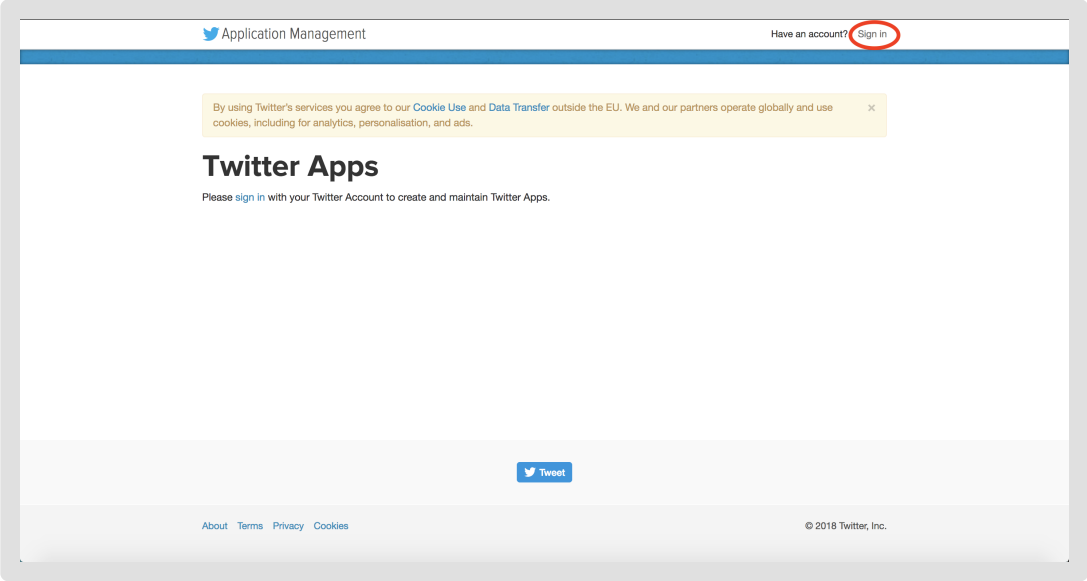
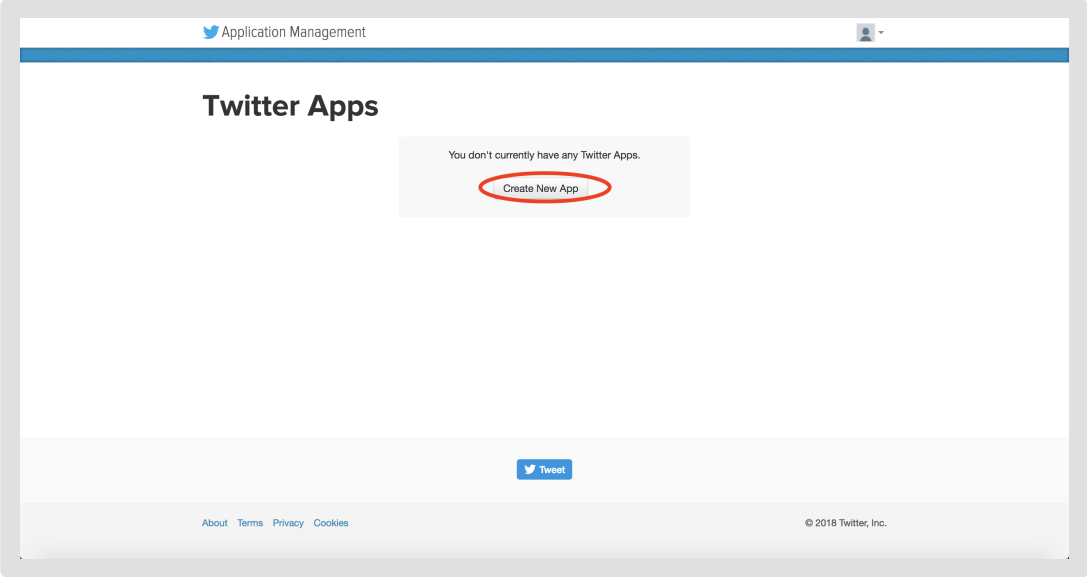
Lab Guide: Sentiment Analysis and Brand Monitoring

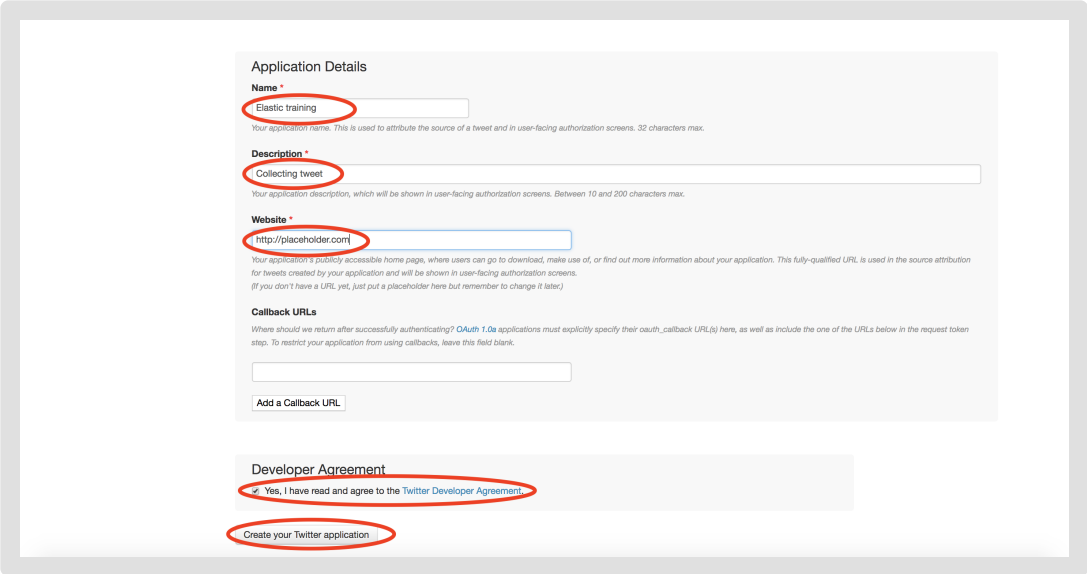
Lab 1: Overview

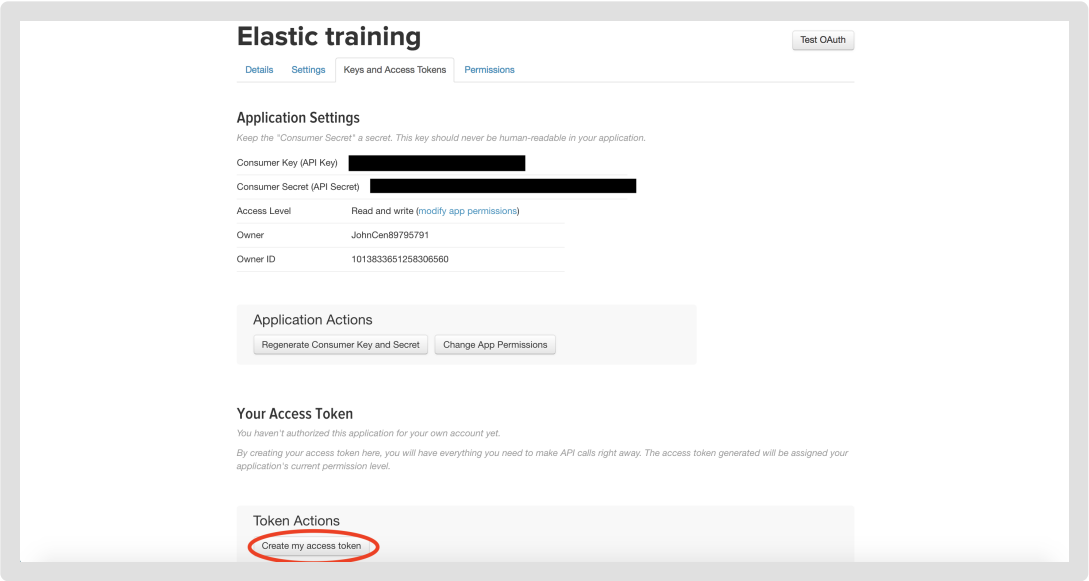
Objective: In this lab, we are going to first create a twitter account and then prepare our environment for the rest of the class.

1. The first step will be to create a Twitter app. In order to do that you will need a twitter account. Use your twitter account or create a new one: https://apps.twitter.com to create an application.


Application Management
Have an account? Sign in
By using Twitter's services you agree to our Cookie Use and Data Transfer outside the EU. We and our partners operate globally and use cookies, including for analytics, personalisation, and ads.
Twitter Apps
Please sign in with your Twitter Account to create and maintain Twitter Apps.
Tweet
About Terms Privacy Cookies
© 2018 Twitter, Inc.


Application Management
Twitter Apps
You don't currently have any Twitter Apps.
Create New App
Tweet
About Terms Privacy Cookies
© 2018 Twitter, Inc.


