



Lab Guide: Elasticsearch Engineer I

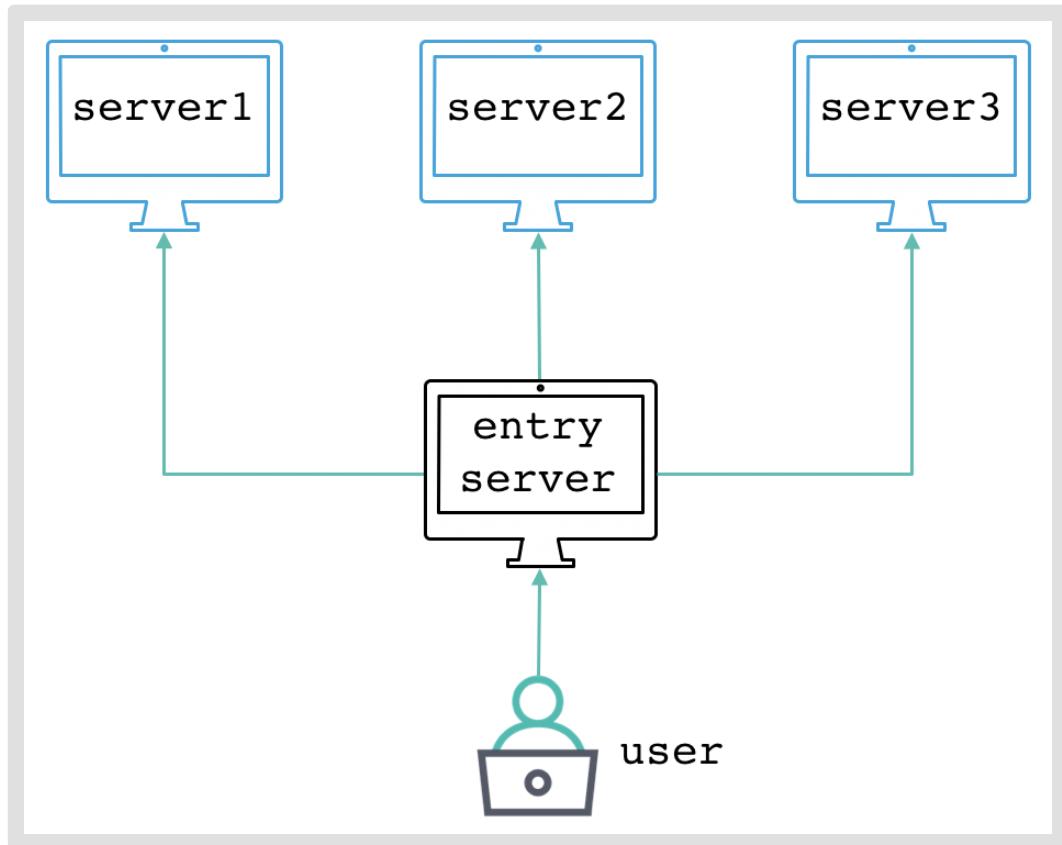
- [Lab 1.1: Elastic Stack Overview](#)
- [Lab 1.2: Getting Started with Elasticsearch](#)
- [Lab 1.3: CRUD Operations](#)
- [Lab 1.4: Searching Data](#)
- [Lab 2.1: Relevance](#)
- [Lab 2.2: Full-Text Queries](#)
- [Lab 2.3: Combining Queries](#)
- [Lab 2.4: Implementing a Search Page](#)
- [Lab 3.1: Metrics Aggregations](#)
- [Lab 3.2: Bucket Aggregations](#)
- [Lab 3.3: Combining Aggregations](#)
- [Lab 4.1: What is a Mapping?](#)
- [Lab 4.2: Text and Keyword Strings](#)
- [Lab 4.3: The Inverted Index and Doc Values](#)
- [Lab 4.4: Custom Mappings](#)
- [Lab 5.1: Master Nodes](#)
- [Lab 5.2: Node Roles](#)
- [Lab 5.3: Understanding Shards](#)
- [Lab 5.4: Distributed Operations](#)
- [Lab 6.1: HTTP Response and Shard Allocation Issues](#)
- [Lab 6.2: Monitoring](#)
- [Lab 6.3: Diagnosing Performance Issues](#)

Lab 1.1: Elastic Stack Overview

Objective: The goal of this lab is to introduce you to the Strigo lab environment and give you a brief overview of its architecture.

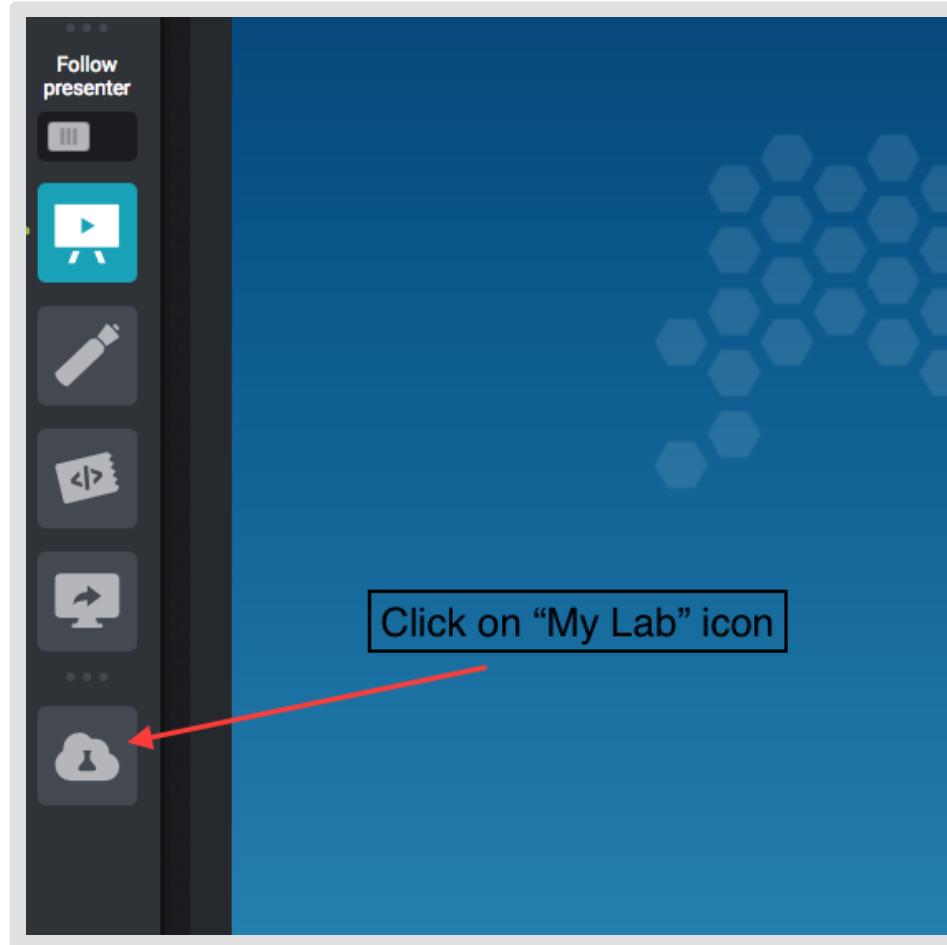
NOTE: All labs will be done using the Strigo environment. The lab environment was tailored for this training and has everything you need to complete the labs successfully. After the course is completed and your lab environment is no longer available, you will be able to complete the labs on your local machine by downloading the zip file available in your course page at training.elastic.co. Use [these instructions](#) as a reference on how to run the labs in your own machine.

1. Take a moment to look at the picture below, which shows the server architecture you are going to use in the labs:



Notice there are four different servers available. The `entry server` is an entry point to the other three servers, and it is the server in which you are automatically logged in to every time you open a new terminal tab in your lab environment.

2. From within the Virtual Environment UI, click on the "**My Lab**" icon in the left toolbar:



A command prompt will open, and you will be logged in to a Linux machine that is configured to provide easy SSH access to three servers: `server1`, `server2` and `server3`. You will deploy an Elasticsearch cluster on these 3 servers throughout the *Engineer I* course. For this first lab, you will perform all of the tasks on `server1`.

3. SSH onto `server1`:

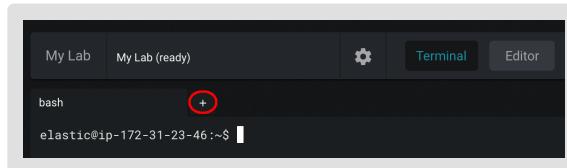
```
ssh server1
```

4. View the contents of the **elastic** user home folder by entering `ls -l`. You should see Elasticsearch, Filebeat, Kibana and Logstash tar files, which were all simply downloaded from our website. (You will see other files in the home folder as well):

```
[elastic@server1 ~]$ ls -l
total 1502160
drwxrwxr-x 5 elastic elastic      4096 Sep 26 20:43 datasets
-rw-rw-r-- 1 elastic elastic 285082863 Sep 26 20:43 elasticsearch
-rw-rw-r-- 1 elastic elastic 25265083 Sep 26 20:43 filebeat-7.3
-rw-rw-r-- 1 elastic elastic 236770201 Sep 26 20:43 kibana-7.3.1
```

```
-rw-rw-r-- 1 elastic elastic 171784034 Sep 26 20:43 logstash-7.3  
-rw-rw-r-- 1 elastic elastic      551290 Sep 26 20:43 postgresql-9
```

5. Within Virtual Environment, open a new console by clicking the plus sign "+" next to the tab of your current console, and you should see a command prompt:



6. Use your new console window to SSH onto server1 .

[Hide answer](#)

```
ssh server1
```

7. In the next labs you will start Elasticsearch, use Filebeat and Logstash to ingest data into Elasticsearch, as well as visualize data with Kibana.

Summary: In this lab you familiarized yourself with the lab environment, which you are going to use throughout this course.

End of Lab 1.1

Lab 1.2: Getting Started with Elasticsearch

Objective: In this lab, you will start and stop Elasticsearch and update some of the configuration settings.

1. Extract Elasticsearch using the following command:

```
tar -xf elasticsearch-7.3.1-linux-x86_64.tar.gz
```

2. Start Elasticsearch:

```
./elasticsearch-7.3.1/bin/elasticsearch
```

3. Wait for Elasticsearch to startup. You should see a message showing that it started. (The output order may vary a bit.)

```
[2019-10-02T21:22:29,879][INFO ][o.e.h.AbstractHttpServerTransport:131] o.e.h.a.TransportService$2{[2019-10-02T21:22:29,880][INFO ][o.e.n.Node{[2019-10-02T21:22:29,979][INFO ][o.e.g.GatewayService{[2019-10-02T21:22:30,228][INFO ][o.e.c.m.MetaDataIndexTemplateService{[2019-10-02T21:22:30,302][INFO ][o.e.c.m.MetaDataIndexTemplateService{[2019-10-02T21:22:30,328][INFO ][o.e.c.m.MetaDataIndexTemplateService{[2019-10-02T21:22:30,360][INFO ][o.e.c.m.MetaDataIndexTemplateService{[2019-10-02T21:22:30,409][INFO ][o.e.c.m.MetaDataIndexTemplateService{[2019-10-02T21:22:30,450][INFO ][o.e.c.m.MetaDataIndexTemplateService{[2019-10-02T21:22:30,480][INFO ][o.e.c.m.MetaDataIndexTemplateService{[2019-10-02T21:22:30,517][INFO ][o.e.c.m.MetaDataIndexTemplateService{[2019-10-02T21:22:30,543][INFO ][o.e.x.i.a.TransportPutLifecycleService{[2019-10-02T21:22:30,656][INFO ][o.e.l.LicenseService{[2019-10-02T21:22:30,659][INFO ][o.e.x.s.s.SecurityStatusChangeLog{
```

4. Using the other tab that you connected to **server1** in the previous lab, enter the following request to get basic cluster and node information:

```
curl -X GET "http://localhost:9200/"
```

When you execute this request, you should see a response similar to:

```
{  
  "name" : "server1",  
  "cluster_name" : "elasticsearch",  
  "cluster_uuid" : "UCvdeC0gTJ2UEa__qmcP5w",  
  "version" : {  
    "number" : "7.3.1",  
    "build_flavor" : "default",  
    "build_type" : "tar",  
    "build_hash" : "4749ba6",  
    "build_date" : "2019-08-19T20:19:25.651794Z",  
    "build_snapshot" : false,  
    "lucene_version" : "8.0.0",  
    "minimum_wire_compatibility_version" : "6.8.0",  
    "minimum_index_compatibility_version" : "6.0.0-beta1"
```

```
  },
  "tagline" : "You Know, for Search"
}
```

- What is the version of your Elasticsearch instance?

[Hide answer](#)

The version of the node is value of the `version.number` field (**7.3.1** above)

- What is the name of your node?

[Hide answer](#)

The name of the node is the value of the `name` field (**server1** above).

- What is the name of your cluster?

[Hide answer](#)

In the output above, the cluster is named "**elasticsearch**". Your cluster should be named "**elasticsearch**" also.

- How did these names get assigned to your node and cluster?

[Hide answer](#)

By default Elasticsearch will use "**elasticsearch**" as the cluster name and the hostname that the machine has when it starts as the node name. In the next lab steps you will see how to change these settings.

5. Stop Elasticsearch by hitting **Ctrl+c** in the terminal tab where you have Elasticsearch running. You should see the following output, and the command prompt should also appear:

```
[2019-10-02T21:40:09,926][INFO ][o.e.x.m.p.NativeController] [se
[2019-10-02T21:40:09,927][INFO ][o.e.n.Node           ] [ser
[2019-10-02T21:40:09,934][INFO ][o.e.x.w.WatcherService] [ser
[2019-10-02T21:40:10,004][INFO ][o.e.n.Node           ] [ser
[2019-10-02T21:40:10,004][INFO ][o.e.n.Node           ] [ser
[2019-10-02T21:40:10,016][INFO ][o.e.n.Node           ] [ser
[elastic@server1 ~]$
```

6. The current cluster and node names have been automatically defined. Let's change both to values that make more sense:

- Open the `elasticsearch.yml` file using a text editor of your choice: nano, emacs, or vi. If you are not familiar with command-line editors, we suggest you use nano (click [here](#) for tips on using nano). You will find the file in the `elasticsearch-7.3.1/config` directory. For example, if you want to use nano:

```
nano elasticsearch-7.3.1/config/elasticsearch.yml
```

- Notice that every line in this file is commented out. This is a good reference if you want to quickly lookup settings names. However, having a configuration file with so many comments in between the settings can be confusing. At the end of the file, Set the name of the cluster to `my_cluster`:

[Hide answer](#)

```
cluster.name: "my_cluster"
```

- Save your changes to `elasticsearch.yml`. To save changes to a file using nano, hit **Ctrl+X** to exit nano, then press "y" to save your changes. Hit "**Enter**" when prompted for a file name.
- Now, start Elasticsearch again, with the node name `node1`.

[Hide answer](#)

```
./elasticsearch-7.3.1/bin/elasticsearch -E node.name=node1
```

7. Now check the cluster information to see what changed:

[Hide answer](#)

```
curl -X GET "http://localhost:9200/"
```

The output should be very similar to the previous execution, except that the cluster and node names have changed. Elasticsearch internally uses cluster and node IDs—the cluster and node names are there for humans. This means that you can change cluster and node names at any point without affecting the existing data, but you still need to restart the cluster (so it's probably not something you would do very often in your production environment).

8. Let's take a look at the log files that Elasticsearch produces.

- First, list the contents of the log folder.

[Hide answer](#)

```
ls -l elasticsearch-7.3.1/logs/
```

- There are a lot of different log files in there. Some file names start with `elasticsearch` while others start with `my_cluster`. Why is that?

[Hide answer](#)

There are different log files with different purposes. For example, you have index and search slow logs that will be discussed later.

Regardless of the log type, all log files contain the cluster name as a prefix. `elasticsearch` was the initial cluster name and then you changed it to `my_cluster`, hence multiple files with both prefixes.

9. Elasticsearch is running using the default heap size, which is 1GB. Let's change it to 512MB. Open the `elasticsearch-7.3.1/config/jvm.options` file using a text editor of your choice. Update both initial (**-Xms**) and max (**-Xmx**) heap sizes to 512MB:

[Hide answer](#)

```
# Xms represents the initial size of total heap space  
# Xmx represents the maximum size of total heap space  
  
-Xms512m  
-Xmx512m
```

10. Next, you need to restart the node so your changes can take effect, so stop the Elasticsearch process again, by hitting **Ctrl+c** in the terminal tab where Elasticsearch is running.

11. Start Elasticsearch **with the exact command that is shown below**. Don't worry about the `http.host` configuration right now, as you are going to learn more about it later.

```
./elasticsearch-7.3.1/bin/elasticsearch -E node.name=node1 -E ht
```

Summary: In this lab you started and stopped Elasticsearch and updated some of the configuration settings.

End of Lab 1.2

Lab 1.3: CRUD Operations

Objective: In this lab, you will install Kibana, define the structure of a few documents, and use Kibana **Console** to perform CRUD operations.

1. Using Elasticsearch from the command line can be hard and time consuming. Kibana **Console** allows you to write Elasticsearch commands efficiently. First, on `server1`, extract Kibana with the following command:

```
tar -xf kibana-7.3.1-linux-x86_64.tar.gz
```

2. Start Kibana:

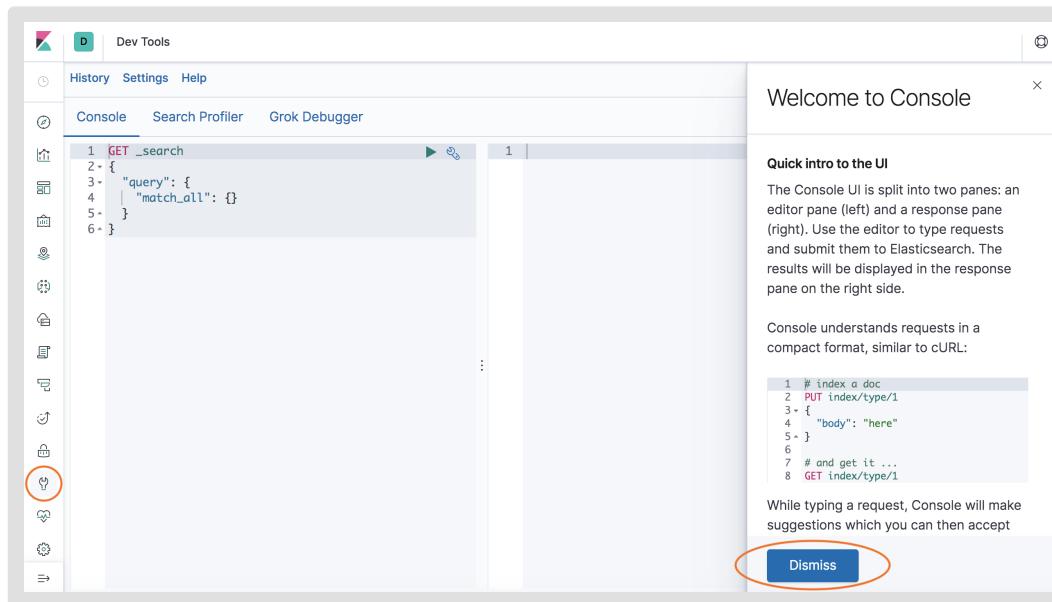
```
./kibana-7.3.1-linux-x86_64/bin/kibana --host=0.0.0.0
```

You should see the following output:

```
log [21:11:37.412] [info][listening] Server running at http://  
log [21:11:37.418] [info][server][Kibana][http] http server ru
```

3. Now that Kibana is running, click [here](#) to connect.

4. In Kibana, click on the **Dev Tools** button in the side navigation pane to open the **Console** application:



5. Kibana has several handy [keyboard shortcuts](#) that will make your time spent in **Console** more efficient. To view the keyboard shortcuts, click on "**Help**" in the upper-right corner. For example, pressing **Ctrl + Enter** (**Cmd + Enter** on a Mac) submits a request without having to use your mouse.

