Lab Guide: Elasticsearch Engineer II

Lab Setup: Welcome to Engineer II

Objective: In this lab, you will access your classroom lab instance and startup a 3-node Elasticsearch cluster, along with a Kibana instance.

1. From within the Strigo 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 named `server1`, `server2` and `server3` that have Elasticsearch downloaded and ready to be executed.

2. SSH onto `server1`:

```
ssh server1
```

3. View the contents of the `elastic` user home folder by entering `ls -l`. You should see Elasticsearch and Kibana directories:

```
[elastic@server1 ~]$ ls -l
total 8
drwxr-xr-x  9 elastic elastic 4096 Nov 26 15:20 elasti
drwxr-xr-x 11 elastic elastic 4096 Nov 16 02:41 kibana
```

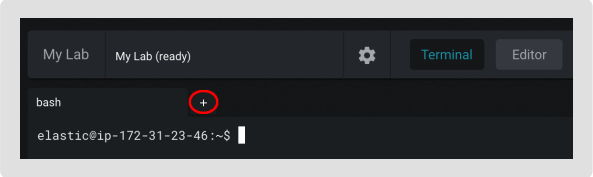
4. View the contents of `elasticsearch/config/elasticsearch.yml` file using a text editor of your choice: nano, emacs, or vi. If you are not familiar with command-line editors, we suggest you use nano (click here for tips on using nano). This is not the default config file, but one provided for this training. Notice the `node.name` is set to an environment variable named **NODENAME**. (We added a command into `/etc/profile` to dynamically define this variable.)

```
cluster.name: my_cluster
node.name: ${NODENAME}
network.host: _site_
discovery.seed_hosts: ["server1", "server2", "server3"]
```

5. Start Elasticsearch using the following command:

```
./elasticsearch/bin/elasticsearch -E cluster.initial_m
```

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



7. SSH onto `server2`:

```
ssh server2
```

8. Once `node1` is started, start Elasticsearch on `server2`, which will startup `node2`:

```
./elasticsearch/bin/elasticsearch -E cluster.initial_m
```

9. Now, you will add one more node to your cluster. So, open a new console tab and SSH onto `server3`:

```
ssh server3
```

10. Once `node2` is started, start Elasticsearch on `server3`, which will startup `node3`:

```
./elasticsearch/bin/elasticsearch -E cluster.initial_m
```

11. Finally, you will start Kibana on `server1`. So, open a new console tab and SSH onto `server1`:

```
ssh server1
```

12. Start Kibana on `server1` using the following command:

```
./kibana/bin/kibana
```

13. Now that Kibana is running, click here to connect.

14. Check if all nodes are running via the Kibana Dev Tools Console before going to the next step.

```
GET _cat/nodes
```

```
172.18.0.2 30 72 10 0.46 0.41 0.18 dim * node1
172.18.0.3 37 72  9 0.46 0.41 0.18 dim - node2
172.18.0.4 32 72  4 0.46 0.41 0.18 dim - node3
```

15. Now that you have your 3-node cluster up and running, you can access the blogs search page and play around with it.

Summary: In this lab, you accessed your classroom lab instance and started up a 3-node Elasticsearch cluster, along with a Kibana instance.

End of Lab Setup

Lab 1.1: Denormalization

Objective: In this lab, you will explore how to denormalize data and how denormalized data relate to mappings.

If you haven't finished the previous lab or had any issues with your cluster, you can reload the labs for this module by running the following command:

```
./load_lab.sh 1
```

By doing this, you will lose any previous work that is not required for this lab state and the Kibana and Elasticsearch processes will be running in the background. If you need to kill these processes you can use their process ID that has been saved into `elasticsearch_pid` and `kibana_pid` files. To stop Elasticsearch run `kill $(cat elasticsearch_pid)` To stop Kibana run `kill $(cat kibana_pid)`

1. Based on the following simplified Java classes, create a sample blog document using the JSON syntax. The blog document should be fully denormalized. *(**TIP:** think about the output of a toString method in the `Blogs` class.)*

```
class Blog {
  String id;
  String title;
  String[] products;
  Author[] authors;

  /* list of methods */
```

```
  ...
}

class Author {
  String name;
  Company company;

  /* list of methods */
  ...
}

public class Company {
  String name;
  Country country;

  /* list of methods */
  ...
}

public class Country {
  String name;
  String code;

  /* list of methods */
  ...
}
```

Show answer

2. Now that the document is defined, the next step is to define the mappings. Create a new index named `denormalized_blogs` with a mapping that corresponds to the denormalized document above. *(**TIP:** you can either create the mappings manually, one field at a time, or index the document into a temporary index, and then adjust the mappings.)*

   Show answer

3. Now, execute the following command to index three different blogs.

```
POST denormalized_blogs/_bulk
{"index":{"_id":1}}
{"id":"1","title":"Time Series with Kibana","authors":[
{"index":{"_id":2}}
{"id":"2","title":"Memory Issues We'll Remember","autho
{"index":{"_id":3}}
{"id":"3","title":"Making Kibana Accessible","authors"
```

4. Write and execute a query that returns all blogs written by a `chris` who works at `Globex`.

   Show answer

5. Write and execute an aggregation that returns the top 5 companies that have an author who wrote a blog.

   Show answer

6. Write and execute a query that returns all blogs written by any `alex` who works at `Elastic`. How many documents do you expect to get back?

   Show answer

   NOTE: The response of the previous query was unexpected. You are going to understand the why in the next lesson.

Summary: In this lab, you explored how to denormalize data and how denormalized data relate to mappings.

End of Lab 1.1

Lab 1.2: Nested and Join Data Types

Objective: In this lab, you will be using the nested data type. You will also see how the results of regular and `nested` queries and aggregations differ.

1. In the previous lab, you wrote and executed the following query to return all blogs written by any `alex` who works at `Elastic`.

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

```
    "bool": {
      "must": [
        {
          "match": {
            "authors.name": "alex"
          }
        },
        {
          "match": {
            "authors.company.name.keyword": "Elastic"
          }
        }
      ]
    }
  }
}
```

However, the query above returns a document that contains an `alex` who works at `ACME`. As discussed in this lesson, this happens because internally Lucene flattens the data. Next, we will fix that.

2. **EXAM PREP**: Use the `denormalized_blogs` mapping to create a new index named `nested_blogs` that has `authors` mapped as `nested` instead of `object`.

   Show answer

3. Why don't you have to also map `company` and `country` as `nested`?

Show answer

4. Now that you have a new index we need to index some data. Run the following command to load the same documents into the new index.

```
POST nested_blogs/_bulk
{"index":{"_id":1}}
{"id":"1","title":"Time Series with Kibana","authors":[
{"index":{"_id":2}}
{"id":"2","title":"Memory Issues We'll Remember","autho
{"index":{"_id":3}}
{"id":"3","title":"Making Kibana Accessible","authors"
```

5. **EXAM PREP**: Next, modify the query from the first step to execute a `nested` query. It should return a single document now.

Show answer

6. Next, we are going to compare a normal aggregation to a `nested` aggregation. Run the following aggregation and analyze the results. Are there any problems?

```
GET denormalized_blogs/_search
{
  "size": 0,
  "aggs": {
    "NAME": {
      "terms": {
```

```
          "field": "authors.company.name.keyword"
        },
        "aggs": {
          "NAME": {
            "terms": {
              "field": "authors.name.keyword"
            }
          }
        }
      }
    }
  }
```

Show answer

7. Based on the previous aggregation, write a nested version and execute it against the `nested_blogs` index. You do not want see any author as a sub-bucket of a company that they don't work for.

Show answer

Summary: In this lab, you implemented a nested object field. Nested and join datatypes are both good skills to know, but do not forget the most important rule of modeling data in Elasticsearch, which is to ***denormalize your data***! Both nested objects and parent/child relationships require expensive joins at runtime that are a common cause of performance issues, and you should avoid nested types and parent/child relationships whenever you can.

End of Lab 1.2

Lab 1.3: Field Modeling and the Elastic Common Schema

Objective: In this lab you will explore granular field modeling. You will also define field aliases.

1. Index the following three documents into an index called `version_test`:

```
POST version_test/_bulk
{"index":{"_id":1}}
{"version":"5.6.1"}
{"index":{"_id":2}}
{"version":"6.0.0"}
{"index":{"_id":3}}
{"version":"6.8.1"}
```

   These documents represent three versions: **5.6.1**, **6.0.0** and **6.8.1**.

2. Write a query that returns all documents with a minor version **6**.

   Show answer

3. To illustrate how much easier it is when you structure your data at ingestion time, index the same data into a new index

`version_fixed`, applying granular fields:

```
POST version_fixed/_bulk
{"index":{"_id":1}}
{"version":{ "display_name": "5.6.1", "major": 5, "minc
{"index":{"_id":2}}
{"version":{ "display_name": "6.0.0", "major": 6, "minc
{"index":{"_id":3}}
{"version":{ "display_name": "6.8.1", "major": 6, "minc
```

4. Write a query that returns all documents with a minor version **6** from this new index.

Show answer

5. To illustrate what happens when you do not follow a naming convention, index the following two simplified sample log events:

```
POST loglevel_test1/_doc/
{
  "level": "info"
}

POST loglevel_test2/_doc/
{
  "log_level": "warn"
}
```

Now, both indexes contain a document with a field representing the **log level**. However, in the `loglevel_test1` index this field is

called `level`, while in the `loglevel_test2` index it is called `log_level`.

6. Write a `terms` aggregation that returns the top **log levels** from accross the two `loglevel_test1` and `loglevel_test2` indexes.

   Show answer

7. Let's fix the problem using field aliases. By adding a field alias with the same name to the mappings of both indexes, pointing to our two different **log level** fields, you will be able to simply execute a `terms` aggregation on that field alias. Let's adopt the convention from Elastic Common Schema, and call the field alias `log.level`.

   Update the mapping of the `loglevel_test1` index and add a `log` field of type `object`, with a property `level` to it. Map `level` as an `alias` that points to the `level.keyword` field. Do the same for the `loglevel_test2` index, but now point the `level` alias to the `log_level.keyword` field.

   Show answer

8. To validate that everything has been set up correctly, execute a `terms` aggregation on the `log.level` alias in both indexes.

   Show answer

Summary: In this lab you have seen that by modeling your data granularly, and by using field aliases, you can avoid computationally expensive queries and aggregations.

End of Lab 1.3

Lab 1.4: Analyzers

Objective: In this lab, you will practice with the _**analyze** API to explore the different ways to analyze text. Also, you will use your text analysis and mappings skills to re-create the `blogs` index with a better configuration.

1. Test the following text using the _**analyze** API. Do not specify an analyzer - just use the default one:

```
"Introducing beta releases: Elasticsearch and Kibana D
```

Show answer

2. What was the default analyzer used in the previous step?

Show answer

3. Test the text again, but use the **whitespace** analyzer. What is the difference?

Show answer

4. Change the analyzer a few more times, testing the **stop**, **keyword** and **english** analyzers on the same text and comparing the different outputs. Which analyzer do you think works the best for the blogs website?

Show answer

5. Let's go one level lower than analyzers. Analyze the following text using the **standard** tokenizer and the **lowercase** and **snowball** filters. Now, compare the results with the output of the **english** analyzer. What is different between these two techniques?

```
"text": "This release includes mainly bug fixes."
```

Show answer

6. Using _**analyze**, configure and test an analyzer that satisfies the following:

   - uses the **standard** tokenizer
   - uses the **lowercase** token filter
   - uses the **asciifolding** token filter

   Test your analyzer with the exact text here by copy-and-pasting it:

```
"text": "Elasticsearch é um motor de buscas distribuído
```

   Run the command with and without **asciifolding** to see its effect on special characters.