Application Details
Name *
Elastic training
Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.
Description *
Collecting tweet
Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.
Website *
http://placeholder.com
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)
Callback URLs
Where should we return after successfully authenticating? OAuth 1.0a applications must explicitly specify their oauth_callback URL(s) here, as well as include the one of the URLs below in the request token step. To restrict your application from using callbacks, leave this field blank.
Add a Callback URL
Developer Agreement
Yes, I have read and agree to the Twitter Developer Agreement.
Create your Twitter application



2. Let's start by going into server1:

```
ssh server1
```

3. Extract the two folders:

```
tar -zxf elasticsearch-6.3.1.tar.gz
tar -zxf kibana-6.3.1-linux-x86_64.tar.gz
```

4. You can now start elasticsearch.

```
./elasticsearch-6.3.1/bin/elasticsearch -E network.host=0.0
```

5. We should as well extract Logstash.

```
ssh server2
```

Extract logstash:

```
tar -zxf logstash-6.3.1.tar.gz
```

6. In order to make sure that we can use our Twitter tokens without them being visible let's use the Logstash keystore:

```
./logstash-6.3.1/bin/logstash-keystore create
```

You can now add each one of your tokens. Make sure to not paste your token before the prompt tell you so in order to keep your tokens as secret as possible:

```
./logstash-6.3.1/bin/logstash-keystore add CONS_KEY
```

Then copy your consumer key. Reiterate the process for CONS_KEY_SEC (consumer key secret), OAUTH_TO (Access token), OAUTH_TO_SEC (Access token secret)

List all your tokens to make sure you didn't forgot one:

```
./logstash-6.3.1/bin/logstash-keystore list
```

7. We are going to install some Logstash plugins during this class. Since we don't have any SSL certificate we need to remove "https" in the Gemfile.

```
vim logstash-6.3.1/Gemfile
```

Replace **"https://rubygems.org"** by **"http://rubygems.org"**

8. You can now start Kibana on the server1.

```
./kibana-6.3.1-linux-x86_64/bin/kibana --host=0.0.0.0
```

9. We need to add a dataset that we are going to use in this training. In the server1 there is folder, datasets, with a subfolder called snapshot. We are going to use it in order to load our data. Open the **elasticsearch.yml** file (in folder **./elasticsearch-6.3.1/config**) and add the following line:

```
path.repo: "/home/elastic/datasets/snapshot"
```

   Restart your elasticsearch node. To stop it, go into the terminal tab where it is running and press **ctrl + c**, then restart it.

10. Once Elasticsearch is running go into Kibana go into **Dev Tools** and type the following:

```
PUT /_snapshot/tweet
{
  "type": "fs",
  "settings": {
    "location": "/home/elastic/datasets/snapshot"
  }
}
```

   This is defining the repository for the snapshot. Now let's use the snapshot to restore the dataset:

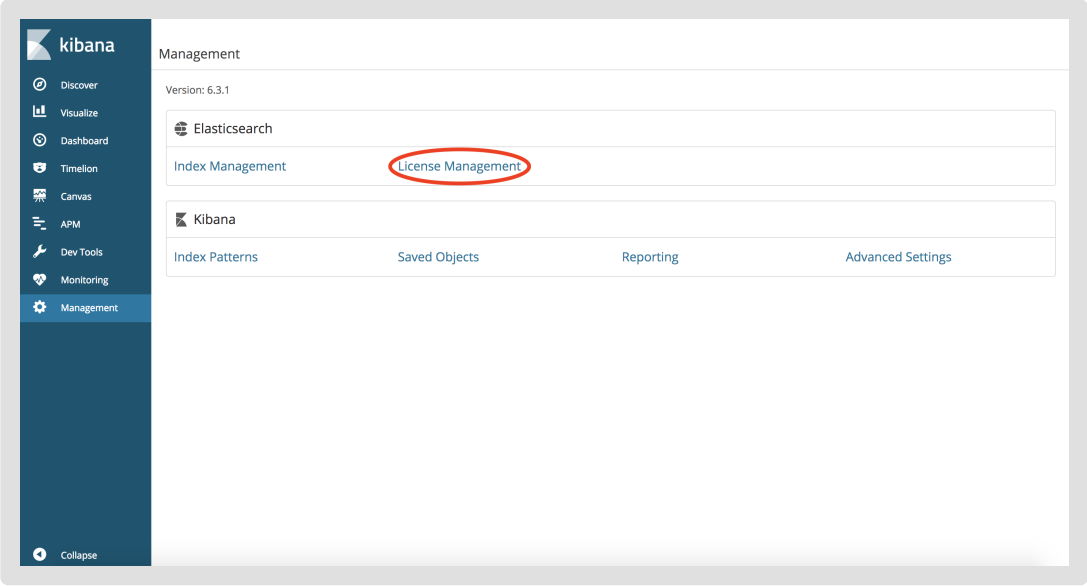
```
POST /_snapshot/tweet/dump/_restore
```

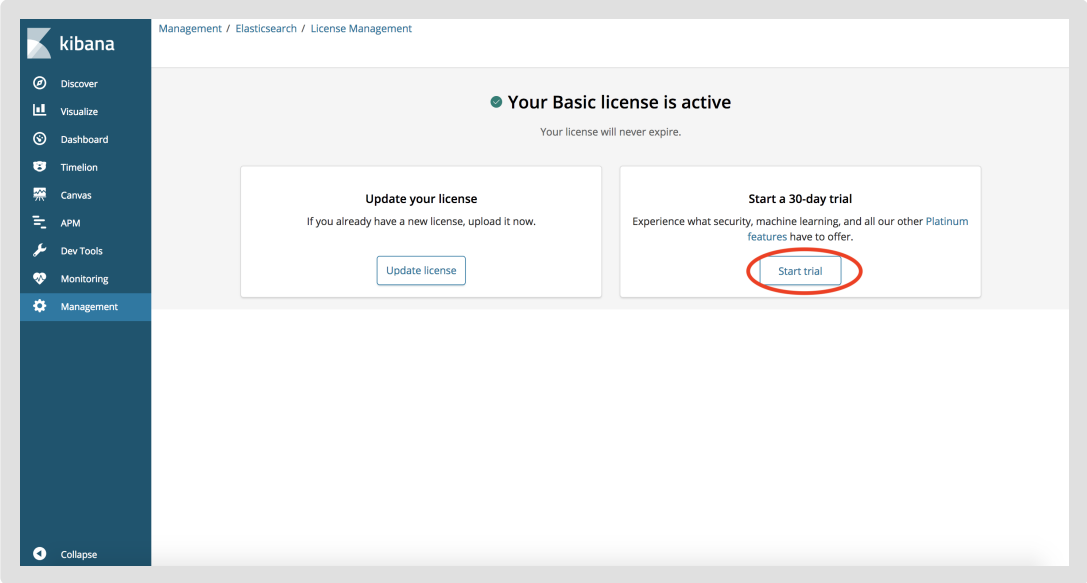
11. In the Kibana console verify if the snapshot worked properly:

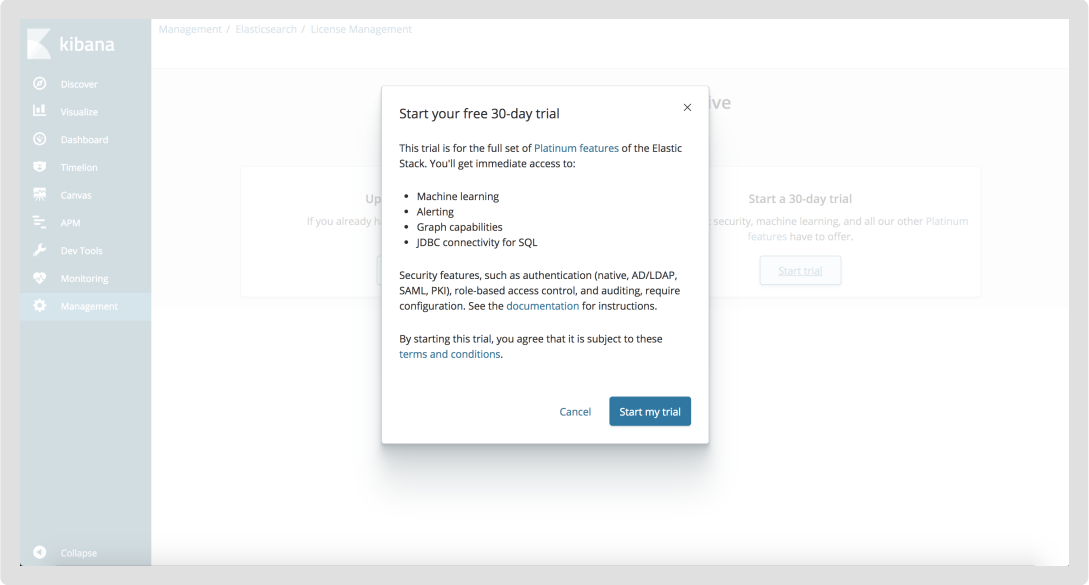
```
GET _cat/indices
```

You should see an index called twitter-ml-clean.

12. We are going to use Elasticsearch Machine Learning in this class. To do so we need to activate the Elasticsearch license.







Summary: In this lab you should have created a twitter application, which should give you access to different tokens and you should have activated the free license and set up your cluster for the next labs!

End of Lab 1

Lab 2: Twitter input

Objective: In this lab, you are going to use the Twitter account you just created in order to collects tweets using different methodologies.

1. In the folder **pipelines** you are going to find a file called **lab2.conf**, edit this file. But before that you will need to ssh into server2.

```
ssh server2
vim pipelines/lab2.conf
```

Let's change the input in order to collect random tweets.

```
input {
  twitter {
      consumer_key => "${CONS_KEY}"
      consumer_secret => "${CONS_KEY_SEC}"
      oauth_token => "${OAUTH_TO}"
      oauth_token_secret => "${OAUTH_TO_SEC}"
      use_samples => true
      full_tweet => true
      type => "tweet" # Apply a new field to the JSON colle
      # it is pretty usefull when working with multiple inpu
  }
}
```

2. You probably realised that we used prune in the pipeline, so we need to install it. Hopefully a single line will do the trick:

```
logstash-6.3.1/bin/logstash-plugin install logstash-filter-p
```

It should take only few seconds. Once it's done, launch the pipeline.

```
./logstash-6.3.1/bin/logstash -f pipelines/lab2.conf
```

3. Let's see if our pipeline is collecting tweets properly. Wait a little bit and then Go in **Kibana** and count the number of documents present in the **twitter** index.