TIP: Click on "**Help**" again to close the Help panel.

6. Notice there is a `match_all` query already written in the **Console**. Just skip it for now. We will talk about it later. Below the `match_all` query, write a request that returns the node's basic information. Then run it by clicking the green "play" button that appears to the right of the command (or using the **Ctrl/Cmd + Enter** keyboard shortcut).

[Hide answer](#)

The request below is the same request that you executed with curl in the terminal in the previous lab.

GET /

7. Next, you will focus on the dataset. Imagine a dataset that is row-oriented (e.g. spreadsheet or a traditional database). How would you write a JSON document based on sample entries that look like the following? Think about field structure and empty fields:

id	title	category	date	author_first_name	author_last_name	author_company
----	-------	----------	------	-------------------	------------------	----------------

id	title	category	date	author_first_name	author_last_name	author_company
1	Better query execution	Engineering	July 15, 2015	Adrien	Grand	Elastic
2	The Story of Sense		May 28, 2015	Boaz	Leskes	

[Hide answer](#)

Each entry contains a flattened structure, so **author_first_name** is not specifically related to **author_last_name** and **author_company**. While in relational databases you can use relations to model it, in document-oriented stores we preserve structures within a single document. So the entry could be written as:

```
{
  "id": "1",
  "title": "Better query execution",
  "category": "Engineering",
  "date": "July 15, 2015",
  "author": {
    "first_name": "Adrian",
    "last_name": "Grand",
    "company": "Elastic"
  }
}
```

Notice that author is now an object with 3 fields. For the second entry we can do the same thing to the author structure, but there are also empty fields. In a document store, empty fields can just be omitted.

```
{
  "id": "2",
  "title": "The Story of Sense",
  "date": "May 28, 2015",
  "author": {
    "first_name": "Boaz",
    "last_name": "Leskes"
  }
}
```

```
    }  
}
```

Notice we do not have `category` or `author.company` fields in this document. Other options would be to index these fields with a `null` value (similar effect as omitting) or empty strings (completely different effect: the field exists and its value is an empty string).

8. **EXAM PREP:** Now that we have defined the documents, let's index them. Notice that the `id` field defined inside the documents is just a normal field like any other. The actual document `_id` is defined in the request URL when you index the documents. Index both JSON documents into the `my_blogs` index using their respective ids.

[Hide answer](#)

```
PUT my_blogs/_doc/1  
{  
  "id": "1",  
  "title": "Better query execution",  
  "category": "Engineering",  
  "date": "July 15, 2015",  
  "author": {  
    "first_name": "Adrian",  
    "last_name": "Grand",  
    "company": "Elastic"  
  }  
}  
  
PUT my_blogs/_doc/2  
{  
  "id": "2",  
  "title": "The Story of Sense",  
  "date": "May 28, 2015",  
  "author": {  
    "first_name": "Boaz",  
    "last_name": "Leskes"  
  }  
}
```

9. **EXAM PREP:** The index operation can also be executed without specifying the `_id`. In such a case, you use a **POST** instead of a **PUT** and an `_id` will be generated automatically. Index the following document without an id and check the `_id` in the response. (Make sure you use **POST**.)

```
{  
  "id": "57",  
  "title": "Phrase Queries: a world without Stopwords",  
  "date": "March 7, 2016",  
  "category": "Engineering",  
  "author": {  
    "first_name": "Gabriel",  
    "last_name": "Moskovicz"  
  }  
}
```

[Hide answer](#)

```
POST my_blogs/_doc/  
{  
  "id": "57",  
  "title": "Phrase Queries: a world without Stopwords",  
  "date": "March 7, 2016",  
  "category": "Engineering",  
  "author": {  
    "first_name": "Gabriel",  
    "last_name": "Moskovicz"  
  }  
}
```

10. **EXAM PREP:** Use a **GET** command to retrieve the document with `_id` of **1** from the `my_blogs` index.

[Hide answer](#)

```
GET my_blogs/_doc/1
```

11. **EXAM PREP:** Delete the document with the `_id` of 1 from the `my_blogs` index. Verify it was deleted by trying to **GET** it again. You should get the following response:

```
{  
  "_index": "my_blogs",  
  "_type": "_doc"  
  "_id": "1",  
  "found": false  
}
```

[Hide answer](#)

```
DELETE my_blogs/_doc/1  
GET my_blogs/_doc/1
```

12. **OPTIONAL:** Perform a bulk operation that executes the following operations on the `my_blogs` index from the previous step:

- Update document 2: set `category` to "**Engineering**" and set `author.company` to "**Elastic**"
- Index a new document with `_id` of 3 that contains the following fields and values:

```
{  
  "title": "Using Elastic Graph",  
  "category": "Engineering",  
  "date": "May 25, 2016",  
  "author": {  
    "first_name": "Mark",  
    "last_name": "Harwood",  
    "company": "Elastic"  
  }  
}
```

[Hide answer](#)

```
POST my_blogs/_bulk  
{"update": {"_id": 2}}
```

```
{"doc": {"category": "Engineering", "author.company": "Elastic"}},  
{"index": {"_id": 3}},  
{"title": "Using Elastic Graph", "category": "Engineering", "date": ""}  
  
GET my_blogs/_search
```

13. **OPTIONAL:** Use `_mget` to retrieve the documents with the ids **2** and **3** from your `my_blogs` index and verify your two bulk operations were both successful.

[Hide answer](#)

```
GET my_blogs/_mget  
{  
  "docs": [  
    {"_id": 2},  
    {"_id": 3}  
  ]  
}
```

14. **OPTIONAL:** What happens if you copy and paste the following curl request to **Console**?

```
curl -X GET "http://localhost:9200/"
```

[Hide answer](#)

Console is smart enough to understand the syntax and apply the **Console** format to it.

15. **OPTIONAL:** Sometimes it is also handy to copy from **Console** to the terminal. Click the wrench icon close to the play button and select **Copy as cURL**. Now go back to a terminal and paste it.

16. Finally, delete the `my_blogs` index.

[Hide answer](#)

```
DELETE my_blogs
```

Summary: In this lab, you installed Kibana, defined the structure of a few documents, and used Kibana **Console** to perform CRUD operations.

End of Lab 1.3

Lab 1.4: Searching Data

Objective: In this lab, you will see how quickly and easily the Elastic Stack can be used to ingest data from a database, collect data from log files, and search data. You will use Logstash to ingest the blogs from a Postgres database into Elasticsearch and use Filebeat to ingest the web access log data from log files.

1. If you completed the previous labs, Elasticsearch and Kibana should be running in the foreground at two separate tabs. Next, you are going to start Logstash. Open a new tab and SSH onto server1 :

```
ssh server1
```

WARNING: If you are running the labs in your own environment click `Show answer` to see how to load the data.

[Hide answer](#)

If you are running the labs in your own environment, you don't need to install the Postgres database. Just execute the following steps:

- First, download and extract Logstash 7.3.1 into the course folder.
- Then, open a terminal and go to the course folder.
- Finally, you can use the following command to ingest the data from a csv file.

```
cat datasets/blogs.csv | logstash-7.3.1/bin/logstash -f data
```

- Now, you can skip steps 2, 3, and 4 and go directly to step 5.

- The first dataset you are going to ingest into Elasticsearch is the blogs dataset using Logstash, which has been downloaded already. Extract Logstash using the following command:

```
tar -xf logstash-7.3.1.tar.gz
```

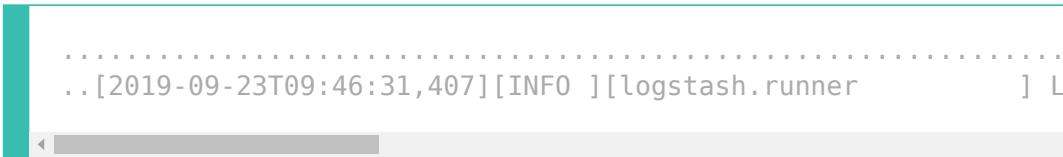
- Logstash is going to retrieve the blogs from a SQL database and index them into your running Elasticsearch instance. This is accomplished using a Logstash config file that has been provided for you named `blogs_sql.conf`. You do not need to modify this file in any way, but here are its contents. Notice the `input` is all blogs from a table named `blogs`, and the `output` is an Elasticsearch index named `blogs`:

```
input {  
    jdbc {  
        jdbc_connection_string => "jdbc:postgresql://db_server:5432/  
        jdbc_driver_class => "org.postgresql.Driver"  
        jdbc_driver_library => "/home/elastic/postgresql-42.2.9.jar"  
        jdbc_user => "postgres"  
        jdbc_password => "password"  
        statement => "SELECT * from blogs"  
    }  
}  
  
filter {  
    mutate {  
        remove_field => ["@version", "host", "message", "@timestamp"]  
    }  
}  
  
output {  
    stdout { codec => "dots"}  
    elasticsearch {  
        index => "blogs"  
    }  
}
```

4. To run Logstash with this config file, execute the following command:

```
./logstash-7.3.1/bin/logstash -f datasets/blogs_sql.conf
```

Logstash will take some time to start, so make sure to wait before you go to the next step. Eventually the output will show dots, and each dot represents a document being indexed into Elasticsearch. When Logstash is finished, you will see a "Logstash shut down" message:



```
.....[2019-09-23T09:46:31,407][INFO ][logstash.runner] .....[2019-09-23T09:46:31,407][INFO ][logstash.runner] Logstash shut down
```

5. The `blogs` table in the database contains 1594 entries. Verify that all those entries are in Elasticsearch.

[Hide answer](#)

You can check that by running a `_search` on the `blogs` index with the following command:

```
GET blogs/_search
```

You should see the following response:

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1594,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      ...
    ]
}
```

```
    ...
  ]
}
```

6. Every `_search` request to Elasticsearch will have a query when executed. The `match_all` query is implicit if no query is defined. Scroll up in **Console** and execute the default `match_all` query and see that the results include all indices, because you reached the `_search` endpoint without specifying an index.
7. Another option is to execute a `_count` request instead of a `_search`. Try the following example:

```
GET blogs/_count
```

You should see a slightly different output. Because you are executing a count, no hits are returned in the response. Notice, the `match_all` query is still implicit.

```
{
  "count" : 1594,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  }
}
```

8. Now that we have all blogs indexed, let's see how the search page works. Access the [blogs search page](#) and play around with it.

The screenshot shows the Elasticsearch search interface. At the top left is the elastic logo. A search bar contains the placeholder "Search...". To its right, a message says "There are 1594 results for "" (41 milliseconds)". Below the search bar are two sections: "Categories" and "Dates (yyyy-MM-dd)". The "Categories" section includes links for "Engineering (440)", "(333)", "Releases (238)", "This week in Elasticsearch and Apache Lucene (156)", and "User Stories (122)". The "Dates" section has input fields for "from" and "to" dates, both set to "dd/mm/yyyy". Below these are sorting options: "Sort by: Score" and a dropdown arrow. A search button with a magnifying glass icon is at the bottom.

Congratulations, the blogs data are in Elasticsearch and the webpage is working! Make sure to familiarize yourself with different searches, filtering by category and date, and also paging. Over the next chapters you are going to learn how to write the queries and aggregations used to build this application.

The screenshot shows the Elasticsearch search interface with the query "lucene" entered into the search bar. At the top right, it says "There are 248 results for "lucene" (45 milliseconds)". Below the search bar are the same "Categories" and "Dates" sections as the previous screenshot. The search results on the right include:

- Investing in Apache Lucene** [_score: 15.301029]
 - As you know, Apache Lucene is at the heart of Elasticsearch, and when we formed the company around Elasticsearch, we made a commitment to make sure we drive Apache Lucene forward. With , and , we already have a very strong group of Lucene committers that help to harden Lucene, amount of time in Lucene a few months ago, the opportunity to help innovation in the Lucene world happen at an even faster pace.
- This Week in Elasticsearch and Apache Lucene: Algorithms that power Lucene and Elasticsearch** [_score: 14.187443]
 - With this weekly series, we're bringing you an update on all things Elasticsearch and Apache Lucene at Top NewsVideo of my talk is now online: Algorithms that power Lucene and Elasticsearch — Adrien Grand, (@pjoumouz) Elasticsearch Core Apache Lucene Watch This Space , where we'll share more news on the whole
- Store compression in Lucene and Elasticsearch** [_score: 14.174288]
 - [Engineering] Back in 2011 if you asked Lucene to index and store some text content, odds are high that your inverted Lucene 4 and codecsBut then in 2012 perspectives changed after the release of Lucene 4.0. By the way, out of the eleven 4.x Lucene releases, 6 of them changed the index format! Lucene 5.0 still has the same default codec as Lucene 4.1 but now allows you to use (the compression,a new article about Lucene and Elasticsearch internals!

Paging controls "« 1 2 3 4 5 »" are at the bottom right.

9. Next, let's index the access logs dataset. You will use Filebeat to monitor the log files and ingest every line. Filebeat is downloaded already on server1 . Open a new Terminal tab and extract it using the following command:

```
tar -xf filebeat-7.3.1-linux-x86_64.tar.gz
```

WARNING: If you are running the labs in your own environment click [Show answer](#) to see how to load the data.

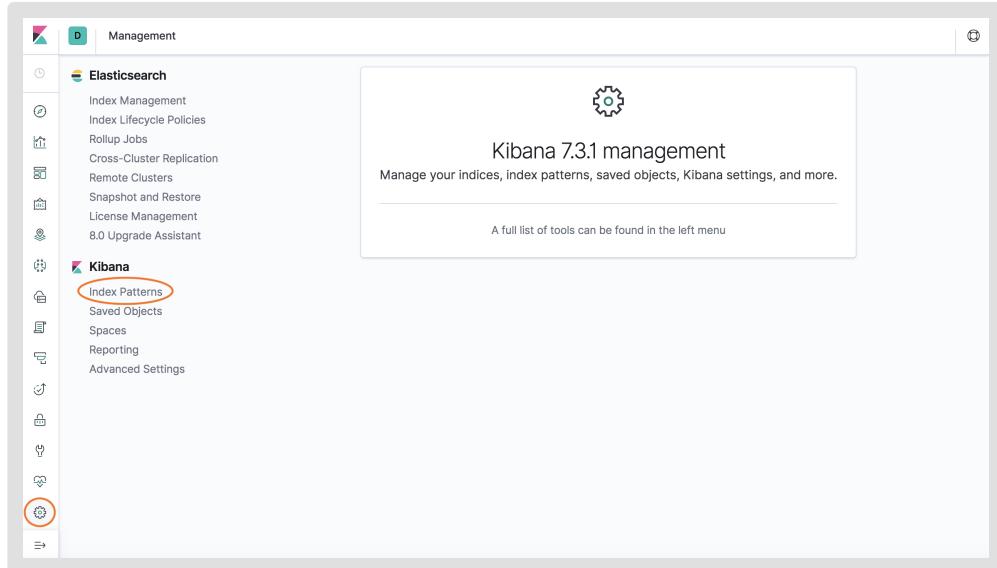
[Hide answer](#)

If you are running the labs in your own environment, execute the following steps:

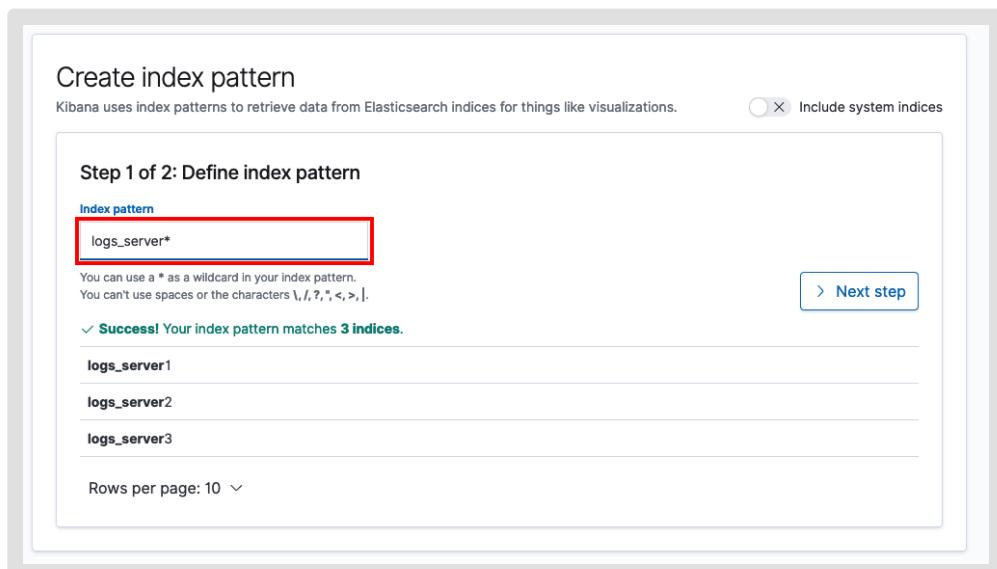
- First, download and extract Filebeat 7.3.1 into the course folder.
 - Then, in the datasets folder, extract the elastic_blog_curated_access_logs.tar.gz file.
 - In the datasets folder, edit the filebeat.yml to use the correct dataset path. Set paths under filebeat.inputs to <course_folder>/datasets/elastic_blog_curated_access_logs_server*/*.log Make sure to update <course_folder> to the correct path in your environment.
 - Then, open a terminal and go to the course folder.
 - Finally, execute the next step using this terminal tab.
10. Run the following command to start Filebeat using the filebeat.yml configuration file that has been provided. There will be no output - the command prompt will just hang:
- ```
./filebeat-7.3.1-linux-x86_64/filebeat -c datasets/filebeat.yml
```
11. There are 1,751,476 events that occurred between **2017-04-01** and **2017-09-01** across multiple log files. They will be ingested into 3 indices called logs\_server1 , logs\_server2 and logs\_server3 . Use one of the following commands to check how many files have been ingested so far:
- ```
GET logs_server1,logs_server2,logs_server3/_count  
GET logs_server*/_count
```
- The ingestion might take a few minutes, feel free to go to the next task before all logs are indexed. When all of the log events are indexed, you can kill the Filebeat process using **Ctrl+c**.
12. **OPTIONAL:** Now that you have the blogs and access log data in Elasticsearch, you will use Kibana to analyze it. Click [here](#) to connect to Kibana.

13. OPTIONAL: You need to tell Kibana which indices to analyze, which is accomplished by defining an **Index Pattern**:

- Click on the **Management** link in the left-hand toolbar of Kibana, then click on **Index Patterns**:



- Type `logs_server*` in the **Index pattern** field:



- Click **Next step** and select `@timestamp` as the **Time Filter field name** field:

Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

X Include system indices

Step 2 of 2: Configure settings

You've defined **logs_server*** as your index pattern. Now you can specify some settings before we create it.

Time Filter field name Refresh

The Time Filter will use this field to filter your data by time.
You can choose not to have a time field, but you will not be able to narrow down your data by a time range.

> Show advanced options

< Back **Create index pattern**

- Click the **Create index pattern** button. You should see a page that shows the 43 fields of the web log documents.

14. **OPTIONAL:** Now that we have set up our index pattern, let's visualize the data. You will first create a table with the top 5 most popular blogs:

- Click on **Visualize** in the left-hand toolbar of Kibana, and then click the **Create a visualization** button:

Visualize

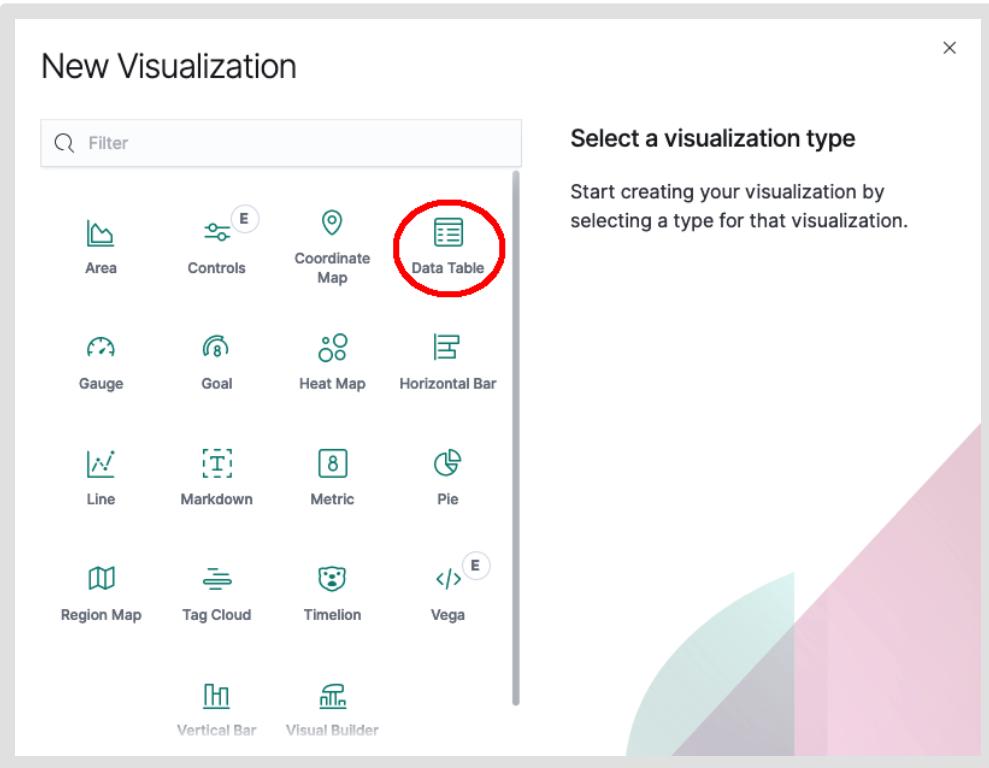
Visualize

Create your first visualization

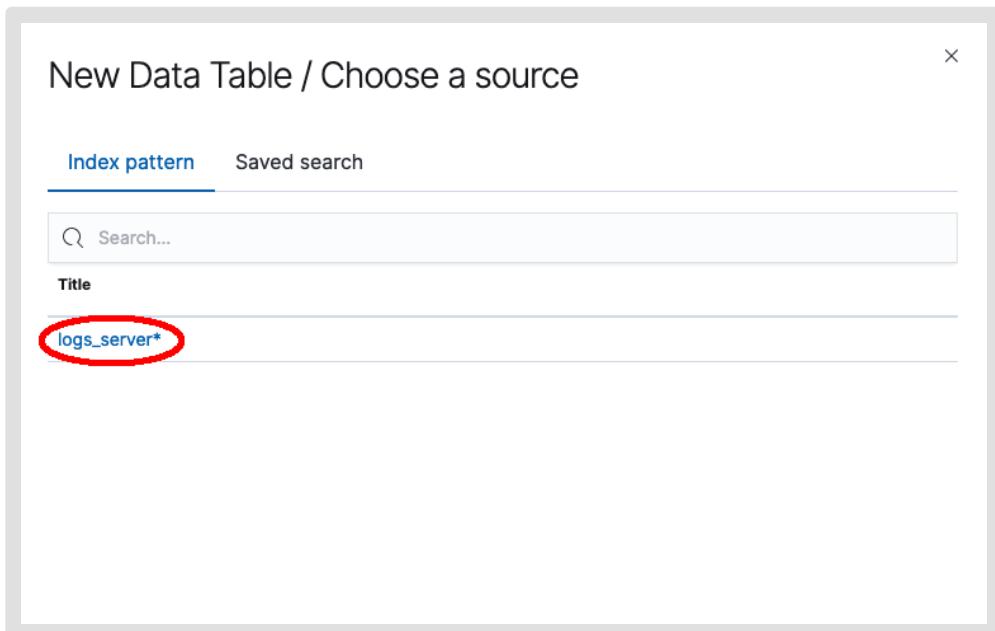
You can create different visualizations, based on your data.

Create new visualization

- Select **Data Table**:



- Click on `logs_server*`. (This is the index pattern we created, which defines from which indices we will read data.)



- Click on Split Rows under **Buckets**:

The screenshot shows the Kibana Metrics visualization interface. On the left, there's a sidebar with various icons and sections like 'Metrics' and 'Buckets'. A context menu is open over the 'Buckets' section, with the 'Add' option highlighted and circled in red. Other options in the menu include 'ADD BUCKET', 'Split rows', and 'Split table'.

- Select Terms under Aggregation:

The screenshot shows the Kibana Metrics visualization interface. The 'Aggregation' dropdown in the sidebar is set to 'Terms', which is highlighted and circled in red. Below it, the 'Field' dropdown is set to 'Select a field'.

- Select originalUrl.keyword under Field:

The screenshot shows the Kibana Metrics visualization interface. The 'Field' dropdown in the sidebar is set to 'OriginalUrl.keyword', which is highlighted and circled in red.

- Click on Apply changes (the button that looks like a "play" button):

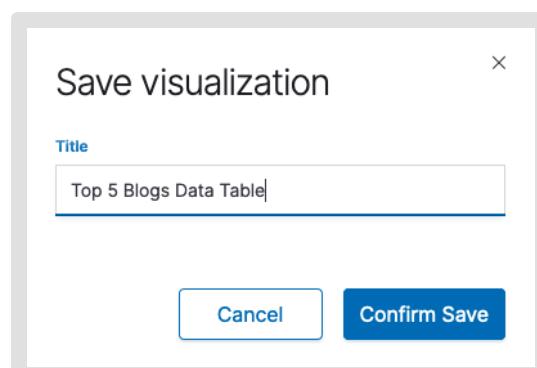
The screenshot shows the Kibana interface with the visualization titled "Top 5 Blogs Data Table". The top navigation bar includes Save, Share, Inspect, Refresh, and a Time Picker set to "Last 15 minutes". The visualization panel on the left is titled "logs_server*" and contains sections for Metrics, Buckets, and Fields. A red circle highlights the "More" button (three dots) in the top right corner of the visualization panel.

- Kibana tells you "No results found". That's because Kibana defaults to showing the last 15 minutes of data. Our data is older than that. In the top right, use the Time Picker to change **Last 15 minutes** into **Last 3 years**:

The screenshot shows the Kibana interface with the visualization titled "Top 5 Blogs Data Table". The top navigation bar includes Save, Share, Inspect, Refresh, and a Time Picker set to "Last 3 years". The visualization panel on the left is titled "logs_server*" and contains sections for Metrics, Buckets, and Fields. A red circle highlights the "3" in the "Quick select" dropdown, another highlights "years", and a third highlights the "Apply" button.

You should see the 5 most popular URLs on Elastic's blog site.

- Save your visualization by clicking on **Save** in the top-left menu and name the visualization "**Top 5 Blogs Data Table**". Then click the **Confirm Save** button to save the visualization.



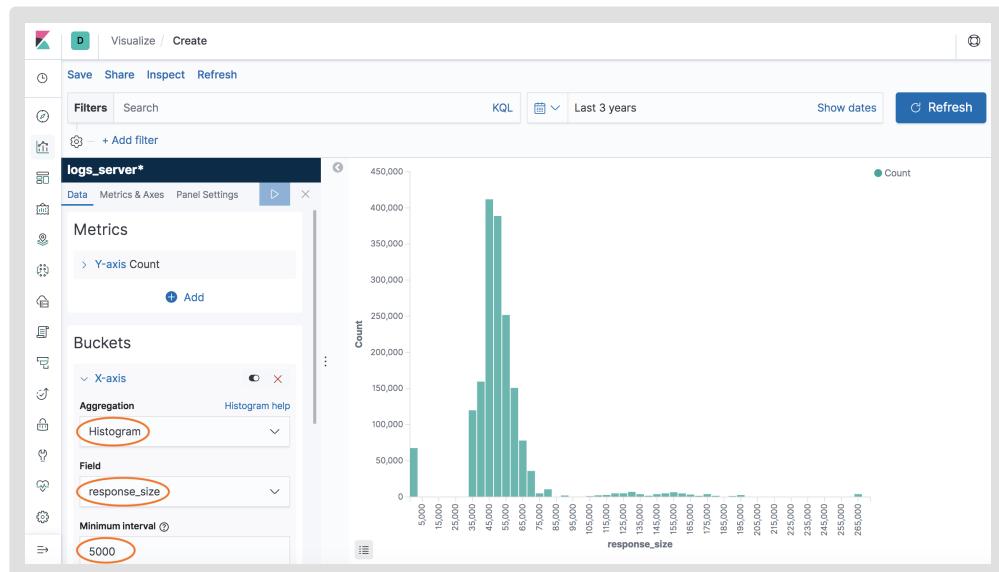
- Click the "**Visualize**" link in the left-hand column of Kibana and you should see your data table in the list of saved visualizations.

15. OPTIONAL: Let's build another visualization - a histogram that shows the distribution of the size of the responses from our website:

- Click on **Visualize** and then on the "+" button to create a new visualization:

The screenshot shows the Kibana Visualize interface. On the left is a vertical toolbar with various icons. In the center, there is a search bar and a table titled "Visualizations". The table has columns for "Title", "Type", and "Actions". One entry, "Top 5 Blogs Data Table", is listed under "Type" as a "Data Table". At the top right of the main area, there is a blue button labeled "+ Create new visualization", which is circled in red.

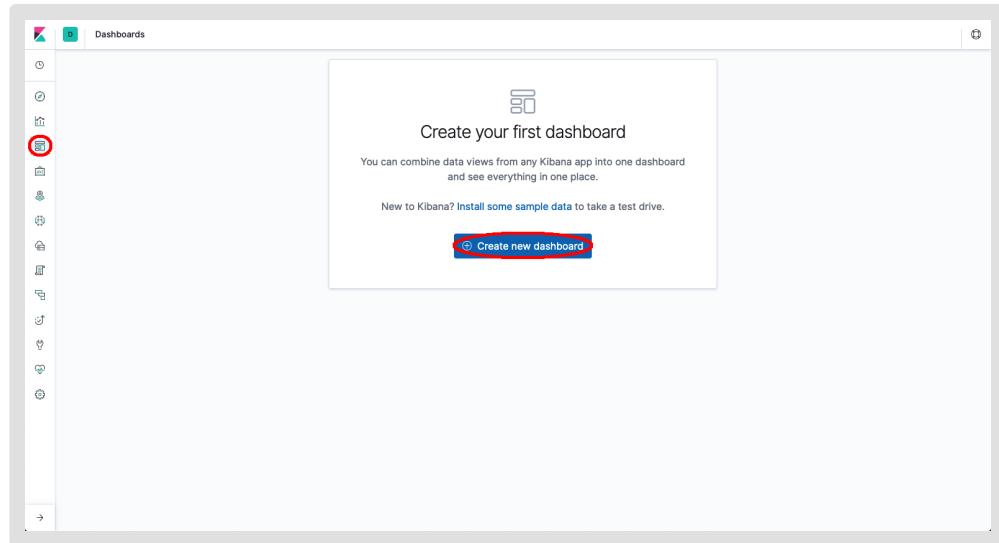
- Select **Vertical Bar**, then select the `logs_server*` index pattern.
- Under **Buckets** choose **X-Axis**
- Under **Aggregation**, choose **Histogram**.
- Select **Field: response_size** and enter **5000** for the **Minimum Interval**.
- Click **Apply changes** to view the bar chart:



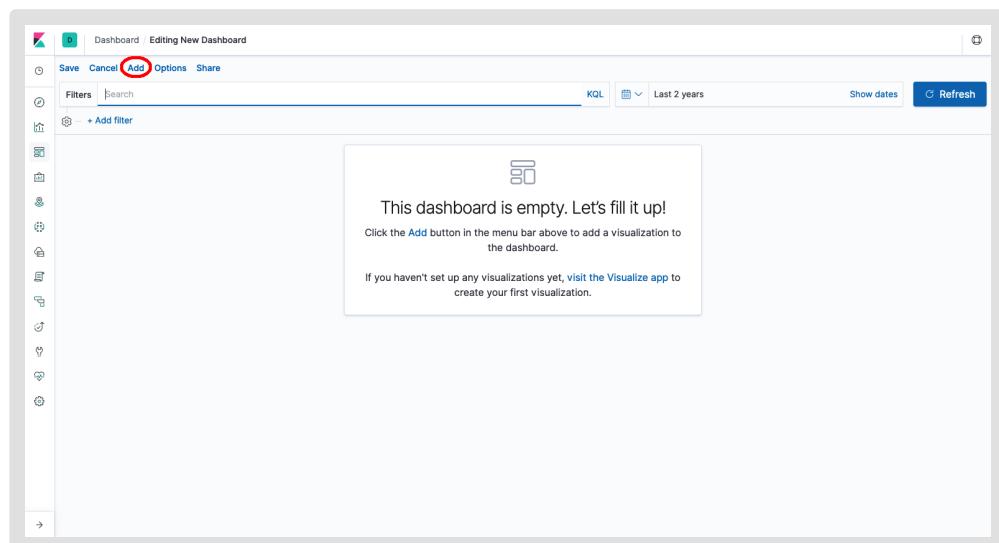
- Save the visualization, giving it the name "**Response Size Bar Chart**"

16. OPTIONAL: After creating visualizations, you can combine them onto a single dashboard:

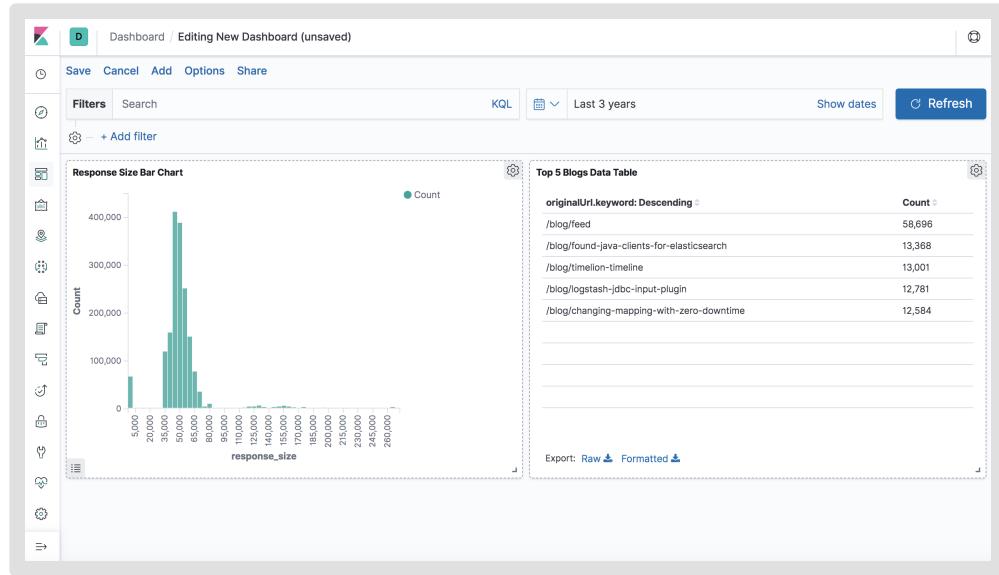
- Click on **Dashboard** in the left-hand toolbar of Kibana, then click on the **Create new dashboard** button:



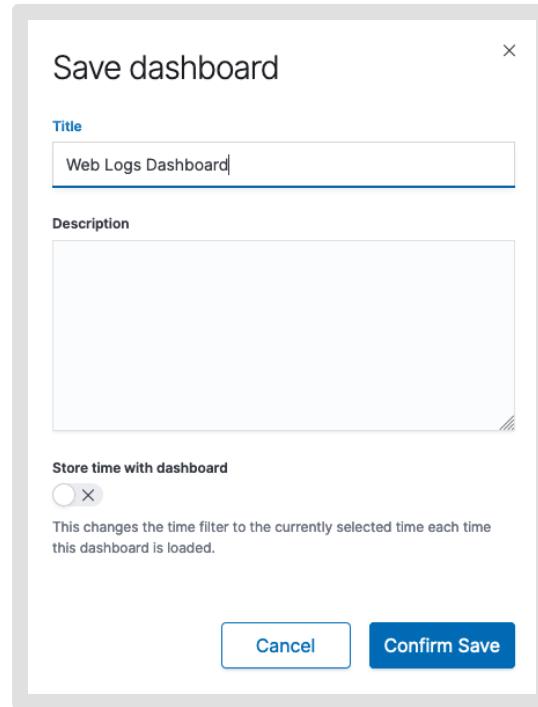
- Click on the **Add** menu item and add both of the visualizations that you saved earlier:



- Close the "**Add Panels**" dialog.
- Feel free to resize the visualizations and move them around:



- Save your dashboard by clicking **Save** in the menu and naming your dashboard "**Web Logs Dashboard**". Then click the **Confirm Save** button to save the dashboard.



- You should now see your saved dashboard on the "**Dashboard**" page of Kibana, so you can come back and view the dashboard at any time.

Summary: In this lab, you used the Elastic Stack to ingest data from a database, to collect data from log files, and to search data. You used Logstash to ingest the blogs from a Postgres database into Elasticsearch and used Filebeat to ingest the web access log data from log files.

End of Lab 1.4

Lab 2.1: Relevance

Objective: In this lab, you will familiarize yourself with the Query DSL. You will write various queries that search documents in the `blogs` index using the Search API and the `match` query.

1. Write and execute a query that matches all the documents in the `blogs` index. You should have a total of 1594 hits:

```
"hits": {  
    "total": 1594
```

[Hide answer](#)

```
GET blogs/_search  
{  
    "query": {  
        "match_all": {}  
    }  
}
```

2. Add the `size` parameter to your previous request and set it to `100`. You should now see 100 blogs in the results.

[Hide answer](#)

```
GET blogs/_search  
{  
    "size": 100,  
    "query": {  
        "match_all": {}  
    }  
}
```

3. You probably noticed that the `content` field is large, which makes it more difficult to navigate the result set. In these cases where the output is too verbose you can use `_source` to filter the output, but keep in mind that this

is done in the JSON level and therefore it adds overhead to Elasticsearch (though it decreases the amount of data transferred in the network).

- You can use `_source` with `excludes` pattern to remove the fields you don't want. For example, run the query below, which does not include the `content` field in the output:

```
GET blogs/_search
{
  "size": 100,
  "_source": {
    "excludes": ["content"]
  },
  "query": {
    "match_all": {}
  }
}
```

- You can also use `_source` with `includes` pattern to specify the fields you want. For example, run the following query below, which includes only the `author` and `title` in the output:

```
GET blogs/_search
{
  "size": 100,
  "_source": {
    "includes": ["author","title"]
  },
  "query": {
    "match_all": {}
  }
}
```

- Alternatively, when just including fields you can list them in an array without the `includes` clause. For example, the previous query can be simplified as follows:

```
GET blogs/_search
{
  "size": 100,
```

```
_source": ["author","title"],  
"query": {  
    "match_all": {}  
}  
}
```

4. Write and execute a `match` query for blogs that have the term "**elastic**" in the `title` field. You should get 260 hits:

```
"hits": {  
    "total": 260,
```

[Hide answer](#)

```
GET blogs/_search  
{  
    "query": {  
        "match": {  
            "title": "elastic"  
        }  
    }  
}
```

5. Now run a `match` query for "**elastic stack**" in the `title` field. Why did the number of hits go up by adding a term to the `match` query?

