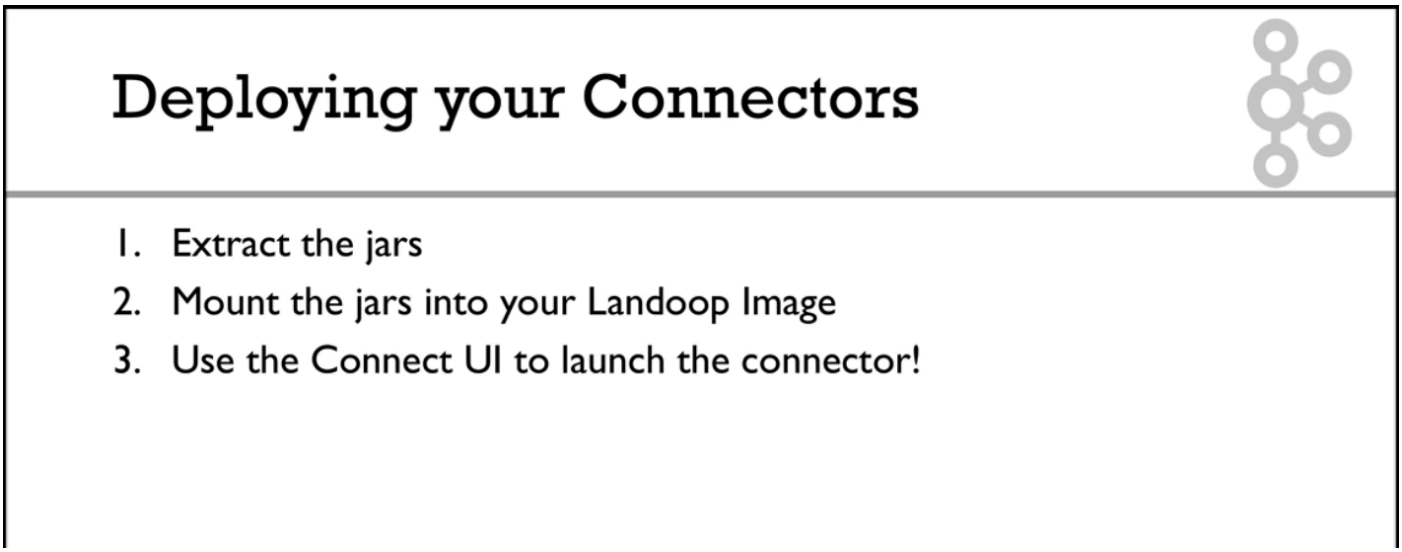
Building and running a connector in standalone mode

- 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





Deploying Connector in Landoop Cluster



Step 1: Running the Kafka Cluster using landoop image in docker.

docker run -it --rm -p 2181:2181 -p 3030:3030 -p 8081:8081 -p 8082:8082 -p 8083:8083 -p 9092:9092 -e ADV_HOST=127.0.0.1 -e RUNTESTS=0 -v "$(pwd)":/connectors/kafka-connect-github-source landoop/fast-data-dev

Here mounting to the “connectors/kafka-connect-github-source” folder is mandatory.