Hide answer:

```
GET twitter/_count
```

4. Let's stop the pipeline and collect tweets based on other criterion. Delete the **twitter** index.

   Hide answer:

   In the terminal where your pipeline is running press ctrl + c.

   In the Kibana console:

   ```
   DELETE twitter
   ```

5. Let's collect tweets from specific area. To do this we are going to use the location contained in a tweet. We are going to collect tweets from the US west coast, UK and Australia. For every box we need to define 2 points, The SW one and the NE one.
   - For the US west coast:
     - SW: lon: -124.264886 lat: 31.598581
     - NE: lon:-117.570610 lat: 48.536975
   - For the UK:
     - SW: lon: -12.006055 lat: 49.984609
     - NE: lon: 1.220102 lat: 59.406623
   - For Australia:
     - SW: lon: 112.310377 lat: -39.696116

- NE: lon: 154.248758 lat: -10.751004

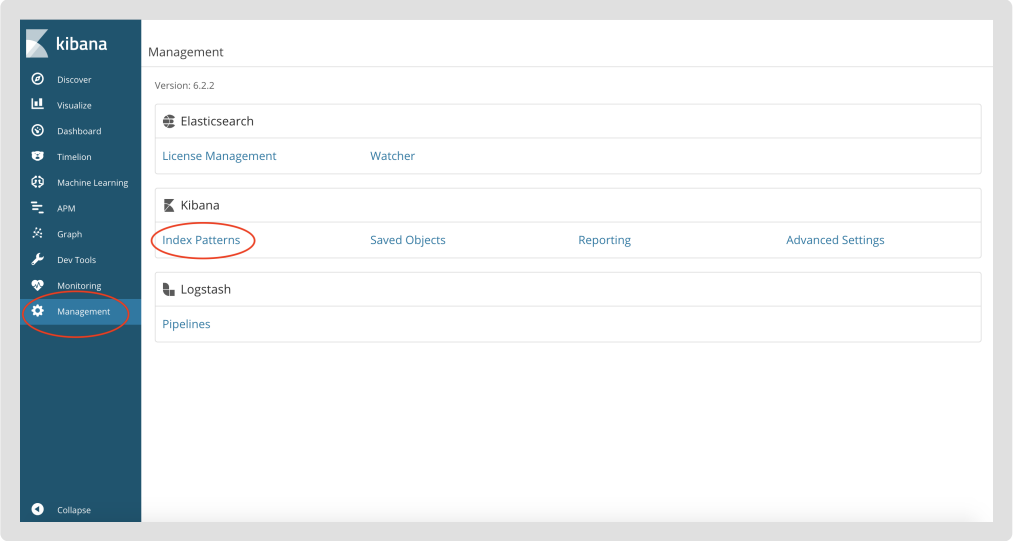
  Update the configuration file to collect tweets from those area.

Hide answer:

```
input {
  twitter {
      consumer_key => "${CONS_KEY}"
      consumer_secret => "${CONS_KEY_SEC}"
      oauth_token => "${OAUTH_TO}"
      oauth_token_secret => "${OAUTH_TO_SEC}"
      locations => "-124.264886,31.598581,-117.570610,4
      full_tweet => true
      type => "tweet"
  }
}
```

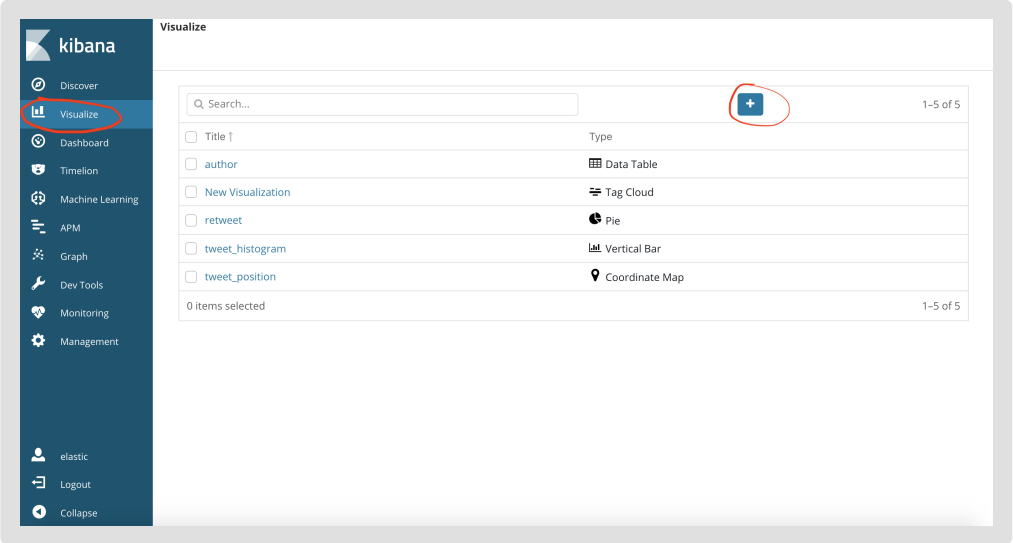
6. Let's verify if the tweets that we are collecting really comes from the desired location. To do that we are going to build a coordinate map in **Kibana**. All the points should be on the US west coast, Australia or UK.

   a. The first step will be to create an index pattern, go into management and then select index pattern:

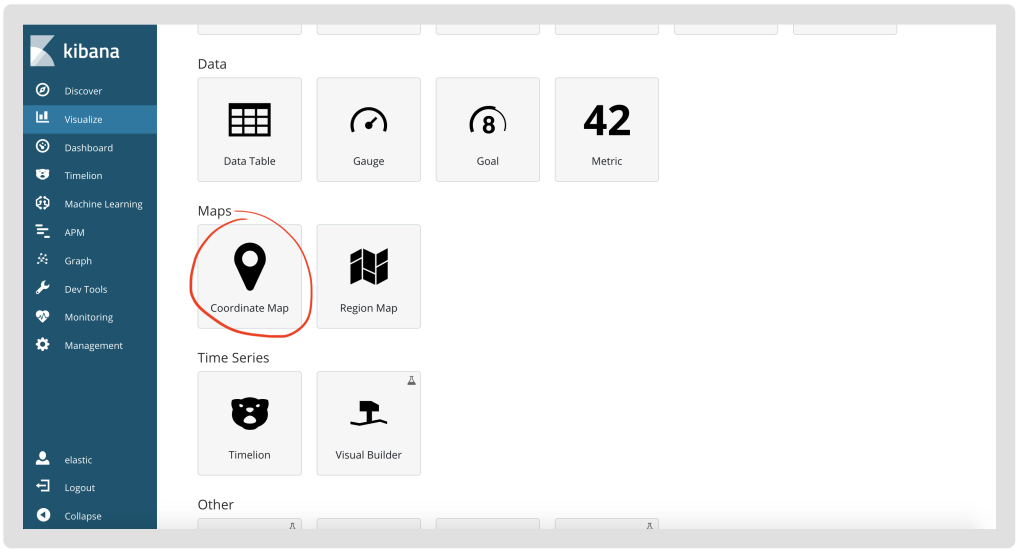


b. Type the name of the index pattern **twitter** and select the field **tweet_created_at** as the timefield:

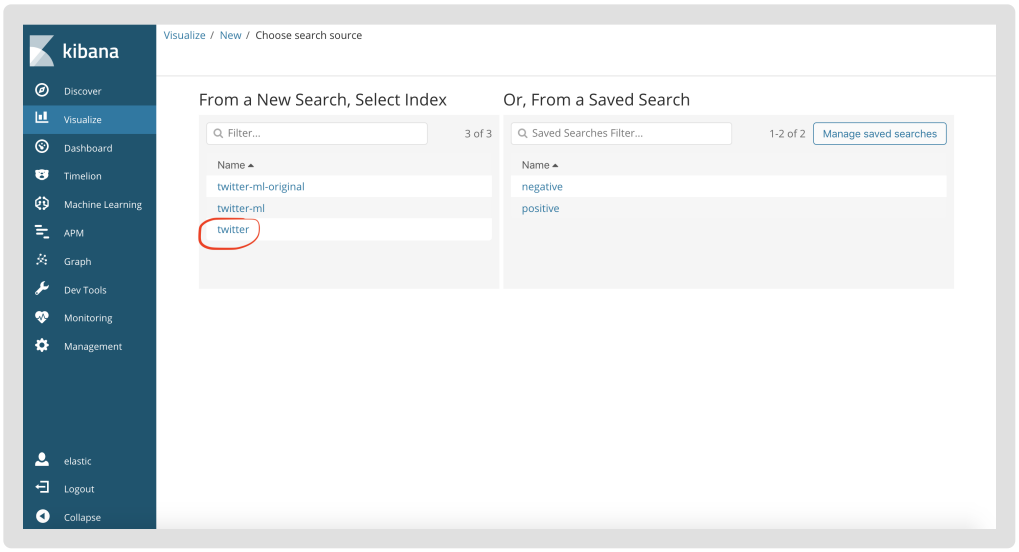
c. Click on **Visualize** and then the **+** button.



