EJERCICIOS ALMACENAMIENTO Y PROCESAMIENTO DE DATOS

[ELK Stack](#_wrde8sb1wcez)

[Elasticsearch](#_76kbb197dwq8)

[Logstash](#_qgjrugsx23vy)

[Kibana](#_15ytchjpmsxh)

[Stream Processing](#_lbzl1xc8fs15)

[Spark Streaming Demo](#_thhxqzwpswo1)

# ELK Stack

## Elasticsearch

**Objective**: Exercise you will see how quickly and easily the Elastic Stack can be used to search a dataset. You will startup Elasticsearch and Kibana, then run queries from Kibana to search an indexed dataset.

**Note**: The datasets we are going to use are on git repo on following folder: https://github.com/rlopezherrero/GFT-EDEM-MasterData/tree/master/AlmacenamientoProcesamiento/docker/data

* 1. Launch docker compose with full ELK stack

$ GFT-EDEM-MasterData/AlmacenamientoProcesamiento/docker/

$ docker-compose up -d

* 1. Wait a couple of minutes to ensure that the stack is running.
  2. Open your Web browser and go to [http://localhost:9200](http://localhost:9200/). You should see the start page that verifies Elasticsearch is running on your localhost - a simple JSON object that will look similar to:

{"name" : "elasticsearch",

"cluster\_name" : "docker-cluster",

"cluster\_uuid" : "UJuJfAFlR-m0kyuyLd0mew",

"version" : {

"number" : "7.3.0",

"build\_flavor" : "default",

"build\_type" : "docker",

"build\_hash" : "de777fa",

"build\_date" : "2019-07-24T18:30:11.767338Z",

"build\_snapshot" : false,

"lucene\_version" : "8.1.0",

"minimum\_wire\_compatibility\_version" : "6.8.0",

"minimum\_index\_compatibility\_version" : "6.0.0-beta1"

},

"tagline" : "You Know, for Search"

* 1. Based on previous output:

o What is the version of Elasticsearch instance?

o What is the name of your node?

o What is the name of your cluster?

* 1. Analyse the data set we are going to use (data/products). The dataset you are going to be using is a collection of products sold in stores. The fields in the dataset consist of:

o grp\_id: a unique ID for each row

o upc12: a 12-digit string containing the Universal Product Code

o brandName: the name of the company that makes the product

o productName: the name of the product

o customerRating: an integer value between 1 and 5, where 5 is the highest rating

o price: a float value representing the price of the product

o quantitySold: an integer value representing the number of units sold

Review this small sample of what the data looks:

20,204040000000,Usda Produce,Plums Black,3,1.39,68603

21,753950001954,Doctor's Best,Doctor's Best Best Curcumin C3 Complex 1000mg Tablets - 120 Ct,2,1.75,651857

22,016000288829,Betty Crocker,Betty Crocker Twin Pack Real Potatoes Scalloped 2 Pouches For 2 Meals - 2 Pk,2,4.22,527899

23,070670009658,Reese,Reese Mandarin Oranges Segments In Light Syrup,2,3.73,104348

24,688267084225,Smart Living,Smart Living Charcoal Lighter Fluid,5,3.20,637769

25,044100117428,Hood,Hood Latte Iced Coffee Drink Vanilla Latte,5,5.99,166777

* 1. The products dataset was indexed already into your Elasticsearch instance. Go to<http://localhost:9200/_cat/indices?v>. Your **products** index should appear in the list of indices. (Notice the size may vary, but it should contain 110,435 documents):

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

**green open products 7J9gdrEPS3GJqzjDBs7SIA 2 0 110435 0 26.6mb 26.6mb**

green open .kibana 2cg2sj3VTNeNfHuESfFkzA 1 0 1 0 3.7kb 3.7kb

* 1. To view the products data, use the Search API with the following URL:<http://localhost:9200/products/_search>. You should see 10 products (10 is the default search size). The products displayed will likely vary from the output below, but notice the output is not very pretty:

{"took":57,"timed\_out":false,"\_shards":{"total":2,"successful":2,"skipped":0,"failed":0},"hits":{"total":110435,"max\_score":1.0,"hits":[{"\_index":"products","\_type":"product","\_id":"50550","\_score":1.0,"\_source":{"brandName":"Riceland","productName":"Riceland Extra Long Grain Rice","quantitySold":733733,"price":11.87,"grp\_id":"50550","type":"csv","upc12":"035200055154","customerRating":2}},{"\_index":"products","\_type":"product","\_id":"50551","\_score":1.0,"\_source":{"brandName":"Silverpoint","productName":"Silverpoint Quad Planning Pad","quantitySold":450550,"price":11.8,"grp\_id":"50551","type":"csv","upc12":"075755510708","customerRating":1}},

……

* 1. Change the URL to include the pretty argument: <http://localhost:9200/products/_search?pretty>. The output should now look much nicer:

{

"took" : 25,

"timed\_out" : false,

"\_shards" : {

"total" : 1,

"successful" : 1,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : {

"value" : 10000,

"relation" : "gte"

},

"max\_score" : 1.0,

"hits" : [

{

"\_index" : "products",

"\_type" : "product",

"\_id" : "588",

"\_score" : 1.0,

"\_source" : {

"productName" : "Diamond Shelled Pecans",

"customerRating" : 4,

"price" : 1.62,

"brandName" : "Diamond",

"upc12" : "070450142513",

"type" : "csv",

"grp\_id" : "588",

"quantitySold" : 797270

}

},

….

* 1. Based on the output of previous search:

o How many documents were indexed?

o What is the \_type of each document?

* 1. Next, you are going to use Kibana.
  2. To verify Kibana is running, open your Web browser and go to [http://localhost:5601](http://localhost:5601/) . The Kibana application should appear.
  3. Click on the **Dev Tools** button (left side tool icon) in the side navigation pane to open the **Console** application:
  4. Notice there is a **match\_all** query already written in the **Console**. Go ahead and run it by clicking the green "play" button that appears to the right of the command, or using the **Ctrl/Cmd + Enter** keyboard shortcut. This search hits all documents in all indexes of your cluster. Notice the output is similar to the output you saw a few minutes ago, but displayed nicely in the Console screen.
  5. Now let's search for products. Enter the following query into the **Console**, just below your first command. It answers the question "Which products have peanut or butter in the product name?" Feel free to copy-and-paste the command:

GET products/\_search

{

"size": 25,

"query": {

"match": {

"productName": "peanut butter"

}

}

}

Click the green play button next to this new query, or you can use the keyboard shortcut **Command+Enter**. The output will show the top 25 results of products with "peanut" or "butter" in the **productName** field

* 1. Based on the results of your previous search:

o How many products in the dataset match "peanut butter" in the product name?

o What was the max\_score of the results?

o Would you say that your top results were relevant to "peanut butter"?

* 1. Change the term **"peanut butter"** to **"Peanut Butter"** and run the query again. What is different in the results, if anything?
  2. Run a search for **"kasmati rice"** in the **"productName"**, then answer the following questions:

o How many hits are there?

o What was the \_score of the top hit?

o View the scores of the next few hits. Why do you think the top hit had a much higher score than all the other hits?

* 1. Compare the following search to the previous ones you executed. Notice the field being searched on is now customerRating. Run the following query, which answers the query: "Find all products with a customer rating of 4":

GET products/\_search

{

"query": {

"match" : {

"customerRating" : 4

}

}

}

Notice the score for each hit is simply **1**. Why do you think all matching documents have the same score?

* 1. Write a query that finds all products whose price field is exactly 10.00. You should get 44 hits.

## Logstash

**Objective**: Learn Logstash basics implementing several pipelines and working with several plugins.

* 1. First test that your logstash installation is working properly, writing on the standard input:

$ Docker exec -it logstash bash

$ bin/logstash -e “input { stdin { } } output { stdout {} }” --path.data /tmp/

* 1. Create a Logstash pipeline called my-first-pipeline.conf to consume from the filebeat on port 5044 and write to the std output. Add following line to the output section for this: stdout { codec => rubydebug }:

vi my-first-pipeline.conf

input {

beats {

port => "5044"

}

}

# The filter part of this file is commented out to indicate that it is

# optional.

# filter {

#

# }

output {

stdout { codec => rubydebug }

}

* 1. Before launching it, verify that your pipeline configuration is fine.

bin/logstash -f my-first-pipeline.conf --config.test\_and\_exit --path.data /tmp

* 1. If the configuration is fine, launch this pipeline.