```
"hits": {  
    "total": 263,
```

[Hide answer](#)

```
GET blogs/_search  
{  
    "query": {  
        "match": {  
            "title": "elastic stack"  
        }  
    }  
}
```

```
    }  
}
```

The search returns blogs with "**elastic**" or "**stack**", so there are additional hits from documents with only "**stack**" in the title .

6. Your search for "**elastic stack**" cast a very wide net. The top hits look good, but the precision is not great for many of the lower-scoring hits, especially those products that have "**elastic**" in the name but not "**stack**". Change the operator of your previous match query to and , then run it again. Notice this increases the precision, as there are now only 70 hits:

```
"hits": {  
    "total": 70,
```

[Hide answer](#)

```
GET blogs/_search  
{  
    "query": {  
        "match": {  
            "title": {  
                "query": "elastic stack",  
                "operator": "and"  
            }  
        }  
    }  
}
```

7. Run a query that answers the question: "*Which blogs have **performance** or **optimizations** or **improvements** in the **content** field?*" You should get the following hits:

```
"hits": {  
    "total": 374,
```

[Hide answer](#)

```
GET blogs/_search
{
  "query": {
    "match": {
      "content": "performance optimizations improvements"
    }
  }
}
```

8. **OPTIONAL:** Run a query that answers the question: "*Which blogs have a content field that includes at least 2 of the terms performance or optimizations or improvements?*" You should get the following hits this time:

```
"hits": {
  "total": 82,
```

[Hide answer](#)

```
GET blogs/_search
{
  "query": {
    "match": {
      "content": {
        "query" : "performance optimizations improvements",
        "minimum_should_match" : 2
      }
    }
  }
}
```

9. **OPTIONAL:** Let's analyze the hits from the "**performance optimizations improvements**" search:

- What was the maximum _score ?

[Hide answer](#)

The exact max score should be around 9.97

- o If you were searching for "**performance optimizations improvements**" to do some fine tuning in your deployment, do you see any issues with some of the results from the group of documents with the highest score?

[Hide answer](#)

Most of them talk about improvements in a specific release of the Elastic Stack. Those blog posts don't talk about performance tuning a cluster. We will see how to improve it later.

Summary: In this lab, you became familiar with the Query DSL. You wrote various queries that searched documents in the `blogs` index using the Search API and the `match` query.

End of Lab 2.1

Lab 2.2: Full-Text Queries

Objective: In this lab, you will deepen your Query DSL knowledge. You will write various queries that search documents in the `blogs` index using queries like `match` , `match_phrase` , and `multi-match` .

1. Write a query for each of the following searches:

- o blogs that have the word "**search**" in their `content` field.

[Hide answer](#)

There are 432 blogs with the word "**search**" in their `content` field.

```
GET blogs/_search
{
  "query": {
    "match": {
      "content": "search"
    }
  }
}
```

```
    }  
}
```

- blogs that have "**search**" or "**analytics**" in their content field.

[Hide answer](#)

There are 476 blogs with the terms "**search**" or "**analytics**" in their content field.

```
GET blogs/_search  
{  
  "query": {  
    "match": {  
      "content": "search analytics"  
    }  
  }  
}
```

- blogs that have "**search** and "**analytics**" in their content field.

[Hide answer](#)

There are 110 blogs with the terms "**search** and "**analytics**" in their content field.

```
GET blogs/_search  
{  
  "query": {  
    "match": {  
      "content": {  
        "query": "search analytics",  
        "operator": "and"  
      }  
    }  
  }  
}
```

- Run a `match_phrase` search for "**search analytics**" in the content field that returns the top 3 hits. You should get 6 hits total.

[Hide answer](#)

There are 6 blogs with the terms "**search**" and "**analytics**" in the content field where the term "**analytics**" is one position after the term "**search**".

```
GET blogs/_search
{
  "size": 3,
  "query": {
    "match_phrase": {
      "content": "search analytics"
    }
  }
}
```

3. The phrase "**search and analytics**" is fairly common in the blog content. Update the previous `match_phrase` query so that it allows for 1 term (any word - not just "**and**") to appear between "**search**" and "**analytics**". How many hits do you see now?

[Hide answer](#)

There are 41 blogs with the terms "**search analytics**" in the content field where the term "**analytics**" is one or two positions after the term "**search**".

```
GET blogs/_search
{
  "query": {
    "match_phrase": {
      "content": {
        "query": "search analytics",
        "slop": 1
      }
    }
  }
}
```

4. Run a query on the `blogs` index for "**open source**" in the content field. Then run a second query for "**open source**" on the title field. Compare

the total hits and top hits returned from both queries. Do the top hits look relevant? Which query had more hits?

[Hide answer](#)

```
GET blogs/_search
{
  "query": {
    "match": {
      "content": "open source"
    }
  }
}
```

```
GET blogs/_search
{
  "query": {
    "match": {
      "title": "open source"
    }
  }
}
```

Searching for "**open source**" in the `title` returns hits that look more relevant, and it also returns far fewer hits.

5. Combine the hits of the last two searches by writing a `multi_match` query that searches both the `content` and `title` fields for "**open source**". Does the `multi_match` deliver more or fewer hits? How did this affect the relevance?

[Hide answer](#)

```
GET blogs/_search
{
  "query": {
    "multi_match": {
      "query": "open source",
      "fields": [
        "title",
        "content"
      ]
    }
  }
}
```

```
        "content"  
    ]  
}  
}  
}
```

The `multi_match` delivered many more hits, and the top hits maintained their relevance.

6. **EXAM PREP:** Modify your `multi_match` query by giving the `title` field a boost of 2. How does the score of the top hit compare to the previous query without the boost?

[Hide answer](#)

```
GET blogs/_search  
{  
  "query": {  
    "multi_match": {  
      "query": "open source",  
      "fields": [  
        "title^2",  
        "content"  
      ]  
    }  
  }  
}
```

The score of the top hit was multiplied by 2.

7. **EXAM PREP:** Boost affects the score without impacting recall or precision. Remove the boost and modify your `multi_match` query to perform a phrase query, which increases precision (perhaps at the expense of recall). Did the increase in precision return more or fewer hits?

[Hide answer](#)

```
GET blogs/_search  
{  
  "query": {
```

```
"multi_match": {  
    "query": "open source",  
    "fields": [  
        "title",  
        "content"  
    ],  
    "type": "phrase"  
}  
}  
}
```

The increased precision reduced the number of hits.

8. **OPTIONAL:** Let's see what happens when a user misspells their query. Run the following request, which searches for "**oven sauce**" in the `title` field. Notice `_source` is used to filter the JSON response to show only the `title` field of the document:

```
GET blogs/_search  
{  
    "_source": "title",  
    "query": {  
        "match": {  
            "title": "oven sauce"  
        }  
    }  
}
```

Were there any hits returned?

[Hide answer](#)

The query does not return any hits.

9. **EXAM PREP:** Try increasing the recall (perhaps at the expense of precision) by adding the `fuzziness` parameter, permitting a maximum of 2 edits per word. Did the increased recall return more or fewer hits? How relevant are they?

[Hide answer](#)

```
GET blogs/_search
{
  "_source": "title",
  "query": {
    "match": {
      "title": {
        "query" : "oven sauce",
        "fuzziness": 2
      }
    }
  }
}
```

There were lots of hits. Many of them do not seem relevant - allowing 2 edits on a 4 letter word with 2 vowels is not great for precision.

10. **EXAM PREP:** Modify your query so that Elasticsearch uses the `auto` fuzziness level. Were more or fewer hits returned? How relevant are they?

[Hide answer](#)

```
GET blogs/_search
{
  "_source": "title",
  "query": {
    "match": {
      "title": {
        "query" : "oven sauce",
        "fuzziness": "auto"
      }
    }
  }
}
```

There are fewer hits, and they seem more relevant than the majority of hits from the previous request.

Summary: In this lab, you deepened your Query DSL knowledge. You wrote various queries that searched documents in the `blogs` index using queries like `match` , `match_phrase` , and `multi-match` .

End of Lab 2.2

Lab 2.3: Combining Queries

Objective: In this lab, you will learn how to logically combine multiple query clauses to query multiple fields effectively and to manage precision and recall to refine or broaden the query scope.

1. In one of the previous labs you wrote a query that answers the question: "*Which blogs have a **content** field that includes **at least 2 of the terms performance or optimizations or improvements?***" Write a query that answers the same question, but using `bool` instead of `match`. You should get the following hits:

```
"hits": {  
    "total": 82,
```

[Hide answer](#)

```
GET blogs/_search  
{  
    "query": {  
        "bool": {  
            "should": [  
                {  
                    "match": {  
                        "content": "performance"  
                    }  
                },  
                {  
                    "match": {  
                        "content": "optimizations"  
                    }  
                },  
                {  
                    "match": {  
                        "content": "improvements"  
                    }  
                }  
            ]  
        }  
    }  
}
```

```
"match": {  
    "content": "improvements"  
}  
}  
],  
"minimum_should_match": 2  
}  
}  
}
```

2. **EXAM PREP:** It looks like releases usually come with performance optimizations and improvements. Assuming that you are not interested in upgrading your deployment, change the previous query (the `should` version) so that it `must_not` contain "**released**" or "**releases**" or "**release**" in the `title` field. Your top hits look better now, and notice the number of total hits dropping down to 47.

[Hide answer](#)

```
GET blogs/_search  
{  
    "query": {  
        "bool": {  
            "should": [  
                {  
                    "match": {  
                        "content": "performance"  
                    }  
                },  
                {  
                    "match": {  
                        "content": "optimizations"  
                    }  
                },  
                {  
                    "match": {  
                        "content": "improvements"  
                    }  
                }  
            ]  
        }  
    }  
}
```

```
        ],
        "minimum_should_match": 2,
        "must_not": [
            {
                "match": {
                    "title": "release releases released"
                }
            }
        ]
    }
}
```

3. **EXAM PREP:** In the previous query, let's say you are more interested in blogs about Elasticsearch. How could you rank the results so that the documents that mention `"elasticsearch"` in the `title` score higher? (*TIP: you must have two separate should clauses. One clause with a minimum_should_match will work like a must . The other clause will influence the score.*)

[Hide answer](#)

To implement this new requirement you need to change the structure of the current query. As said in the tip, you must have two separate `should` clauses. Move the original `should` clause (the one with a `minimum_should_match`) inside a `must` . To have a logical clause inside another logical clause, you need to use an inner `bool` query. Then, add the new `should` clause that will influence the score to the main `bool` query.

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "bool": {
            "should": [
              {
                "match": {
                  "content": "performance"
                }
              }
            ]
          }
        }
      ]
    }
  }
}
```

```
        }
    },
{
    "match": {
        "content": "optimizations"
    }
},
{
    "match": {
        "content": "improvements"
    }
}
],
"minimum_should_match": 2
}
}
],
"must_not": [
{
    "match": {
        "title": "release releases released"
    }
}
],
"should": [
{
    "match": {
        "title": "elasticsearch"
    }
}
]
}
}
```

4. **OPTIONAL:** If you are curious about what your query actually means when executed in Lucene, you can use the `_validate` API along with the `rewrite` parameter as shown below.

```
GET blogs/_validate/query?rewrite=true  
{...}
```

The validate API allows a user to validate a potentially expensive query without executing it. You can read more about it [here](#).

- Use the header above to compare the match and should versions of the first step in this lab (copied below). Are they the same query?

```
GET blogs/_search  
{  
  "query": {  
    "match": {  
      "content": {  
        "query" : "performance optimizations improvements",  
        "minimum_should_match" : 2  
      }  
    }  
  }  
}
```

```
GET blogs/_search  
{  
  "query": {  
    "bool": {  
      "should": [  
        {  
          "match": {  
            "content": "performance"  
          }  
        },  
        {  
          "match": {  
            "content": "optimizations"  
          }  
        },  
        {  
          "match": {  
            "content": "improvements"  
          }  
        }  
      ]  
    }  
  }  
}
```

```
        "content": "improvements"
    }
],
"minimum_should_match": 2
}
}
```

[Hide answer](#)

Yes, they are same query when executed at the Lucene level.

```
GET blogs/_validate/query?rewrite=true
{
  "query": {
    "match": {
      "content": {
        "query" : "performance optimizations improvements",
        "minimum_should_match" : 2
      }
    }
  }
}
```

```
{
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "valid" : true,
  "explanations" : [
    {
      "index" : "blogs",
      "valid" : true,
      "explanation" : "(content:performance content:optimiza"
    }
}
```

```
    ]  
}
```

Summary: In this lab, you learned how to logically combine multiple query clauses to query multiple fields effectively and to manage precision and recall to refine or broaden the query scope.

End of Lab 2.3

Lab 2.4: Implementing a Search Page

Objective: In this lab, you will learn how to implement some features that users would expect to find in any good search application, which might enable them to explore and retrieve the documents most relevant to their needs using correct ergonomics and efficient use of system resources.

1. Imagine the blogs search page is implemented using the `multi_match` query below and that a user is searching for "meetups".

```
GET blogs/_search  
{  
  "query": {  
    "bool": {  
      "must": [  
        {  
          "multi_match": {  
            "query": "meetups",  
            "fields": ["title", "content"]  
          }  
        }  
      ]  
    }  
  }  
}
```

2. Now, imagine the user selected **News** on the left side bar. Update the query above to add the user selection. The result should be 5 documents.

meetups

Categories

- (117)
- This week in Elasticsearch and Apache Lucene (74)
- News (9)
- User Stories (4)
- Culture (3)

[Hide answer](#)

The left side selection should be implemented inside a `filter`. If you implement a `match` query on the `category` field, it will work for **News**, but it might not work for compound categories. Therefore, the correct solution is to use the `keyword` field, which in this particular case is `category.keyword`.

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "multi_match": {
            "query": "meetups",
            "fields": ["title", "content"]
          }
        }
      ],
    }
  }
},
```

```
"filter": {  
    "match": {  
        "category.keyword": "News"  
    }  
}  
}  
}  
}
```

3. **EXAM PREP:** Next, imagine the user also selected **Culture** on the left side bar. Update the query above to add the new user selection. The result should be 7 documents. (*TIP: you need to implement an OR logic combined with an AND logic. Which boolean query allows you to run ORs?*)

[Hide answer](#)

The solution is not straight forward. Adding a clause to the filter has an AND behavior and returns no results. To create a proper OR behavior, all you need is a `should` with both clauses (the default `minimum_should_match` is 1). Then, because this is a **yes** or **no** question, you should make use of filters. Finally, to add a `should` clause inside a `filter` clause, you need to add a `bool` clause in between.

```
GET blogs/_search  
{  
    "query": {  
        "bool": {  
            "must": [  
                {  
                    "multi_match": {  
                        "query": "meetups",  
                        "fields": [  
                            "title",  
                            "content"  
                        ]  
                    }  
                }  
            ],  
            "filter": [  
                {  
                    "bool": {  
                        "should": [  
                            {"term": {  
                                "category": "Meetups"  
                            }},  
                            {"term": {  
                                "category": "Culture"  
                            }}  
                        ]  
                    }  
                }  
            ]  
        }  
    }  
}
```

```
"bool": {  
    "should": [  
        {  
            "match": {  
                "category.keyword": "News"  
            }  
        },  
        {  
            "match": {  
                "category.keyword": "Culture"  
            }  
        }  
    ]  
},  
}  
]  
}  
}  
}
```

4. You might be thinking: "*Is this the best solution for such a common problem?*". The answer is no. There is a better way to implement the query above. There is a set of [Term Level Queries](#) that don't perform analysis. The **Terms Query** is a great option to implement facetting filters. The query would then look like the one below:

```
GET blogs/_search  
{  
    "query": {  
        "bool": {  
            "must": [  
                {  
                    "multi_match": {  
                        "query": "meetups",  
                        "fields": [  
                            "title",  
                            "content"  
                        ]  
                    }  
                }  
            ]  
        }  
    }  
}
```

```
        }
    },
],
"filter": [
{
    "terms": {
        "category.keyword": [
            "News",
            "Culture"
        ]
    }
}
]
```

5. **EXAM PREP:** Next, imagine the user adds a date to the search, so only blogs published in **2017** are returned. Update the query above to only return blogs published in **2017**. The result should be 2 documents.

[Hide answer](#)

```
GET blogs/_search
{
    "query": {
        "bool": {
            "must": [
                {
                    "multi_match": {
                        "query": "meetups",
                        "fields": [
                            "title",
                            "content"
                        ]
                    }
                }
            ],
            "filter": [

```

```
{  
    "terms": {  
        "category.keyword": [  
            "News",  
            "Culture"  
        ]  
    },  
    {  
        "range": {  
            "publish_date": {  
                "gte": "2017-01-01",  
                "lt": "2018-01-01"  
            }  
        }  
    }  
}
```

6. Write a `match_phrase` query that searches for "**elastic stack**" in the `content` field. Sort the results so that they are returned from newest to oldest (based on the `publish_date` field).

[Hide answer](#)

```
GET blogs/_search  
{  
    "query": {  
        "match_phrase": {  
            "content": {  
                "query" : "elastic stack"  
            }  
        }  
    },  
    "sort": [  
        {  
            "publish_date": {  
                "order": "desc"  
            }  
        }  
    ]  
}
```

```
        "order": "desc"
    }
}
]
}
```

7. **EXAM PREP:** Modify your previous query so that the results are sorted first by author name ascending, and then from newest to oldest.

[Hide answer](#)

```
GET blogs/_search
{
  "query": {
    "match_phrase": {
      "content": {
        "query" : "elastic stack"
      }
    }
  },
  "sort": [
    {
      "author.keyword": {
        "order": "asc"
      }
    },
    {
      "publish_date": {
        "order": "desc"
      }
    }
  ]
}
```

8. Our web application only shows three blog hits at a time. Modify the previous query so that it only returns the top 3 hits.

[Hide answer](#)

```
GET blogs/_search
{
  "size": 3,
  "query": {
    "match_phrase": {
      "content": {
        "query" : "elastic stack"
      }
    }
  },
  "sort": [
    {
      "author.keyword": {
        "order": "asc"
      }
    },
    {
      "publish_date": {
        "order": "desc"
      }
    }
  ]
}
```

9. **EXAM PREP:** Suppose a user clicks on page 4 of the search results from your previous query. Write a query that returns the 3 hits of page 4.

[Hide answer](#)

```
GET blogs/_search
{
  "from": 9,
  "size": 3,
  "query": {
    "match_phrase": {
      "content": {
        "query" : "elastic stack"
      }
    }
  }
}
```

```
        }
    },
    "sort": [
        {
            "author.keyword": {
                "order": "asc"
            }
        },
        {
            "publish_date": {
                "order": "desc"
            }
        }
    ]
}
```

10. **EXAM PREP:** Modify the previous query to implement highlighting on the `content` field. It should enclose the matched search terms with a `<mark>` HTML tag.

[Hide answer](#)

```
GET blogs/_search
{
    "from": 9,
    "size": 3,
    "query": {
        "match_phrase": {
            "content": {
                "query" : "elastic stack"
            }
        }
    },
    "sort": [
        {
            "author.keyword": {
                "order": "asc"
            }
        },
        {
            "publish_date": {
                "order": "desc"
            }
        }
    ]
}
```

```
{  
    "publish_date": {  
        "order": "desc"  
    }  
},  
"highlight": {  
    "fields": {  
        "content" : {}  
    },  
    "pre_tags": ["<mark>"],  
    "post_tags": ["</mark>"]  
}  
}
```

Summary: You have implemented most of the search features used in our blog search web application! In this lab, you controlled the presentation of search hits using sort, pagination and highlighting, as well as filtering results by category.

End of Lab 2.4

Lab 3.1: Metrics Aggregations

Objective: In this lab, you will become familiar with writing metrics aggregations to answer some questions about the `logs_server*` indices.

1. What is the runtime of the fastest request? (*TIP: set `size` to 0.)

[Hide answer](#)

It is 0.0 milliseconds.

