accusoft®

Log Analysis with Elasticsearch and Kibana

Aleksey Kondratov
Software Engineer, PrizmDoc Suite

Overview

From time to time, I have had to review and analyze various logs. Over time, log file size may reach >100 MB, which is problematic for analysis. Previously, I used Notepad++ and Autogrep for searching and querying logs. Sometimes, I created custom parsers in Python when a deep analysis was required. Then, I used Excel for displaying results. Creating custom solutions can take too much time and cause headaches. Recently, I learned about the Elasticsearch platform. Immediately, I recognized it was an excellent choice for this kind of analysis. In this blog, I will describe the minimum steps required to create some basic researching and analysis for a log which contains JSON entries. I will not cover advanced concepts of Elasticsearch such as working with Aggregations and Analyzers. This article is an introduction to a great tool. So let's start!

Goals

1. Set up and run Elasticsearch
2. Generate a log sample for analysis with Elasticsearch
3. Create an index and mapping in Elasticsearch
4. Upload data to Elasticsearch
5. Set up Kibana and use it for analysis

Prerequisites

The following apps need to be installed before starting:

- Docker app with Engine which is no older than version 18.09.1
  - (This is my local version where I run all apps).
- Python3
- Curl app (it may be specific to your platform, check here).
  - This is not required; you are free to use any application that generates rest-api requests like Postman, for example.
- A terminal or console for running apps from the command line

What Is Elasticsearch?

Elasticsearch is a search engine which has a lot of capabilities for data analysis and analytics. It also can be used as a NoSQL database. The Elasticsearch (ES) uses Apache Lucene for searching and is written in Java. You can find more details on the official ES web page here.

Set Up Elasticsearch with Docker

It would be redundant to set up ES from packages and then make additional configurations to run it. The easiest and fastest way to set up Elasticsearch with Docker is to use an ES Docker image. It is available in the Docker hub here. Run the following commands to download and run Elasticsearch:

docker pull docker.elastic.co/elasticsearch/elasticsearch:6.7.1

docker run -p 9200:9200 -p 9300:9300 -e “discovery.type=single-node” docker.elastic.co/elasticsearch/elasticsearch:6.7.1

Note: The latest version of ES is 7.0.0, but for my examples in this blog, I’m using version 6.7.1. This is the previous version and I’m using it because it seems to be more stable. My experience has shown me that the latest version always has some unpredictable problems.

To check if Elasticsearch has started, run the following command:

curl -X GET localhost:9200/_cat/health

You should see something very similar to the following:

docker-cluster yellow 1 1 7 7 0 0 5 0 - 58.3%

You may ask, "What does the 'yellow' status" mean?" The answer is very simple. Elasticsearch is a cluster platform; it runs on several nodes. If it is run on one node, it does not make backup copies of existing indexes. When these backup copies are absent, then it reports a yellow status. If you are interested in learning more of the details, there is a good explanation in this article. Our Elasticsearch is running on a single node, so the "yellow" status (check the Docker command where the container starts) is OK and it is ready. Let's define basic terms used by ES: index, mapping, and document. To understand this better, we can compare it with terms from a Relational Database. Let's compare the following:

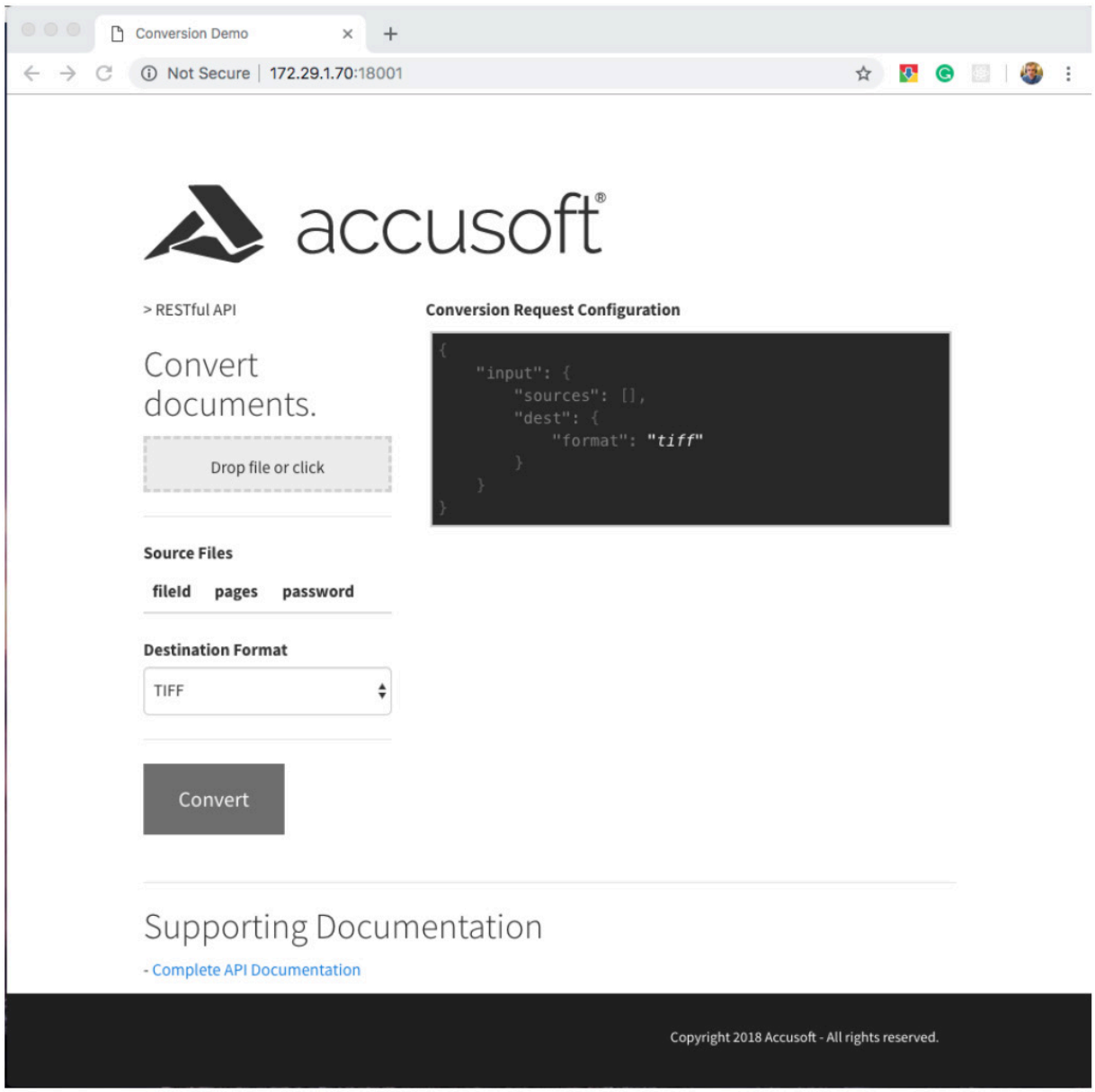
- **Index** - This is the same thing as a table in the database. Regarding our case, this is the entire log we are going to save in ES.
- **Mapping** - It looks like a schema definition, and it includes type definitions for all fields used by the index. Also, it may contain additional parameters which allows ES to store, ignore, or make indexing for a field of the index (a.k.a., table).
- **Document** - This is a JSON object which is stored in ES. It looks like a row in the table. For our case, it will be just a single line from the log file.

Getting a Log File for Analysis

I'm going to take a real log for analysis from the product PrizmDoc Viewer. You may request a trial edition here. It allows conversions for multiple file types (CAD, PDF, MS Office formats) to PDF, SVG, and raster images (PNG, JPEG, TIFF). Check out the online demo. I used a standalone version which is provided as a CCS sample with PrizmDocServer.

The idea is to take the log from the Content Conversion Service (CCS) as this service is working behind the scenes and suggests a public RESTful API for conversion. If this topic is interesting, check out this additional information. The demo sample utilizes this API and makes it available for the end client.

I installed the PrizmDoc Server on my VirtualBox Windows instance, ran the CCS Demo sample, and made several conversions from PDF and doc formats to raster and PDF ones.



Then I grabbed the ContentConversionService.log from the "log" folder under the "C:\Prizm" installation directory. It helps to review the structure of the log we are going to use as a sample:

- The whole log file has a ndjson format. A file by this format is a collection of JSON objects, separated by newlines '\n'. In a simple way, it looks like this:
  {"name":"ContentConversionService","pid":2134, "gid":"Ldr..",...}
  {"name":"ContentConversionService","pid":2134, "gid":"CVHX..",...}
  {"name":"ContentConversionService","pid":2134, "gid":"JFx..",...}

  The reason I took this log for analysis is because it has the structure which is almost ready to push to Elasticsearch. Note that it is possible to use any other log which has the ndjson format.

- There is a small problem with the log structure because it has the field “src” which may be a simple string value in one case and a JSON object on another one:

  “src”:”C:\\Prizm\\cache\\WorkfileCache\\hbyqgOZEX6PuThiE3duV9w\\WorkfileContents.pdf”
  *“src”:[[“path”:”C:\\Prizm\\cache\\WorkfileCache\\hbyqgOZEX6PuThiE3duV9w\\WorkfileContents.pdf”,”pages”:null,”password”:”******”]]*

- This inconsistency provokes ES to fail to accept such entries because it does not support fields with variable complex types. I fixed it by manually updating the entries:

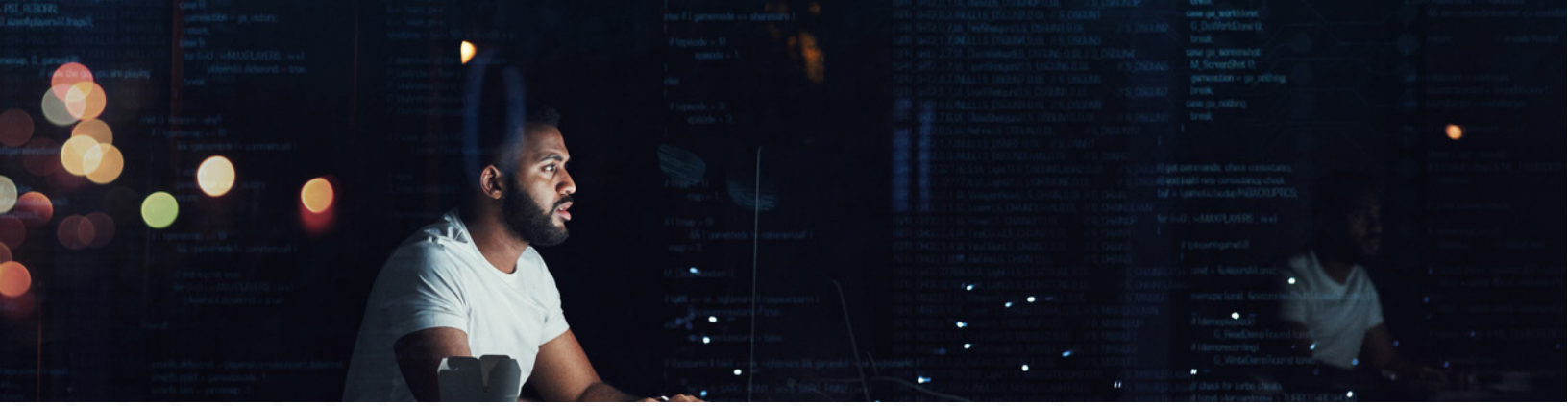
  “src”:[[“path”:”C:\\Prizm\\cache\\WorkfileCache\\hbyqgOZEX6PuThiE3duV9w\\WorkfileContents.pdf”]]

Pushing Data to Elasticsearch

At this point, it looks like everything is ready for uploading data to ES. We just need to make some additional preparations:

1. We need to create an index. Let’s set the “ccs” name for it by executing the following:
   curl -X PUT “localhost:9200/ccs”

   If it is successful, you will see:
   {“acknowledged”:true,”shards_acknowledged”:true,”index”:”ccs”}



2. Any index requires mapping. This step is optional because ES has an internal mechanism to generate it if it is not set. But, I don't want to drop it because it allows us to get a more clear understanding of the data which we are going to upload, save, and index. So, let's review log entries and try to select fields that we want to analyze. Here is the typical ccs log record:


```
[
  "name": "ContentConversionService",
  "hostname": "win2012-tests",
  "pid": 7092,
  "taskid": 9,
  "gid": "hT5YrJZzGYMrWmOVdsRANg",
  "level": 30,
  "taskBegin": true,
  "parent": {
      "name": "LoadBalancer",
      "pid": 7552,
      "taskid": 53
   },
   "reqAccepted": true,
   "req": {
      "method": "POST",
      "path": "/v2/contentConverters",
      "port": 19010
   },
   "msg": "",
   "time": "2019-04-18T19:32:02.074Z",
   "v": 0
}
```


I think I will need indexing in the following fields:

- pid - this pid of the running service
- taskid - an identifier of the internal subtask
- gid - this a global cross identifier for all services; it allows us to track requests through taskBegin - the flag which indicates starting the request
- msg - some message - e.g. "Begin: convert", it is empty for above sample
- time - the time when an action occurred

In reviewing other log entries, I found additional fields which can be indexed: type, httpStatusCode, src, err. ES supports simple types (text, keyword, date, long, double, boolean, it), more complex ones which may contain some hierarchical data like JSON (objecs, nested) and some special types (geo_point, geo_shape and completion). Please see details here. As a result, we should map all fields we find interesting to the JSON scheme which can be used as a mapping for the “ccs” index:

```
{
"properties": {
  "time": {
    "type": "date"
  },
  "pid": {
    "type": "long"
  },
  "taskid": {
    "type": "long"
  },
  "type": {
    "type": "keyword"
  },
  "msg": {
    "type": "text"
  },
  "httpStatusCode": {
    "type": "long"
  },
  "src": {
     "type": "object",
                    "properties": {
                          "path":{
                                "type": "text"
                          }
                    }
  },
  "err": {
    "type": "object",
    "properties": {
      "message": {
        "type": "text"
      }
    }
  }
 }
}
```

Now we can save the above JSON as the file "mapping.json" and put it to ES as mapping for the ccs index. Execute the following command:

curl -H "Content-type: application/json" -X PUT "localhost:9200/ccs/_mapping/_doc" -d @mapping.json

To check if it is saved correctly, re-execute the command:

curl -H "Content-type: application/json" -X GET "localhost:9200/ccs/_mapping?pretty" -d @mapping.json

and you will get the mapping schema back in the answer.

3. Now everything is ready for posting log data to Elasticsearch. We created the "ccs" index, generated the mapping for it, and we have a log sample which contains JSON entries. There is just one problem. We can't send the log directly to Elasticsearch. Sure, it supports bulk uploading for a set of records from the file; however, each record should be indexed before each line in the row. The following example should be inserted:

   { "index" : { "_index" : "ccs", "_type" : "_doc", "_id" : "1" } }

   This is not the preferred way, but it can work if there is a small log which contains only a couple hundred entries. But, if there are millions of entries, then this can be a real problem. Instead, I will use a help Library from Elasticsearch. It will allow me to make this operation very trivial. This library is called "elasticsearch" and it is available for a variety of programming languages. I'll select Python3, but you can use another language that is more comfortable for you if you like:

   - Add the library "elasticsearch"

     pip install elasticsearch

   - Open any text editor and write the following Python code:

     ```
     from elasticsearch import Elasticsearch

     es = Elasticsearch(['localhost'], port = 9200)
     with open('ContentConversionService.log','rt') as logFile:
     ln = logFile.readline()
     ln_count = 1
     while ln:
        print(str(ln_count) + ': ' + ln)
        es.index(index = 'ccs', id = ln_count, body = ln)
        ln = logFile.readline()
        ln_count += 1
     ```

Bringing It All Together

Save the above code as a Python file and execute it. If you do not have any issues, then all your data is in the Elasticsearch storage. Now let's connect to Kibana and start the analysis of the uploaded log.