Show answer

7. In many use cases, the built-in analyzers are not perfect. Imagine we want to search for **C++** or **IT**. Both the **standard** and

the **english** analyzers will not help much. Test the following sentence using the analyze API:

```
"text": "C++ can help it and your IT systems."
```

Show answer

8. **EXAM PREP**: To solve the problem discussed above, you need to write your own custom analyzer that handles text like C++ and IT in a better manner. Create an index named `analysis_test` that has an analyzer named `my_analyzer` which satisfies the following:

    - allow queries for **C++** to match only documents that contain **C++** (**TIP:** transform **c++** and **C++** into **cpp**)
    - allow queries for **IT** to match only documents that contain **IT** and not **it**. (**TIP:** transform **IT** into _**IT**_ before lowercase)
    - lowercase all text
    - remove the default stop words
    - remove the following terms as well: **can, we, our, you, your, all**

Show answer

9. **OPTIONAL:** Run the following two queries, which search for **blogs** that contain **c++** in the `content` field and for **IT** in the `title` field.

    Notice you get poor results in both cases because of how the default `standard` analyzer processes **c++** and **IT**.

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

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

10. **EXAM PREP**: Now, setup a new blogs index with the same data, but using a better, more appropriate analyzer:
    - Create a new index named `blogs_analyzed` that uses your custom `my_analyzer` from the previous step
    - Use the mappings from `blogs` and add a multi-field to **both** the `content` and `title` fields named `my_analyzer`. These multi-fields should be of type `text` and set the `analyzer` to `my_analyzer`.

Show answer

11. **OPTIONAL:** Run the following command to index the current `blogs` into your new `blogs_analyzed` index:

```
POST _reindex?wait_for_completion=false
{
  "source": {"index": "blogs"},
  "dest":   {"index": "blogs_analyzed"}
}
```

12. **OPTIONAL:** Rerun the queries from before against the new index using the `.my_analyzer` fields and compare the results.

Show answer

Summary: In this lab, you have practiced with the _**analyze** API to explore the different ways to analyze text. Also, you have used your text analysis and mappings skills to re-create the `blogs` index with a better configuration.

End of Lab 1.4

Lab 2.1: Changing Data

Objective: In this lab, you will fix some of the fields in the `blogs` index using some of the tools you learned, including the Reindex API and the Update By Query API.

If you haven't finished the previous lab or had any issues with your cluster, you can reload the labs for this module by running the following command:

```
./load_lab.sh 2
```

By doing this, you will lose any previous work that is not required for this lab state and the Kibana and Elasticsearch processes will be running in the background. If you need to kill these processes you can use their process ID that has been saved into `elasticsearch_pid` and `kibana_pid` files. To stop Elasticsearch run `kill $(cat elasticsearch_pid)` To stop Kibana run `kill $(cat kibana_pid)`

1. You are going to make several modifications to the `blogs` index, but we want to be careful and not do anything that ruins our original blogs. Therefore, you will reindex all of the blogs into a new index named `blogs_fixed` that you will use throughout the lab. Run the following PUT command, which creates `blogs_fixed` with an appropriate mapping:

```
PUT blogs_fixed
{
  "mappings": {
    "properties": {
      "author": {
        "type": "text",
        "fields": {
          "keyword": {
            "type": "keyword"
          }
```

```
      }
    },
    "category": {
      "type": "keyword"
    },
    "content": {
      "type": "text"
    },
    "locales": {
      "type": "keyword"
    },
    "publish_date": {
      "type": "date"
    },
    "seo_title": {
      "type": "text",
      "fields": {
        "keyword": {
          "type": "keyword"
        }
      }
    },
    "title": {
      "type": "text"
    },
    "url": {
      "type": "text",
      "fields": {
        "keyword": {
          "type": "keyword"
```

```
          }
        }
      },
      "number_of_views": {
        "type": "integer"
      },
      "reindexBatch": {
        "type": "byte"
      }
    }
  }
}
```

2. Notice the `blogs_fixed` mapping has a byte field named `reindexBatch`. Even though you didn't learn how to use scripts yet, you are going to increment this field every time you update the documents. This is a best practice and allows you to handle issues that might happen in the middle of updating. When needed, use the code below to update the `reindexBatch` value.

```
"script": {
  "source": "ctx._source['reindexBatch'] = 1;"
}
```

3. Reindex `blogs` into `blogs_fixed`. Make sure to use the script above to set the `reindexBatch` to 1. You should see 1,594 blogs created in `blogs_fixed`. View some of the documents in `blogs_fixed` and verify that the `reindexBatch` field was added properly.

Show answer

4. In the previous lesson you learned how to define a custom analyzer. Let's make that analyzer also available in our new `blogs_fixed` index by adding it to its settings. The analysis settings are non dynamic and can't be updated while the index is open. To update the analysis settings you need to close the index. Use the following command to close the `blogs_fixed` index.

```
POST blogs_fixed/_close
```

   Note that a closed index is blocked for read/write operations and does not allow all operations that opened indices allow. It is **not** possible to index documents or to search for documents in a closed index.

5. Add the custom analyzer that you defined in the previous lesson.

```
PUT blogs_fixed/_settings
{
  "analysis": {
    "char_filter": {
      "cpp_it": {
        "type": "mapping",
        "mappings": [
          "c++ => cpp",
          "C++ => cpp",
          "IT => _IT_"
        ]
```

```
      }
    },
    "filter": {
      "my_stop": {
        "type": "stop",
        "stopwords": [
          "can",
          "we",
          "our",
          "you",
          "your",
          "all"
        ]
      }
    },
    "analyzer": {
      "my_analyzer": {
        "tokenizer": "standard",
        "char_filter": [
          "cpp_it"
        ],
        "filter": [
          "lowercase",
          "stop",
          "my_stop"
        ]
      }
    }
  }
}
```

6. You can now reopen this index. When opening an index, the master node is responsible for restarting the shards to make them aware of the changed settings.

```
POST blogs_fixed/_open
```

7. Update the `blogs_fixed` mapping to add a multi-field `my_analyzer` for the field `content` and `title`.

Show answer

8. Now try to run a query on this new field.

```
GET blogs_fixed/_search
{
  "query": {
    "match": {
      "content.my_analyzer": "c++"
    }
  }
}
```

You didn't get any results why?

Show answer

9. **EXAM PREP**: Using an `_update_by_query`, update all the documents in `blogs_fixed` with a `reindexBatch` equal to 1. Use the script given above to update the `reindexBatch` to 2 on all documents.

Show answer

10. Run your `_update_by_query` command again. Assuming that all documents were previously updated in the first `_update_by_query`, this time the update should return very quickly and show 0 documents being updated.

11. Try again to search on the `my_analyzer` field, you should get 2 documents.

```
GET blogs_fixed/_search
{
  "query": {
    "match": {
      "content.my_analyzer": "c++"
    }
  }
}
```

12. Delete all the documents where the `category` is `Releases`. You should delete 238 documents.

Show answer

Summary: In this lab, you fixed the mapping of the blogs index by using the `_reindex` API. You also learned how to update documents inside an index by using update and delete by query.

End of Lab 2.1

Lab 2.2: Ingest Pipelines

Objective: In this lab, you will use ingest pipelines to clean the data of your `blogs_fixed` index.

1. Let's clean up the `locales` field in `blogs_fixed`. Run the following `terms` aggregation and analyze the results. What could be improved on the `locales` field?

```
GET blogs_fixed/_search
{
  "size": 0,
  "aggs": {
    "locale_terms": {
      "terms": {
        "field": "locales",
        "size": 10
      }
    }
  }
}
```

2. **EXAM PREP**: The `locales` field is empty for 1,290 documents, which is over 90% of the index. These particular documents should have the English locale "**en-en**". Also, as discussed in the lecture, this field would be much easier to search and aggregate

on if it was indexed as an array instead of a single comma-separated list of values. To fix `locales`, write a pipeline that satisfies the following criteria:

- The name of the pipeline is `fix_locales`
- The first processor is a `set` processor that checks if the `locales` field is an empty string. If it is empty, assign it the value "**en-en**". If it is not empty, leave the field as is. **TIP:** To check if a field is empty you can use `ctx['field'].empty` in the `if` option of your processor.
- The second `set` processor should set `reindexBatch` to 3 for every document
- The third processor is a `split` processor that splits the `locales` field into an array, using a comma as the separator

Show answer

3. Test the following two documents on your `fix_locales` pipeline:

```
{
  "locales": "de-de,fr-fr,ja-jp,ko-kr"
}

{
  "locales": ""
}
```

If successful, the results should look like:

```
{
  "docs": [
    {
      "doc": {
        ...
        "_source": {
          "locales": [
            "de-de",
            "fr-fr",
            "ja-jp",
            "ko-kr"
          ],
          "reindexBatch": 3
        },
        ...
      }
    },
    {
      "doc": {
        ...
        "_source": {
          "locales": [
            "en-en"
          ],
          "reindexBatch": 3
        },
        ...
      }
    }
```

```
  ]
}
```

Show answer

4. **EXAM PREP**: Using an `_update_by_query`, update all documents in `blogs_fixed` with a `reindexBatch` equal to 2. Use the `fix_locales` pipeline to update the documents.

   Show answer

5. **OPTIONAL:** Check the stats of your pipeline by running the following query.

   ```
   GET _nodes/stats/ingest?filter_path=nodes.*.ingest.pipe
   ```

   This shows how many documents have been processed for each processor in your pipeline.

6. Run the `terms` aggregation again on the `locales` field. You should see completely different results this time. Notice each locale is broken out into a single value, so you are getting an accurate count of each individual locale:

   ```
   "aggregations" : {
     "locale_terms" : {
       "doc_count_error_upper_bound" : 0,
       "sum_other_doc_count" : 0,
       "buckets" : [
         {
   ```

```
        "key" : "en-en",
        "doc_count" : 1290
      },
      {
        "key" : "fr-fr",
        "doc_count" : 43
      },
      {
        "key" : "de-de",
        "doc_count" : 41
      },
      {
        "key" : "ko-kr",
        "doc_count" : 37
      },
      {
        "key" : "ja-jp",
        "doc_count" : 34
      },
      {
        "key" : "zh-chs",
        "doc_count" : 8
      }
    ]
  }
}
```

7. Use the following query to view the `locales` of some documents from `blogs_fixed`. Notice the values of the

`locales` are all in an array now:

```
GET blogs_fixed/_search
{
  "size": 100,
  "_source": "locales"
}
```

8. **EXAM PREP**: Your front-end developer forgot to tell you that he is expecting locales to be separated by an `underscore` (_) instead of a `hyphen` (-). For instance, `en-en` should be `en_en`. To fix it, write a pipeline that satisfies the following criteria:
   - The name of the pipeline is `underscore_locales`
   - Using a foreach processor and a gsub processor, replace all the `hyphen` by `underscore` in the locales.
   - The second set processor should set reindexBatch equal to 4 for every document

Show answer

9. Test your pipeline on this document.

```
"locales": [
    "de-de",
    "fr-fr",
    "ja-jp",
    "ko-kr",
```

```
    "zh-chs"
  ]
```

If successful, the results should look like:

```
"_source" : {
  "locales" : [
    "de_de",
    "fr_fr",
    "ja_jp",
    "ko_kr",
    "zh_chs"
  ],
  "reindexBatch" : 4
}
```

Show answer

10. Using an `_update_by_query`, update all documents in `blogs_fixed` with a `reindexBatch` equal to 3 with the `underscore_locales` pipeline.

Show answer

11. Use the previous `_search` query to verify that your documents are successfully updated.

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

```
  "_source": "locales"
}
```

Summary: In this lab, you saw how to use ingest pipeline to apply modifications to your documents.

End of Lab 2.2

Lab 2.3: Painless Scripting

Objective: In this lab, you will familiarize yourself with the `painless` scripting language.

WARNING: Even though the dot notation (e.g. `ctx.title`) is commonly used in scripting languages and in the painless documentation, it **doesn't** accept some special characters (e.g `ctx.@timestamp`). If you want to access the field `@timestamp` you need to use the brackets notation (e.g. `ctx['@timestamp']`. Therefore, we use the brackets notation in all the examples.