bin/logstash -f my-first-pipeline.conf --config.reload.automatic --path.data /tmp

* 1. Configure a filebeat logging, first logon filebeat container:

Docker exec --user root -it filebeat bash

* 1. Create a filebeat config to capture and publish logs.

cp filebeat.yml filebeat-training.yml

* 1. Edit filebeat-training.yml

filebeat.inputs:

- type: log

enabled: true

paths:

- /tmp/datasets/apache\_logs/logstash-tutorial-dataset.log

output.logstash:

# Array of hosts to connect to.

hosts: ["logstash:5044"]

* 1. Launch filebeat process:

filebeat -e -c filebeat-training.yml -d "publish"

* 1. Take a look to the logstash, you should see following events on the console:

{

"offset" => 24464,

"message" => "86.1.76.62 - - [04/Jan/2015:05:30:37 +0000] \"GET /style2.css HTTP/1.1\" 200 4877 \"http://www.semicomplete.com/projects/xdotool/

\" \"Mozilla/5.0 (X11; Linux x86\_64; rv:24.0) Gecko/20140205 Firefox/24.0 Iceweasel/24.3.0\"",

"source" => "C:\\Users\\role\\Documents\\Formacion\\Introduction ELK\\course-ELK-introduction\\datasets\\apache\_logs\\logstash-tutorial-datase

t.log",

"@version" => "1",

"prospector" => {

"type" => "log"

},

"beat" => {

"name" => "NBVAL730",

"hostname" => "NBVAL730",

"version" => "6.2.3"

},

"host" => "NBVAL730",

"tags" => [

[0] "beats\_input\_codec\_plain\_applied"

],

"@timestamp" => 2018-04-16T11:36:42.598Z

}

….

* 1. Remove data registry in order to repopulate log messages (/usr/share/filebeat/data/registry)

rm -r /usr/share/filebeat/data/registry

* 1. Now, that we see pipeline is working, let’s tune logstash config. We are going to update logstash pipeline using a specific filter that parses unstructured data and generates a structured json. For doing this we will use grok plugin with the following property : "message" => "%{COMBINEDAPACHELOG}" .

vi my-first-pipeline.conf

input {

beats {

port => "5044"

}

}

filter {

grok {

match => { "message" => "%{COMBINEDAPACHELOG}"}

}

}

output {

stdout { codec => rubydebug }

}

* 1. You should see something like this on the logstash console:

{

"request" => "/presentations/logstash-monitorama-2013/images/kibana-search.png",

"agent" => "\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_9\_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.77 Safari/537.36\"",

"offset" => 325,

"auth" => "-",

"ident" => "-",

"verb" => "GET",

"prospector" => {

"type" => "log"

},

"source" => "/path/to/file/logstash-tutorial.log",

"message" => "83.149.9.216 - - [04/Jan/2015:05:13:42 +0000] \"GET /presentations/logstash-monitorama-2013/images/kibana-search.png HTTP/1.1\" 200 203023 \"http://semicomplete.com/presentations/logstash-monitorama-2013/\" \"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_9\_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.77 Safari/537.36\"",

"tags" => [

[0] "beats\_input\_codec\_plain\_applied"

],

"referrer" => "\"http://semicomplete.com/presentations/logstash-monitorama-2013/\"",

"@timestamp" => 2017-11-09T02:51:12.416Z,

"response" => "200",

"bytes" => "203023",

"clientip" => "83.149.9.216",

"@version" => "1",

"beat" => {

"name" => "My-MacBook-Pro.local",

"hostname" => "My-MacBook-Pro.local",

"version" => "6.0.0"

},

"host" => "My-MacBook-Pro.local",

"httpversion" => "1.1",

"timestamp" => "04/Jan/2015:05:13:42 +0000"

}

* 1. Modify the pipeline to index all logs on local elastic instance. Check on your local kibana instance if an index for logstash data has been created (Management --> create index pattern). You should see 101 log events loaded with proper json format.

input {

beats {

port => "5044"

}

}

# The filter part of this file is commented out to indicate that it is

# optional.

filter {

grok {

match => { "message" => "%{COMBINEDAPACHELOG}"}

}

}

output {

elasticsearch {

hosts => [ "elasticsearch:9200" ]