```
GET logs_server*/_search  
{  
    "size": 0,
```

```
"aggs": {  
    "fastest_request_time": {  
        "min": {  
            "field": "runtime_ms"  
        }  
    }  
}
```

2. What is the runtime of the slowest request?

[Hide answer](#)

It is 59756.0 milliseconds.

```
GET logs_server*/_search  
{  
    "size": 0,  
    "aggs": {  
        "slowest_request_time": {  
            "max": {  
                "field": "runtime_ms"  
            }  
        }  
    }  
}
```

3. You can use the `stats` aggregation when you want to calculate all the main metrics (min, max, avg, and sum). Update the aggs above to use `stats` instead of `max`. What is the average runtime?

[Hide answer](#)

The average runtime is approximately 361.64 milliseconds.

```
GET logs_server*/_search  
{  
    "size": 0,  
    "aggs": {
```

```
"request_time_stats": {  
    "stats": {  
        "field": "runtime_ms"  
    }  
}  
}
```

If you want even more details, you can use the `extended_stats` aggs to calculate `sum_of_squares`, `variance`, `std_deviation`, and `std_deviation_bounds`.

4. Median often is a better option than average, as a single result can significantly impact the average. Calculate the median runtime and verify if 95% of the requests take less than 500 milliseconds.

[Hide answer](#)

The median is calculated using the `percentile` aggregation. The median runtime is approximately 103.510 milliseconds, which is much lower than the average. And 95% of the requests take less than approximately 684.404 milliseconds.

```
GET logs_server*/_search  
{  
    "size": 0,  
    "aggs": {  
        "runtime_median_and_95": {  
            "percentiles": {  
                "field": "runtime_ms",  
                "percents": [  
                    50,  
                    95  
                ]  
            }  
        }  
    }  
}
```

5. Assuming 500 milliseconds is our runtime goal, what percentage of the requests is within this time?

[Hide answer](#)

The `percentile_ranks` aggregations allows you to provide a value and get back the percentile it represents. Approximately 94.3% of the requests take 500 milliseconds or less.

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "runtime_goal": {
      "percentile_ranks": {
        "field": "runtime_ms",
        "values": [
          500
        ]
      }
    }
  }
}
```

6. How many distinct URL requests were logged in the `logs_server*` indices? URL requests are indexed in the `originalUrl` field. You should get around 37,000 as the result.

[Hide answer](#)

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "my_url_value_count": {
      "cardinality": {
        "field": "originalUrl.keyword"
      }
    }
  }
}
```

```
    }  
}
```

7. **EXAM PREP:** Add a query that limits the scope of your aggregation in the previous step to only documents that contain the term "**elastic**" in the `originalUrl` field. You should get around 4,500 as the result.

[Hide answer](#)

```
GET logs_server*/_search  
{  
  "size": 0,  
  "query": {  
    "match": {  
      "originalUrl": "elastic"  
    }  
  },  
  "aggs": {  
    "my_url_value_count": {  
      "cardinality": {  
        "field": "originalUrl.keyword"  
      }  
    }  
  }  
}
```

Summary: In this lab, you became familiar with writing metrics aggregations to answer some questions.

End of Lab 3.1

Lab 3.2: Bucket Aggregations

Objective: In this lab, you will become familiar with writing bucket aggregations to answer some questions.

1. How many requests are there for each of the 6 `status_code` values?
(*TIP: set size to 0.)

[Hide answer](#)

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "status_code_buckets": {
      "terms": {
        "field": "status_code"
      }
    }
  }
}
```

2. **EXAM PREP:** A `terms` aggregation is sorted by `doc_count` by default. Modify your previous search so that its `terms` are sorted alphabetically.

[Hide answer](#)

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "status_code_buckets": {
      "terms": {
        "field": "status_code",
        "order": {
          "_key": "asc"
        }
      }
    }
  }
}
```

3. The following query has 165 hits. Our search page has a UI that allows users to filter those hits by category, and notice that our UI shows how many

hits belong to each category. Add an aggregation to the following query that returns the number of hits in each category.

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": {
        "multi_match": {
          "query": "open source",
          "fields": [
            "title^2",
            "content"
          ],
          "type": "phrase"
        }
      }
    }
  },
  "highlight": {
    "fields": {
      "title": {},
      "content": {}
    },
    "require_field_match": false,
    "pre_tags": [
      "<mark>"
    ],
    "post_tags": [
      "</mark>"
    ]
  }
}
```

[Hide answer](#)

```
GET blogs/_search
{
  "query": {
```

```
"bool": {  
    "must": {  
        "multi_match": {  
            "query": "open source",  
            "fields": [  
                "title^2",  
                "content"  
            ],  
            "type": "phrase"  
        }  
    }  
},  
"highlight": {  
    "fields": {  
        "title": {},  
        "content": {}  
    },  
    "require_field_match": false,  
    "pre_tags": [  
        "<mark>"  
    ],  
    "post_tags": [  
        "</mark>"  
    ]  
},  
"aggs": {  
    "category_terms": {  
        "terms": {  
            "field": "category.keyword",  
            "size": 10  
        }  
    }  
}
```

4. Write an aggregation that returns the `response_size` distribution for the logs indices with an interval of 10000.

[Hide answer](#)

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "runtime_histogram": {
      "histogram": {
        "field": "response_size",
        "interval": 10000
      }
    }
  }
}
```

5. **EXAM PREP:** Notice that some of the returned buckets have few documents. Update the aggregation to exclude the buckets that have less than 1000 documents.

[Hide answer](#)

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "runtime_histogram": {
      "histogram": {
        "field": "response_size",
        "interval": 10000,
        "min_doc_count": 1000
      }
    }
  }
}
```

6. How many log requests are there for each week?

[Hide answer](#)

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "logs_by_week": {
      "date_histogram": {
        "field": "@timestamp",
        "calendar_interval": "week"
      }
    }
  }
}
```

7. Modify the previous aggregation to return how many requests per minute.
What happened?

[Hide answer](#)

The request looks like the following:

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "logs_by_week": {
      "date_histogram": {
        "field": "@timestamp",
        "interval": "minute"
      }
    }
  }
}
```

Elasticsearch returns the following error:

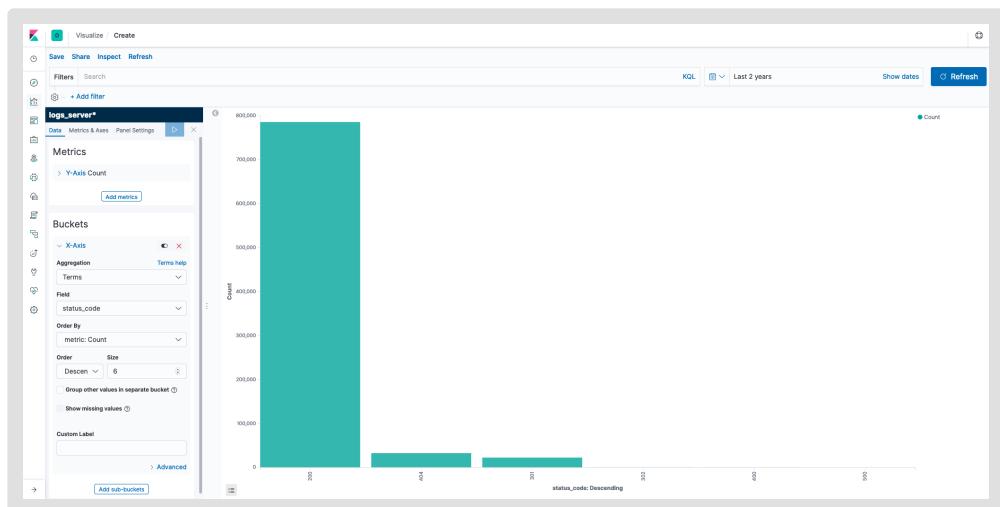
```
{
  "type": "too_many_buckets_exception",
  "reason": "Trying to create too many buckets. Must be less than or equal to 2^31 - 1 (2147483647). Got 2147483648."}
```

```
"max_buckets": 10000
}
```

It means the requested aggregation would generate too many buckets and put the cluster in danger. Therefore, Elasticsearch will not execute it. The maximum number of buckets allowed in a single response is limited by a dynamic cluster setting named `search.max_buckets`. As you can see in the response, it defaults to 10000 and requests that try to return more than the limit fail with an exception.

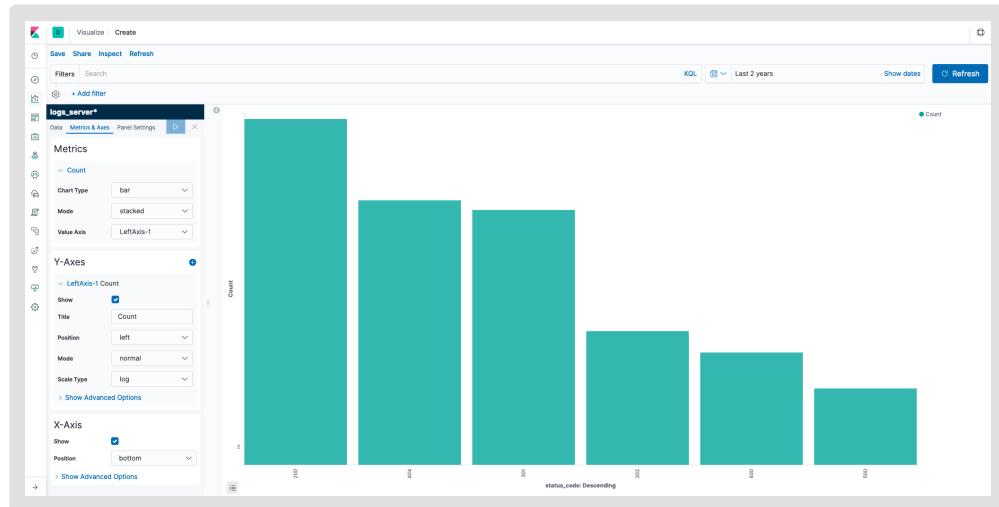
8. OPTIONAL: Congratulations, you have answered a lot of questions using aggregations in Elasticsearch. Now you will render an aggregation in Kibana:

- Click on the **Visualize** icon on the left hand side bar
- Click on **+** to create a visualization
- Click on the **Vertical Bar** icon
- Click on the hyperlinked index pattern `logs_server`*
- In the **Buckets** control select Buckets type X-Axis
- In the **Aggregation** drop-down select Terms (you may have to scroll down)
- In the **Field** drop-down select `status_code`
- Set **Size** to 6
- Click on the **Apply Changes** icon and you should see the following bar chart (make sure your time range is "Last 5 years"):



- Now Click on **Metrics & Axes**

- In the Y-Axes control click on **LeftAxis-1**
- Change **Scale Type** from **linear** to **log** and click **Apply Changes** again to improve visibility of the smaller buckets:



Summary: In this lab, you became familiar with writing bucket aggregations to answer some questions.

End of Lab 3.2

Lab 3.3: Combining Aggregations

Objective: In this lab, you will become familiar with writing aggregations that combine multiple metrics and/or bucket aggregations to answer some questions.

1. How many blog requests are there for each week? (*TIP: *set size to 0.)

[Hide answer](#)

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "logs_by_week": {
      "date_histogram": {
        "field": "@timestamp",
        "interval": "1w"
      }
    }
  }
}
```

```
        "calendar_interval": "week"
    }
}
}
```

2. **EXAM PREP:** For each week of blog requests, how many requests were received from each of the 6 values of `status_code` ?

[Hide answer](#)

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "logs_by_week": {
      "date_histogram": {
        "field": "@timestamp",
        "calendar_interval": "week"
      },
      "aggs": {
        "status_code_buckets": {
          "terms": {
            "field": "status_code"
          }
        }
      }
    }
  }
}
```

3. **EXAM PREP:** What is the median runtime for each `status_code` ? (**TIP:** This is not per week.)

[Hide answer](#)

```
GET logs_server*/_search
{
```

```
"size": 0,  
"aggs": {  
    "status_code_buckets": {  
        "terms": {  
            "field": "status_code"  
        },  
        "aggs": {  
            "runtime": {  
                "percentiles": {  
                    "field": "runtime_ms",  
                    "percents": [50]  
                }  
            }  
        }  
    }  
}
```

4. **EXAM PREP:** What are the top 3 URLs accessed from each of the top 20 cities? Analyze the results closely and notice there is a common set of URLs for most cities.

[Hide answer](#)

```
GET logs_server*/_search  
{  
    "size": 0,  
    "aggs": {  
        "top_cities": {  
            "terms": {  
                "field": "geoip.city_name.keyword",  
                "size": 20  
            },  
            "aggs": {  
                "top_urls": {  
                    "terms": {  
                        "field": "originalUrl.keyword",  
                        "size": 3  
                    }  
                }  
            }  
        }  
    }  
}
```

```
        }
    }
}
}
```

5. Change the `terms` aggregation of the top 3 URLs to a `significant_terms` aggregation and compare the results of the two different queries. Notice how the URLs have changed to be less generic and more specific topics.

[Hide answer](#)

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "top_cities": {
      "terms": {
        "field": "geoip.city_name.keyword",
        "size": 20
      },
      "aggs": {
        "top_urls": {
          "significant_terms": {
            "field": "originalUrl.keyword",
            "size": 3
          }
        }
      }
    }
  }
}
```

6. **EXAM PREP:** Write a query that searches for "`elasticsearch siem`" in the `content` field and use this scope of documents to list only the `title` field of the top three blogs of each one of the top 5 categories.

[Hide answer](#)

```
GET blogs/_search
{
  "size": 0,
  "query": {
    "match": {
      "content": "elasticsearch siem"
    }
  },
  "aggs": {
    "top5_categories": {
      "terms": {
        "field": "category.keyword",
        "size": 5
      },
      "aggs": {
        "top3_blogs": {
          "top_hits": {
            "size": 3,
            "_source": ["title"]
          }
        }
      }
    }
  }
}
```

Summary: In this lab, you became familiar with writing aggregations that combine multiple metrics and/or bucket aggregations to answer some questions.

End of Lab 3.3

Lab 4.1: What is a Mapping?

Objective: In this lab, you will become familiar with how to read an index mapping and how to define your own mappings.

1. Let's start this lab by analyzing the dynamic mapping behavior. Index the following sample document into a temporary index named `tmp_index`:

```
POST tmp_index/_doc/1
{
  "@timestamp": "2019-07-08T03:00:00.000Z",
  "ip" : "105.32.126.44",
  "bytes" : 8261,
  "coordinates" : {
    "lat" : 30.42769722,
    "lon" : -87.70082
  }
}
```

2. View the mappings of `tmp_index` that were dynamically created.

[Hide answer](#)

The output of:

```
GET tmp_index/_mappings
```

should return the following:

```
{
  "tmp_index" : {
    "mappings" : {
      "properties" : {
        "@timestamp" : {
          "type" : "date"
        },
        "bytes" : {
          "type" : "long"
        },
        "coordinates" : {
          "properties" : {
```

```
        "lat" : {  
            "type" : "float"  
        },  
        "lon" : {  
            "type" : "float"  
        }  
    },  
    "ip" : {  
        "type" : "text",  
        "fields" : {  
            "keyword" : {  
                "type" : "keyword",  
                "ignore_above" : 256  
            }  
        }  
    }  
}
```

3. The @timestamp field was mapped as type date , bytes as type long , coordinates as an object with two properties lat and lon of type float , and finally ip was mapped as type text (which will be discussed in the next lesson).

Which of the fields' mappings could be improved?

[Hide answer](#)

@timestamp and bytes have probably been mapped correctly, but coordinates could be better mapped as geo_point and ip could be better mapped as ip .

4. To illustrate why the mapping of the ip field needs to be improved, execute the following query to find documents with an IP within the 105.32.0.0/16 range:

```
GET tmp_index/_search
{
  "query": {
    "term": {
      "ip": "105.32.0.0/16"
    }
  }
}
```

Even though the IP in our document falls within this range, the document is not a match. This is because it has been mapped as type `text` instead of `ip`. Let's improve the mapping!

5. **EXAM PREP:** Create a new index `my_logs` with the `ip` field mapped explicitly as type `ip`.

[Hide answer](#)

```
PUT my_logs
{
  "mappings": {
    "properties": {
      "ip": {
        "type": "ip"
      }
    }
  }
}
```

6. Index the following document with the same IP as before into `my_logs`:

```
POST my_logs/_doc
{
  "ip" : "105.32.126.44"
}
```

7. Execute the query again, now against the new index. The document should now be returned as a hit:

```
GET my_logs/_search
{
  "query": {
    "term": {
      "ip": "105.32.0.0/16"
    }
  }
}
```

8. Let's also fix the `location` field. To see why, execute the following `geo_distance` query on the `tmp_index` index.

```
GET tmp_index/_search
{
  "query": {
    "geo_distance": {
      "distance": "200km",
      "coordinates": {
        "lat": 30,
        "lon": -87
      }
    }
  }
}
```

What is the result?

[Hide answer](#)

The query fails, with the following message: "failed to find geo_point field [coordinates]" . The `coordinates` field is not mapped as a `geo_point` .

9. Add a new field `coordinates` to the mapping of the `my_logs` index, and map the field as type `geo_point` .

[Hide answer](#)

```
PUT my_logs/_mapping
{
  "properties": {
    "coordinates": {
      "type": "geo_point"
    }
  }
}
```

10. Now that the `ip` and `coordinates` fields have been fixed, let's index the same document to `my_logs`.

```
POST my_logs/_doc/1
{
  "@timestamp": "2019-07-08T03:00:00.000Z",
  "ip" : "105.32.126.44",
  "bytes" : 8261,
  "coordinates" : {
    "lat" : 30.42769722,
    "lon" : -87.70082
  }
}
```

11. To validate that the `coordinates` field is now mapped correctly, run the previous `geo_distance` query again, but now against the new index:

```
GET my_logs/_search
{
  "query": {
    "geo_distance": {
      "distance": "200km",
      "coordinates": {
        "lat": 30,
        "lon": -87
      }
    }
}
```

```
}
```

The document should now be returned as a hit.

Summary: In this lab, you became familiar with how to read an index mapping and how to define your own mappings.

End of Lab 4.1

Lab 4.2: Text and Keyword Strings

Objective: In this lab you will take a close look at how strings are mapped dynamically and learn how different analyzers work.

1. Let's start this lab by indexing this sample document into the temporary index tmp_index :

```
POST tmp_index/_doc/2
{
  "country_name": "United Kingdom"
}
```

2. How has this field been dynamically mapped by Elasticsearch?

[Hide answer](#)

The output of:

```
GET tmp_index/_mappings
```

shows that this field has been mapped as a `text` field, with a multifield called `keyword` of type `keyword`. This is how Elasticsearch will dynamically map any string by default.