1. Let's continue to clean up the `blogs_fixed` index by cleaning the `seo_title` field. Run the following query, and notice that 1,085 blog posts have an empty `seo_title` field (which is 80% of our blogs!).

```
GET blogs_fixed/_search
{
```

```
  "query": {
    "bool": {
      "filter": {
        "match": {
          "seo_title.keyword": ""
        }
      }
    }
  }
}
```

2. **EXAM PREP**: Define an ingest pipeline that satisfies the following criteria:
   - The name of the pipeline is `fix_seo_title`
   - Add a `script` processor that checks if the `seo_title` is equal to an empty string "". If it is, set `seo_title` to the value of the document's `title` field. **TIP:** As you are going to run this script in an ingest pipeline, the syntax for accessing the fields of a document is `ctx['field_name']` (without the `_source`). For example, to access the `seo_title` in your script, use `ctx['seo_title']`.
   - Set the value of `reindexBatch` to 5 for every document

   Show answer

3. You should always test a pipeline with some sample documents before running it on your entire index! Test your `fix_seo_title` pipeline using the following two documents:

```
{
   "title": "Where in the World is Elastic? - Elastic{(
   "seo_title": ""
}

{
   "title": "This week in Elasticsearch and Apache Luce
   "seo_title": "What's new in Elasticsearch and Apache
}
```

Show answer

4. **EXAM PREP**: Run an `_update_by_query` on `blogs_fixed`, sending each document through your `fix_seo_title` pipeline. Your update by query should only update documents that have a `reindexBatch` value equal to 4. You should see that all 1,356 documents are updated.

   Show answer

5. Run the following query again, and notice now that none of your documents have an empty `seo_title` field:

```
GET blogs_fixed/_search
{
  "query": {
    "bool": {
      "filter": {
        "match": {
          "seo_title.keyword": ""
```

```
        }
      }
    }
  }
}
```

6. Next, imagine that every document in `blogs_fixed` should have the number of views of that blog. Run an `update_by_query` with a script that sets the `number_of_views` field to 0 on every document.

   Show answer

7. Next, you will write a script that updates the number of views that a blog gets based on the web access logs. Run the following query and copy the document `_id` returned, as you will need it in later steps:

```
GET blogs_fixed/_search
{
  "query": {
    "bool": {
      "filter": {
        "match": {
          "url.keyword": "/blog/elasticsearch-storage-t
        }
      }
    }
  }
}
```

You should get one hit: a blog titled *"The true story behind Elasticsearch storage requirements"*. Now run the following query, which hits every log entry for this blog on May 12, 2017. You should get 41 hits:

```
GET logs_server*/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "@timestamp": {
              "gte": "2017-05-12",
              "lt": "2017-05-13"
            }
          }
        },
        {
          "match": {
            "originalUrl.keyword": "/blog/elasticsearch
          }
        }
      ]
    }
  }
}
```

8. Using an `_update` and an inline script, add 41 to the `number_of_views` field of the blog above. You will need the `_id` of the blog, which you already copied from the result of the first query in the previous step.

   Show answer

9. Get the blog post that you just modified and verify that its `number_of_views` is now 41.

   Show answer

10. This script seems to work fine, let's store this script so you can get the most out of caching. Write a script that satisfies the following criteria:

    - The script is stored in the cluster state with the id `add_to_number_of_views`
    - The script increments `number_of_views` by the amount of the value in a parameter named `new_views`

    Show answer

11. Run the following query and notice there are 11 hits for the same blog received on May 13, 2017:

```
GET logs_server*/_search
{
  "query": {
    "bool": {
      "filter": [
        {
```

```
          "range": {
            "@timestamp": {
              "gte": "2017-05-13",
              "lt": "2017-05-14"
            }
          }
        },
        {
          "match": {
            "originalUrl.keyword": "/blog/elasticsearch
          }
        }
      ]
    }
  }
}
```

Using the script `add_to_number_of_views`, increment the `number_of_views` for this blog by 11 and verify that the new value of `number_of_views` is 52.

Show answer

12. **OPTIONAL:** Because an ingest pipeline cannot send requests to another Elasticsearch index, you cannot automatically update every `number_of_views`. Doing so requires an external script or tools like Logstash.

    Let's simulate the number of views for the other blogs. Write a script `add_random_number_of_views` which sets the field `number_of_views` to a random integer between 0 and 10,000.

You will use this script in an ingest pipeline so make sure to use the syntax accordingly.

TIP: You should use the class Random

Show answer

13. **OPTIONAL:** Create an ingest pipeline `number_of_views` which uses the script that you previously created.

Show answer

14. **OPTIONAL:** Update every blogs except the one that you already updated (the url is `/blog/elasticsearch-storage-the-true-story`). You should update 1,355 documents.

Show answer

Summary: In this lab, you learned how to define a script using the `painless` language. You learned how to store scripts and use it to update some documents of your index.

End of Lab 2.3

Lab 3.1: Securing Elasticsearch

Objective: In this lab, you will secure a one node Elasticsearch cluster with Elastic security and then you will also create users and

roles for your secure cluster. **This entire lab should be considered as preparation for the certification exam.**

If you haven't finished the previous labs or had any issues with your cluster, you can reload the labs for this module by running the following command:

```
./load_lab.sh 3
```

By doing this, you will lose any previous work that is not required for this lab state and the Kibana and Elasticsearch processes will be running in the background. If you need to kill these processes you can use their process ID that has been saved into `elasticsearch_pid` and `kibana_pid` files. To stop Elasticsearch run `kill $(cat elasticsearch_pid)` To stop Kibana run `kill $(cat kibana_pid)`

1. Elastic Stack security features enable you to lock down your cluster and setup users and roles. In this lab you are going to secure a separate one node cluster. First, open a new tab and ssh to `server7`.

   ```
   ssh server7
   ```

2. Next, configure `node7` and enable elastic security. In the `elasticsearch.yml` configuration file, **delete all existing lines and replace with the following lines**:

   ```
   cluster.name: my_secure_cluster
   node.name: ${NODENAME}
   network.host: _site_
   xpack.security.enabled: true
   discovery.type: single-node
   ```

NOTE: Ensure you have cleared the original contents of the `elasticsearch.yml` file and replaced the contents, exactly, with what is listed above.

3. Start Elasticsearch.

Show answer

4. Open a new terminal and try running the following command:

```
curl 'server7:9200/_cat/nodes?pretty'
```

You should get the following security error because you are trying to access a secure cluster without any credentials.

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "security_exception",
        "reason" : "missing authentication token for RE
        "header" : {
          "WWW-Authenticate" : "Basic realm=\"security\
        }
      }
    ],
    "type" : "security_exception",
    "reason" : "missing authentication token for REST r
    "header" : {
      "WWW-Authenticate" : "Basic realm=\"security\" ch
    }
  },
  "status" : 401
}
```

5. You need to create some credentials. Login to `server7` and run the `elasticsearch-setup-passwords` script:

```
ssh server7
./elasticsearch/bin/elasticsearch-setup-passwords inter
```

You will be prompted for a password for each one of the reserved users. Just enter **nonprodpwd** for all the passwords to keep things simple:

```
Initiating the setup of passwords for reserved users el
You will be prompted to enter passwords as the process
Please confirm that you would like to continue [y/N]y


Enter password for [elastic]:
Reenter password for [elastic]:
Enter password for [apm_system]:
Reenter password for [apm_system]:
Enter password for [kibana_system]:
Reenter password for [kibana_system]:
Enter password for [logstash_system]:
Reenter password for [logstash_system]:
Enter password for [beats_system]:
Reenter password for [beats_system]:
Enter password for [remote_monitoring_user]:
Reenter password for [remote_monitoring_user]:
Changed password for user [apm_system]
Changed password for user [kibana_system]
Changed password for user [kibana]
Changed password for user [logstash_system]
Changed password for user [beats_system]
Changed password for user [remote_monitoring_user]
Changed password for user [elastic]
```

6. Now, run the same command as before but with the username and password. (**WARNING**: Never provide password in plain text in sensitive environments.)

```
curl 'http://elastic:nonprodpwd@server7:9200/_cat/nodes
```

You should get the following response.

```
172.18.0.8 10 98 10 0.41 0.37 0.17 dilmrt * node7
```

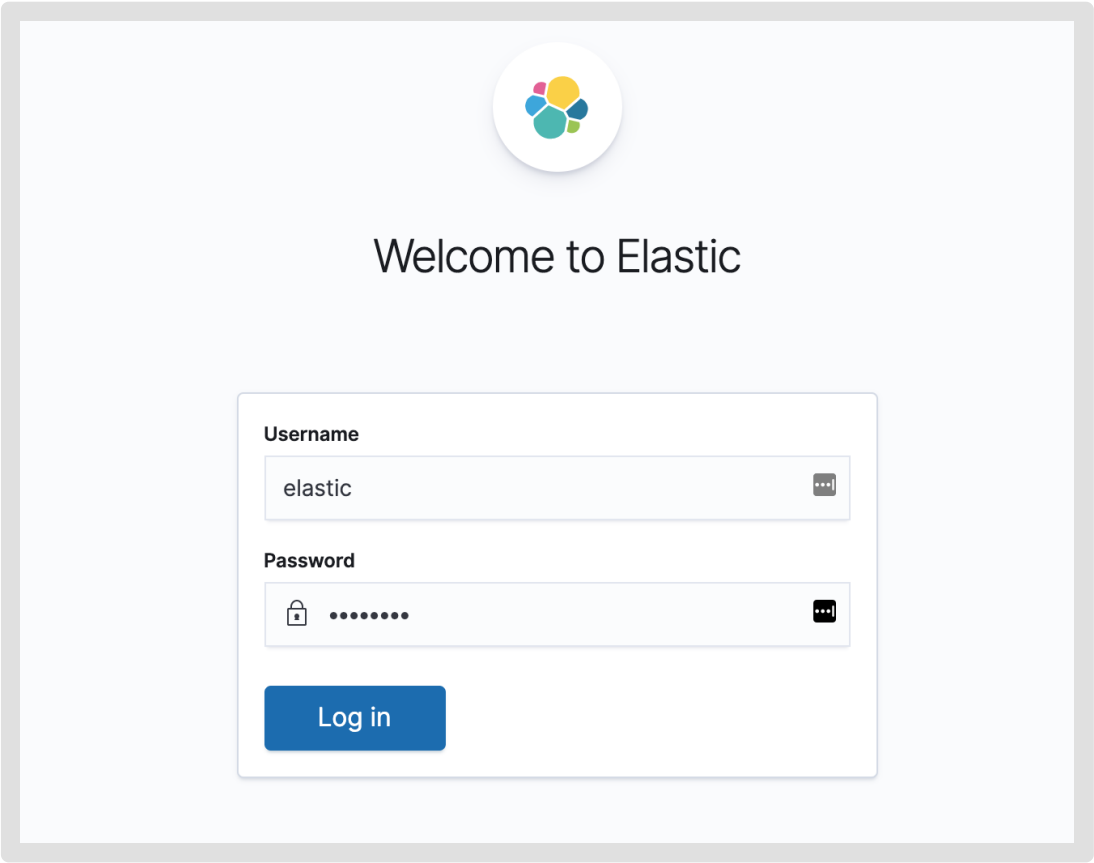
This also means that Kibana will not be able to connect without credentials.

7. Next, you are going to see how to use Kibana with security. Navigate to the terminal window that has Kibana up and running. Stop Kibana by pressing `ctrl+c` and then run the following command to start Kibana connecting to `node7`. (**WARNING**: Never provide password in plain text in sensitive environments.)

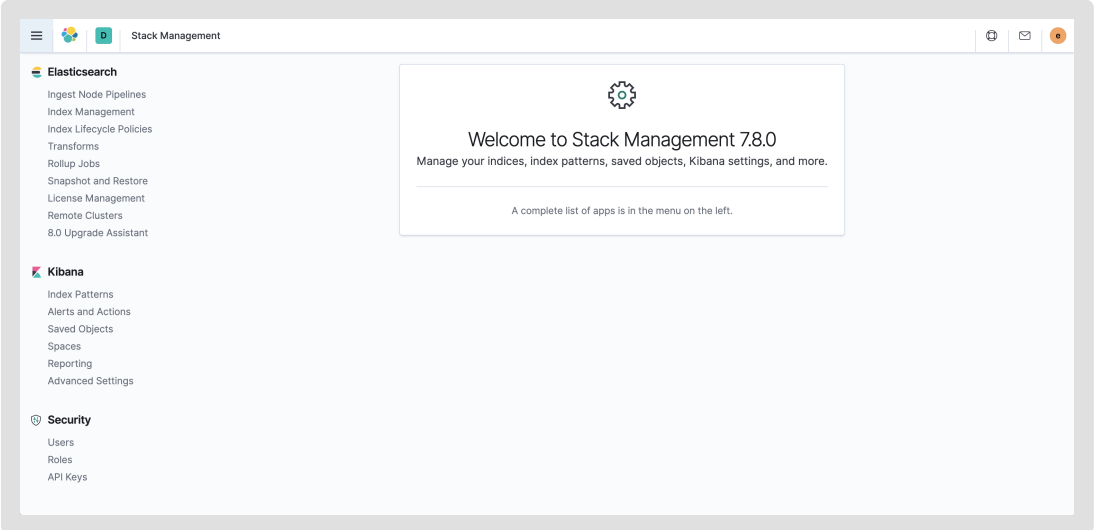
```
./kibana/bin/kibana --elasticsearch.hosts="http://serve
```

8. Try connecting to your Kibana instance from a new web browser tab, and you should see a login page.

9. Log in as the "**elastic**" user with password "**nonprodpwd**":



10. From the Kibana navigation menu, select on the "**Stack Management**" (under **Management**). Notice the **Security** section that allows you to configure users, roles and API keys:



11. Create a new role named `read_only` that satisfies the following criteria:

- The user has no cluster privileges
- The user has access to indices that match the pattern `*`
- The index privileges are only `read`

Show answer

12. Create a new user named `read_only_user` that satisfies the following criteria:
    - password is "**nonprodpwd**"
    - enter **Read Only User** for the name of the user
    - use your own email address
    - assign the user to **two** roles: `read_only` and `kibana_admin`

    NOTE: Make sure to add the `kibana_admin` role, otherwise you won't be able to log in to Kibana with `read_only_user`.

    Show answer

13. Log out and log in again as `read_only_user` user. Navigate to the Console and run the commands below. Notice that the only successful command is the `_search`-request, as it only reads data.