}

}

* 1. Check the status of the logstash to see your pipeline running. You should get just one pipeline running an output similar to this. Go to this link <http://localhost:9600/_node/pipelines?pretty>

{

"host" : "NBVAL730",

"version" : "6.2.3",

"http\_address" : "127.0.0.1:9600",

"id" : "ac74dbff-9d8f-4b8f-99b5-0f3a71e5365b",

"name" : "NBVAL730",

"pipelines" : {

"main" : {

"workers" : 4,

"batch\_size" : 125,

"batch\_delay" : 50,

"config\_reload\_automatic" : true,

"config\_reload\_interval" : 3000000000,

"dead\_letter\_queue\_enabled" : false

}

}

}

## Kibana

In this exercise we will play with Kibana. For that we have following datasets:

* The complete works of William Shakespeare, suitably parsed into fields GFT-EDEM-MasterData/tree/master/AlmacenamientoProcesamiento/docker/data/shakespeare). It has following schema:

{

"line\_id": INT,

"play\_name": "String",

"speech\_number": INT,

"line\_number": "String",

"speaker": "String",

"text\_entry": "String",

}

* A set of fictitious accounts with randomly generated data. fields GFT-EDEM-MasterData/tree/master/AlmacenamientoProcesamiento/docker/data/accounts).

{

"account\_number": INT,

"balance": INT,

"firstname": "String",

"lastname": "String",

"age": INT,

"gender": "M or F",

"address": "String",

"employer": "String",

"email": "String",

"city": "String",

"state": "String"

}

* A set of randomly generated log files. fields dlp/Sessions/STO-Storage/STO-003 /docker/data/random\_logs). It has a big schema, but among all the fields following ones are the important ones on the Laboratory:

{

"memory": INT,

"geo.coordinates": "geo\_point"

"@timestamp": "date"

}

* 1. Before we load those data sets, we need to specify some data type mappings. To do that, go to the Dev tools on your local kibana instance (<http://localhost:5601/app/kibana#/>) and launch the following for Shakespeare data :

PUT /shakespeare

{

"mappings": {

"properties": {

"speaker": {"type": "keyword"},

"play\_name": {"type": "keyword"},

"line\_id": {"type": "integer"},

"speech\_number": {"type": "integer"}

}

}

}

* 1. Check on the result panel that this has been successfully executed. With a result similar to this:

{

"acknowledged": true,

"shards\_acknowledged": true,

"index": "shakespeare"

}

* 1. For logstash data, we need also to specify geopoint type mapping. Replicate this for the 3 indexes ( logstash-2015.05.18, logstash-2015.05.19, logstash-2015.05.20 ). Also check that the result is successful.

PUT /logstash-2015.05.18

{

"mappings": {

"properties": {

"geo": {

"properties": {

"coordinates": {

"type": "geo\_point"

}

}

}

}

}

}

* 1. Now, let’s do a bulk load of the 3 datasets. Use following command as an example:

On windows using Curl Power Shell (remember you need to be on dataset folder accounts , shakespeare, random\_logs, etc..):

curl -Uri<http://localhost:9200/bank/_bulk?pretty> -ContentType application/x-ndjson -Method Post -InFile .\accounts.json

On Mac/Linux/Unix:

curl -H "Content-Type: application/x-ndjson" -XPOST "http://localhost:9200/bank/\_bulk?pretty" --data-binary @accounts.json

* 1. For Shakespeare data index name is ‘shakespeare’, and for random\_logs, format is on logstash schema and you don’t need to specify index.
  2. Check that your created indexes exists and contains loaded data, use curl command for this. You should get something like this:

green open .kibana\_task\_manager cgNUew3ATWm63zpIz7aaVw 1 0 2 0 12.7kb 12.7kb

yellow open bank D2OMkGX4T\_qEBG\_mfzKrfg 1 1 1000 0 427.6kb 427.6kb

yellow open shakespeare gsSXWPmqSH-mBkWWmW6jpg 1 1 111396 0 19.5mb 19.5mb

yellow open logstash-2015.05.20 Q02jsR\_gQjq8Qixqrfw4SA 1 1 4750 0 18.2mb 18.2mb

green open .kibana\_1 yK\_Cb0FmQnaqf1WLjhepgg 1 0 6 0 35.4kb 35.4kb

yellow open logstash-2015.05.18 zN8qlTwTS9C0fW5VsvEKWg 1 1 4631 0 19.3mb 19.3mb

yellow open products q9G0BdT4RBOXanoGQwc\_cA 1 1 110435 0 28.5mb 28.5mb

yellow open logstash-2015.05.19 Qf78PQfATPexdga86DyCXg 1 1 4624 0 19.8mb 19.8mb

* 1. Create index patterns for 3 datasets. You can use following patterns: bank\*, shakespeare\* and logstash-2015.05\*. Take into consideration that logstash dataset contains a time series.
  2. Now, go to the discover tab and get from bank index, the accounts that have a balance higher than 47500 and the account number is lower than 100. (You should bet 5 results).
  3. Using Shakespeare index, get the entries that are from play “Henry IV” and on the text contains “London”. (You should get 9 hits).
  4. Using account data, create a Pie chart defining following Ranges:

0 999

1000 2999

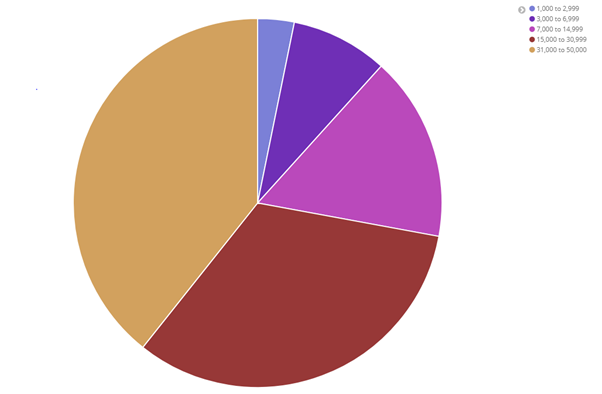
3000 6999

7000 14999

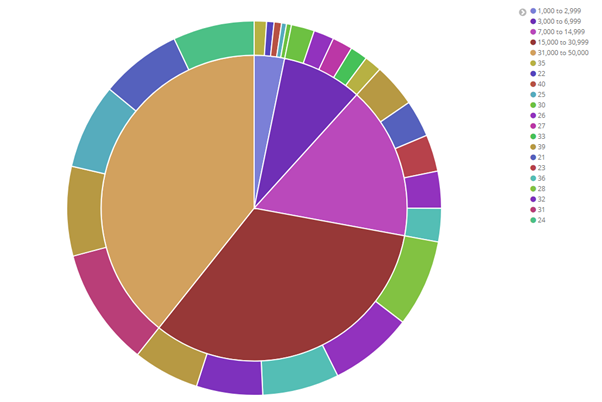
15000 30999

31000 50000

* 1. You should get something similar to this:

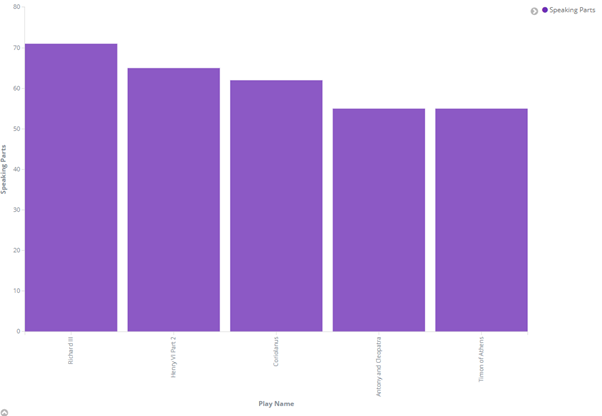


* 1. On same Pie chart add another bucket aggregation to get the age of each account holder inside each balance range. You should get something similar to this:

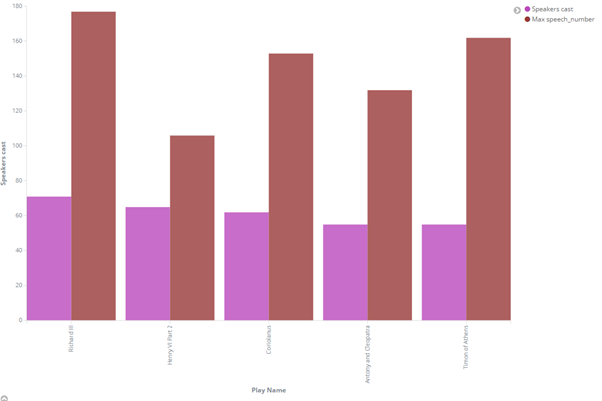


Click on save to use it later, and save it as “Pie Example”.