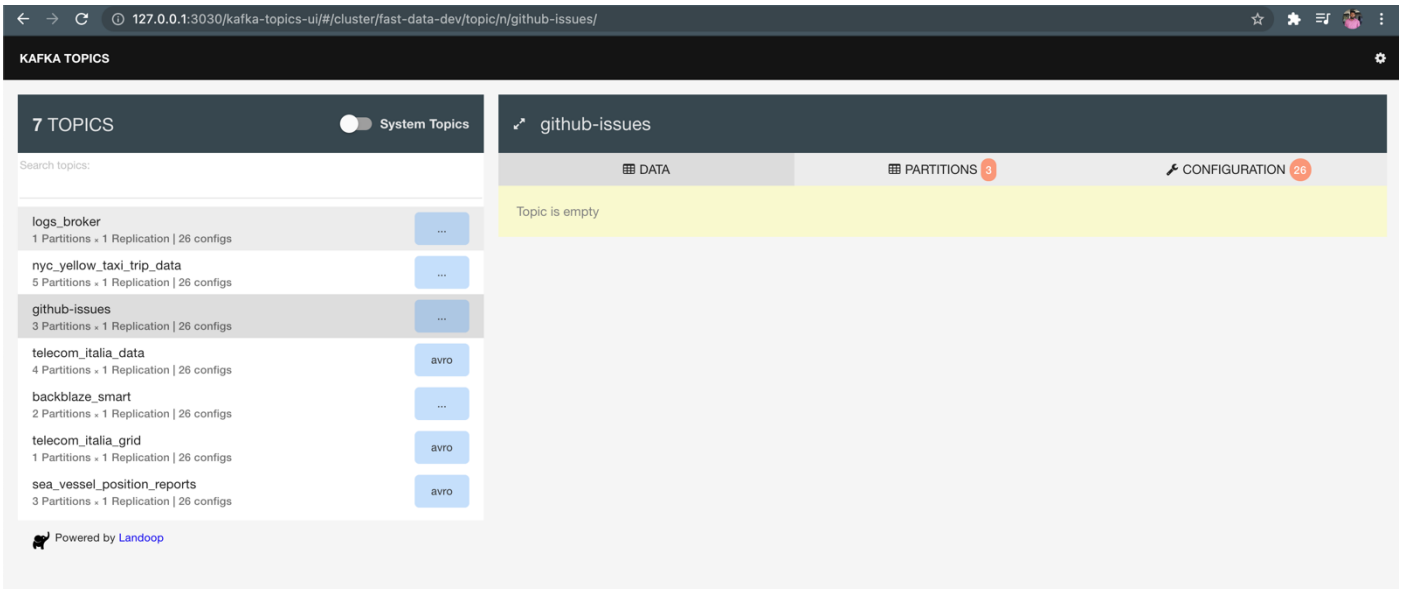
```
→  kafka-connect-github-source docker run -it --rm -p 2181:2181 -p 3030:3030 -p 8081:8081 -p 8082:8082 -p 8083:8083 -p 9092:9092 -
e ADV_HOST=127.0.0.1 -e RUNTESTS=0 -v "$(pwd)":/connectors/kafka-connect-github-source landoop/fast-data-dev
Setting advertised host to 127.0.0.1.
Operating system RAM available is 3088 MiB, which is less than the lowest
recommended of 4096 MiB. Your system performance may be seriously impacted.
Starting services.
This is Lenses.io's fast-data-dev. Kafka 2.6.2-L0 (Lenses.io's Kafka Distribution).
You may visit http://127.0.0.1:3030 in about a minute.
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/01-zookeeper.conf" during parsing
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/02-broker.conf" during parsing
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/03-schema-registry.conf" during parsing
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/04-rest-proxy.conf" during parsing
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/05-connect-distributed.conf" during parsing
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/06-caddy.conf" during parsing
```

Step 2: Creating a new topic

docker run --rm -it --net=host landoop/fast-data-dev bash

kafka-topics --create --topic github-issues --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181

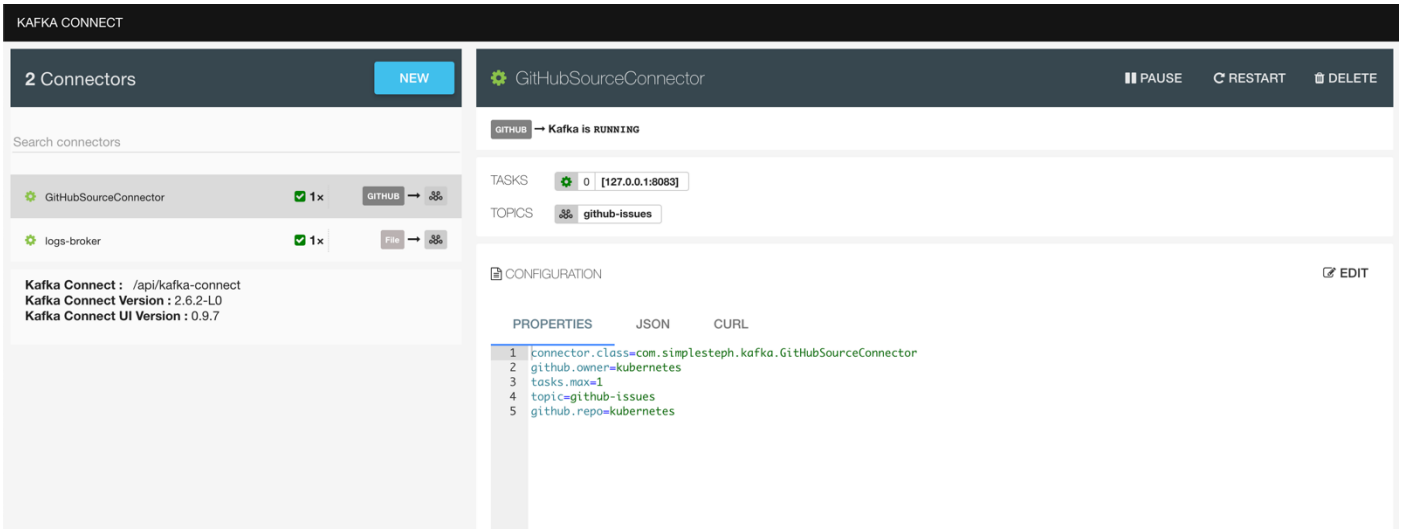
```
→  kafka-connect-github-source docker run --rm -it --net=host landoop/fast-data-dev bash
root@fast-data-dev / $
root@fast-data-dev / $ kafka-topics --create --topic github-issues --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:218
1
Created topic github-issues.
```