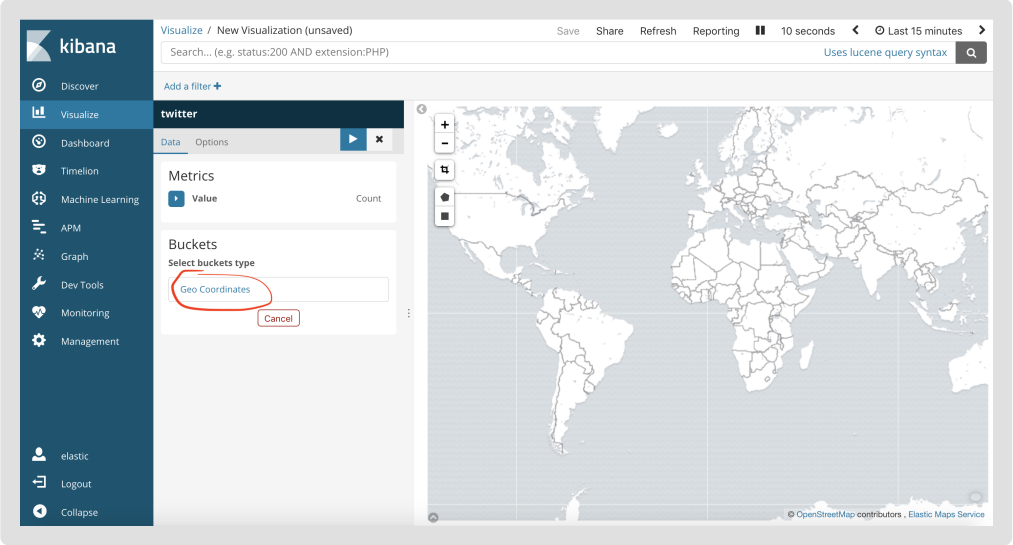
d. Choose **Coordinate map**



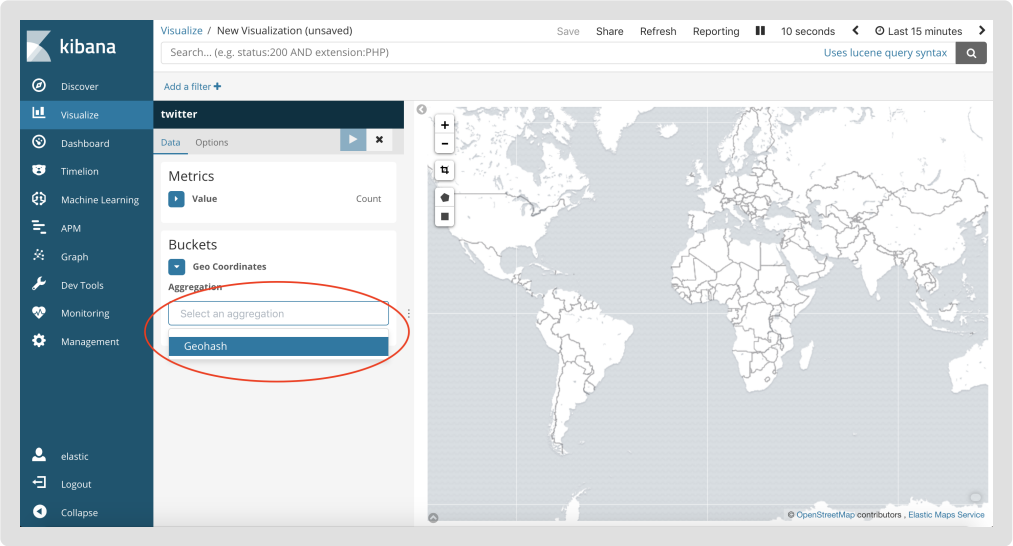
e. Select **twitter** index template



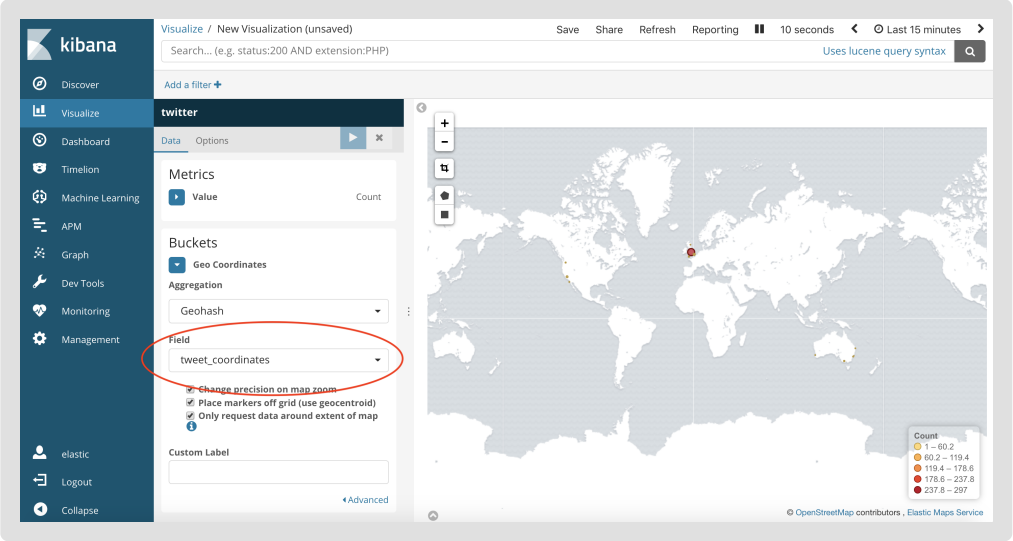
f. Click on **Geo Coordinates**



g. Select the aggregation **Geohash**



h. Select the field **tweet_coordinates**



i. Click on the **play button**

All the point should be on the US west coast, UK and Australia.

7. You can now stop the pipeline using **ctrl + c** in the terminal.

8. Now let's try to collect tweets from a specific user. Go on this website: http://mytwitterid.com and type your twitter pseudo. It will give you an ID. Use this ID to build a pipeline that will collect your tweets. You can find your id in the twitter app that you created earlier. Once you have saved the configuration file, start the pipeline.

Hide answer:

```
input {
  twitter {
      consumer_key => "${CONS_KEY}"
      consumer_secret => "${CONS_KEY_SEC}"
      oauth_token => "${OAUTH_TO}"
      oauth_token_secret => "${OAUTH_TO_SEC}"
      follows => ["YOUR_ID"]
      full_tweet => true
```

```
        type => "tweet"
    }
}
```

9. Now go on twitter and tweet the following:

   " **Testing the twitter input for Logstash! https://www.elastic.co/guide/en/logstash/current/plugins-inputs-twitter.html #ElasticTraining #Elastic** ".

10. Now search for your tweet into your Elasticsearch using your ID and the field **author_id**.

Hide answer:

```
GET twitter/_search
{
  "query": {
    "match": {
      "author_id": "YOUR_TWITTER_ID"
    }
  }
}
```

11. Let's try to collect tweets that contained at least one of the two following word: **HBO** or **Netflix**.

Hide answer:

```
input {
  twitter {
    consumer_key => "${CONS_KEY}"
    consumer_secret => "${CONS_KEY_SEC}"
    oauth_token => "${OAUTH_TO}"
    oauth_token_secret => "${OAUTH_TO_SEC}"
    keywords => ["Netflix", "HBO"]
```

```
    full_tweet => true
    type => "tweet"
  }
}
```

12. Start your new pipeline and check if your are collecting tweets by counting the number of document in your index.

Hide answer:

```
GET twitter/_count
```

Summary: Well done!! In this lab, you used Logstash to collect tweets using different kind of filter. You also learned how to use Kibana to check if your way of collecting tweets worked properly.

End of Lab 2

Lab 3: External API and REST filter

Objective: In this lab, you will see how to build the external classifier and then how to send events to the external API using the rest filter.

1. First let's see what are the data is that we are going to use to train our algorithm. We are going to use a dataset included in the

NLTK. The first step will be to build our virtual environment for python that will include the different library that we are going to use.

Go into the **sentiment_api** folder in the server2:

```
cd sentiment_api
```

Now we will need to build the virtual environement:

```
virtualenv -p python3 venv
source venv/bin/activate
```

2. The first line will build a virtual environment to avoid us having to install the library globally. The second command line will activate the virtual environement. Now you can install the different libraries that we are going to use.

```
pip install -r requirements.txt
```

This will automatically download and install all the libraries include in the requirements file.

Now that we have all the needed libraries, let's start exploring the dataset.

```
python3
```

The above command will start a python shell. We need to import the library that contains the dataset.

```
from nltk.corpus import movie_reviews
import nltk
```

```
nltk.download('movie_reviews')
nltk.download('punkt')
```

The corpus being loaded, let's start to look at it. Movie_reviews is an object that contains different attributes and methods. We can use them to display the different labels present in the dataset.

```
movie_reviews.categories()
```

This line of code is going to output the following: ["pos", "neg"], "pos" is standing for positive and "neg" is standing for negative. Now if you want to display all the reviews that are positive you can type the following. If I want to retrieve all the positive review I can type:

```
movie_reviews.fileids("pos")
```

This will display all the files that contain a positive review. Note that every file start with "pos". Let's look at one of them:

```
file = movie_reviews.fileids("pos")[0]
movie_reviews.raw(file)
```

Now you can see what's the content of the first positive review. Let's count how many positive and negative review we have:

```
print("Number of positive review:", len(movie_reviews.fileid
print("Number of negative review:", len(movie_reviews.fileid
```

Ideally you would prefer a dataset that will reflect the real-world proportions. Let's now train an algorithm that will take this previous datasets in order to label unseen data. The Text blob

library can use a classifier from NLTK: **NaiveBayesAnalyzer**. By default this classifier is going to use the previous corpus.

```
from textblob import TextBlob
from textblob.sentiments import NaiveBayesAnalyzer
blob = TextBlob("This movie is awesome", analyzer=NaiveBaye
blob.sentiment
```

So if I type "This movie is awesome" we are going to compute the probability of this text to be positive or negative depending on the word from the previous dataset. The output is:

Sentiment(classification='pos', p_pos=0.6928102445189113, p_neg=0.30718975548108857)

It tells you that the probability of being positive is around 0.69 and the probability of being negative is 0.31, so the text will be labeled as positive, since the probability is higher.