* 1. Now let’s create a Vertical Bar Chart using Shakespeare dataset. We want so see number of speakers cast per play. You should get something similar to this:



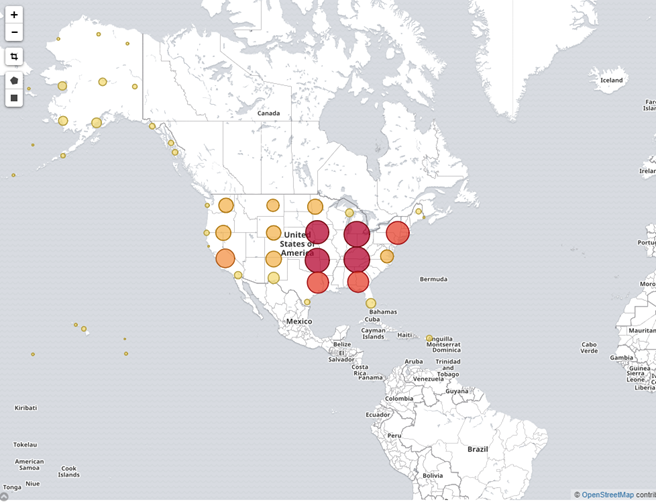
* 1. Using same bar char, we want to get the maximum number of speeches for an individual actor on every play.



Click save, and save it as “Bar chart example”.

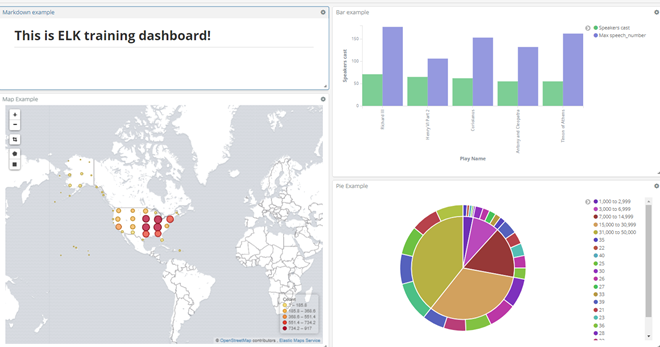
* 1. Now, using Logstash dataset, create a Coordinate Map with following parameters:
     1. Use geo.coordinates as the geo\_point.
     2. Configure the Time window to absolute between May 18 2015 and May 20 2015.

You will get something similar to this:



Click save, and save it as “Bar Map example”.

* 1. Now let’s create a dashboard and integrate all visualization panes done. You should obtain something like the below:



# Stream Processing

## Streaming window type

Imagine that you work for an e-commerce Company. And you have a web, where customers buy your products. Yo need to do the following:

* Every 4 hours you want to pack together all buyed products in order to be sent by the main logistic warehouse to the distribution channel.
* In order to identify your advertising campaigns you want to see at any time last 5 minutes number of buyed products for each type.

How would you implement this?

## Spark Streaming Demo

Main purpose of this exercise is to execute an Streaming word count application over a tumbling window and see the result. For doing that we will execute this [class](https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/streaming/NetworkWordCount.scala). Those are the steps that you need to follow:

* Launch a spark master docker container on a command interface:

docker run --name spark-master -h spark-master -e ENABLE\_INIT\_DAEMON=false -d bde2020/spark-master:2.4.5-hadoop2.7

* Check the container is running:

docker ps

* On same console init a ssh session on spark-master container:

docker exec -it spark-master bash

* Init a netcat server:

nc -lkp 9999

* Open another command interface and connect to the same master container:

docker exec -it spark-master bash

* Now launch a Spark streaming job:

./spark/bin/run-example streaming.NetworkWordCount localhost 9999

* Now start writing lines on the netcat console
* See the output on Spark Submit console. What do you think is the window used and the functionality implemented?

Hace un micro-batching de 5 segundos. Es un ejemplo de ventana fija. Ambos servidores tienen el mismo puerto, escribes con el netcat y Spark lo lee.