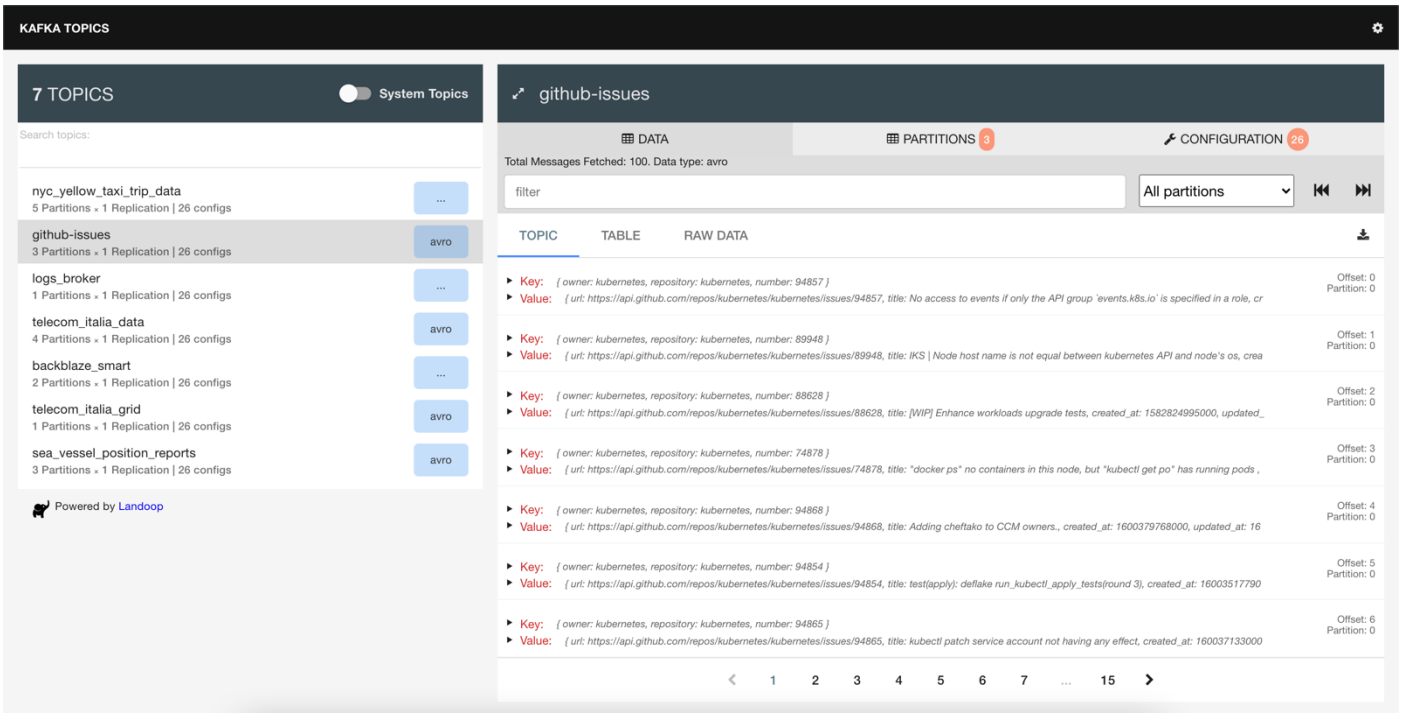
Step 2: Creating a new connector UI landoop UI.

All the JARs are loaded in target mount directory in the docker run command. So that we could able to see the GitHubSourceConnector in Connector New UI in landoop.

And the we created a github source source connector.



Step 3: Checking the topic data.



Development guide for Connector

https://docs.confluent.io/3.2.0/connect/devguide.html

Guideline for Developing Connector

https://www.confluent.io/wp-content/uploads/Partner-Dev-Guide-for-Kafka-Connect.pdf?x18424

https://gist.github.com/jcustenborder/b9b1518cc794e1c1895c3da7abbe9c08#schema-types

Setting up Kafka Connect in Production (1/2)

Here is what we know

- We know how to use connectors
- We know how to create connectors
- We know how to add connectors
- We know how to use the Landoop Kafka Connect UI

All of that is great, but you may have realised this is only for development purposes.

… so how do we do things in production?

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