You can see that every time you are using the commands:

```
blob = TextBlob("This movie is really bad", analyzer=NaiveB
blob.sentiment
```

It will take few seconds to be executed. It is because every time it will need to recompute the probabity based on the dataset, so what you can do is use a Blobber factory, that will help dry your code (https://en.wikipedia.org/wiki/Don%27t_repeat_yourself).

```
from textblob import Blobber
text_blob = Blobber(analyzer=NaiveBayesAnalyzer())
text_blob("This is a test").sentiment
```

Now only the first execution will take time. The others will be much faster.

Maybe, you would prefer using another dataset in order to do the sentiment analysis. An option can be to use your own data. So you are going to build your own classifier:

```
from textblob.classifiers import NaiveBayesClassifier
train = [('This movie is just... Wow', 'pos'),
  ('I just wasted my time', 'neg')]
classifier = NaiveBayesClassifier(train)
classifier.classify("Wow this movie is incredible!")
```

Keep in mind that the training dataset is extremely important. Don't ever expect an algorithm with bad training data to give you good results. Garbage in, Garbage out.

```
exit()
```

Exit the python shell.

The code for the API has already been written down. Let's just start the API.

```
python sentiment_server.py
```

Let's have a look at **sentiment_server.py**. The method that is interesting is **predict_a_tweet**, it expects a JSON with a field **submit**. The content of that field will then be send to a function **clean_text** to be cleaned of token like **@username**. Then using the trained algorithm we compute the probability of the tweet to be positive. If this value is higher than **0.55** the label will be positive, if the probability is between **0.55 and 0.45** then we are not sure enough and the label will be neutral, lastly if the probability is lower than **0.45** then the label is negative. We are

sending back a JSON containing two fields: **sentiment** and **sentiment_score**, the first field is the label: "positive", "negative" or "neutral" the second field is the probability of being positive.

Our webserver is up and running. It is now time to set up Logstash in order to send the tweets to the API. The first step will be to install the rest filter.

```
logstash-6.3.1/bin/logstash-plugin install logstash-filter-r
```

3. In the file **lab3.conf**, try to find the rest filter in the configuration file and edit it accordingly. We need to specify that Logstash needs to send the field **tweet_content** to the URL: **http://localhost:5000/predict** using a **POST** request. And we want to put the result in a field called **rest**.

Hide answer:

```
rest {
  request => {
    url => "http://localhost:5000/predict"
    method => "POST"
    params => {
      "submit" => "%{tweet_content}"
    }
    headers => { "Content-Type" => "application/json" }
  }
  target => 'rest'
}
```

Summary: Well done!! Now every tweet is send to the API to get a label, and has been enriched in Logstash.

End of Lab 3

Lab 4: Elasticsearch Machine Learning

Objective: Objective: In this last lab, we are going to see how we can use the Elasticsearch build-in machine learning features in order to detect anomalies in the data. Those could be a certain amount of tweets that are not normal for this time of the day, or a user that is tweeting too much in comparison with other users.

1. Do a **match_all** query on the **twitter-ml-clean** index.

   Hide answer:

   ```
   GET twitter-ml-clean/_search
   {
     "query": {
       "match_all": {}
     }
   }
   ```

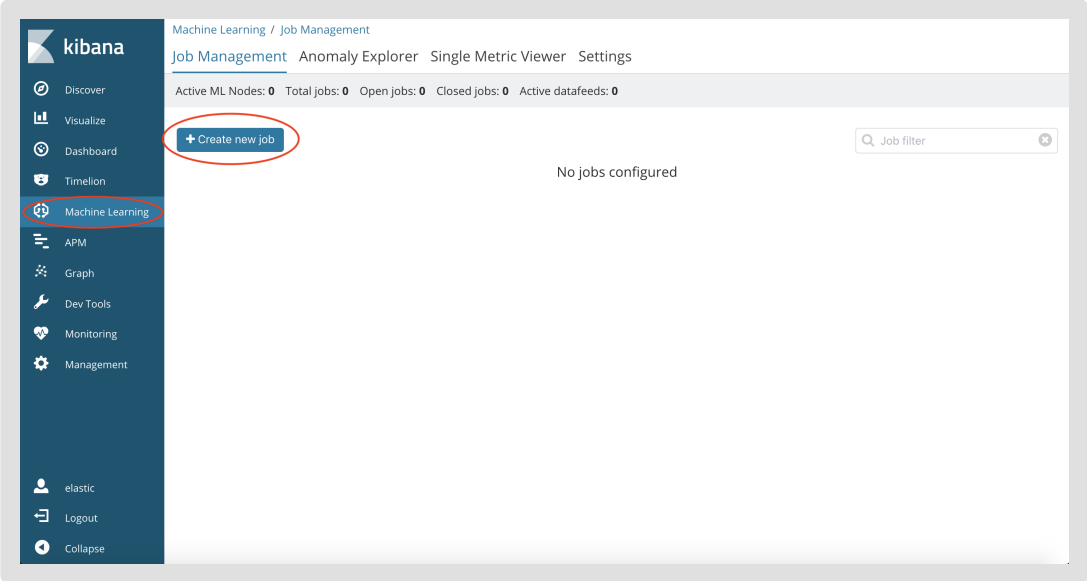
   Or you could simply do:

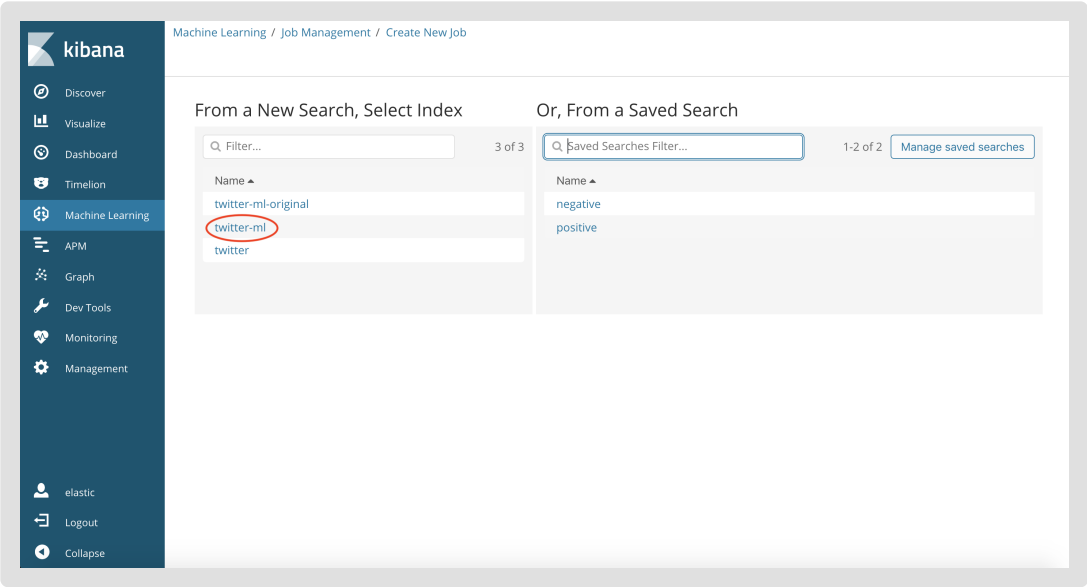
   ```
   GET twitter-ml-clean/_search
   ```

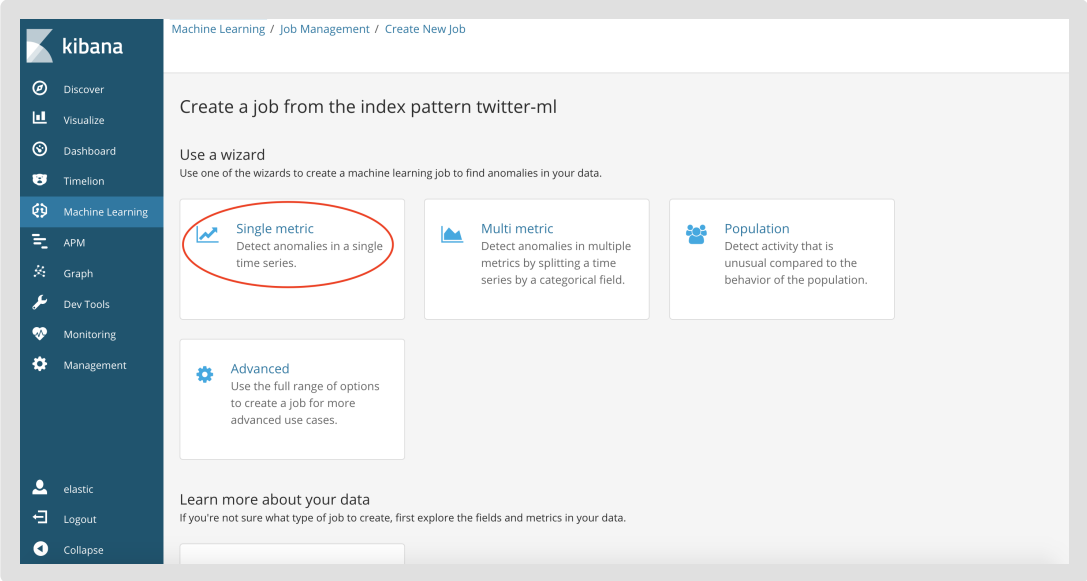
2. Let's run our first Machine Learning job. We want to know if at some point in time the number of tweet over time present some anomaly in its pattern. Go into **Machine Learning**, click on **Create new job**, select the **twitter-ml-clean** index pattern,