Set Up Kibana

Kibana is a client application for Elasticsearch. It allows to manage and navigate the stack where it is running. It contains rich possibilities to visualize and analyze Elasticsearch data. Again, let's use an image of Kibana from the Docker hub and run it in the container:

docker pull docker.elastic.co/kibana/kibana:6.7.1
docker run --link id_of_elasticsearch:elasticsearch -p 5601:5601 docker.elastic.co/kibana/kibana:6.7.1

Where **id_of_elasticsearch** is the container id for running elasticsearch. To get this, run the following command:

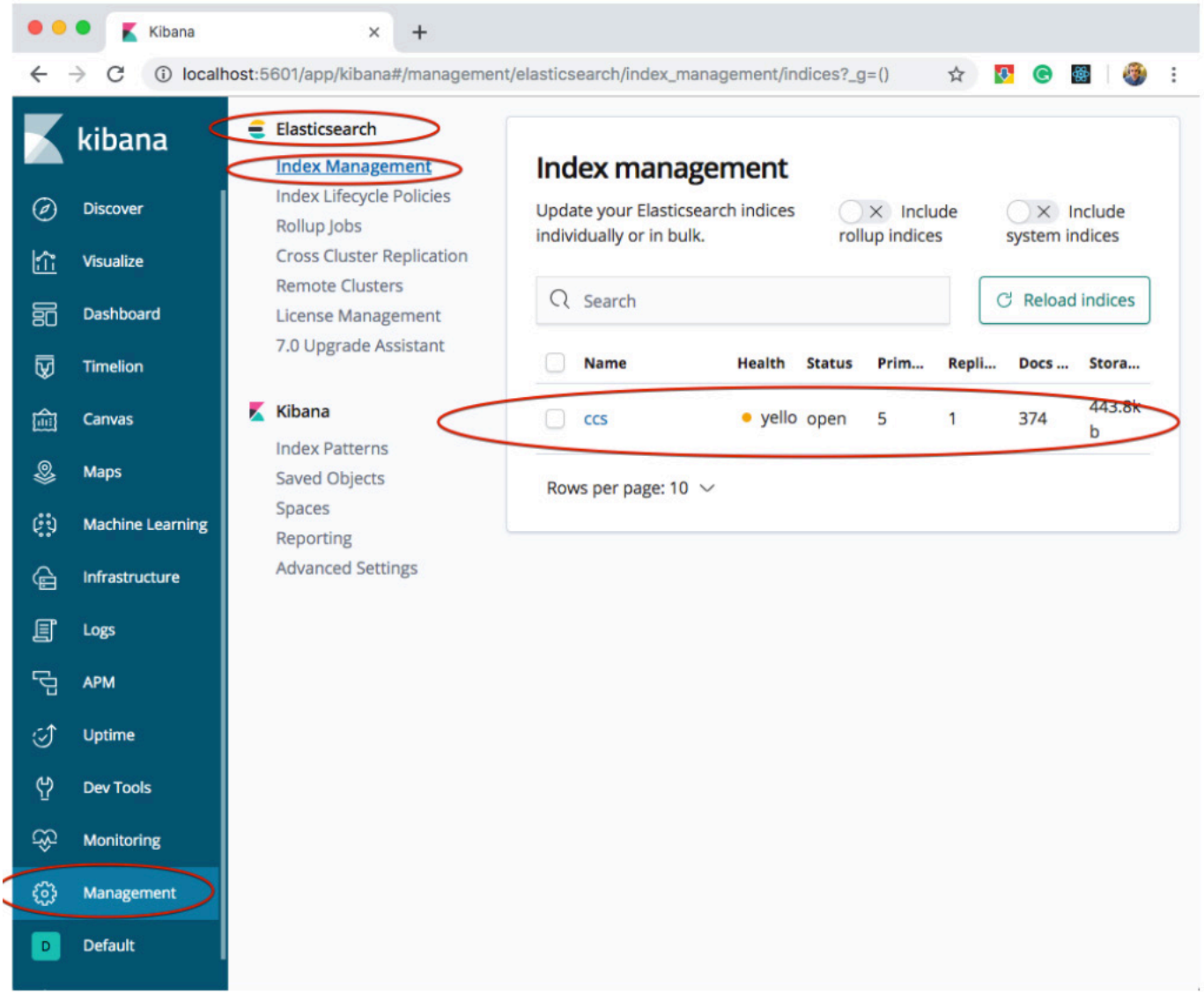
docker ps

and you will get a list of running containers, select the CONTAINER ID value for the image docker.elastic.co/elasticsearch/elasticsearch:6.7.1 from the provided table.

