Kakfa-connect-workshop

Requirements:

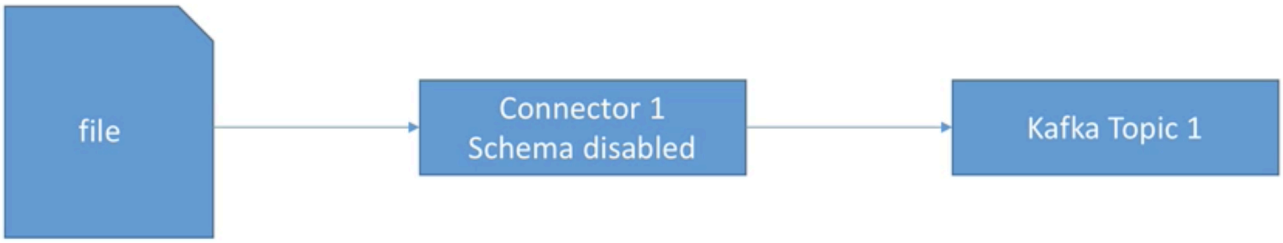
Github repo: https://github.com/rvanrijn/workshop-apache-kafka-connect

---

Lab 1) FileStreamSourceConnector in standalone mode

Goal:

- Read a file and load the content directly into Kafka.
- Run a connector in standalone mode (useful for development).



Learning:

- Understand how to configure a connector in standalone mode.

Steps:

Go to the repository and start the kafka cluster

$ docker-compose up kafka-cluster

Kakfa-connect-workshop

Use one of these based on your OS:

Linux / Mac

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

Windows Command Line:

docker run --rm -it -v %cd%:/tutorial --net=host landoop/fast-data-dev bash

Windows Powershell:

docker run --rm -it -v ${PWD}:/tutorial --net=host landoop/fast-data-dev bash

we launch the kafka connector in standalone mode:

cd /workshop-apache-connect/source/demo-1

This directory contains three files:

- **worker.properties**
- **file-stream-demo-standalone.properties**
- **demo-file.txt**

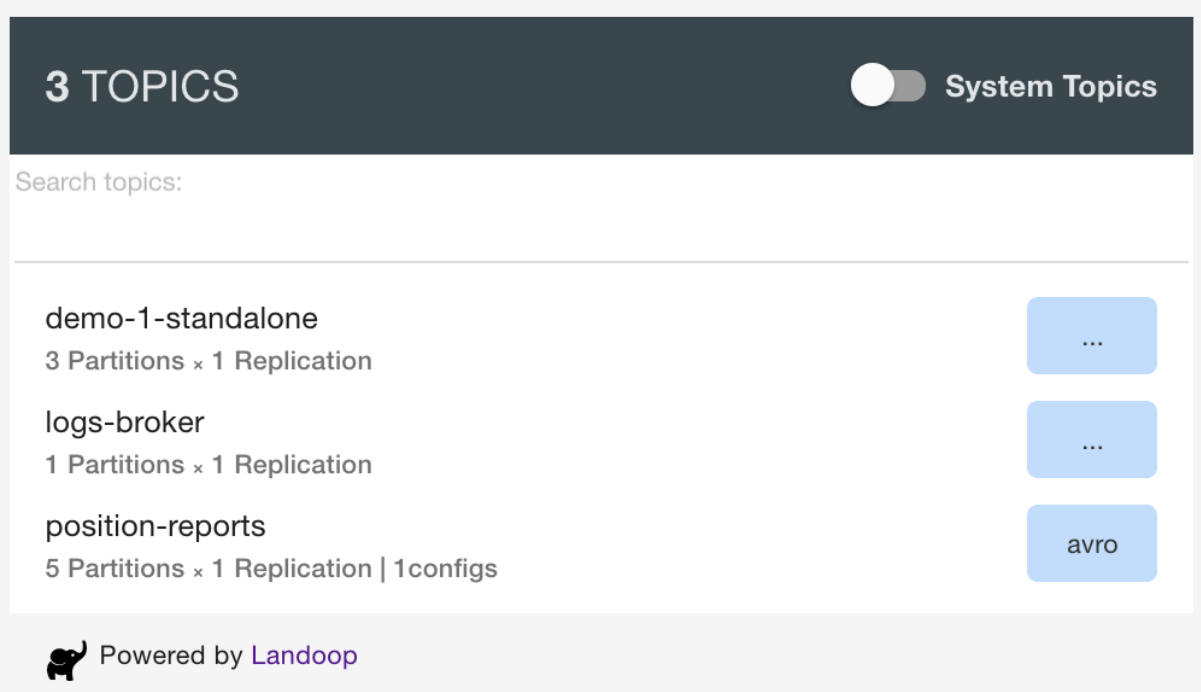
create the topic we write to with 3 partitions