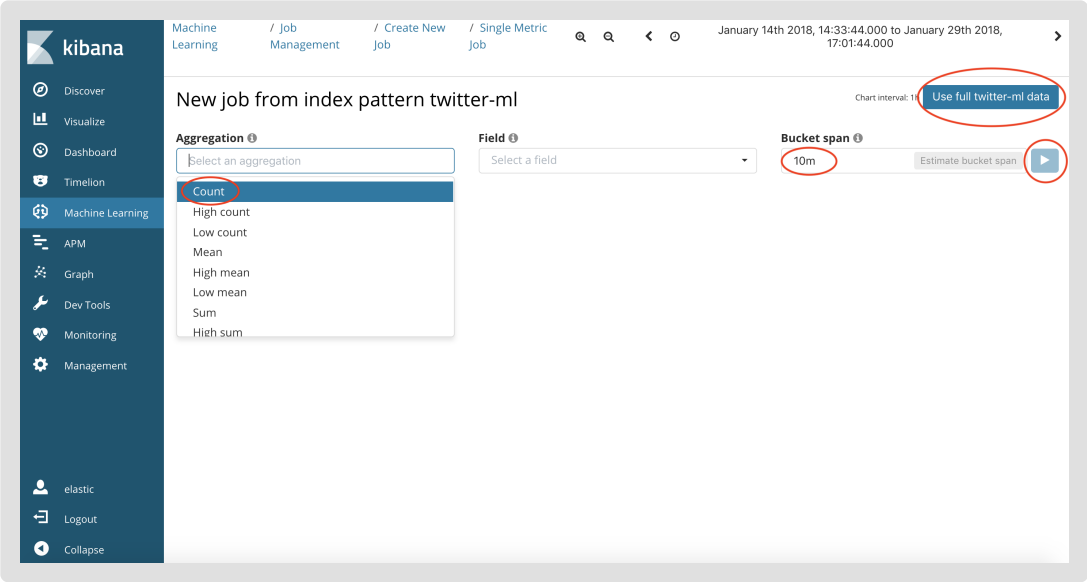
choose **Single metric**. You can now configure the job. We want to group our documents by bucket of **10 minutes** and do a **count** on those bucket. Click on the button **use full twitter-ml-clean data**. Give a **name** to your job and press the button **Create job**, and then view a job: **View Results**

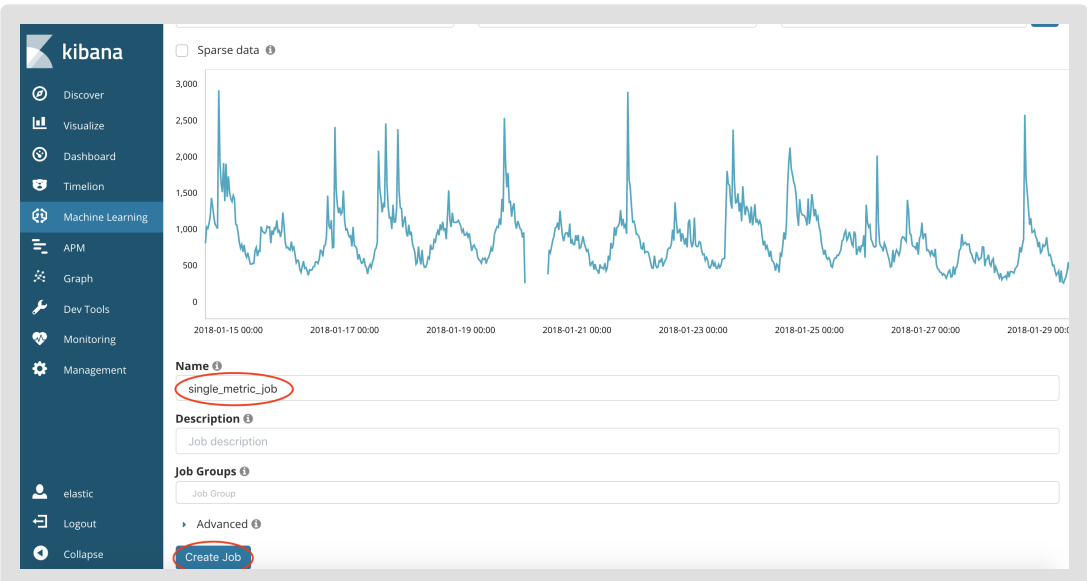
Hide answer:

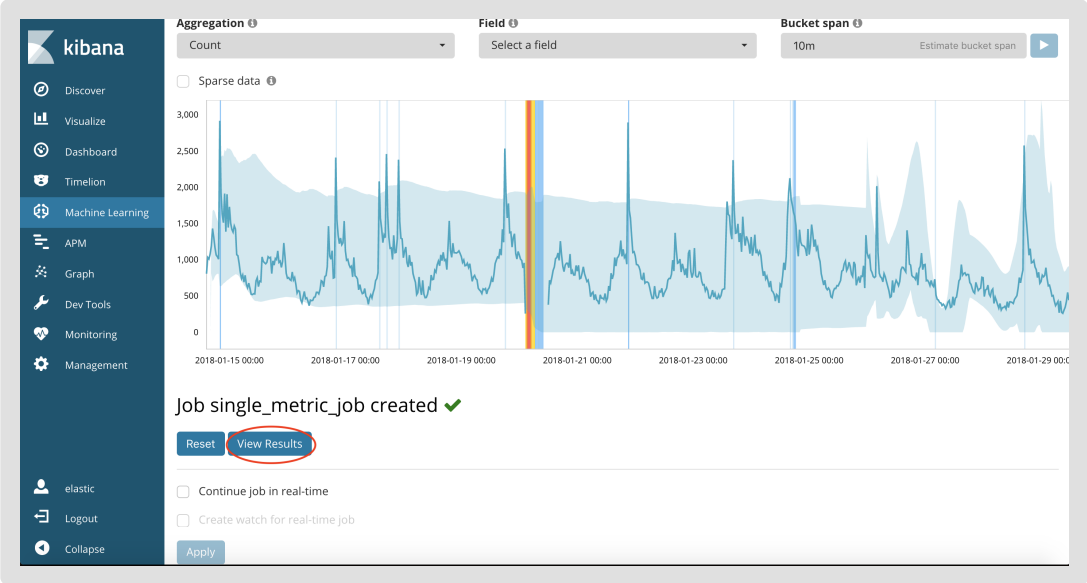





kibana
Machine Learning / Job Management / Create New Job
Discover
Visualize
Dashboard
Timelion
Machine Learning
APM
Graph
Dev Tools
Monitoring
Management
elastic
Logout
Collapse
Create a job from the index pattern twitter-ml
Use a wizard
Use one of the wizards to create a machine learning job to find anomalies in your data.
Single metric
Detect anomalies in a single time series.
Multi metric
Detect anomalies in multiple metrics by splitting a time series by a categorical field.
Population
Detect activity that is unusual compared to the behavior of the population.
Advanced
Use the full range of options to create a job for more advanced use cases.
Learn more about your data
If you're not sure what type of job to create, first explore the fields and metrics in your data.


kibana
Machine Learning / Job Management / Create New Job / Single Metric Job
January 14th 2018, 14:33:44.000 to January 29th 2018, 17:01:44.000
New job from index pattern twitter-ml
Use full twitter-ml data
Aggregation
Select an aggregation
Count
High count
Low count
Mean
High mean
Low mean
Sum
High sum
Field
Select a field
Bucket span
10m
Estimate bucket span


kibana
Sparse data
3,000
2,500
2,000
1,500
1,000
500
0
2018-01-15 00:00
2018-01-17 00:00
2018-01-19 00:00
2018-01-21 00:00
2018-01-23 00:00
2018-01-25 00:00
2018-01-27 00:00
Name
single_metric_job
Description
Job description
Job Groups
Job Group
Advanced
Create Job



3. Play with the time picker to navigate into your timeline. Why is the blue area at the end of the timeline different than at the beginning?

Hide answer:

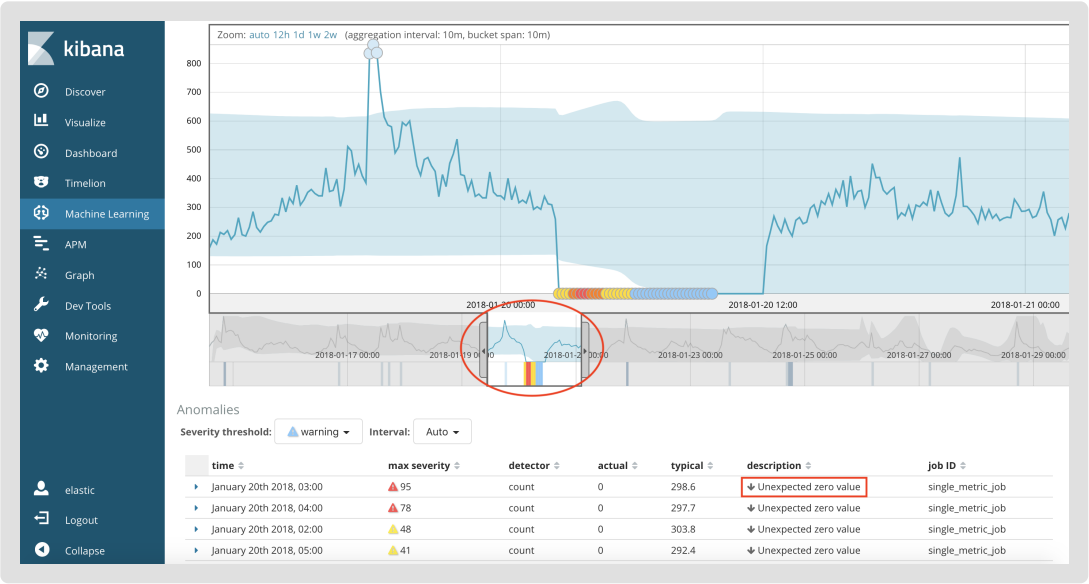
The time picker is surrounded by the red square. The blue area is the area in which your model expect the time series that you are analyzing to be in. At the beginning the model doesn't have enough data to define properly this area, at a certain point in time you have enough data to have an accurate model, so the blue area fit more accurately your time series.