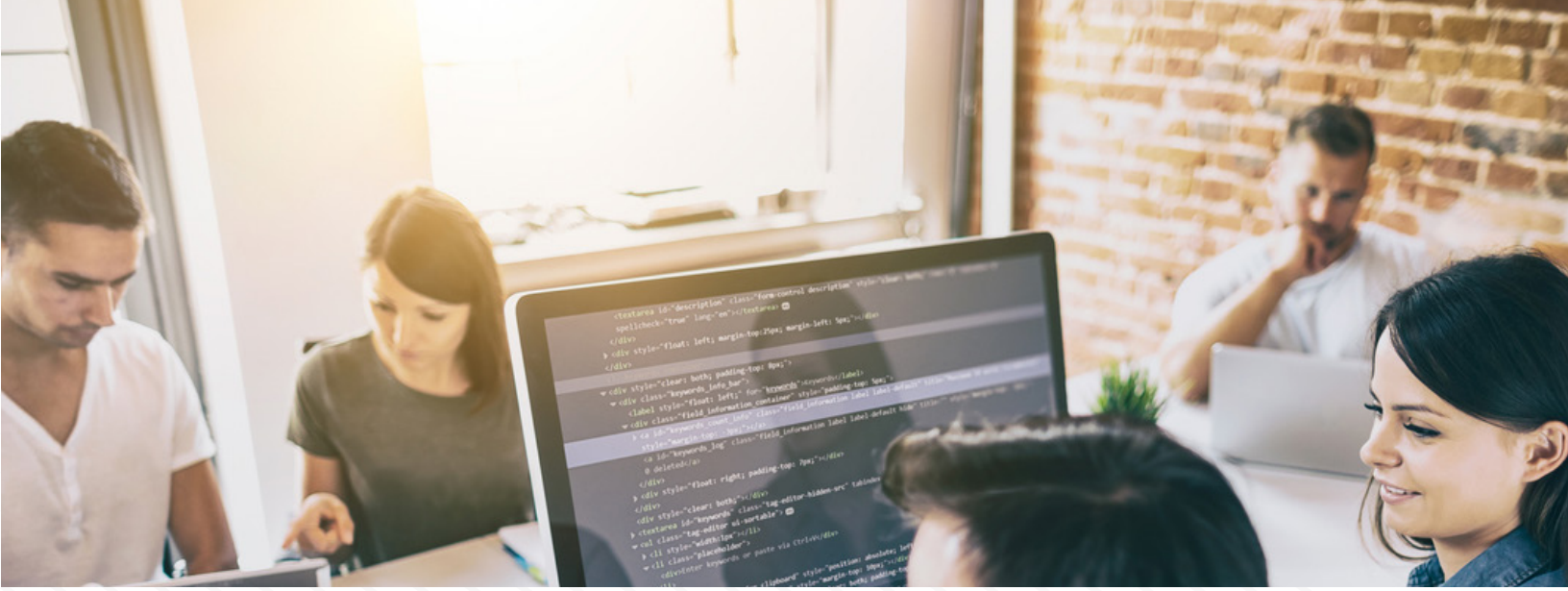
Now you can open a browser and navigate to "localhost:5601". If Kabana is running, you will

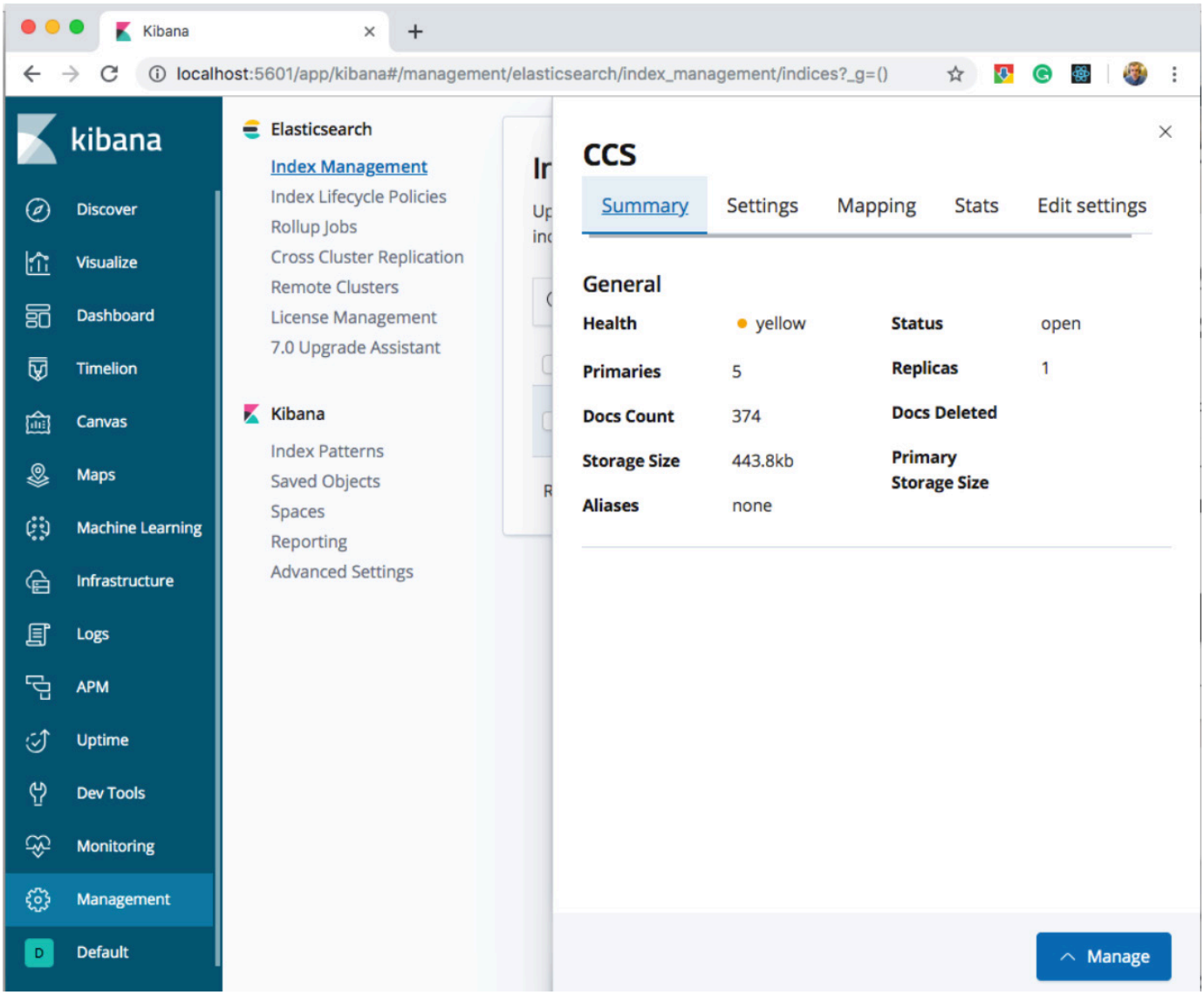
see its splash window.