```
"country_name" : {  
    "type" : "text",  
    "fields" : {  
        "keyword" : {  
            "type" : "keyword",  
            "ignore_above" : 256  
        }  
    }  
}
```

3. To see the different behavior of the `text` and `keyword` fields, try to query both the `country_name` and the `country_name.keyword` field for the search term `kingdom`. Which of the two queries do you expect to return a hit?

```
GET tmp_index/_search  
{  
    "query": {  
        "match": {  
            "country_name": "kingdom"  
        }  
    }  
}  
  
GET tmp_index/_search  
{  
    "query": {  
        "match": {  
            "country_name.keyword": "kingdom"  
        }  
    }  
}
```

[Hide answer](#)

The second query returns no hits. You can only query `keyword` field for exact values.

4. How would you have to rewrite the query so it returns the sample document you indexed earlier?

[Hide answer](#)

The `keyword` field is not analyzed. As a result, you can only find this document if you use the exact string `United Kingdom` (exactly as it occurs in the sample document) as the query string, with all the exact capitalization and whitespaces:

```
GET tmp_index/_search
{
  "query": {
    "match": {
      "country_name.keyword": "United Kingdom"
    }
  }
}
```

5. To help you understand how text analysis works, Elasticsearch provides an `_analyze` API. For example, to see what would happen to the string `"United Kingdom"` if you applied the `standard` analyzer, you can use:

```
GET _analyze
{
  "text": "United Kingdom",
  "analyzer": "standard"
}
```

What two tokens are the output of the request above?

[Hide answer](#)

The two tokens are `united` and `kingdom` (both lowercased).

6. Let's take a closer look at analyzers. Compare the output of the `_analyze` API on the string `"Nodes and Shards"` using the `standard` analyzer and using the `english` analyzer.

[Hide answer](#)

```
GET _analyze
{
  "text": "Nodes and Shards",
  "analyzer": "standard"
}
```

```
GET _analyze
{
  "text": "Nodes and Shards",
  "analyzer": "english"
}
```

The standard analyzer outputs three tokens: nodes , and , and shards . The english analyzer however only outputs two tokens: node and shard . There are two things that happened here:

First, the english analyzer removed so called "stop words". Certain words, like the word "and" in English do not add a lot of relevance to queries. Almost every document will contain the word "and", if your documents are in English. That's why you can choose to ignore those words to save on disk space, by not indexing those.

Secondly, the tokens nodes and shards were stemmed by the english analyzer, meaning they were reduced to their root form. In this case, the plural nodes and shards became the singular node and shard . By applying stemming to both the indexed data as well as the queries, the user does not need to worry about whether a search term is singular or plural.

7. Considering the datasets that you are using in this training, what would be good fields to apply the english analyzer to?

[Hide answer](#)

Given that most of the blog posts are in English, the english analyzer would work great on fields like title and content in the blogs dataset.

8. **EXAM PREP:** Add a new field named title to the mapping of the tmp_index index. Map the field as type text , with the english analyzer.

[Hide answer](#)

```
PUT tmp_index/_mapping
{
  "properties": {
    "title": {
      "type": "text",
      "analyzer": "english"
    }
  }
}
```

9. Index the following document into the `tmp_index` index:

```
POST tmp_index/_doc/3
{
  "title": "Nodes and Shards"
}
```

10. Validate that you can indeed find this document when you query the `title` field for `node` (singular):

```
GET tmp_index/_search
{
  "query": {
    "match": {
      "title": "node"
    }
  }
}
```

Summary: In this lab you learned how to work with `text` and `keyword` fields and took a closer look at the behavior of the `english` analyzer.

End of Lab 4.2

Lab 4.3: The Inverted Index and Doc Values

Objective: In this lab you are going to take a closer look at the inverted index and doc values.

1. In the previous lab you indexed a document with the value "United Kingdom" for the `country_name` field. Run the following `terms` aggregation on all the documents in the `tmp_index` index to find the most common values for `country_name`:

```
GET tmp_index/_search
{
  "size": 0,
  "aggs": {
    "top_countries": {
      "terms": {
        "field": "country_name",
        "size": 10
      }
    }
  }
}
```

What is the result?

[Hide answer](#)

The operation returns an error. Instead of aggregating on the `country_name` field, you should aggregate on the `country_name.keyword` field, because `keyword` fields have doc values enabled.

```
GET tmp_index/_search
{
  "size": 0,
  "aggs": {
    "top_countries": {
      "terms": {
```

```
        "field": "country_name.keyword",
        "size": 10
    }
}
}
}
```

2. Add two new fields `region_name` and `city_name` to the mapping of the `tmp_index` index. Map both as type `keyword`. Disable the inverted index for `region_name`. Disable doc values for `city_name`.

[Hide answer](#)

```
PUT tmp_index/_mapping
{
  "properties": {
    "region_name": {
      "type": "keyword",
      "index": false
    },
    "city_name": {
      "type": "keyword",
      "doc_values": false
    }
  }
}
```

3. Index the following sample document:

```
POST tmp_index/_doc/4
{
  "region_name": "Hertfordshire",
  "city_name": "Hatfield"
}
```

4. Out of the following two operations, which one do you expect to fail? Why?

```
GET tmp_index/_search
{
  "query": {
    "match": {
      "region_name": "Hertfordshire"
    }
  }
}

GET tmp_index/_search
{
  "size": 0,
  "aggs": {
    "top_regions": {
      "terms": {
        "field": "region_name"
      }
    }
  }
}
```

[Hide answer](#)

You have disabled the inverted index for `region_name`. As a result, you cannot query that field. The `region_name` still has doc values enabled though, so the `terms` aggregation can execute without any problems.

5. Try the same for the `city_name` field. Which one will fail now?

```
GET tmp_index/_search
{
  "query": {
    "match": {
      "city_name": "Hatfield"
    }
  }
}

GET tmp_index/_search
```

```
{  
    "size": 0,  
    "aggs": {  
        "top_regions": {  
            "terms": {  
                "field": "city_name"  
            }  
        }  
    }  
}
```

[Hide answer](#)

This time it is the `terms` aggregation that fails. You have disabled doc values for `city_name`. As a result, you cannot aggregate on that field. The `city_name` still has the inverted index enabled though, so you can still query that field.

6. Let's take a close look at the logs in the `logs_server*` indexes, and see how you could optimize the way those documents are indexed. Here is a sample document:

```
{  
    "@timestamp" : "2017-05-25T00:30:26.806Z",  
    "user_agent" : "Amazon CloudFront",  
    "input" : {  
        "type" : "log"  
    },  
    "ecs" : {  
        "version" : "1.0.0"  
    },  
    "level" : "info",  
    "response_size" : 54067,  
    "log" : {  
        "offset" : 40587,  
        "file" : {  
            "path" : "/home/elastic/datasets/elastic_blog_curated_acce  
        }  
    },  
    "status_code" : 200,  
    "offset" : 40587,  
    "file" : {  
        "path" : "/home/elastic/datasets/elastic_blog_curated_acce  
    }  
}
```

```
"geoip" : {  
    "city_name" : "Seoul",  
    "country_name" : "Republic of Korea",  
    "country_code2" : "KR",  
    "country_code3" : "KR",  
    "continent_code" : "AS",  
    "region_name" : "Seoul",  
    "location" : {  
        "lat" : 37.5111,  
        "lon" : 126.9743  
    }  
,  
    "http_version" : "1.1",  
    "originalUrl" : "/kr/blog/introducing-machine-learning-for-the-elastic-",  
    "method" : "GET",  
    "host" : "server1",  
    "agent" : {  
        "version" : "7.1.1",  
        "type" : "filebeat",  
        "ephemeral_id" : "8c726d99-51d6-4669-bebc-9461e100731b",  
        "hostname" : "server1",  
        "id" : "16bc0a64-d4a5-4f90-bcc8-08b76b604b55"  
    },  
    "runtime_ms" : 107,  
    "language" : {  
        "url" : "/blog/introducing-machine-learning-for-the-elastic-",  
        "code" : "ko-kr"  
    }  
}
```

There is some information in these logs that you may never use in your Kibana dashboards. In other words, you will never aggregate or query on those fields. For example, the fields inside of the `log` object. These fields contain information about the log file that this entry was ingested from:

```
"log" : {  
    "offset" : 40587,  
    "file" : {
```

```
        "path" : "/home/elasticsearch/datasets/elasticsearch_blog_curated_access  
    }
```

Similarly, the `agent` object contains information about the Filebeat instance that shipped the log file:

```
"agent" : {  
    "version" : "7.1.1",  
    "type" : "filebeat",  
    "ephemeral_id" : "8c726d99-51d6-4669-bebc-9461e100731b",  
    "hostname" : "server1",  
    "id" : "16bc0a64-d4a5-4f90-bcc8-08b76b604b55"  
}
```

7. Let's assume we are never going to query or aggregate on the `log` and `agent` objects. Let's optimize Elasticsearch such that the fields inside these objects don't take up disk space for an inverted index or doc values.

Create a new index `logs_fixed` with a mapping in which the `log` and `agent` objects have been completely disabled.

[Hide answer](#)

```
PUT logs_fixed  
{  
    "mappings": {  
        "properties": {  
            "log": {  
                "enabled": false  
            },  
            "agent": {  
                "enabled": false  
            }  
        }  
    }  
}
```

In the next lab you are going to continue to optimize the `logs_fixed` index and you will see how much disk space can be saved.

Summary: In this labs, you explored what happens when you disable the inverted index and doc values for a field.

End of Lab 4.3

Lab 4.4: Custom Mappings

Objective: In this lab you are going to further optimize the `logs_fixed` index by using a dynamic template. You are also going to see how some of the mapping parameters work.

1. **EXAM PREP:** Update the mappings for the `logs_fixed` index (that you created in the last lab) with a dynamic template:

- o that matches all unmapped fields with a value of JSON type `string`
- o and maps those as type `keyword`

[Hide answer](#)

```
PUT logs_fixed/_mapping
{
  "dynamic_templates": [
    {
      "my_string_fields": {
        "match_mapping_type": "string",
        "mapping": {
          "type": "keyword"
        }
      }
    }
  ]
}
```

2. **EXAM PREP:** With the mappings for the `logs_fixed` index in place, let's populate the index. Copy over all documents from the `logs_server1` index to the `logs_fixed` index by using the `_reindex` API:

```
POST _reindex?wait_for_completion=false
{
  "source": {
    "index": "logs_server1"
  },
  "dest": {
    "index": "logs_fixed"
  }
}
```

This will run in the background and may take a few minutes to complete.

3. Retrieve the mapping of the `logs_fixed` index. How have the string fields (like `geoip.country_name`) been dynamically mapped?

[Hide answer](#)

The output of `GET logs_fixed/_mapping` will show you that all string fields have been mapped only as type `keyword`.

4. Use `GET _cat/indices?v` to get a listing of all the indexes in your cluster. Compare the `store.size` of the `logs_fixed` and `logs_server1` indexes. Which index takes up less disk space?

[Hide answer](#)

The `logs_fixed` index takes up about 15% less disk space. By disabling fields you will not use (in the last lab), and by mapping all strings as type `keyword` you have optimized Elasticsearch for more efficient storage.

5. **EXAM PREP:** Let's review some of the mapping parameters, like `copy_to` and defining default null values. Create a new index named `surveys` that is going to hold some survey results. Create the index with only four fields in its mapping:

- o A field named `all_feedback` of type `text`

- A field named `instructor_feedback` of type `text` that gets copied to the `all_feedback` field
- A field named `labs_feedback` of type `text` that is also copied to the `all_feedback` field
- A field named `course_rating` of type `integer` in which `null` values default to 1, and also has coercion disabled

[Hide answer](#)

```
PUT surveys
{
  "mappings": {
    "properties": {
      "all_feedback": {
        "type": "text"
      },
      "instructor_feedback": {
        "type": "text",
        "copy_to": "all_feedback"
      },
      "labs_feedback": {
        "type": "text",
        "copy_to": "all_feedback"
      },
      "course_rating": {
        "type": "integer",
        "null_value": 1,
        "coerce": false
      }
    }
  }
}
```

6. Put the following document into `surveys` :

```
PUT surveys/_doc/1
{
  "instructor_feedback": "She was great!",
```

```
    "labs_feedback": "Labs were hard!"  
}
```

7. Write a query that searches the `all_feedback` field for the term `great`.
The document above should be a hit.

[Hide answer](#)

```
GET surveys/_search  
{  
  "query": {  
    "match": {  
      "all_feedback": "great"  
    }  
  }  
}
```

8. What is the value of `course_rating` for your document in `surveys`? Write a `range` query that finds all documents with a `course_rating` greater than or equal to 1.

[Hide answer](#)

The document does not have a `course_rating` field. The `null_value` parameter is only applied to fields that exist and have `null` as their value.
The following query does not match the document in `surveys`.

```
GET surveys/_search  
{  
  "query": {  
    "bool": {  
      "filter": {  
        "range": {  
          "course_rating": {  
            "gte": 1  
          }  
        }  
      }  
    }  
  }  
}
```

```
    }  
}
```

9. Review the following two **PUT** commands and predict what the result is of each one. Run each **PUT** command to see if you are correct:

```
PUT surveys/_doc/2  
{  
  "course_rating": null  
}  
  
PUT surveys/_doc/3  
{  
  "course_rating": "8"  
}
```

[Hide answer](#)

- The document with id of 2 indexes fine, and its indexed value of `course_rating` is 1. To verify, run the `range` query again from the previous step.
- The document with id of 3 does not get indexed because "8" can not be coerced into an integer, since coercion is disabled for the `course_rating` field.

10. It is always a good practice to cleanup your tests. Make sure to delete the indices that were created for practice in this lab.

```
DELETE surveys,tmp_index,my_logs,logs_fixed
```

Summary: In this lab you saw how you can optimize Elasticsearch for more efficient storage using dynamic templates. You also explored some of the mapping parameters like `copy_to` and `coerce`.

End of Lab 4.4

Lab 5.1: Master Nodes

Objective: In this lab, you will add a node to the cluster as well as become familiar with the cluster state and the discovery module.

1. Before you make changes to your cluster, let's start by understanding its current state. Use the Cluster State API to answer the following questions.

- What is the cluster name?
- How many nodes are there in the cluster?
- Which node is the elected master node?
- How many indices are there in the cluster?

You probably know the answers to these questions because your current cluster only has 1 node and uses a lot of default settings, but these are great questions to ask about any cluster.

[Hide answer](#)

```
GET _cluster/state
```

The cluster name should be `my_cluster`. You only have a single node, which is the elected master node. You should have at least 5 indices in the cluster.

2. Another way to answer some of the previous questions is to use the `_cat` API, which often returns less information per API, but it is typically easier to read. Run each of the following commands:

```
GET _cat/nodes?v
```

The `nodes` command gives you information about all the nodes in the cluster, including their roles. You can see that `node1` is a `data`, `ingest` and `master-eligible` node (`dim`) and that `node1` is the `elected-master` in the cluster. (Remember to use `?v` to see the header.)

3. Next, you will scale your cluster by adding another node. But first, you need to make a few changes to the `node1` configuration.

- The `node.name` setting is currently defined in the command line when you execute Elasticsearch. Update the configuration file to have the node name set to `node1`, so you don't need to define it in the command line.

[Hide answer](#)

Add the `node.name` setting at the end of the configuration file. You should have something like the following:

```
cluster.name: my_cluster  
node.name: node1
```

4. Currently, `node1` transport protocol is bound to `localhost` and, therefore, nodes cannot connect to it. Update the configuration file to bind both transport and HTTP protocols to a site-local address.

[Hide answer](#)

Add the `network.host` setting at the end of the configuration file. You should have something like the following:

```
cluster.name: my_cluster  
node.name: node1  
network.host: _site_
```

The `network.host` setting will update both `bind_host` and `publish_host` for both the transport and HTTP protocols. The `_site_` value binds to any site-local addresses on the system, in this case the server IP.

5. As discussed in this lesson, when you start a brand new Elasticsearch cluster for the very first time, there is a cluster bootstrapping step, which determines the set of master-eligible nodes whose votes are counted in the very first election. Even though this is a one node-only cluster, you should set the `initial_master_nodes` setting.

[Hide answer](#)

Add the `cluster.initial_master_nodes` setting at the end of the configuration file. As this is a one node cluster, you should only add `node1` to the list. You should have something like the following:

```
cluster.name: my_cluster  
node.name: node1  
network.host: _site_  
cluster.initial_master_nodes: ["node1"]
```

- As discussed in this lesson, nodes should be able to discover the cluster using the seed hosts. Even though the cluster currently has a single node, there will be more soon. Update the configuration file so that `node1` discovers the cluster via `server1` or `server2` or `server3`.

[Hide answer](#)

Add the `discovery.seed_hosts` setting at the end of the configuration file.

```
cluster.name: my_cluster  
node.name: node1  
network.host: _site_  
cluster.initial_master_nodes: ["node1"]  
discovery.seed_hosts: ["server1", "server2", "server3"]
```

- Now, stop Elasticsearch on `node1` and then start it using the following command:

```
./elasticsearch-7.3.1/bin/elasticsearch
```

- Notice that Kibana will fail to connect because now the HTTP protocol is not bound to `localhost` anymore:

```
log [21:06:43.707] [warning][admin][elasticsearch] Unable to  
log [21:06:43.708] [warning][admin][elasticsearch] No living
```

- To solve this problem, stop Kibana and start it using the following command:

```
./kibana-7.3.1-linux-x86_64/bin/kibana --elasticsearch.hosts="ht
```

10. Next, you need to setup and start node2 . First, open a new terminal tab in the Virtual Environment UI and ssh onto server2 :

```
ssh server2
```

11. Then, Extract Elasticsearch on server2 :

```
tar -zxf elasticsearch-7.3.1-linux-x86_64.tar.gz
```

12. Now, configure the node with the following basic characteristics.

- it joins my_cluster
- the name of the node is node2
- binds and publishes both the transport and HTTP protocols to the site-local address

[Hide answer](#)

Add the following settings at the end of the `elasticsearch.yml` configuration file.

```
cluster.name: my_cluster
node.name: node2
network.host: _site_
```

13. Next, update the configuration file with the following discovery and cluster formation settings.

- node1 is the only initial master node in the cluster
- it discovers the cluster via server1 , server2 , or server3

[Hide answer](#)

The `cluster.initial_master_nodes` setting is very important to avoid node2 to create its own cluster if it cannot connect to node1 for any reason (e.g. typo in the cluster name). The `discovery.seed_hosts` setting allows node2 to connect to node1 and node3 (after it is added). The `elasticsearch.yml` should look like the following:

```
cluster.name: my_cluster  
node.name: node2  
network.host: _site_  
cluster.initial_master_nodes: ["node1"]  
discovery.seed_hosts: ["server1", "server2", "server3"]
```

14. Next, configure `node2` to use only 512 megabytes of memory. (*TIP:* make sure to set both the min and max heap size to **512m**.)

[Hide answer](#)

In the `jvm.options` configuration file you will see a `-Xms` and a `-Xmx` setting configured to `1gb`. Update both values to `512m`.