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

Check if Topics are visible in the browser, visit: **http://localhost:3030/kafka-topics-ui/#/**

Now go to to the next page.

Kakfa-connect-workshop

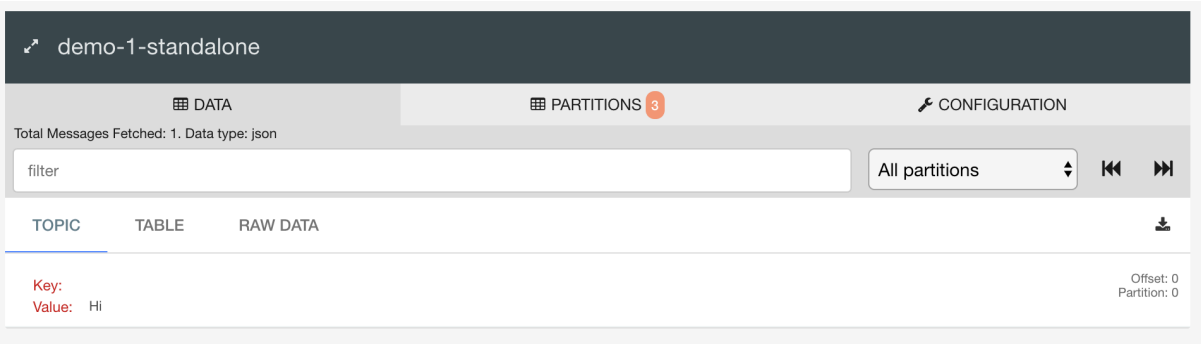


The newly created topic should be visible in the topics list.

Within the **/workshop-apache-connect/source/demo-1 run:**

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

write some data to the demo-file.txt! and check if the data is written the topic! Check:
http://localhost:3030/kafka-topics-ui/#/cluster/fast-data-dev/topic/n/demo-1-standalone/

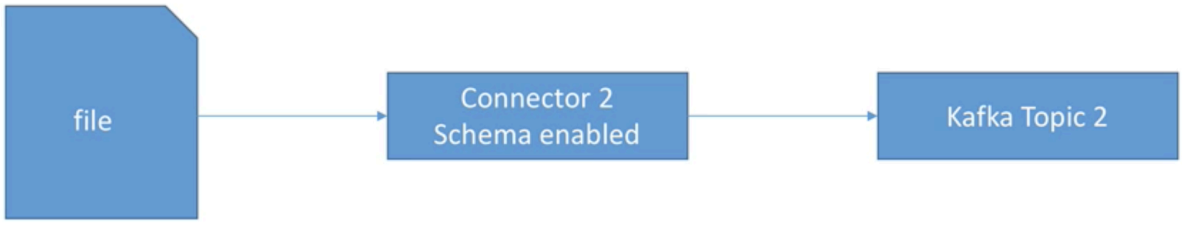


Kakfa-connect-workshop

Lab 2) FileStreamSourceConnector in distributed mode:

Goal:

- Read a file and load the content directly into Kafka
- Run in distributed mode on our 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