Now, go to the Management tab and select the “Index Management” in the Elasticsearch group:

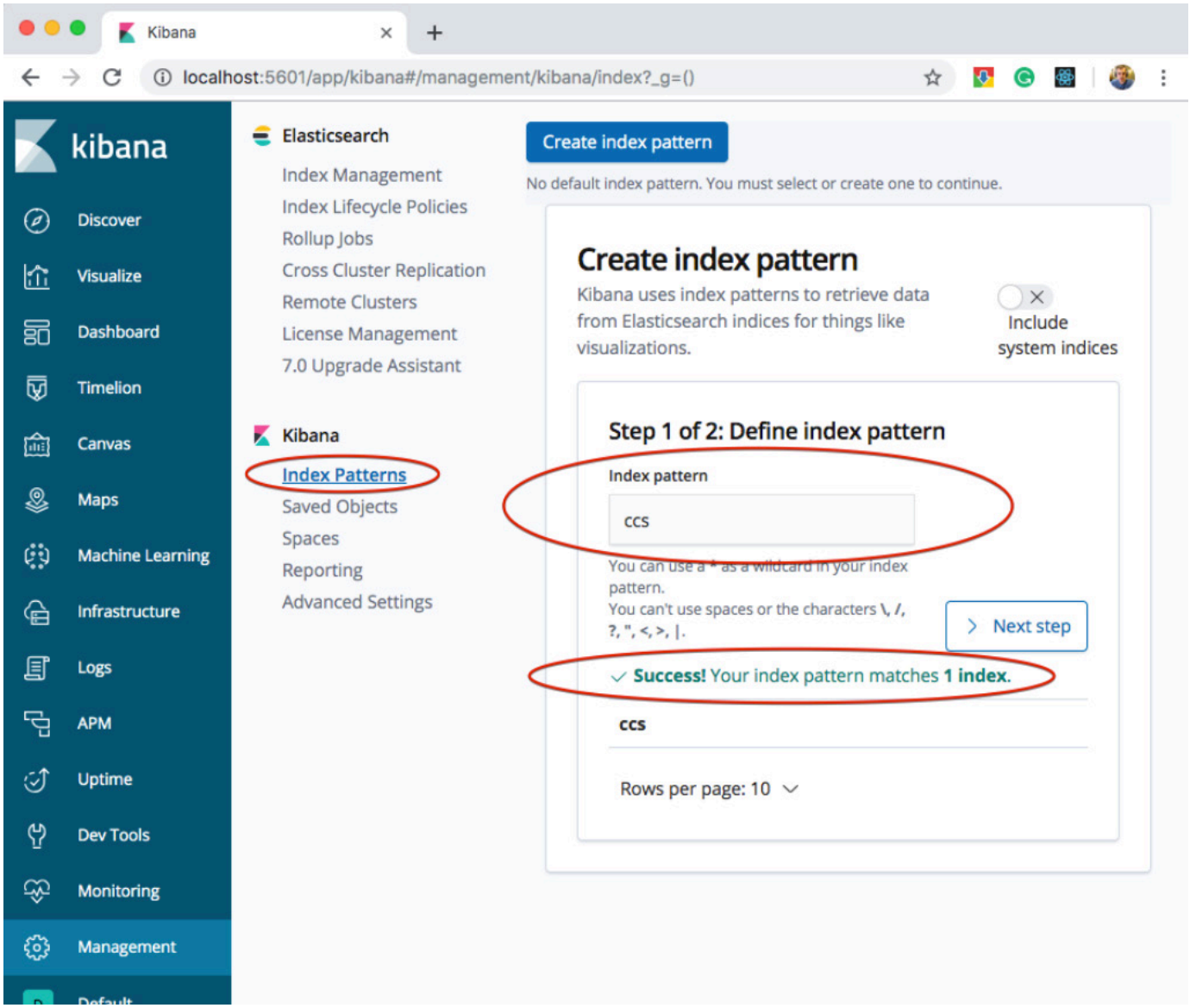


You should see that the index “ccs” is available. You can click on “ccs” and check the Summary information, Settings, Mapping, Stats, and Edit settings.