```
GET /

GET _search

PUT new_index/_doc/1
{
```

```
  "security_test": "this will fail"
}
```

Summary: In this lab, you secured a one node cluster using Elastic security and then you saw how to create a user that has limited access to specific indices. This lab uses a one node only cluster to simplify the lab. To learn how to properly secure a multi-node cluster we recommend you take the Fundamentals of Securing Elasticsearch course.

End of Lab 3.1

Lab 3.2: Development vs. Production Mode

Objective: In this lab, you will try to startup a node that does not pass the bootstrap checks. You will also see how the request cache works.

1. Before you start this lab, you should cleanup the previous lab (Securing Elasticsearch):
    - Stop elasticsearch on `node7` and close the terminal tab
    - Stop Kibana and restart it without any command line parameter
    - Refresh your Kibana tab

Show answer

2. Next, you will better understand bootstrap checks. Open a new tab and ssh to `server6`. Edit the Elasticsearch `jvm.options` file and set the initial heap size to `512m` and the maximum heap size to `1g`.

   Show answer

3. Try to startup `node6`. What happens?

   Show answer

4. Set both the min and max heap size on `server6` to `512m`, but do not startup Elasticsearch yet. We will be using `server6` again in a later lab.

5. Now, you are going to explore the request cache. In order to notice the impact of caching, you will need to put some load in the cluster. An aggregation that has to iterate over all documents many times will help. In this example, you will run an aggregation that has three terms aggregations (one "root" aggregation, one "sub-bucket" aggregation, and one "sub-sub-bucket" aggregation). Run the following search command multiple times and notice that the `took` time is going to decrease after a few executions and become really small. Remember that the request cache is enabled by default.

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

```
  "aggs": {
    "NAME": {
      "terms": {
        "field": "geoip.country_name.keyword",
        "size": 100
      },
      "aggs": {
        "NAME": {
          "terms": {
            "field": "geoip.region_name.keyword",
            "size": 100
          },
          "aggs": {
            "NAME": {
              "terms": {
                "field": "geoip.city_name.keyword",
                "size": 1
              }
            }
          }
        }
      }
    }
  }
}
```

6. If you really want to verify if the speed is related to the request cache, execute the same search with the `request_cache` parameter set to `false`.

```
GET logs_*/_search?request_cache=false
{
  "size": 0,
  "aggs": {
    "NAME": {
      "terms": {
        "field": "geoip.country_name.keyword",
        "size": 100
      },
      "aggs": {
        "NAME": {
          "terms": {
            "field": "geoip.region_name.keyword",
            "size": 100
          },
          "aggs": {
            "NAME": {
              "terms": {
                "field": "geoip.city_name.keyword",
                "size": 1
              }
            }
          }
        }
      }
    }
  }
}
```

7. The request cache does not work by default if the search returns hits. Update the `size` to 10 and run the aggregation a few times without the `request_cache` parameter. Notice that the execution time is high and it does not matter how many times you execute it, it will not be under 50ms.

```
GET logs_*/_search
{
  "size": 10,
  "aggs": {
    "NAME": {
      "terms": {
        "field": "geoip.country_name.keyword",
        "size": 100
      },
      "aggs": {
        "NAME": {
          "terms": {
            "field": "geoip.region_name.keyword",
            "size": 100
          },
          "aggs": {
            "NAME": {
              "terms": {
                "field": "geoip.city_name.keyword",
                "size": 1
              }
            }
          }
        }
      }
```

```
    }
  }
}
```

8. You can force any request to be cached by appending `?request_cache=true` to the URI. Add this parameter to the previous search and run it again.

```
GET logs_*/_search?request_cache=true
{
  "size": 10,
  "aggs": {
    "NAME": {
      "terms": {
        "field": "geoip.country_name.keyword",
        "size": 100
      },
      "aggs": {
        "NAME": {
          "terms": {
            "field": "geoip.region_name.keyword",
            "size": 100
          },
          "aggs": {
            "NAME": {
              "terms": {
                "field": "geoip.city_name.keyword",
                "size": 1
              }
```

```
                }
            }
        }
    }
  }
}
```

Notice that the first request will take a long time to execute. Then, starting with the second request, executions will be really fast, which means the request cache is being used. Remember that the request cache will not work on indices under constant indexing operations.

Summary: In this lab, you saw how a node will not startup properly if it fails the bootstrap checks. You also saw how the request cache works.

End of Lab 3.2

Lab 3.3: Scaling Elasticsearch

Objective: In this lab, you will test over allocation of shards. You will also explore how to scale for reads with replicas and some settings to optimize write throughput during an initial load.

1. Create an index called `temp1` with the following command:

```
PUT temp1
```

2. Run the following `_cat` command to review the default number of primary and replica shards as well as how the shards of `temp1` got allocated:

```
GET _cat/shards/temp1?v&h=index,shard,prirep,state,node
```

You should get a response similar to the one below. The shards might be allocated to other nodes.

```
index shard prirep state   node
temp1 0     p      STARTED node2
temp1 0     r      STARTED node3
```

Index `temp1` has one primary and one replica shard, the default configuration. Even though these settings work great for several applications, you might want to have more primary shards to over allocate your shards and take advantage of your four data nodes, assuming that you are going to have a lot of data being stored in this index.

3. Create one index called `temp3` that has three primary shards and zero replica shards, and disable refresh. You are creating `temp3` with `index.refresh_interval` set to `-1` and `index.number_of_replicas` set to `0` because you are going to load a large amount of data at once in the next steps.

Show answer

4. Before loading data into `temp3`, run the following `_cat` command to review its shard allocation:

```
GET _cat/shards/temp3?v&h=index,shard,prirep,state,node
```

You should get a response similar to the one below:

```
index shard prirep state   node
temp3 0     p      STARTED node2
temp3 1     p      STARTED node3
temp3 2     p      STARTED node1
```

Note that you should have one primary shard allocated per data node.

5. Reindex all documents from `logs_server*` into `temp3`. Set `wait_for_completion` to `false`, so you can proceed to the next steps.

Show answer

6. How many documents do you get if you run the following command to check how your data load is working? Why?

```
GET temp3/_count
```

Show answer

7. Use the following command to monitor whether your reindex process has finished (it will take a couple of minutes).

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

You will know that it finished when the response no longer contains that reindex task. If you want to cross check, you can also run the following command and check whether `docs.count` for `temp3` is above `1400000` (it's very unlikely that it will be `1751477`, as you disabled `refreshing`).

```
GET _cat/indices/temp3?v
```

8. Now that you finished your initial load, enable refresh on `temp3` and bring it back to the `1s` default configuration.

Show answer

9. How many documents do you get now by running the following command? Why?

```
GET temp3/_count
```

Show answer

10. After your initial load finished and you enabled back the refresh interval, it is also a good idea to add some replicas to your index. Run the following command to auto-expand the number of replicas for the `temp3` index based on the number of data nodes in the cluster.

```
PUT temp3/_settings
{
  "index.auto_expand_replicas": "0-all"
}
```

IMPORTANT: Note that the auto-expanded number of replicas does not take any other allocation rules into account, such as shard allocation awareness, filtering or total shards per node, and this can lead to the cluster health becoming **yellow** if the applicable rules prevent all the replicas from being allocated.

11. Run the following `_cat` command to review `temp3` shard allocation. How many replica shards do you have after enabling auto-expanded replicas?

```
GET _cat/shards/temp3?v&h=index,shard,prirep,state,node
```

Show answer

12. Suppose that your read throughput decreased and you don't need `node3` anymore. So, stop this node.

13. How many replica shards the `temp3` index has after you stopped `node3` ?

Show answer

14. Read throughput increased back again! Restart `node3` to scale with replicas. Run the `_cat` command from the previous step

again to verify that `temp3` has again two replica shards for every primary shard.

15. You are done with these tests, so it is a good idea to clean up the indices you just created to run them. Run the following command to delete `temp1` and `temp3` indices.