2. FileStreamSourceConnector in distributed mode:

create the topic we're going to write to

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

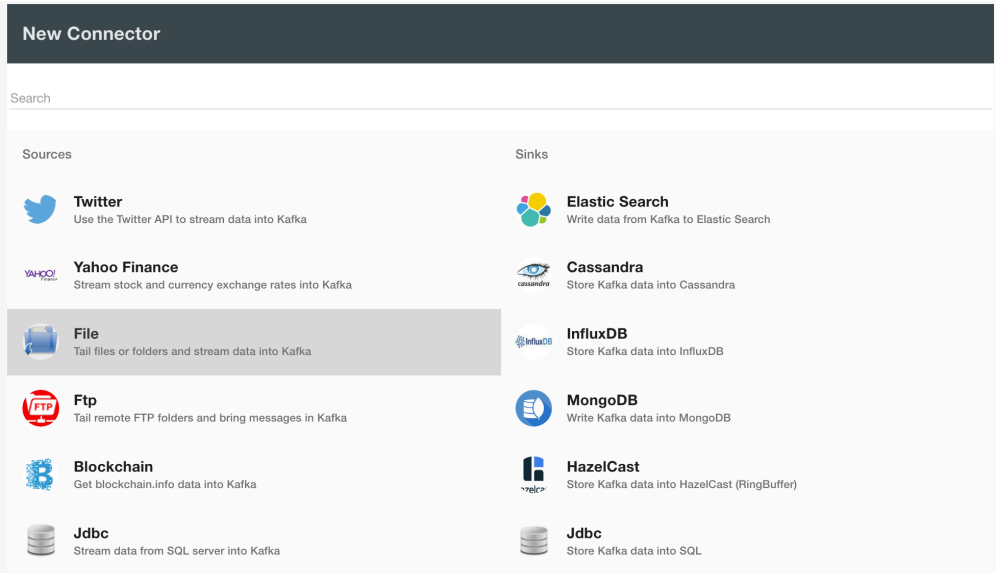
kafka-topics --create --topic demo-2-distributed --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181

you can now close the new shell

head over to 127.0.0.1:3030 -> Connect UI

Kakfa-connect-workshop

Create a new connector -> File Source



Paste the configuration from source/demo-2/file-stream-demo-distributed.properties

Hit CREATE

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

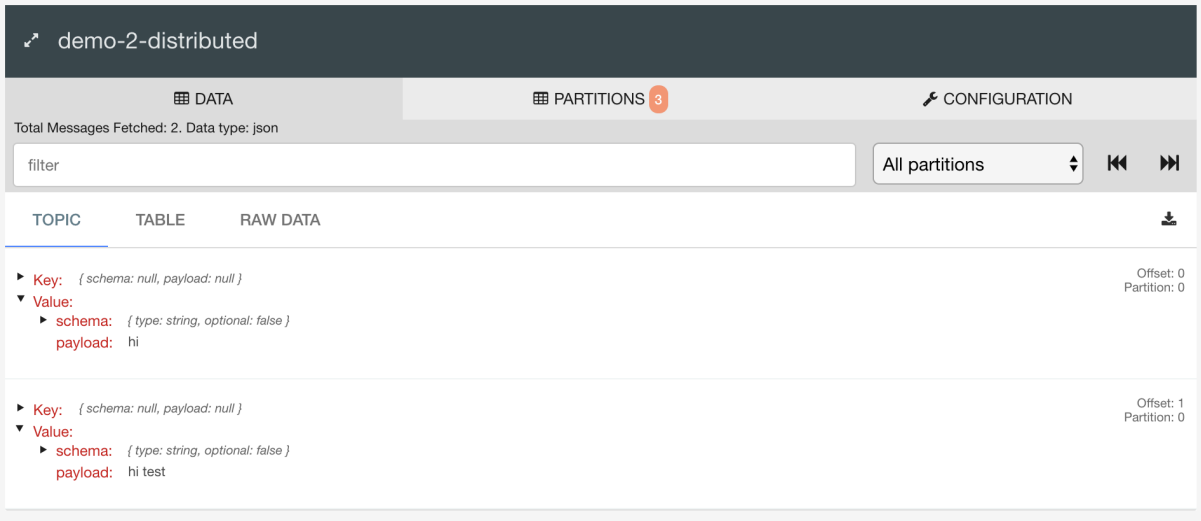
$ echo "hi" >> demo-file.txt

Kakfa-connect-workshop

$ echo "hello" >> demo-file.txt

$ echo "from the other side" >> demo-file.txt

observe we now have json as an output, even though the input was text!