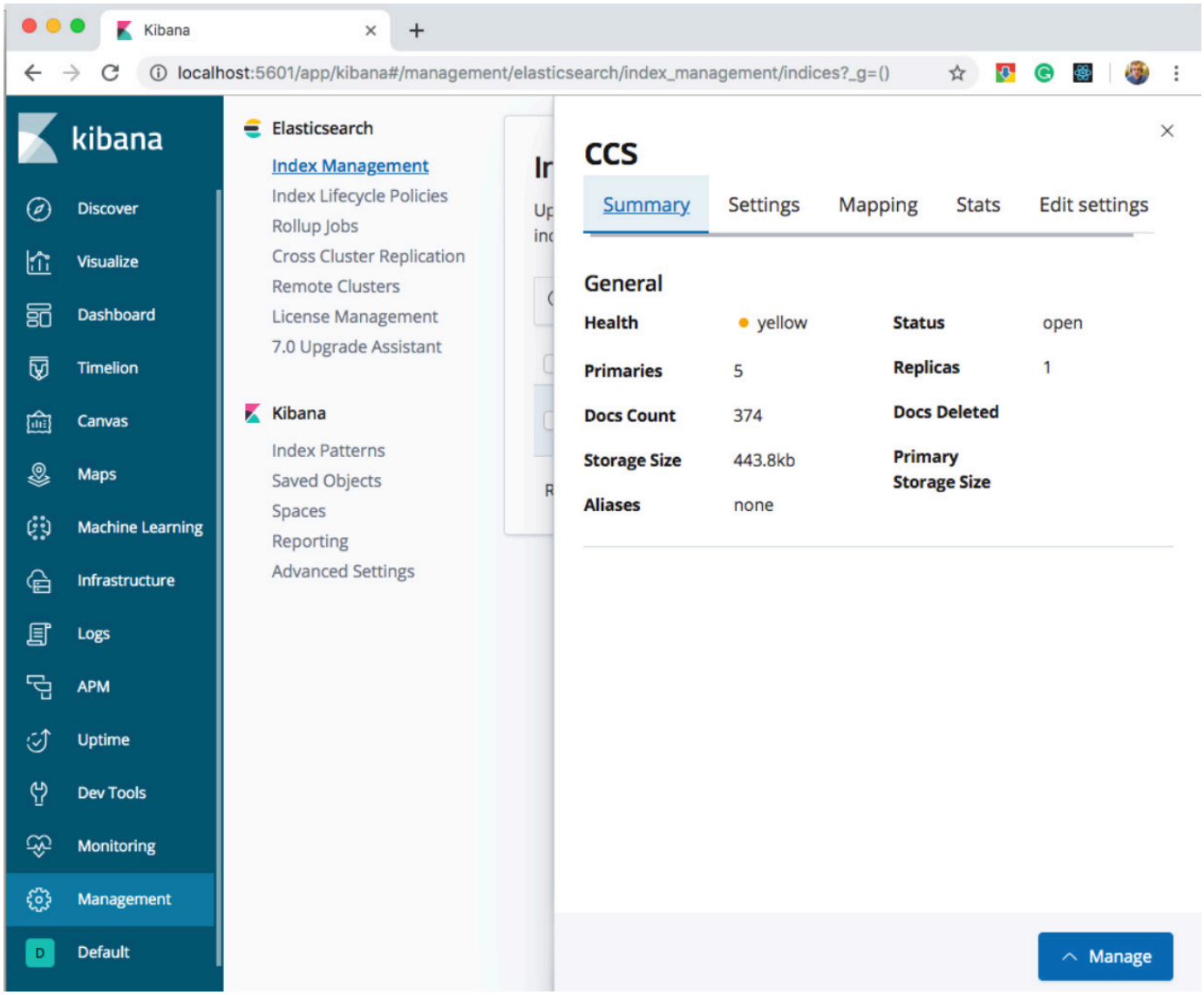



Kibana
localhost:5601/app/kibana#/management/elasticsearch/index_management/indices?_g=()
kibana
Discover
Visualize
Dashboard
Timelion
Canvas
Maps
Machine Learning
Infrastructure
Logs
APM
Uptime
Dev Tools
Monitoring
Management
Default
Elasticsearch
Index Management
Index Lifecycle Policies
Rollup Jobs
Cross Cluster Replication
Remote Clusters
License Management
7.0 Upgrade Assistant
Kibana
Index Patterns
Saved Objects
Spaces
Reporting
Advanced Settings
CCS
Summary
Settings
Mapping
Stats
Edit settings
General
Health
yellow
Status
open
Primaries
5
Replicas
1
Docs Count
374
Docs Deleted
Storage Size
443.8kb
Primary Storage Size
Aliases
none
Manage