4. Find the critical anomalies (The one in red). What are the description of those anomalies?

Hide answer:

Select the part of the time series that contains the critical anomalies and go into the anomalies listing. You can see on the right the reason of the anomalies. The anomaly should be "Unexpected zero value". Those anomalies are there because

the machine that was collecting data was down during this period of time.



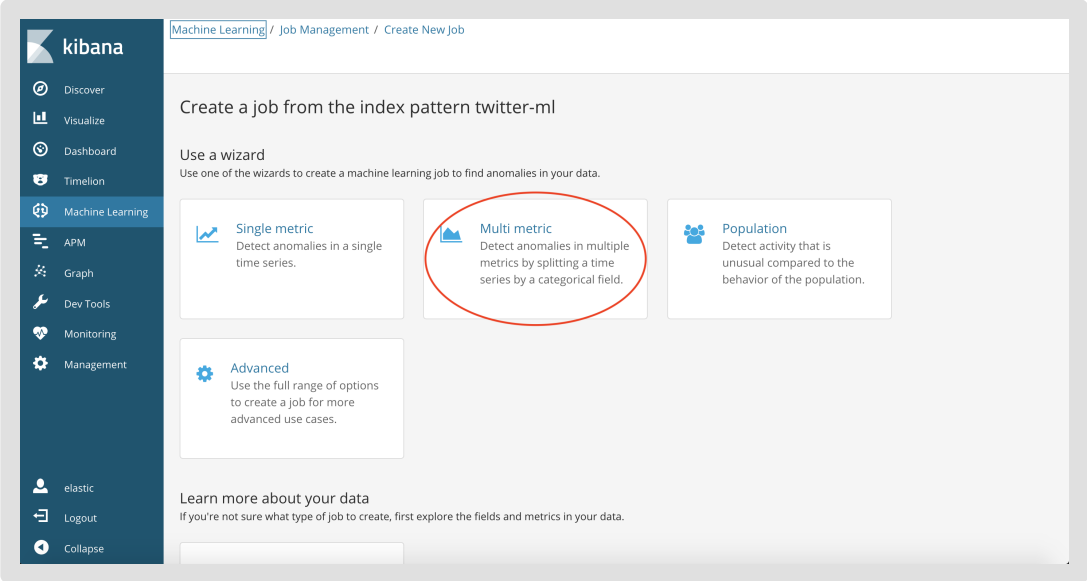
5. Go to the last anomaly what is the "description" of this anomaly?

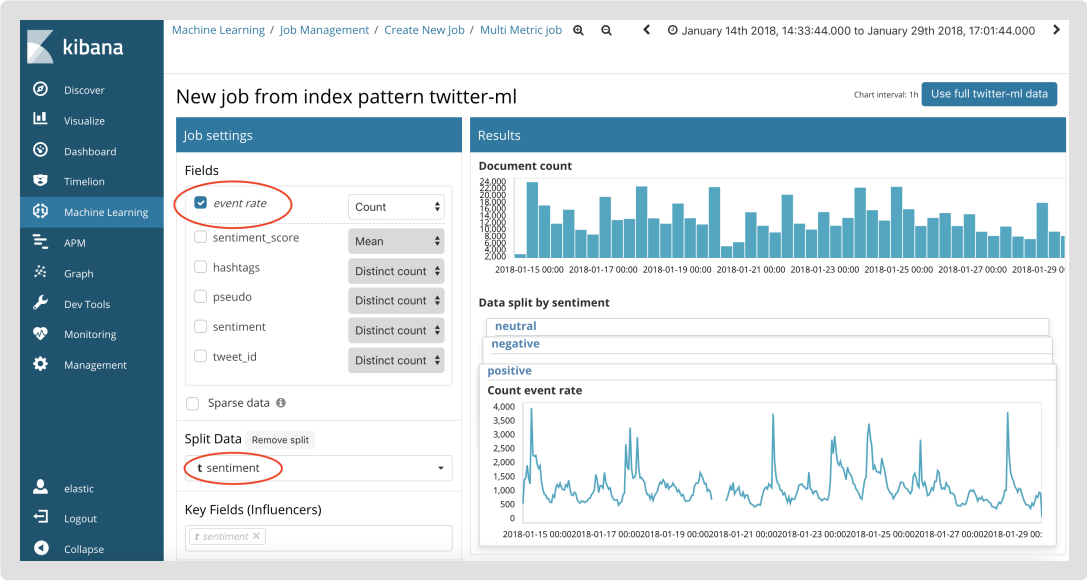
Hide answer:

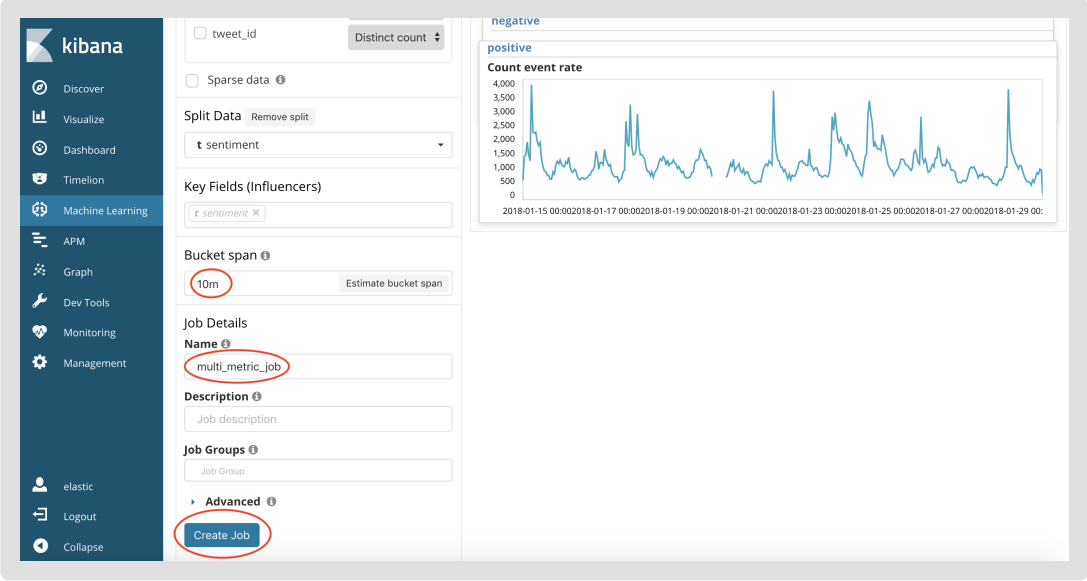
This anomaly is due to values in the time serie 4 times higher than what it should "normally" be. That's means that a lot of tweets has been tweeted around this time.

6. We can identify when there is a change in the frequency of the events. Let's now leverage the sentiment that are present in our tweets. Redo the same steps than before but this time use the **multimetric job** instead. Select the field **event rate** and choose to build multiple time series based on the sentiment field. Group the documents by bucket of **10 minutes**, give a name to the job and then create the job.

Hide answer:


kibana
Machine Learning / Job Management / Create New Job
Discover
Visualize
Dashboard
Timelion
Machine Learning
APM
Graph
Dev Tools
Monitoring
Management
elastic
Logout
Collapse
Create a job from the index pattern twitter-ml
Use a wizard
Use one of the wizards to create a machine learning job to find anomalies in your data.
Single metric
Detect anomalies in a single time series.
Multi metric
Detect anomalies in multiple metrics by splitting a time series by a categorical field.
Population
Detect activity that is unusual compared to the behavior of the population.
Advanced
Use the full range of options to create a job for more advanced use cases.
Learn more about your data
If you're not sure what type of job to create, first explore the fields and metrics in your data.