Read the topic data from the cli.

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

Again, observe we now have json as an output, even though the input was text!

Kakfa-connect-workshop

Lab 3) TwitterSourceConnector in distributed mode:

Goal:

- Gather data from Twitter in Kafka Connect Distributed mode

Learning:

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

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

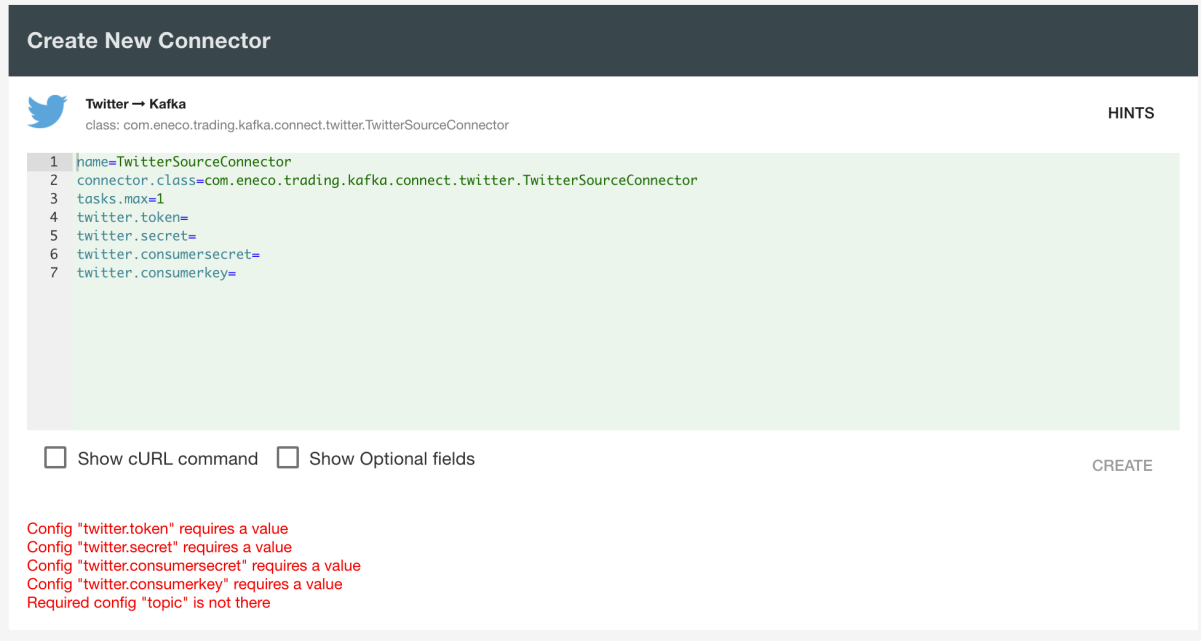
Follow the instructions at: https://github.com/Eneco/kafka-connect-twitter#creating-a-twitter-application

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 agreement. 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.

Kakfa-connect-workshop

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



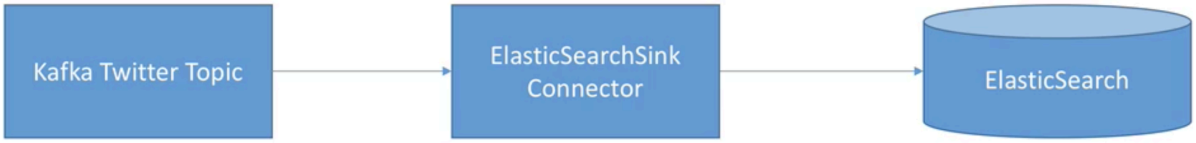
Launch the connector and start seeing the data flowing in!