```
-Xms512m  
-Xmx512m
```

15. Finally, start `node2`. You should see the following output:

```
[2018-11-23T19:03:54,245][INFO ][o.e.c.s.ClusterApplierService]  
[2018-11-23T19:03:54,981][INFO ][o.e.x.s.a.TokenService] [node2]  
[2018-11-23T19:03:55,345][INFO ][o.e.x.s.a.TokenService] [node2]  
[2018-11-23T19:03:55,386][INFO ][o.e.l.LicenseService] [node2]  
[2018-11-23T19:03:55,442][INFO ][o.e.x.s.t.n.SecurityNetty4HttpS  
[2018-11-23T19:03:55,442][INFO ][o.e.n.Node] [node2]
```

[Hide answer](#)

Start Elasticsearch using the following command:

```
./elasticsearch-7.3.1/bin/elasticsearch
```

16. Use `_cat/nodes` to view the nodes in the cluster. You should see two nodes now:

ip	heap.percent	ram.percent	cpu	load_1m	load_5m	load_15m
172.18.0.2	31	98	11	0.01	0.15	0.08
172.18.0.3	37	98	11	0.01	0.15	0.08

[Hide answer](#)

```
GET _cat/nodes?v&s=name
```

17. **OPTIONAL:** Now that you started your 2-node cluster, use the following command to check the voting configuration:

```
GET /_cluster/state?filter_path=metadata.cluster_coordinat...  
...
```

You should get an output similar to the one below, which shows the id of a single master-eligible node that is allowed to vote within your cluster.

```
{  
  "metadata" : {  
    "cluster_coordination" : {  
      "last_committed_config" : [  
        "0vD79L1lQme1hi060uiu7Q",  
      ]  
    }  
  }  
}
```

Why the cluster coordination has a single master-eligible node, if there are two master-eligible nodes in the cluster?

[Hide answer](#)

If there is an even number of master-eligible nodes, Elasticsearch leaves one of them out of the voting configuration to ensure that it has an odd size. This omission does not decrease the failure-tolerance of the cluster. In fact, improves it slightly: if the cluster suffers from a network partition that divides it into two equally-sized halves then one of the halves will contain a majority of the voting configuration and will be able to keep operating.

Summary: In this lab, you started a 2-node cluster as well as became familiar with the cluster state and the discovery module. As discussed in this lesson, having a two node cluster is not a best practice and, in the next lab, you will see a better solution.

End of Lab 5.1

Lab 5.2: Node Roles

Objective: In this lab, you will familiarize yourself with node roles.

1. At the end of the previous lab, we discussed that having an even number of master-eligible node is not a good practice. In cases like this, adding a third node is the preferred solution. However, you might not have the resources to add a third data and master-eligible node to your cluster. Then, the recommended solution is to add a dedicated master-eligible node, which is going to work as a **tiebreaker** in your 3-node cluster. Next, you will configure a dedicated master-eligible node on `server3`. First, open another terminal window in the Virtual Environment UI and `ssh` onto `server3`:

```
ssh server3
```

2. Extract Elasticsearch on `server3`:

```
tar -zxf elasticsearch-7.3.1-linux-x86_64.tar.gz
```

3. **EXAM PREP:** Then, configure the node settings in the appropriate config files (either `elasticsearch.yml` or `jvm.options`) to have the following:

- it joins `my_cluster`
- the name of the node is `node3`
- binds and publishes both the transport and HTTP protocols to the site-local address
- set the min and max heap size to **512m**

[Hide answer](#)

The `elasticsearch.yml` configuration file should look like:

```
cluster.name: my_cluster
node.name: node3
network.host: _site_
```

The `jvm.options` configuration file should contain:

```
-Xms512m  
-Xmx512m
```

4. **EXAM PREP:** Next, update the node settings to have the following:

- o `node1` is the only initial master node in the cluster
- o it discovers the cluster via `server1`, `server2`, or `server3`

[Hide answer](#)

The `elasticsearch.yml` configuration file should look like:

```
cluster.name: my_cluster  
node.name: node3  
network.host: _site_  
cluster.initial_master_nodes: ["node1"]  
discovery.seed_hosts: ["server1", "server2", "server3"]
```

5. Update the node settings to add the following settings:

```
node.master: false  
node.voting_only: true
```

6. Start Elasticsearch on `server3`. What is the issue with this configuration?

[Hide answer](#)

You should have the following error: voting-only node must be master-eligible

7. A voting-only node must be master-eligible even if it cannot endorse the master role. Stop elasticsearch on `server3`.

8. **EXAM PREP:** Finally, update the node settings to implement the following:

- o it is a dedicated voting-only master-eligible node.

[Hide answer](#)

The `elasticsearch.yml` configuration file should look like:

```
cluster.name: my_cluster
node.name: node3
network.host: _site_
cluster.initial_master_nodes: ["node1"]
discovery.seed_hosts: ["server1", "server2", "server3"]
node.data: false
node.ingest: false
node.ml: false
node.master: true
node.voting_only: true
```

9. Start Elasticsearch on `server3`. You should see the following output:

```
[2018-11-23T19:13:18,009][INFO ][o.e.c.s.ClusterApplierService]
[2018-11-23T19:13:18,420][INFO ][o.e.x.s.a.TokenService      ] [nod
[2018-11-23T19:13:18,774][INFO ][o.e.x.s.a.TokenService      ] [nod
[2018-11-23T19:13:18,934][INFO ][o.e.l.LicenseService      ] [nod
[2018-11-23T19:13:19,018][INFO ][o.e.x.s.t.n.SecurityNetty4HttpS
[2018-11-23T19:13:19,019][INFO ][o.e.n.Node                  ] [nod
```

[Hide answer](#)

```
./elasticsearch-7.3.1/bin/elasticsearch
```

10. Verify your nodes are configured the way you want:

```
GET _cat/nodes?s=name
```

You should see an output similar to the one as follows:

```
172.18.0.2 10 98 18 0.74 0.93 0.64 dim * node1
172.18.0.3 8 98 18 0.74 0.93 0.64 dim - node2
172.18.0.4 17 98 48 0.74 0.93 0.64 mv - node3
```

The role of node3 should be only "mv", while the role of both node1 and node2 should be "dim".

- Now that you have a 3-node cluster, use the following command to check the voting configuration:

```
GET /_cluster/state?filter_path=metadata.cluster_coordinat...[REDACTED]
```

You should get an output similar to the one below, which shows the id of three master-eligible nodes that are allowed to vote within your cluster.

```
{
  "metadata" : {
    "cluster_coordinat...[REDACTED]
```

- Check that the voting configuration is working by stopping the master node (probably node1) and check the terminal output in the other nodes. Among the logged lines you will see something similar to the following:

```
[2018-11-23T19:15:47,301][INFO ][o.e.d.z.ZenDiscovery      ] [node1] ...[REDACTED]
[2018-11-23T19:15:47,303][WARN ][o.e.d.z.ZenDiscovery      ] [node1] ...[REDACTED]
{node2}{7IrQlPdBRJaA120lrbdpZQ}{0mVQTduVQJuaQavC6PAHlA}{172.17.1.1}
{node1}{Xe6KFUYCTA6AWRpbw84qaQ}{CKxs8gcTRoiYEiutL4BERw}{172.17.1.1}
{node3}{tWpzRPKfTrujh02kPpMHia}{W4P1zuy_TauSOGGCGMq8Bg}{172.17.1.1}

[2018-11-23T19:15:50,432][INFO ][o.e.c.s.MasterService      ] [node2] ...[REDACTED]
[2018-11-23T19:15:50,461][WARN ][o.e.d.z.PublishClusterStateActi...[REDACTED]
[2018-11-23T19:15:50,465][INFO ][o.e.c.s.ClusterApplierService] ...[REDACTED]
```

Notice the other nodes realized that the elected master left the cluster, and then elected a new master.

13. Next, imagine `node1` is not as powerful as `node2` and you don't want it to run ingest pipelines or machine learning jobs. Update its configuration file to make it a data and master-eligible only node. (**NOTE:** don't disable all ingest nodes in your cluster. Ingest nodes are often used in the Elastic Stack ecosystem—e.g. beats and monitoring.)

[Hide answer](#)

Add the following to the `node1` configuration file:

```
node.ingest: false  
node.ml: false
```

14. Finally, restart `node1` and verify your nodes are configured the way you want:

```
GET _cat/nodes?s=name
```

You should see an output similar to the one as follows:

```
172.18.0.4 11 98 3 0.67 0.22 0.26 mv - node3  
172.18.0.3 19 98 3 0.67 0.22 0.26 dim * node2  
172.18.0.2 13 98 30 0.67 0.22 0.26 dm - node1
```

Your master node might be different, but the role of `node2` should be only "**dim**", while the role of `node1` should be "**dm**" and `node3` should still be "**mv**".

Summary: In this lab, you familiarized yourself with node roles by updating your current nodes to become dedicated nodes.

End of Lab 5.2

Lab 5.3: Understanding Shards

Objective: In this lab, you will become familiar with shard distribution. You will create new indices with a specific number of shards and replicas and analyze shard allocation along with cluster status.

1. First, use the `_cat` API to see information about the indices in the cluster.

```
GET _cat/indices?v
```

The `indices` command gives you information about the indices in the cluster. You can see the number of primary shards, replica shards, documents and deleted documents. You can also see the size on disk for primary shards and in total (primary shards + replica shards). Finally, notice that every index has a name and a uuid.

2. Then, use the `_cat` API to see information about the shards in the cluster. How many shards does the index `logs_server2` have? And in which node are they allocated?

[Hide answer](#)

```
GET _cat/shards?v
```

The `shards` command gives you information about the shards in the cluster. You can see information about primaries and replicas, the number of documents, size on disk, and which node a shard has been allocated. If you are interested in a specific index, add the index name after `/shards`, like the following:

```
GET _cat/shards/logs_server2?v
```

The `logs_server2` index has 1 primary shard and 1 replica shard configured. The allocation of primaries and replicas will be distributed across `node1` and `node2`, as `node3` is a dedicated master-eligible node.

3. Now, use `_cluster/health` to check the status of your cluster, which should be green.

[Hide answer](#)

```
GET _cluster/health
```

4. To better understand shard allocation, create an index named `test1` with 4 primary shards and 2 replicas.

[Hide answer](#)

```
PUT test1
{
  "settings": {
    "number_of_shards": 4,
    "number_of_replicas": 2
  }
}
```

5. Use the following to check the shard allocation of your `test1` index. You should see 4 **STARTED** primary shards, 4 **STARTED** replica shards and 4 **UNASSIGNED** replica shards.

```
GET _cat/shards/test1?v
```

For better readability, you can use the following query to see the results sorted:

```
GET _cat/shards/test1?v&s=shard,prirep
```

6. What is the status of your cluster? Why?

[Hide answer](#)

```
GET _cluster/health
```

It should be yellow because there are unassigned replicas. Remember that Elasticsearch never allocates more than 1 copy of the same shard to the same node.

7. Update the `test1` index to have 1 replicas and check the new shard distribution.

[Hide answer](#)

```
PUT test1/_settings
{
  "settings": {
    "number_of_replicas": 1
  }
}
```

```
GET _cat/shards/test1?v&s=node,shard
```

8. What is the status of your cluster?

[Hide answer](#)

```
GET _cluster/health
```

It should be back to green.

9. It is a good practice to run tests on a new index and, then, cleanup the cluster as you finish the test. Therefore, delete the `test1` index, which was created in this lab.

```
DELETE test1
```

Summary: In this lab, you became familiar with shard distribution. You created new indices with a specific number of shards and replicas and analyzed shard allocation along with cluster status.

End of Lab 5.3

Lab 5.4: Distributed Operations

Objective: In this lab, you will use the `refresh_interval` setting and the `refresh` parameter to better understand how Elasticsearch refreshes.

1. The `refresh_interval` setting defines the maximum time that a document can be unavailable for search after an index operation. Create a new index named `my_refresh_test` with a refresh interval of 1 hour. (The refresh interval is very high, but we will use it for test purposes.)

[Hide answer](#)

```
PUT my_refresh_test
{
  "settings": {
    "index.refresh_interval": "1h"
  }
}
```

2. Execute the following command to index three documents:

```
PUT my_refresh_test/_bulk
{ "index" : { "_id" : "1"}}
{ "level" : "test"}
{ "index" : { "_id" : "2"}}
{ "level" : "test"}
{ "index" : { "_id" : "3"}}
{ "level" : "test"}
```

3. Execute a search request on `my_refresh_test`. How many documents are returned? Why?

[Hide answer](#)

```
GET my_refresh_test/_search
```

The request above should return 0 documents because the `refresh_interval` is very high (1 hour). It will take a long time until the documents are available for search.

4. Send a GET request to retrieve document 1. Do you think the document will be returned? Why or why not?

[Hide answer](#)

```
GET my_refresh_test/_doc/1
```

Document 1 is returned. By default, the GET API is realtime, and is not affected by the refresh rate of the index (when data will become visible for search).

5. There is no point in waiting 1 hour for a refresh to happen. Force a refresh and try running the search again. Will you get any hits this time?

[Hide answer](#)

```
POST my_refresh_test/_refresh
```

Yes, all three documents are hits. Calling `_refresh` forces a shard refresh so that the latest operations are visible immediately.

6. Sometimes in tests (or even in production) you want to write a document and make sure it is available as soon as possible. Regardless of the current `refresh_interval`, you can use the `refresh` parameter in an index request to control it. Index a new document that will be available for search right away, even though the refresh interval is set to 1 hour. The search result should contain 4 documents.

[Hide answer](#)

```
PUT my_refresh_test/_doc/4?refresh=true
{
    "level" : "test"
}
```

```
GET my_refresh_test/_search
```

7. Using `refresh=true` will likely degrade Elasticsearch performance. To make sure that documents are available as soon as the client receives the response without impacting Elasticsearch performance, use the `refresh=wait_for` option. To test it, first change the `refresh_interval` of the `my_refresh_test` index to ten seconds.

[Hide answer](#)

```
PUT my_refresh_test/_settings
{
  "index.refresh_interval": "10s"
}
```

8. Next, index a document using the `wait_for` value. What happens to the request? Why?

[Hide answer](#)

The request hangs for a few seconds before returning the response. This happens because the `wait_for` option only returns the request response after the operation is visible.

```
PUT my_refresh_test/_doc/5?refresh=wait_for
{
  "level" : "test"
}
```

1. It is a good practice to run tests on a new index and, then, cleanup the cluster as you finish the test. Therefore, delete the `my_refresh_test` index, which was created in this lab.

```
DELETE my_refresh_test
```

Summary: In this lab, you have used the `refresh_interval` setting and the `refresh` parameter to better understand how Elasticsearch refreshes.

End of Lab 5.4

Lab 6.1: HTTP Response and Shard Allocation Issues

Objective: In this lab you will become familiar with some of the errors that Elasticsearch returns, and how to understand and fix some of them. Then, you will execute search requests that return partial results and hunt down the cause.

1. Let's start with a very common error. Run the following command in Console. What is the error returned? And how can you fix it?

```
PUT test1/_doc/  
{  
    "test": "test"  
}
```

[Hide answer](#)

The error returned is **405** ("Method not allowed"). The error message from Elasticsearch is saying that POST is allowed, but not PUT, which is correct as our document does not have an id. To fix this problem you can either give an id to the document or use a POST instead of a PUT.

2. Run the following command. What is the error returned? And how can you fix it?

```
GET test2/_doc/1
```

[Hide answer](#)

The error returned is **404** ("Not found"). The error message from Elasticsearch is clear and states that the index does not exist. You probably want to make sure you have the correct index name. If you do, you want to investigate why the index does not exist or was deleted.

3. Run the following command. What is the error returned? And how can you fix it?

```
GET blogs/_search  
{  
    "query": {  
        "match": {  
            "title": "open source software",  
            "minimum_should_match": 2
```

```
    }
}
}
```

[Hide answer](#)

The error returned is a **400** ("Bad Request"). There was a parsing exception as the `match` query does not support multiple fields. In order to use the `minimum_should_match` parameter you need to use the extended form of the `match` query, like the following:

```
GET blogs/_search
{
  "query": {
    "match": {
      "title": {
        "query": "open source software",
        "minimum_should_match": 2
      }
    }
  }
}
```

4. Run the following command. What is the error returned? And how can you fix it?

```
GET blogs/_search
{"query": {"match": {"title": {"query": "open source software," "minim}}
```

[Hide answer](#)

The error returned is **500** ("Internal Server Error"). The error message from Elasticsearch is clear and states that there was a JSON parse exception. The JSON is indeed not valid, which happens often when you have it in a single line, or you have a string in your application with the JSON that should be executed. To fix the above query, add a double quote after **software**.

5. Run the following commands that index two documents and run a query. What is the error returned from the query? And how can you fix it? **NOTE:**

This step and some of the following steps might have slightly different results when you execute them, because Elasticsearch does not always allocate the same shards to the same nodes.

```
PUT test2/_doc/1
{
    "date": "2017-09-10"
}

PUT test3/_doc/1
{
    "date": "September 10, 2017"
}

GET test*/_search
{
    "query": {
        "match": {
            "date": "September"
        }
    }
}
```

[Hide answer](#)

No error code is returned in this example, the status returned is **200** ("OK"). However, the search results state that out of 2 shards, only 1 was successful and that 1 shard failed. WHAT? One shard is missing?! We will look into that in a bit. For now, by looking into the failures, you can see that there was an exception in which **September** could not be parsed as a date. The example is very simple, but this happens often when your search request contains a wildcard in the index name and you have fields with the same name but different types. If the `date` field in the `test2` index should be a "**string**", make sure to define the mapping before indexing the first document (or correct the mapping and reindex the data). If mappings are correct, feel free to ignore the failures and work with the partial results.

6. Now, you are going to simulate the failure of primary shards to see how it looks like. So, first create an index named `test4` with 4 primaries and 0 replicas.

[Hide answer](#)

```
PUT test4
{
  "settings": {
    "number_of_shards": 4,
    "number_of_replicas": 0
  }
}
```

7. Index the following documents inside test4 :

```
PUT test4/_doc/1
{
  "test": "document 1"
}

PUT test4/_doc/2
{
  "test": "document 2"
}

PUT test4/_doc/3
{
  "test": "document 3"
}

PUT test4/_doc/4
{
  "test": "document 4"
}
```

8. Stop node2 to simulate a node failure and see what's going to happen.

9. Run the following query, check the number of hits, and also check the _shards section. How many hits did you get? How many shards successfully executed your query?

```
GET test4/_search
{
  "query": {
    "match": {
      "test": "document"
    }
  }
}
```

[Hide answer](#)

You probably got less than 4 hits. If you check the `_shards` section you will note that despite the fact you have 4 primary shards for `test4` your query didn't reach all of them, that is, the number of successful shards is probably lower than 4, which is the total number of shards.

10. Investigate what is wrong. First, check how the cluster is doing by viewing the cluster health. Is everything ok? What is the problem?

[Hide answer](#)

```
GET _cluster/health
```

The cluster has red status, which means that at least one primary shard is not available. Looking at the Cluster Health API response we can see that there are unassigned shards.

11. **EXAM PREP:** Keep digging it. Figure out which index is red and which shard is unassigned. Feel free to use the `_cluster` or `_cat` APIs.

[Hide answer](#)

Let's figure out which index has the problem. Using the Cluster API, the command would be:

```
GET _cluster/health?level=indices
```

Using the `_cat` API:

```
GET _cat/indices?v
```

As expected, the `test4` index is red. Run the following to check which shard has the issue:

```
GET _cluster/health/test4?level=shards
```

Using the `_cat` API, the similar command would be:

```
GET _cat/shards/test4?v
```

12. **EXAM PREP:** Now that we know which shard has the issue, use the Cluster Allocation Explain API to understand it. Can you discover what the problem is?

[Hide answer](#)

You can run the short form of the allocation API because there is only one unassigned shard in the cluster. If you have more unassigned shards in the cluster, you would need to run the long form.