```
DELETE temp*
```

Summary: In this lab, you tested over allocation of shards. You also explored how to scale for reads with replicas and some settings to optimize write throughput during an initial load.

End of Lab 3.3

Lab 4.1: Cluster Backup

Objective: In this lab, you will learn how to back up and restore your cluster. **This entire lab should be considered as preparation for the certification exam.**

If you haven't finished the previous lab or had any issues with your cluster, you can reload the labs for this module by running the following command:

```
./load_lab.sh 4
```

By doing this, you will lose any previous work that is not required for this lab state and the Kibana and Elasticsearch processes will be running in the background. If you need to kill these processes you can use their process ID that has been saved into `elasticsearch_pid` and `kibana_pid` files. To stop Elasticsearch run `kill $(cat elasticsearch_pid)` To stop Kibana run `kill $(cat kibana_pid)`

1. Let's configure a snapshot lifecycle policy that snapshots your data every night. Before you can do that, you need to configure a repository for your snapshots.

   First stop Elasticsearch on all three of your servers, but leave the terminal tabs open. (This task will be easier if you are SSH'd onto each of your three servers in a separate tab.)

2. Notice there is a shared folder named `/shared_folder` that each server can read/write to. Create a new subfolder of `/shared_folder/` called `my_repo` (you only need to run this command on one of the servers):

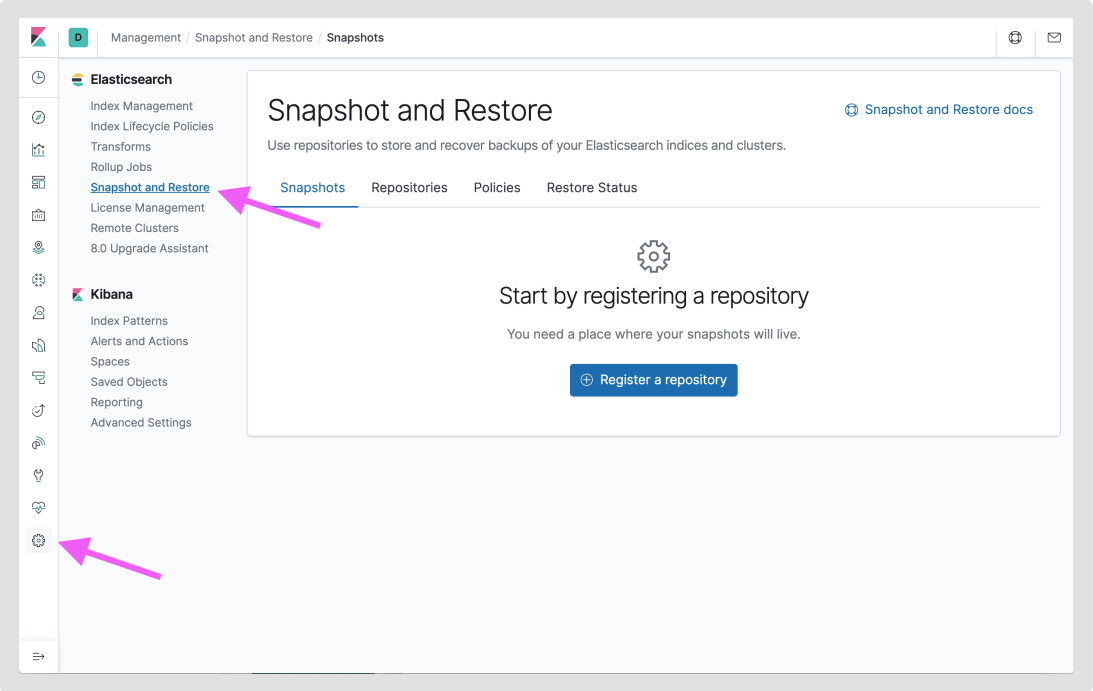
   ```
   mkdir /shared_folder/my_repo
   ```

3. Add the following `path.repo` setting to the `elasticsearch.yml` files of `node1`, `node2` and `node3`:

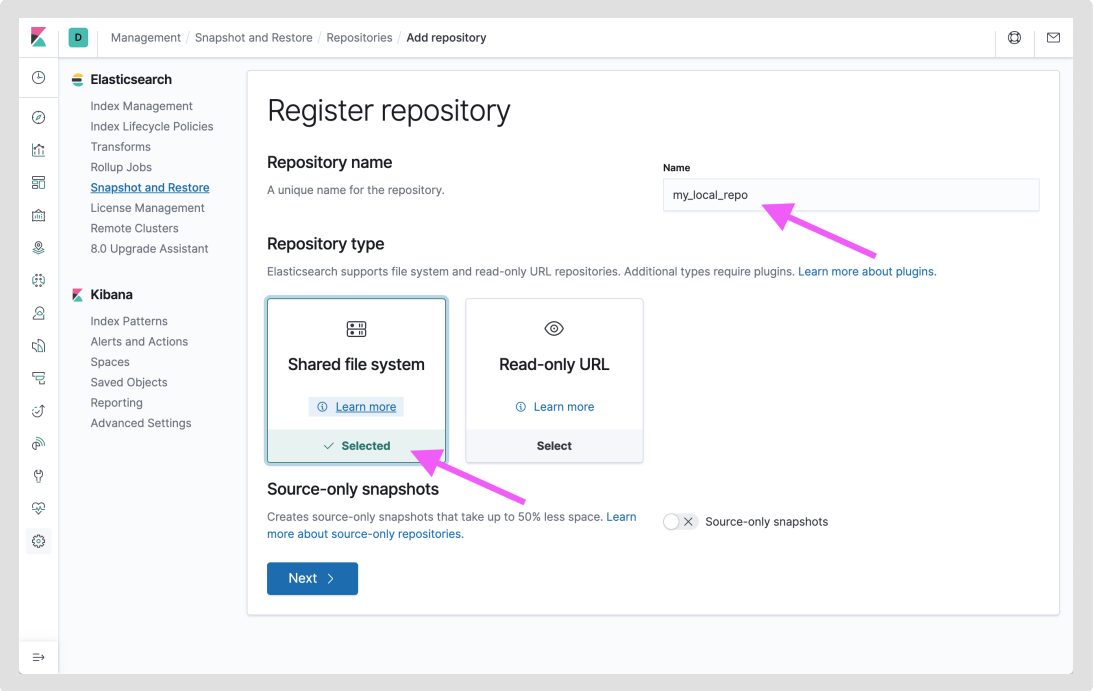
   ```
   path.repo: /shared_folder/my_repo
   ```

4. Start Elasticsearch on all three nodes.

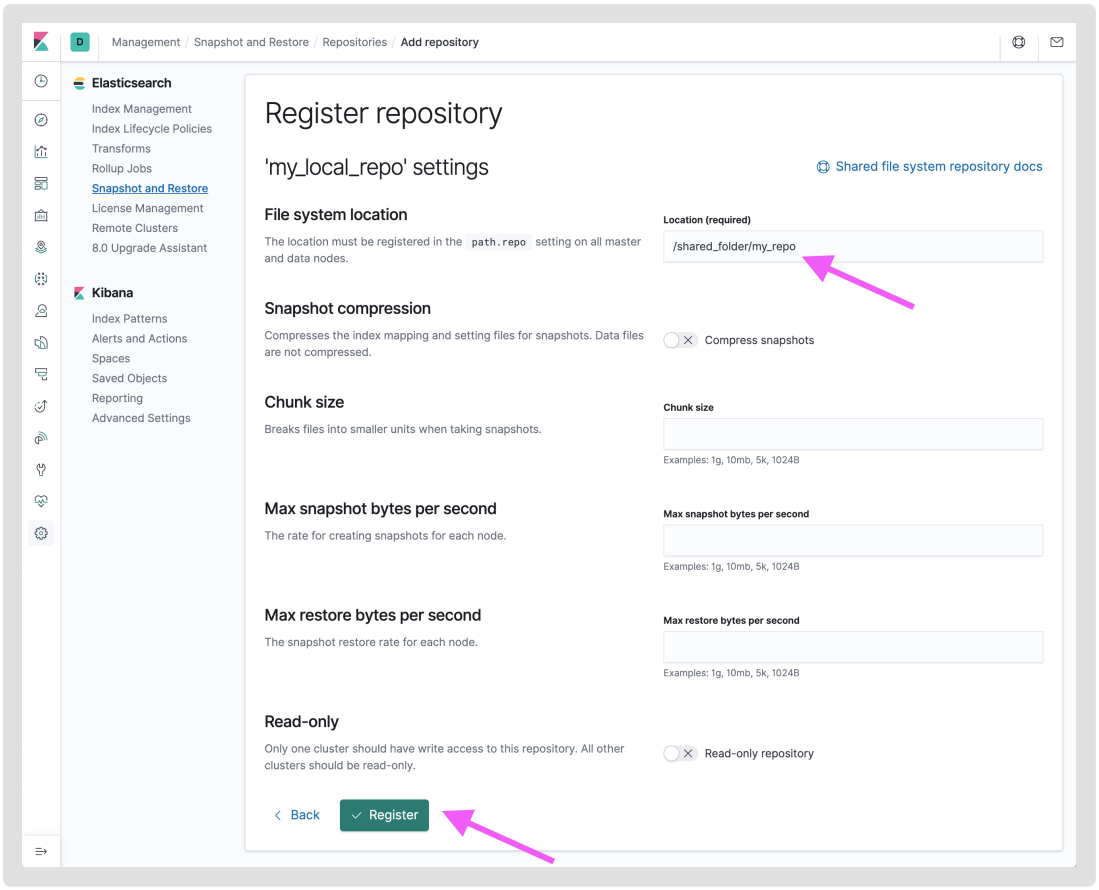
5. In Kibana, click on the last icon on the left hand side to navigate to **Management**, and next click **Snapshot and Restore** under **Elasticsearch**. You should see a blue **Register a repository** button. Click it:



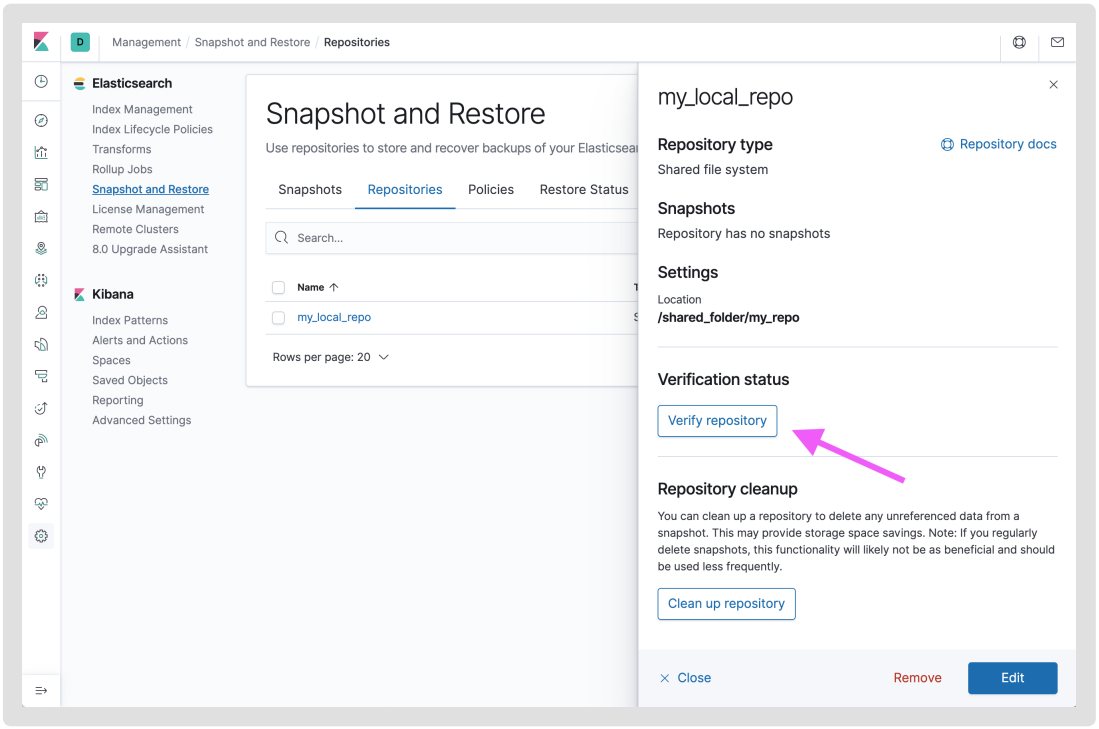
6. Name your repository `my_local_repo` and select **Shared file system** as the repository type. Click **Next**.



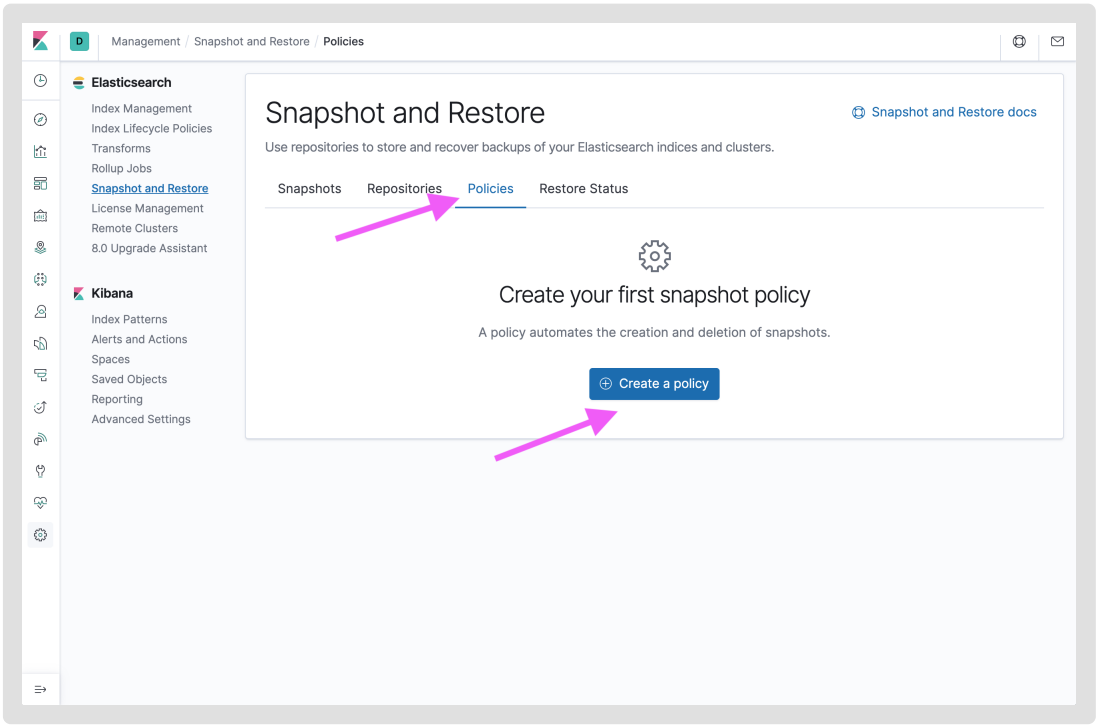
7. Enter `/shared_folder/my_repo` for the **File system location**. Don't enter any values for the other options. Scroll down and click **Register**:



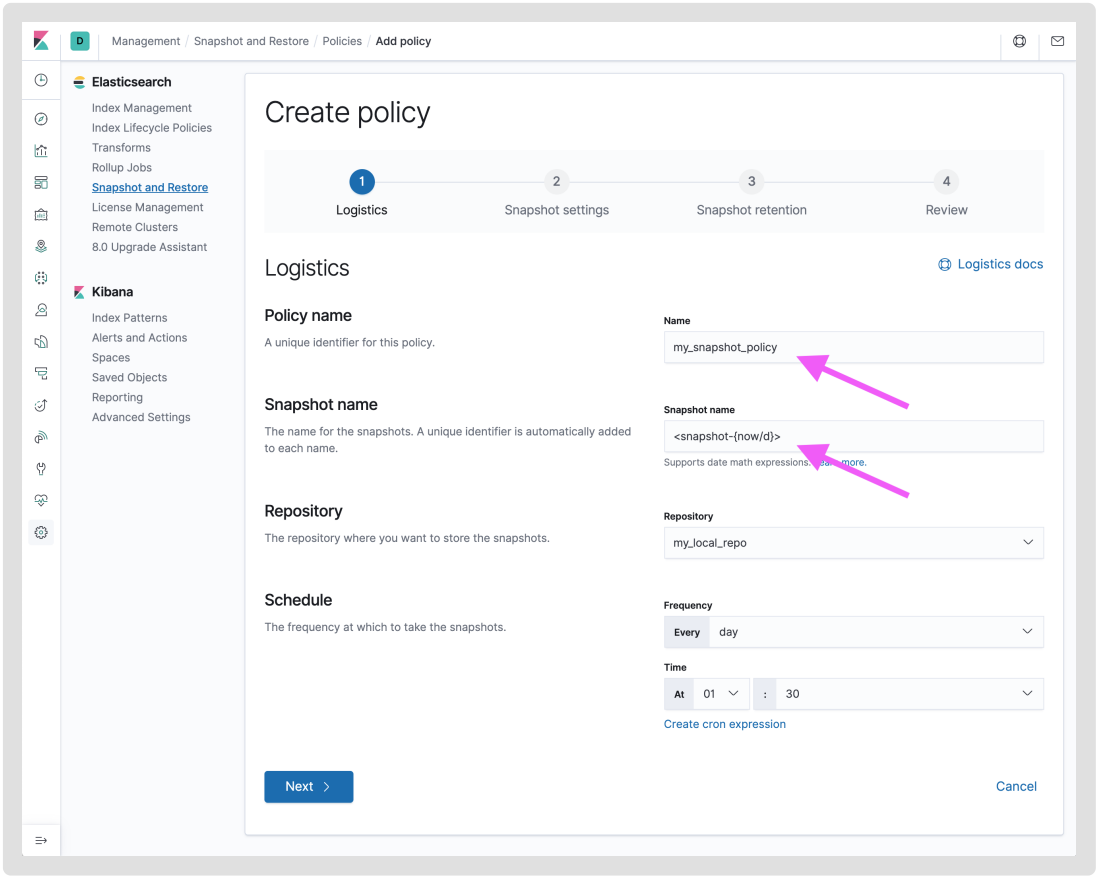
8. Click the **Verify repository** button on the right-hand side. If all went well, you should see that the repository has a status "Connected":



9. Now you're ready to define a snapshot policy. Close the panel on the right-hand side and select the third tab, **Policies**. Next, click the **Create a policy** button:



10. Enter `my_snapshot_policy` for the **Policy name** and `<snapshot-{now/d}>` for the snapshot name.

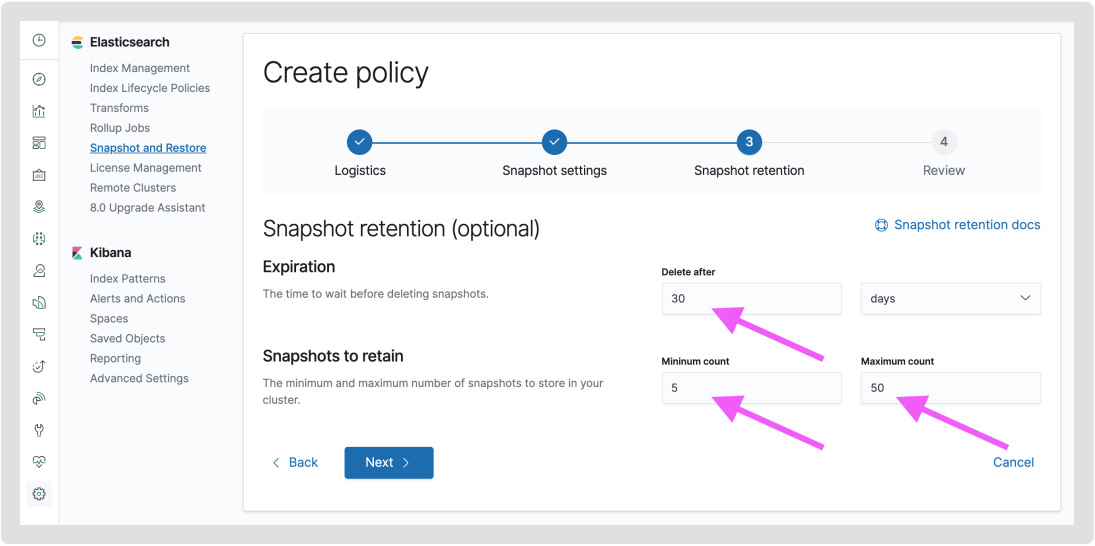


11. There is no need to change any of the other settings. `my_local_repo` should already be selected as the repository. The default schedule that will take a snapshot every night at 1:30 is fine too.

    Click **Next**.

12. On the next page, "Snapshot settings", leave all the defaults, and click **Next**.

13. On the next page, "Snapshot retention", enter `30` days for **Delete after**, `5` for the **Minimum count** and `50` for the **Maximum count**.



14. Click **Next**, review your policy, and click **Create policy**.

15. Everything you have done so far using the Kibana user interface could also have been done using the Elasticsearch API. Let's take a look at your policy using the API.

    Go to Kibana's **Console** by clicking **Dev Tools** from the menu on the left-hand side and execute the following request:

    ```
    GET /_slm/policy/my_snapshot_policy?human
    ```

    This will return information about your policy. You can for example see when it is scheduled to execute next.

16. Let's not wait until 1:30 and test the policy now. You can manually execute a snapshot policy to take a snapshot immediately by using the `_execute` endpoint. Run the following request in Console:

    ```
    POST /_slm/policy/my_snapshot_policy/_execute
    ```

17. Execute the following request again, and you should now see your manual snapshot listed under `last_success`:

```
GET /_slm/policy/my_snapshot_policy?human
```

18. List all the snapshots in your repository by executing the following:

```
GET _snapshot/my_local_repo/_all
```

    You should see your snapshot in the list. Copy the snapshot's name (it starts with `snapshot-`, followed by a date, followed by some alphanumerical characters).

19. Restore the `logs_server*` indices using your snapshot from the previous step. Do not restore the cluster state (your current cluster state is fine). Rename the indices as they get restored to `restored_logs_1`, `restored_logs_2` and `restored_logs_3`.

    Show answer

20. Verify that your `restored_logs_*` indices have been restored with all of their documents. Check that the `restored_logs_*` indices have the same number of documents as the original `logs_server*` indices, for example using the `_cat` API:

```
GET _cat/indices
```

Summary: In this lab, you learned how to define a snapshot lifecycle policy to back up and restore indices.

End of Lab 4.1

Lab 4.2: Upgrading a Cluster

Objective: In this lab, you will perform a rolling restart of your cluster.

1. Run the following in your Kibana **Console**. It will fail. Why?

```
POST _analyze
{
    "tokenizer": "icu_tokenizer",
    "text": ["星球大战是我最喜歡的電影"]
}
```

Show answer

2. On each of the nodes in your cluster, run the following command to install the ICU Analysis plugin (do so from a new Strigo terminal, ssh onto `server1`, `server2` and `server3` while your cluster is up and running):

```
./elasticsearch/bin/elasticsearch-plugin install analys
```

3. Installing a plugin requires the cluster to be restarted. Perform a rolling restart of your cluster by stopping and restarting `node1` first, followed by the other two nodes, one-at-a-time.

4. Run the following command again. It should work this time!

```
POST _analyze
{
    "tokenizer": "icu_tokenizer",
    "text": ["星球大战是我最喜歡的電影"]
}
```

Summary: In this lab, you performed a rolling restart of your cluster.

End of Lab 4.2

Lab 4.3: Topology Awareness

Objective: In this lab, you are going to configure shard allocation awareness along with forced awareness. **This entire lab should be considered as preparation for the certification exam.**

1. You currently have 3 nodes running. However, to better demonstrate shard awareness, you will add an extra node (`node4`). Suppose `node1` and `node3` are in `rack1` and that `node2` and `node4` (newly added) are in `rack2` . You are going

to see what happens with shard allocation when you set this configuration in your cluster and enable shard awareness.

2. Stop `node1` and `node3` and add the attribute `my_rack` set to `rack1` in their configuration files.

   Show answer

3. Start `node1` and `node3` again.

   Show answer

4. Stop `node2` and add the attribute `my_rack` set to `rack2` in its configuration file.

   Show answer

5. Start `node2` again.

   Show answer

6. In a new terminal tab, ssh to `server4` and edit the configuration file so it has the same configuration as `server2`, plus `node.master` set to `false`.

   Show answer

7. Start `node4`.

   Show answer

8. Verify your nodes are running and configured properly by running both of the following cat commands in your Kibana console:

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

9. Verify the shard allocation with the following command. Are there copies of the same shard (both primary and replica shard) allocated to the same rack?

```
GET _cat/shards?h=index,shard,prirep,node&s=index,prire
```

Show answer

10. Use `transient` settings to configure your cluster so that it uses your `my_rack` attribute to implement shard allocation awareness.

Show answer

11. Run the following `cat` command and view your shard allocation carefully:

```
GET _cat/shards?h=index,shard,prirep,node,state&s=index
```

Now, copies of the same shard should be on different racks. Following the `logs_server3` example above, you should see shards that are being relocated like the following:

```
logs_server3         0 p node1 -> 172.18.0.5 dOKFSlHqTF
```

In this case, the primary shard `0` is being relocated from `node1` to `node4`.

12. You currently have 4 nodes running and just saw how shard awareness works. Now we want you to understand forced awareness. Stop `node2` and `node4` which are on `rack2`.

13. Run the following cat command in your Kibana console to see all shards being allocated to `rack1`.