Kakfa-connect-workshop

Lab 4) ElasticSearchSinkConnector in 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

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

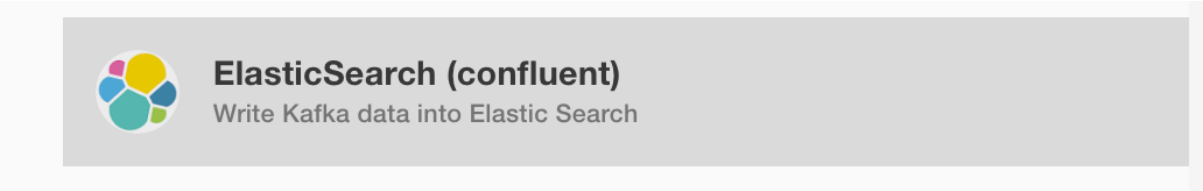
Info here: http://docs.confluent.io/3.2.0/connect/connect-elasticsearch/docs/elasticsearch_connector.html

We make sure elasticsearch is working. Replace 127.0.0.1 by 192.168.99.100 if needed.

Check: http://127.0.0.1:9200/

Kakfa-connect-workshop

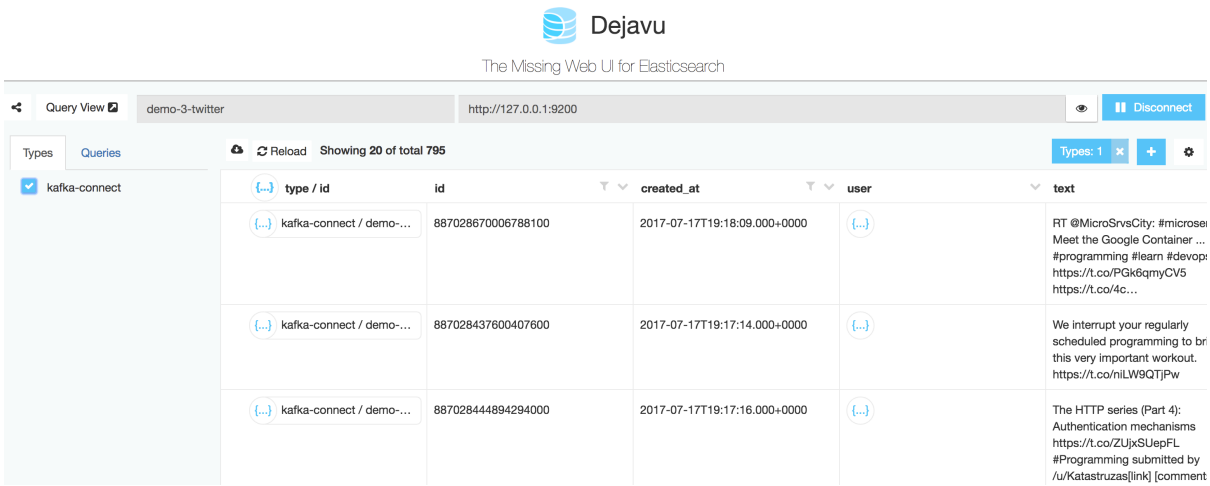
Go to the connect UI and create a new Source connector – ElasticSearch (confluent):



and apply the configuration: sink/demo-elastic/sink-elastic-twitter-distributed.properties

Visualize the data at:

http://127.0.0.1:9200/_plugin/dejavu The type: kafka-connect should appear. See image below:



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

Kakfa-connect-workshop

Lab 5) JdbcSinkConnector in distributed mode:

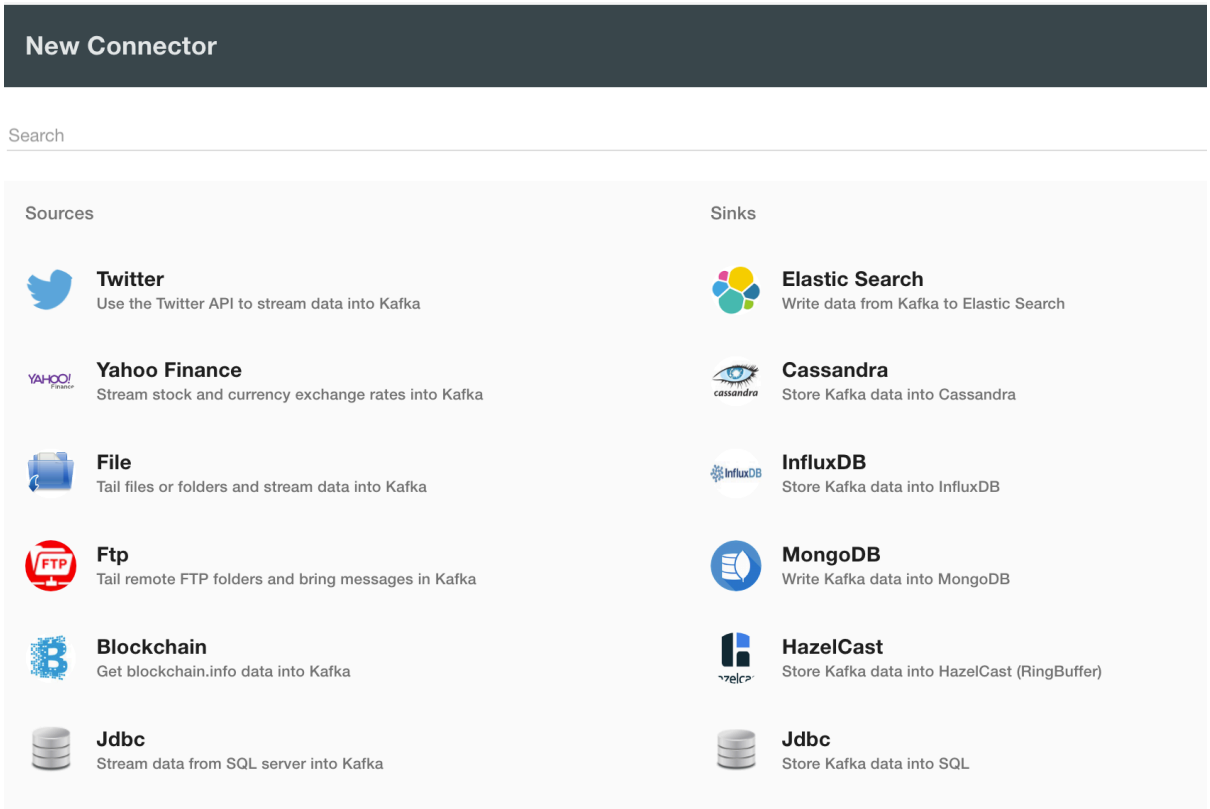
GOAL:

- Start a PostgreSQL instance using Docker
- Run in distributed mode with multiple tasks

Learning:

- Learn about the JDBC Sink Connector

Create a Sink – Jdbc Connector and copy-paste the sink/demo-postgres/ sink-postgres-twitter-distributed.properties



Kakfa-connect-workshop

If all went well the twitter feed should sink(ing) into PostgreSQL. To check if the data is there. Open another CLI and run the following:

```
$ docker run -it --rm --net=host jbergknoff/postgresql-client
postgresql://postgres:postgres@127.0.0.1:5432/postgres
```

And check the following table:

```
psql (9.3.11, server 9.5.7)
WARNING: psql major version 9.3, server major version 9.5.
         Some psql features might not work.
Type "help" for help.
postgres=# \dt
             List of relations
 Schema |      Name      | Type  |  Owner
--------+----------------+-------+----------
 public | demo-3-twitter | table | postgres
(1 row)
postgres=# select count(*) from "demo-3-twitter";
 count
-------
    17
(1 row)
postgres=# \q
```