```
GET _cluster/allocation/explain
```

```
GET _cluster/allocation/explain
{
  "index": "test4",
  "shard": 0,
  "primary": true
}
```

Change the value of "shard" in the request above into the shard that you identified earlier as being unassigned.

13. Analyzing the results, you can see that there are some primary shards that couldn't be allocated because the node that used to hold them left the cluster, which was the expected outcome of our simulation. Now, assume that you fixed the issue of `node2` and restart it.

[Hide answer](#)

```
./elasticsearch-7.3.1/bin/elasticsearch
```

14. Usually it is a good practice to have at least one replica shards to avoid your cluster going red when you lose one node that holds primary shards. Update the settings of `test4` index so it has 1 replica shards.

[Hide answer](#)

```
PUT test4/_settings
{
  "number_of_replicas": 1
}
```

15. Simulate a failure on `node2` again by stopping it and also rerun the following query, check the number of hits, and check the `_shards` section. How many hits did you get? How many shards successfully executed your query?

```
GET test4/_search
{
  "query": {
    "match": {
      "test": "document"
    }
  }
}
```

[Hide answer](#)

Now you should have gotten the 4 hits and all shards successfully executed your query. Even though you lost `node2`, `node1` promoted its replica shards to primary shards so your cluster can work as usual. However, note that probably your cluster is yellow due to some missing replicas.

16. Enough of failure simulation on your cluster. Restart `node2`.

[Hide answer](#)

```
./elasticsearch-7.3.1/bin/elasticsearch
```

17. Now that you have finished your tests and you will not use the created indices anymore, make sure to cleanup your cluster by deleting `test*`

indices.

[Hide answer](#)

You can either delete one index at a time or use the wildcard.

```
DELETE test*
```

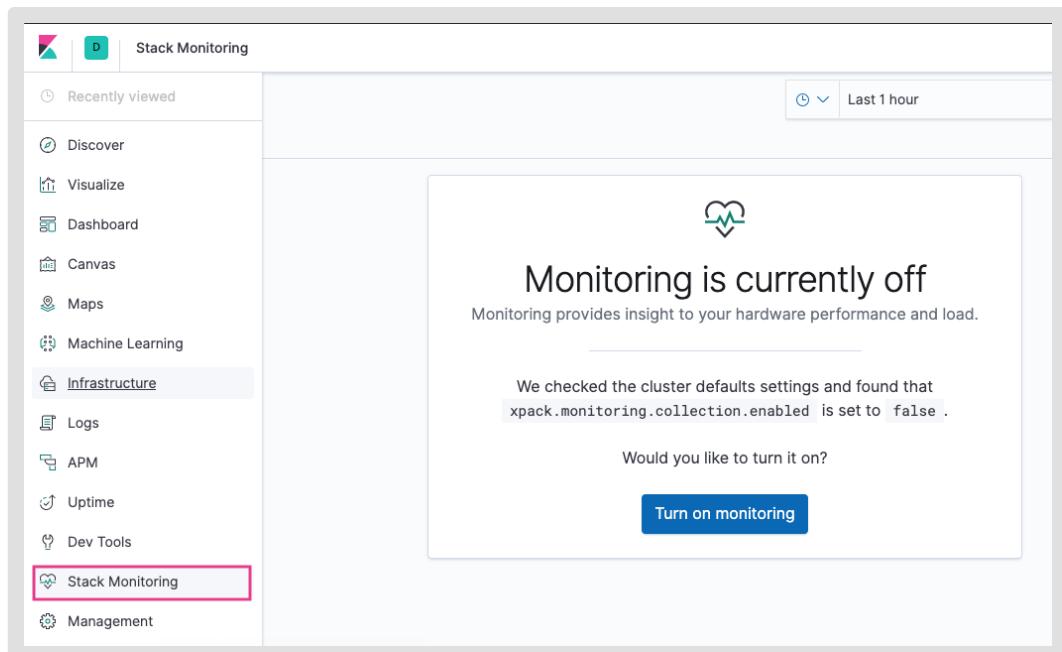
Summary: In this lab you became familiar with some of the errors that Elasticsearch returns, and how to understand and fix some of them. Then you executed search requests that returned partial results and hunted down and fixed the cause.

End of Lab 6.1

Lab 6.2: Monitoring

Objective: In this lab, you will use the Monitoring UI to monitor the health of your cluster.

1. Even though we have monitoring enabled, data collection is disabled by default. To enable it, click on the Monitoring icon in the left toolbar.



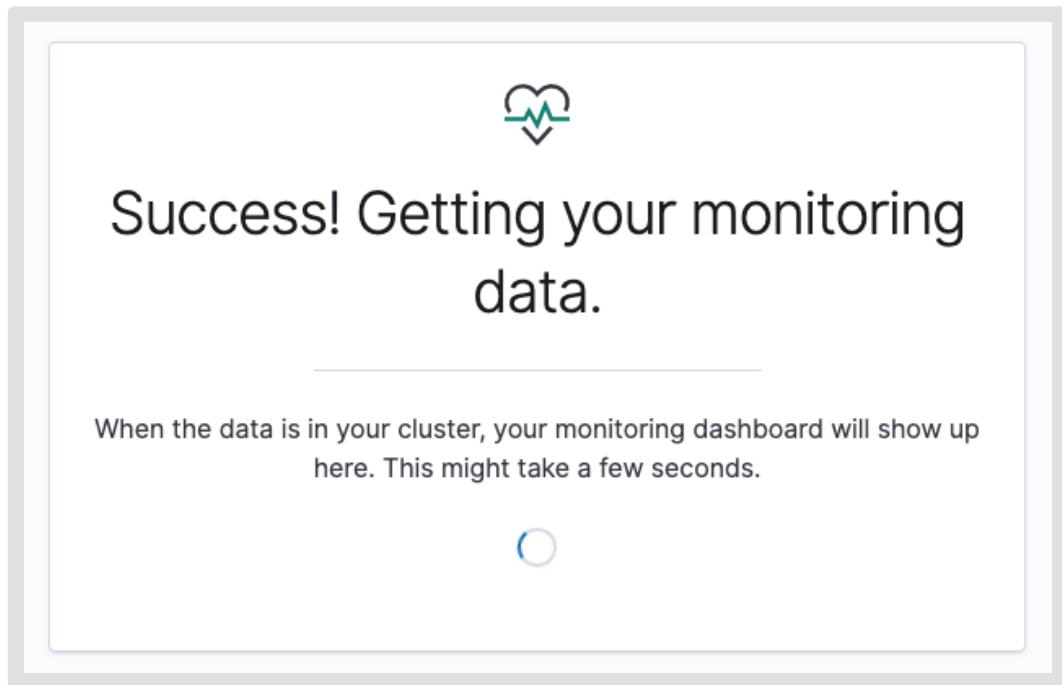
2. Then, click the button **Turn on monitoring** to enable data collection.

The screenshot shows the Elasticsearch Stack Monitoring interface. On the left, a sidebar lists various monitoring tools: Discover, Visualize, Dashboard, Canvas, Maps, Machine Learning, Infrastructure (which is selected and highlighted in grey), Logs, APM, Uptime, Dev Tools, Stack Monitoring, and Management. The main content area has a light blue background. At the top right, there's a time range selector showing 'Last 1 hour'. Below it, a large central box contains a green heart icon with a pulse line. The text reads: 'Monitoring is currently off' followed by 'Monitoring provides insight to your hardware performance and load.' A note below states: 'We checked the cluster defaults settings and found that xpack.monitoring.collection.enabled is set to false.' A question 'Would you like to turn it on?' is followed by a blue rectangular button with white text that says 'Turn on monitoring'. This button is specifically highlighted with a thick pink border.

Note that we could have also used the **Cluster Update Settings API** to enable data collection:

```
PUT /_cluster/settings
{
  "persistent" : {
    "xpack.monitoring.collection.enabled" : true
  }
}
```

3. After enabling data collection it will take a few seconds to start monitoring and collect some data.



4. Finally, you will be able to view your **Clusters** dashboard.

A screenshot of the Elasticsearch Clusters dashboard. The left sidebar has icons for Scripts, Databases, and Clusters, with 'Clusters' selected. The main area shows a cluster named 'my_cluster'. It includes sections for 'Elasticsearch' (Health is green, Basic license) and 'Kibana' (Health is green). Each section contains an 'Overview' card and a detailed metrics card. For Elasticsearch, the overview shows Version 7.1.1 and Uptime 20 minutes. The metrics card shows Disk Available at 81.46% (142.0 GB / 174.3 GB) and JVM Heap at 29.79% (589.8 MB / 1.9 GB). For Kibana, the overview shows Requests 12 and Max. Response Time 75 ms. The metrics card shows Connections 0 and Memory Usage at 18.53% (269.9 MB / 1.4 GB).

5. Change the time interval from "Last 1 hour" to "Last 15 minutes" to view only the data from the last 15 minutes. Since your cluster is just starting to collect metrics, this view will actually provide more details (until more data is collected over a longer interval).

The screenshot shows the Elasticsearch Monitoring interface. On the left, there's a sidebar with various icons. In the center, under the 'Clusters' tab, 'my_cluster' is selected. Below it, the 'Elasticsearch' section displays an 'Overview' with 'Version' 7.1.1 and 'Uptime' 27 minutes. To the right, a detailed view of the refresh settings is shown. The top bar has a date range selector set to 'Last 1 hour'. Below it is a 'Quick select' dropdown with a pink border, showing 'Last 15 minutes' selected. Other options include 'Today', 'This month', 'Today so far', 'Month to date', 'This week', 'This year', 'Week to date', and 'Year to date'. A 'Recently used date ranges' section shows 'Last 1 hour'. At the bottom, a 'Refresh every' section shows '10 seconds' with an 'Apply' button.

6. Update the refresh interval to 5 seconds (instead of the current 10 second interval). You should see events come in twice as often now.

This screenshot is similar to the previous one but shows the refresh interval updated. The 'Quick select' dropdown still has 'Last 15 minutes' selected. However, in the 'Refresh every' section, the input field now contains '5' followed by a dropdown menu set to 'seconds'. The 'Apply' button is also highlighted with a pink box.

7. From the Monitoring home page, click on **Nodes**, then `node2`. Scroll down to the bottom of the page and view the **Shard Legend** section. You should see all of your indices, each with a mix of primary and replica shards, similar to:

The screenshot shows the Elasticsearch Shard Legend interface. At the top, there are six color-coded buttons: Primary (blue), Replica (teal), Relocating (pink), Initializing (light blue), Unassigned Primary (red), and Unassigned Replica (orange). Below these buttons, the word "Indices" is followed by a circular switch icon and the text "System indices". Underneath, four index names are listed: "blogs" (blue bar), "logs_server1" (teal bar), "logs_server2" (light blue bar), and "logs_server3" (orange bar). Each bar has a numerical value "0" inside it.

8. Next, you will simulate a node failure in the cluster. Stop node3 .
9. Go back to the **Nodes** page of Monitoring. Notice that the node3 is now offline and all stats show up as N/A.

The screenshot shows the Elasticsearch Nodes page. At the top, there are tabs for Overview, Nodes (which is selected), Indices, and CCR. Below the tabs, there's a summary table with columns: Status (Green), Nodes (2), Indices (8), Memory (366.2 MB / 1.5 GB), Total Shards (16), Unassigned Shards (0), Documents (1,757,082), and Data (1.4 GB). A search bar labeled "Filter Nodes..." is below the summary table. The main table lists three nodes:

Name	Status	CPU Usage	Load Average	JVM Memory	Disk Free Space	Shards
★ node1 172.18.0.2:9300	Online	5% ↓ 39% max 0% min	0.27 ↓ 5.03 max 0 min	25% ↑ 46% max 15% min	42.7 GB ↓ 47.3 GB max 42.7 GB min	8
■ node2 172.18.0.3:9300	Online	1% ↓ 34% max 0% min	0.27 ↓ 5.03 max 0 min	34% ↓ 69% max 23% min	42.7 GB ↓ 47.3 GB max 42.7 GB min	8
■ node3 172.18.0.4:9300	Offline	N/A	N/A	N/A	N/A	N/A

At the bottom left, it says "Rows per page: 20".

10. Startup the unavailable node.
11. Next, you will put some stress on your cluster by reindexing some data. Open Kibana Dev Tools on a new tab and execute the command below to reindex all logs data (1.7M documents) into a single index named logs_test2 .

```
POST _reindex
{
  "source": {
    "index": "logs_server*"
  },
  "dest": {
    "index": "logs_test2"
```

```
}
```

12. What happened to the cluster overview?

[Hide answer](#)

Both search and indexing rates probably increased during the reindex process.



13. Feel free to explore the Monitoring UI and try to get a sense of the information being provided as well as how to navigate the UI. Before finishing this lab, remember that in production you probably want to have a dedicated monitoring cluster, though during this lab you enabled monitoring in the same cluster for simplicity.

Summary: In this lab, you enabled data collection to monitor the details of your cluster using the Monitoring UI.

End of Lab 6.2

Lab 6.3: Diagnosing Performance Issues

Objective: In this lab, you will explore the cat and task management APIs. You will also learn how to check for slow operations and profiling queries.

1. JSON is great for computers, but it usually is hard for human eyes. The cat API aims to meet this need, so you can use them to check several information about your cluster on a more user friendly way. Use the _cat command alone to list all the available commands:

```
GET _cat
```

2. The shards command is one of the available commands within the cat APIs and it gives you a detailed view of each shard in the cluster. It will tell you whether it is a primary or replica, the number of the docs, the bytes it takes on disk, and the node where it is located. Run _cat/shards to check all this shard information about your cluster.

```
GET _cat/shards
```

3. If you are not familiar with cat APIs it might be difficult to grasp what kind information is being returned. To overcome this you can turn on verbose output and it will show you a header along with the results, which makes it easier to identify what all the information returned is about. Run _cat/shards again with the query string parameter v to turn on verbose output as follows:

```
GET _cat/shards?v
```

4. The cat APIs also allows you to define what columns you want to see in the output. You can use the query string parameter h to do that and if you set it to * you will see all the available information. Run _cat/shards again with verbose output on and also set it to show all available columns as follows:

```
GET _cat/shards?v&h=*
```

5. You probably noticed in the last exercise that showing all the columns might be too much information. If you are not sure what information you want to see, but still don't want to see all of them, you can use the help parameter to check what are the available columns. Use the help parameter on _cat/shards to see what are all the available columns.

```
GET _cat/shards?help
```

6. Suppose you want to see the columns `index`, `shard`, `prirep`, `node`, `search.query_time`, and `search.query_total`. You can run `_cat/shards` again specifying these columns through the `h` parameter as follows.

```
GET _cat/shards?v&h=index,shard,prirep,node,search.query_time,se
```

7. You might also want to sort the output of the cat APIs to make them even easier to read. You can do that through the query string parameter `s` which sorts the table by the columns specified as the parameter value. Run `_cat/shards` again as follows and you will see the output sorted out by `index` and `search.query_time`.

```
GET _cat/shards?v&h=index,shard,prirep,node,search.query_time,se
```

8. Now, you are going to reindex some documents to put some stress on your cluster and use the task management API to check what is going on. So, start by running the following reindex request.

```
POST _reindex?wait_for_completion=false
{
  "source": {
    "index": "logs_server*"
  },
  "dest": {
    "index": "logs_test4"
  }
}
```

9. Even though you can use `_cat/tasks` to check the information about the current tasks in your cluster, it still does not provide all the same tools that are provided by the task management API itself. For this reason, you are

going to rely on the JSON back again. Use the `_tasks` command to retrieve all tasks currently running on all nodes in your cluster.

```
GET _tasks
```

10. You probably noticed that `_tasks` might return too much information. Since you know that a `reindex` is going on in your cluster, you can use the `actions` parameter to filter out the results to show only tasks related to reindex processes. Run the command below to check your reindex.

```
GET _tasks?actions=*reindex
```

11. You can also use the `detailed` request parameter to get more information about the running tasks. This is useful when you need to see what is the difference among the tasks running in your cluster, though it is more costly to execute. Run the command below and notice that the `detailed` parameter will show you a `description` about your `reindex` request, which makes a lot easier to understand what is going on.

```
GET _tasks?actions=*reindex&detailed
```

12. You can also use the task API to check information about a specific task. Use the output of the previous exercise to see what is the id of your reindex task and use it in place of `<id>` in the command below.

```
GET _tasks/<id>
```

Show answer

13. You can use the task management API to cancel long-running tasks. Run the command below to cancel your `reindex`. Remember to change `<id>` by the task id that you used in the previous exercise. (*This command will fail if the task has finished.*)

```
POST _tasks/<id>/_cancel
```

14. Now, you are going to update the shard level slow search log and run a slow aggregation query to see how they appear in the logs. Run the command below to update the slow log level to log any queries that take more than 500ms to run.

```
PUT logs_server*/_settings
{
  "index.search.slowlog" : {
    "threshold.query" : {
      "warn" : "1s",
      "info" : "500ms",
      "debug" : "0ms"
    },
    "level" : "info"
  }
}
```

15. Run the aggregation below. After this aggregation finishes running you can check that it was logged in `elasticsearch-7.3.1/logs/my_cluster_index_search_slowlog.log`.

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "top_10_countries": {
      "terms": {
        "field": "geoip.country_name.keyword",
        "size": 10
      },
      "aggs": {
        "top_5_blogs": {
          "significant_text": {
            "field": "originalUrl.keyword",
            "size": 5
          }
        }
      }
    }
}
```

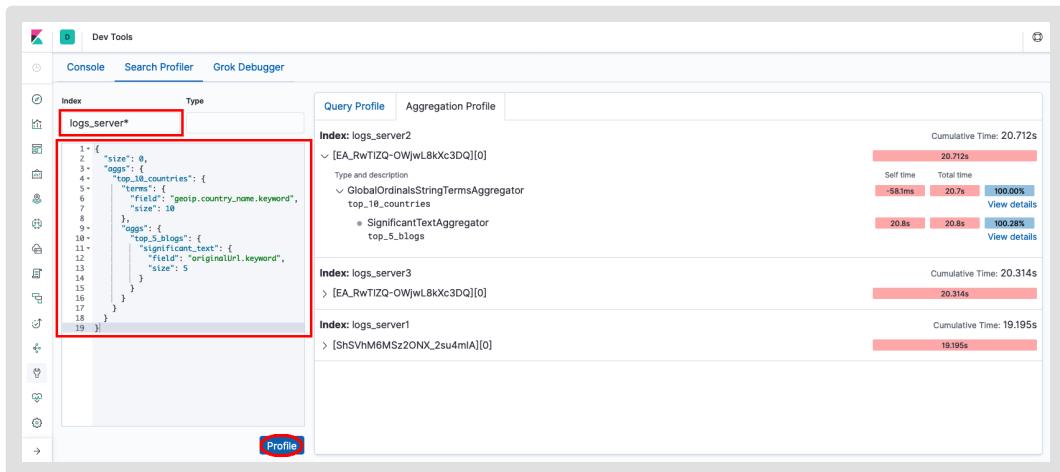
```

        }
    }
}

```

16. When you find a query in your slow logs, a good idea is to use the search profiler in Kibana to check why that query is slow. Use the search profiler in Kibana to check why the query from the previous exercise is slow.

[Hide answer](#)



17. Now that you have finished your tests and you will not use the created indices anymore, make sure to cleanup your cluster by deleting `log_test*` indices.

[Hide answer](#)

You can either delete one index at a time or use the wildcard.

```
DELETE logs_test*
```

Summary: In this lab, you explored the cat and task management APIs. You also learned how to check for slow operations and profiling queries.

End of Lab 6.3

strictly prohibited.