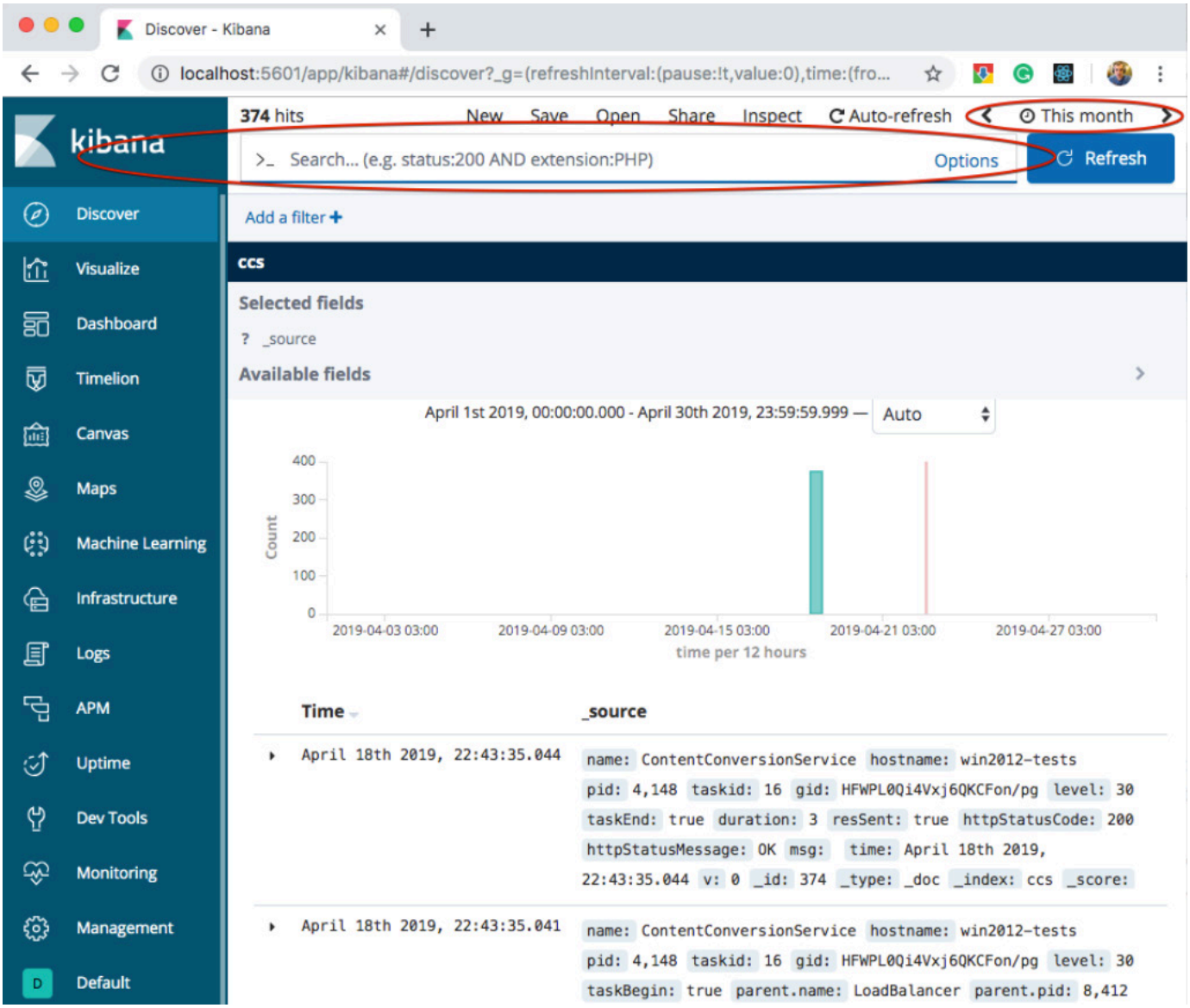
To make the data available for analysis, we need to add an Index Pattern. Just click on the "Index Patterns" item in the Kibana group and enter "ccs" in the input box:



Kibana will report that the index with the name "css" has been found. Then click on "Next Step". It asks to select the field which will be used for time filtering. We have a field "time" for this and when you select it, the button "Create index pattern" becomes active. After you click it, the index pattern is ready and all the log data is available for analysis on the Discover page.



To make the data available for analysis, we need to add an Index Pattern. Just click on the "Index Patterns" item in the Kibana group and enter "ccs" in the input box:



Do not forget to change the time frame in the top right corner; by default, it displays the last 15 minutes. If the log data was not generated in the last 15 minutes, the time frame won't display. There is a special input query box that uses Query Parser Syntax. You can see the details here. This allows you to build any complex queries and create diagrams/graphics for results. To check how it works, enter the following simple query:
err.message : PDF*
It will display all entries with error messages which contain text with the "PDF" keyword. Congratulations!

In summary, the information provided by this article should help you to start working with ELK. Also, it describes the basic terms of Elasticsearch. Now you are able to upload JSON logs with help libraries and connect Kibana. This app contains a lot of tools which allow you to manage ES and analyze the uploaded data.