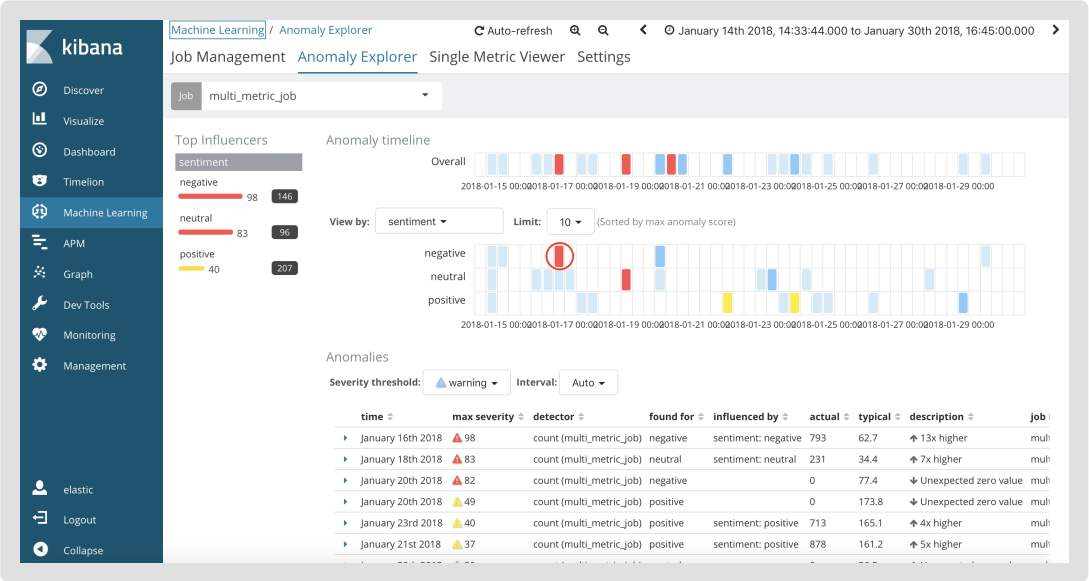

kibana
Machine Learning / Job Management / Create New Job / Multi Metric job
January 14th 2018, 14:33:44.000 to January 29th 2018, 17:01:44.000
New job from index pattern twitter-ml
Chart interval: 1h
Use full twitter-ml data
Job settings
Fields
event rate
Count
sentiment_score
Mean
hashtags
pseudo
sentiment
tweet_id
Distinct count
Sparse data
Split Data
Remove split
t sentiment
Key Fields (Influencers)
Results
Document count
Data split by sentiment
neutral
negative
positive
Count event rate


kibana
tweet_id
Distinct count
Sparse data
Split Data
Remove split
t sentiment
Key Fields (Influencers)
Bucket span
10m
Estimate bucket span
Job Details
Name
multi_metric_job
Description
Job description
Job Groups
Job Group
Advanced
Create Job
negative
positive
Count event rate

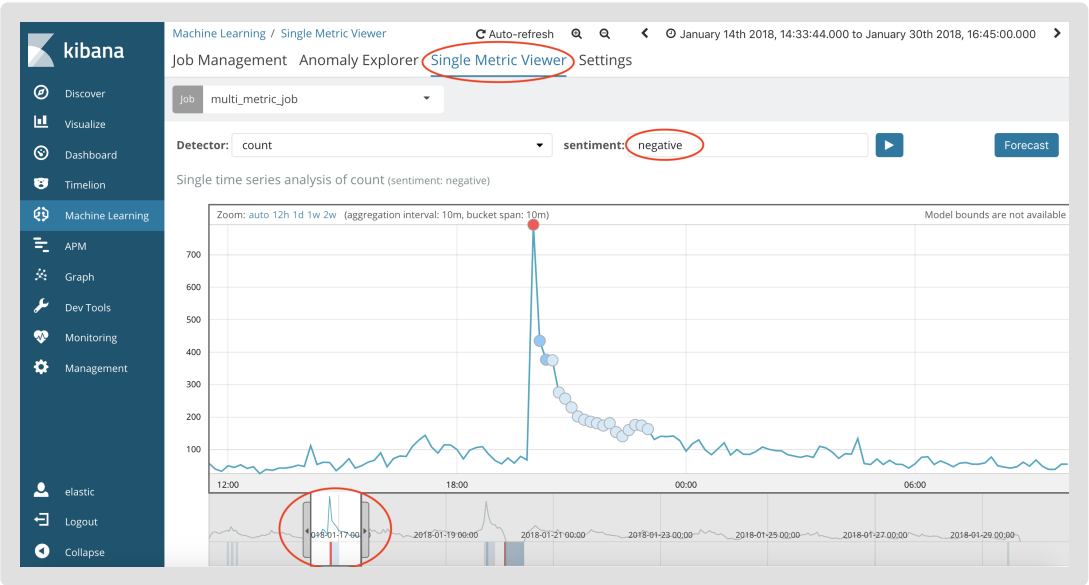
7. Click on **View Results**. The first page you see give you a summary of the different anomalies overall and a summary of the anomalies per time series. Let's say that you are interested in the negative tweets that you have been collecting. Find the first critical anomaly, from the "negative" time series?

Hide answer:

You can simply click on the anomaly:



Or you could click on the **Single metric viewer**, select the time series: **negative** and navigate directly to the anomaly.

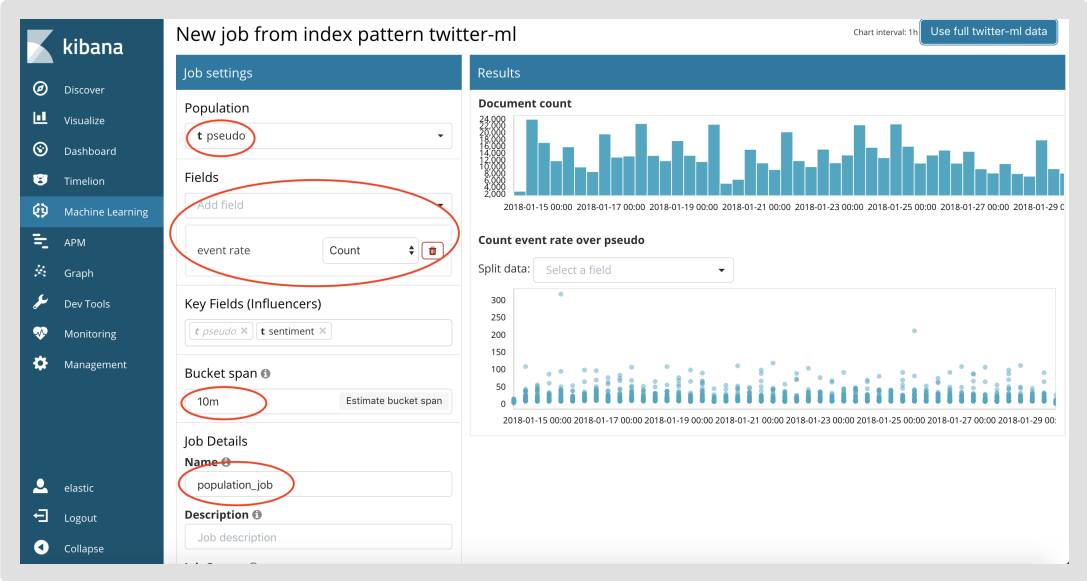


8. What is the **max severity**, the **actual** number of event in the bucket, the **typical** number of events and the description of the anomaly?

Show answer:

9. Let's now try to detect some users that have an abnormal behaviour in our data. To do so you are going to run a "Population" job, the entities that we want to monitor are the user's "pseudo". Use the **Population** job wizard, let's set the population to be **pseudo**, the field to be the **event rate**, a bucket of **10 minutes** and give a name to the job.

Hide answer:



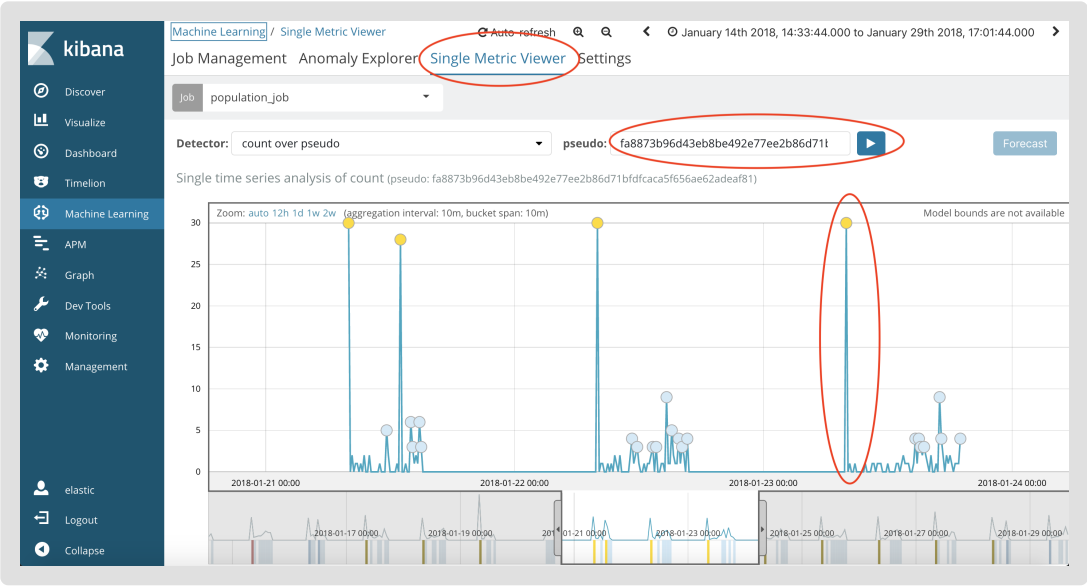
10. This job will take more time than the previous one. Once it's ready click on **View Results**. You should see a summary of the main influencors and their anomaly. Try to find the user with the following pseudo:
fa8873b96d43eb8be492e77ee2b86d71bfdfcaca5f656ae62adeaf81
What is specific about this user?

Hide answer:

This user seem to have a regular pattern, and every time this user is actif, he is generating anomalies, meaning that they tweet much more than other users. image:lab4_10_1.png

11. Go into the single metric viewer and look at this user time series. When does this user publish new tweets? What's special about the way he is publishing things?

Hide answer:



When you are looking at his data, the user seem to often start tweeting around 8-9AM and stop tweeting around 5-6PM, which seem like an "office schedule". Another interesting aspect is that this user when they start tweeting seems, they tweet a lot. Maybe information that arrived during the night need to be tweeted as soon as possible, so everything is done at the first hour.

Summary: Well done!! That was the last lab! Now you know how to collect tweets, enrich those tweets and run machine learning algorithms on top of it!

End of Lab 4