```
GET _cat/shards?h=index,shard,prirep,node,state&s=index
```

This may take a minute or two so be sure to run the command until you see the desired results.

14. Once you see all shards allocated to `rack1` go back to the terminal and restart `node2` and `node4`.

15. To avoid overloading your nodes with all of the newly allocated replicas, you will need to configure forced awareness for `my_rack`. Use the `persistent` settings to configure your cluster so that is uses your `my_rack` attribute to implement the forced awareness for the two zones you want to balance. In this case, `rack1` and `rack2`.

Show answer

16. To test the forced awareness configuration, bring down `rack2` by stopping `node2` and `node4`. Wait a minute and run the

following command in your Kibana console:

```
GET _cat/shards?h=index,shard,prirep,node,state&s=index
```

You should now see only primary shards allocated to `rack1` (`node1` and `node3`).

17. Bring back `rack2` by restarting `node2` and `node4` .

Summary: In this lab, you configured shard allocation and forced awareness.

End of Lab 4.3

Lab 4.4: Multi Cluster Setups

Objective: In this lab, you will configure and test cross cluster search as well as cross cluster replication. **This entire lab should be considered as preparation for the certification exam.**

1. First, you are going to implement cross cluster search by configuring the `server6` node to be in a different cluster.

```
ssh server6
```

2. Modify the `elasticsearch.yml` in `server6` to look like the following.

```
cluster.name: my_cluster_2
node.name: ${NODENAME}
network.host: _site_
discovery.seed_hosts: ["server6"]
cluster.initial_master_nodes: ["node6"]
```

   Save your changes and startup Elasticsearch on `server6`.

3. Use the Kibana Console to create a `comments` index with the following mappings in your original cluster, `my_cluster`.

```
PUT comments
{
  "mappings" : {
    "properties" : {
      "comment" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
          }
        }
      },
      "movie" : {
        "type" : "text",
        "fields" : {
```

```
          "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
          }
        }
      },
      "rating" : {
        "type" : "long"
      },
      "username" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
          }
        }
      }
    }
  }
}
```

4. Run the following commands to index two documents into your current cluster.

```
PUT comments/_doc/7
{"username": "ricardo", "movie": "Star Trek IV: The Voy
```

```
PUT comments/_doc/8
{"username": "sara", "movie": "Wonder Woman","comment"
```

5. Now, in a terminal from any of the servers, run the following curl commands to index some documents into your new cluster on `node6`. Notice the index is named `comments` on both clusters.

```
curl -XPUT "http://server6:9200/comments/_doc/1?pretty"
{"username": "paolo", "movie": "Star Trek IV: The Voyag

curl -XPUT "http://server6:9200/comments/_doc/2?pretty"
{"username": "harrison", "movie": "Blade Runner","comme
```

6. Configure `my_cluster` so that it can run cross cluster searches onto `my_cluster_2`.

Show answer

7. Run a search on `my_cluster` that hits all documents in both `comments` indices (the `comments` index on `my_cluster` *and* the `comments` index on `my_cluster_2`). You should get 4 hits.

Show answer

8. Next, you are going to implement cross cluster replication. First, start a trial license on `my_cluster`.

Show answer

9. Now, use the following `curl` command to start a trial license on `my_cluster_2`.

```
curl -X POST "http://server6:9200/_license/start_trial
```

10. Create a leader index called `my_replicated_blogs` in `my_cluster`.

```
PUT my_replicated_blogs
{
  "settings": {
    "index": {
      "number_of_shards": 1,
      "number_of_replicas": 0,
      "soft_deletes": {
        "enabled": true
      }
    }
  },
  "mappings" : {
    "properties" : {
      "author" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
          }
        }
      },
      "category" : {
        "type" : "text",
```

```
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
      }
    },
    "content" : {
      "type" : "text",
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
      }
    },
    "locales" : {
      "type" : "text",
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
      }
    },
    "publish_date" : {
      "type" : "date"
    },
    "seo_title" : {
      "type" : "text",
```

```
        "fields" : {
          "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
          }
        }
      },
      "some_other_field" : {
        "type" : "text"
      },
      "title" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
          }
        }
      },
      "url" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
          }
        }
      }
    }
```

```
    }
  }
```

11. Use `curl` to connect `my_cluster_2` to `my_cluster`, so it can pull changes to `my_replicated_blogs` and implement cross cluster replication.

    Show answer

12. Use `curl` to create a follower index called `my_replicated_blogs`, which references to `my_cluster` and its leader index `my_replicated_blogs`.

    Show answer

13. Run the following `curl` command and check that there are no blogs yet. Why?

```
curl -X GET "http://server6:9200/my_replicated_blogs/_
```

    Show answer

14. Reindex the blogs from `blogs` index into `my_replicated_blogs` index in `my_cluster`.

    Show answer

15. Run the same `curl` command again and you should see now that the `1594` have been replicated from `my_cluster` into `my_cluster_2`.

Summary: In this lab, you implemented and tested cross cluster search as well as cross cluster replication.

End of Lab 4.4

Lab 5.1: Controlling Shard Allocation

Objective: In this lab, you will configure a 5-node cluster and implement a hot/warm architecture with shard filtering. **This entire lab should be considered as preparation for the certification exam.**

If you haven't finished the previous labs or had any issues with your cluster, you can reload the labs for this module by running the following command:

```
./load_lab.sh 5
```

By doing this, you will lose any previous work that is not required for this lab state and the Kibana and Elasticsearch processes will be running in the background. If you need to kill these processes you can use their process ID that has been saved into `elasticsearch_pid` and `kibana_pid` files. To stop Elasticsearch run `kill $(cat elasticsearch_pid)` To stop Kibana run `kill $(cat kibana_pid)`

1. You currently have a 4-node cluster running on `server1`, `server2`, `server3`, and `server4`. You are going to deploy a 5-node cluster by adding one more node on `server5`. Besides configuring dedicates nodes you will also control the index allocation.

2. Stop all four instances of Elasticsearch, but stay SSH'd onto each server.

3. Open a new tab and SSH onto `server5`.

```
ssh server5
```

4. Now, you are going to implement a hot/warm architecture in your 5-node cluster.

5. Configure your five nodes so that they include the dedicated `roles` and `my_temp` tag names as described in the following table.

Name	Server	my_rack	my_temp	Roles
node1	server1	rack1	hot	data, ingest and master-eligible
node2	server2	rack2	hot	data, ingest and master-eligible
node3	server3	rack1	warm	dedicated data

Name	Server	my_rack	my_temp	Roles
node4	server4	rack2	warm	dedicated data
node5	server5	none	none	dedicated master-eligible voting-only

Note: Make sure you also configure `path.repo: /shared_folder/my_repo` for `node5`.

Show answer

6. Start all nodes and verify they are running and configured properly by running both of the following `_cat` commands in your Kibana console:

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

7. Define a new index named `logs_server4`, if you haven't already.

```
PUT logs_server4
```

8. Configure the `logs_server1`, `logs_server2`, and `logs_server3` indices so that their shards are allocated only to warm nodes, and configure `logs_server4` to be allocated only to hot nodes.

Show answer

9. Verify your shard filtering is working correctly by running the following `_cat` command:

```
GET _cat/shards/logs_server*?v&h=index,shard,prirep,st
```

The shards from `logs_server1`, `logs_server2`, `logs_server3` should be allocated in `warm` nodes, while the shards from `logs_server4` should be allocated in `hot` nodes. You should see a shard allocation similar to the one below:

```
index         shard prirep state   node
logs_server1 0     p      STARTED node4
logs_server1 0     r      STARTED node3
logs_server2 0     p      STARTED node4
logs_server2 0     r      STARTED node3
logs_server3 0     p      STARTED node3
logs_server3 0     r      STARTED node4
logs_server4 0     p      STARTED node2
logs_server4 0     r      STARTED node1
```

Summary: In this lab, you configured a 5-node cluster and implemented a hot/warm architecture with shard filtering.

End of Lab 5.1

Lab 5.2: Index Management

Objective: In this lab, you will learn how to rollover an alias to a new index when the existing index meets a condition you specified.

1. Create a new index named `logs-000001` that meets the following criteria:
   - it has 4 primary shards and 1 replica shard;
   - it uses shard filtering to allocate shards to `hot` nodes;
   - it has an alias named `logs` with `is_write_index` set to `true`.

   Show answer

2. Run the following `_bulk` command, which indexes a few simple log documents into the `logs` alias:

```
POST logs/_bulk
{ "index" : { "_id" : "1"}}
{ "level" : "INFO", "message" : "recovered [20] indices
{ "index" : { "_id" : "2"}}
{ "level" : "WARN", "message" : "received shard failed
{ "index" : { "_id" : "3"}}
{ "level" : "INFO", "message" : "Cluster health status
```

3. Run the following query to view the three documents.

```
GET logs/_search
```

4. Use the following conditions to rollover `logs` into a new index:
    - the maximum age of the index is one day;
    - the maximum number of documents the index should contain is two;
    - the maximum estimated sized of the primary shard of the index is 1 gigabyte.

    Like the old index, the new one should also have 4 primary shards and 1 replica shard and use shard filtering to allocate shards to `hot` nodes.

    Show answer

5. Notice that PUT, POST & DELETE requests for indexing logs do not need to know the name of the index that you are indexing to. You can simply index new log events to `logs` because of the clever use of aliases. Run the following bulk command, which indexes a couple of log events into the new index:

```
POST logs/_bulk
{ "index" : { "_id" : "4"}}
{ "level" : "INFO", "message" : "[node2] started", "dat
{ "index" : { "_id" : "5"}}
{ "level" : "WARN", "message" : "not enough master node
```

6. Run the following query to verify that the two log events ended up in the correct index. You should get 2 hits (the two docs you just indexed):

```
GET logs-000002/_search
```

7. Notice that searching `logs` retrieves all 5 log events:

```
GET logs/_search
```

8. Here is an interesting note about aliases. What do you think happens if you delete the `logs` alias, which currently points to two indices? Run the following command and see what happens:

```
DELETE logs
```

Show answer

9. Prepare `logs-000001` for shrinking by making it read-only and allocating all the shards to `warm` nodes.

Show answer

10. Shrink `logs-000001` into `shrink-logs-000001` with a single primary shard.

Show answer

11. Suppose you are done writing into `shrink-logs-000001`, so force merge all of its segments into a single segment.

    Show answer

12. In a single request, remove the alias `logs` from `logs-000001` and instead assign `logs` to `shrink-logs-000001`.

    Show answer

13. Delete `logs-000001`. Yes, it is fine deleting `logs-000001` since now you are going to use `shrink-logs-000001` instead.

    Show answer

14. Notice that searching `logs` still retrieves all 5 log events:

```
GET logs/_search
```

Summary: In this lab, you learned how to rollover an alias to a new index when the existing index meets a condition you provided.

End of Lab 5.2

Lab 5.3: Index Lifecycle Management

Objective: In this lab, you will explore how to setup index lifecycle management.

1. You are going to use ILM to automatically implement the same hot/warm architecture you implemented in the previous lab along with rollover. First, use Kibana to define an index lifecycle policy named `logs-hot-warm-policy` that implements the following hot and warm phases:
    - **hot phase**
        - it enables rollover;
        - its maximum index size is 1 gigabyte;
        - its maximum documents is 2;
        - its maximum age is 1 day.
    - **warm phase**
        - it uses `my_temp` node attributes to move shards into warm phase on rollover;
        - it shrinks the index into a new index with a single primary shard;
        - it reduces the number of segments to 1.

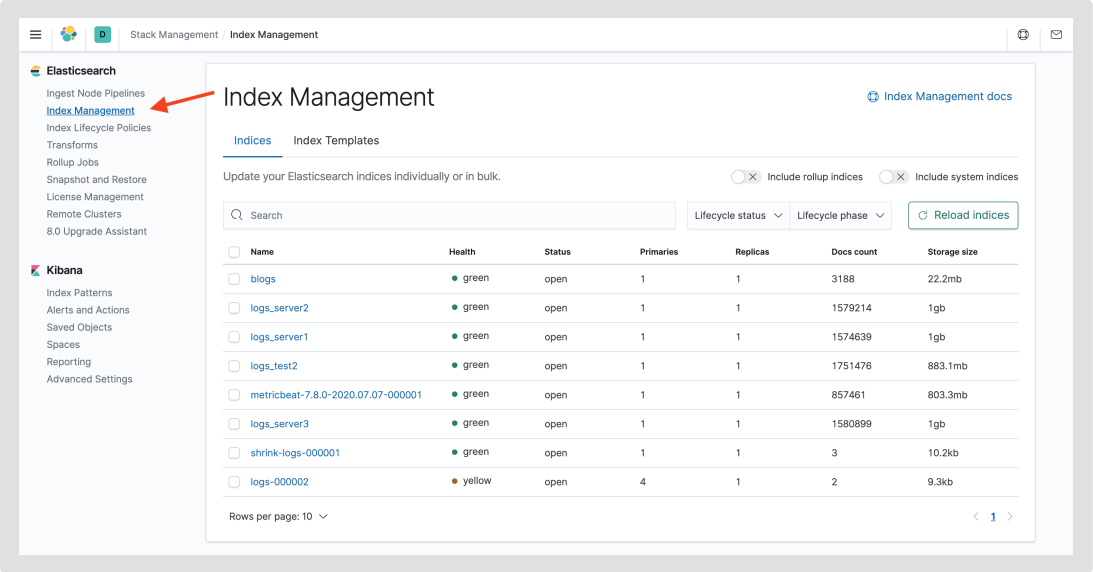
    Show answer

2. To apply this policy to all future log indexes that are going to be created, define an index template named `logs-template` that uses your `logs-hot-warm-policy` and also applies the following settings to new indices that match the pattern `logs-*`:

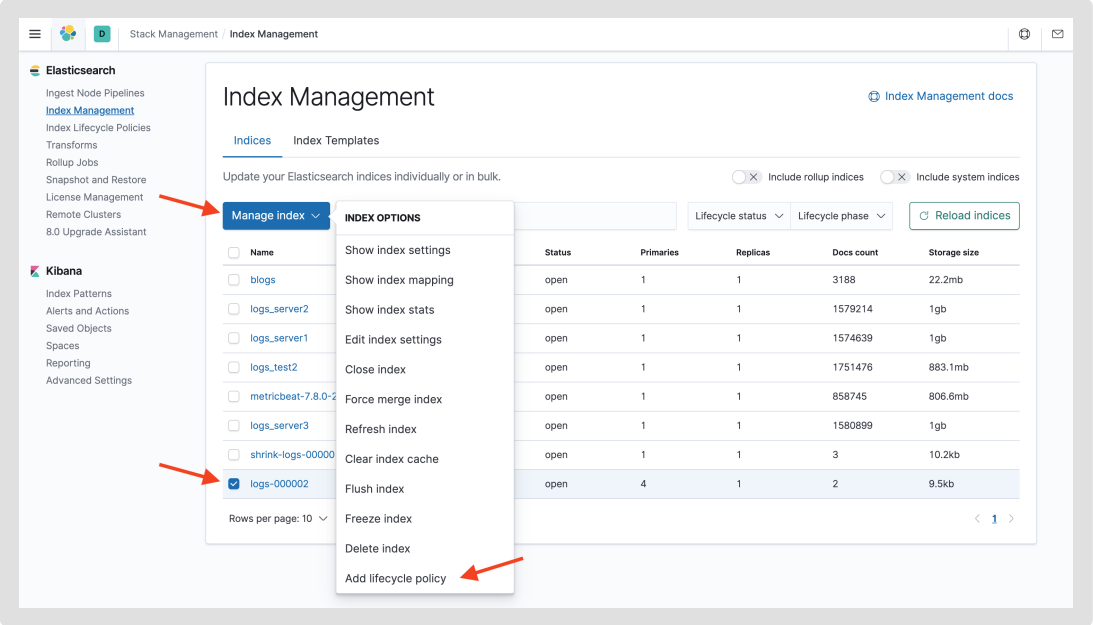
- the number of primary shards is 4;
- the number of replica shards is 1;
- shards are allocated into `hot` nodes;
- the rollover alias is `logs`;

Show answer

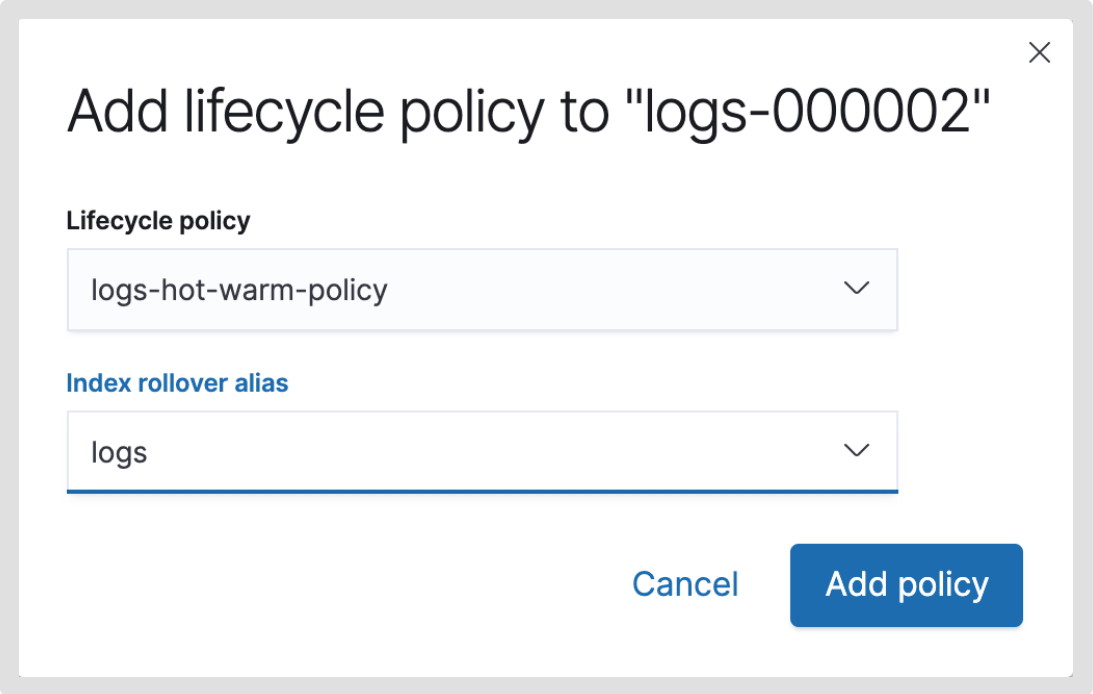
3. You also need to apply the policy to the existing `logs-000002` index. You can again use Kibana for that. First, go back to **Stack Management** via the Kibana navigation menu, and this time select **Index Management** (under **Elasticsearch**).



Check the box of the `logs-000002` index, click the **Manage index** button, and select **Add lifecycle policy**.



Select the `logs-hot-warm-policy` Lifecycle policy and the `logs` Index rollover alias. Finally, click the **Add policy** button.



4. Use the following command to check the lifecycle state of `logs`:

```
GET logs/_ilm/explain
```

You will get a response similar to the one below. Note that `logs` is still aliasing `logs-000002`, though the index already have matched the `max_docs` condition to rollover. This is happening because the conditions are checked periodically and the default lifecycle poll interval is 10 minutes. So the index might grow slightly beyond the specified threshold until the conditions are checked again.


```
{
  "indices" : {
    "shrink-logs-000001" : {
      "index" : "shrink-logs-000001",
      "managed" : false
    },
    "logs-000002" : {
      "index" : "logs-000002",
      "managed" : true,
      "policy" : "logs-hot-warm-policy",
      "lifecycle_date_millis" : 1580486859628,
      "phase" : "hot",
      "phase_time_millis" : 1580492899136,
      "action" : "unfollow",
      "action_time_millis" : 1580492899232,
      "step" : "wait-for-follow-shard-tasks",
      "step_time_millis" : 1580492899265,
      "phase_execution" : {
        "policy" : "logs-hot-warm-policy",
        "phase_definition" : {
          "min_age" : "0ms",
          "actions" : {
            "rollover" : {
```

```
            "max_size" : "1gb",
            "max_age" : "1d",
            "max_docs" : 2
          },
          "set_priority" : {
            "priority" : 100
          }
        }
      },
      "version" : 1,
      "modified_date_in_millis" : 1580492736342
    }
  }
}
```

5. 10 minutes is a long wait for this lab, so use a transient setting to change the lifecycle poll interval to 5 seconds as follows:

```
PUT _cluster/settings
{
    "transient": {
      "indices.lifecycle.poll_interval": "5s"
    }
}
```

6. Check the lifecycle state of `logs` again. What happened?

Show answer

7. Now, run the following `PUT` command and notice that while you send it to `logs`, the document eventually gets written into the new index `logs-000003`:

```
PUT logs/_doc/6
{
  "level": "INFO",
  "message": "[node3] started",
  "date": "2018-07-06"
}
```

8. Notice that searching `logs` retrieves all 6 log events because you configured `logs-template` to also include an alias to the new index, and ILM also updates the aliases from old indices to new shrunken indices:

```
GET logs/_search
```

9. Finally, run the following `cat` command a view your shard allocation carefully:

```
GET _cat/shards/*logs-*?v&h=index,shard,prirep,state,no
```

Note that `hot` data is stored in either `node1` or `node2`, while `warm` data is stored in either `node3` or `node4`. This means that you successfully set your ILM with hot/warm architecture.

```
index              shard prirep state   node
logs-000003        0     p      STARTED node1
```

```
logs-000003          0     r      STARTED node2
logs-000003          1     p      STARTED node2
logs-000003          1     r      STARTED node1
logs-000003          2     p      STARTED node1
logs-000003          2     r      STARTED node2
logs-000003          3     p      STARTED node2
logs-000003          3     r      STARTED node1
shrink-logs-000001 0     p      STARTED node3
shrink-logs-000001 0     r      STARTED node4
shrink-logs-000002 0     p      STARTED node3
shrink-logs-000002 0     r      STARTED node4
```

Summary: In this lab, you explored how to setup index lifecycle management.

End of Lab 5.3

Lab 6.1: Challenges of Distributed Operations

Objective: In this lab, you will familiarize yourself with `dfs_query_then_fetch` search type, deep pagination with `search_after`, and how to check the precision of aggregations.

If you haven't finished the previous lab or had any issues with your cluster, you can reload the labs for this module by running the

following command:

```
./load_lab.sh 6
```

By doing this, you will lose any previous work that is not required for this lab state and the Kibana and Elasticsearch processes will be running in the background. If you need to kill these processes you can use their process ID that has been saved into `elasticsearch_pid` and `kibana_pid` files. To stop Elasticsearch run `kill $(cat elasticsearch_pid)` To stop Kibana run `kill $(cat kibana_pid)`

1. Elasticsearch uses shards to distribute data and scale. However, this creates inaccuracy to the score calculation. Even though in many cases this is not an issue, in some cases it might be. Run the following query and pay attention to the score.

   ```
   GET logs_server*/_search
   {
     "_source": "originalUrl",
     "query": {
       "match": {
         "originalUrl": "search"
       }
     }
   }
   ```

   Notice that variances occur in the scores for the hits related to `/blog/guide/reference/api/search/search-type`, though all of them refer to the same `originalUrl`.

2. Now, run the same query with the `dfs_query_then_fetch` search type and pay attention to the score. Do you see any differences in the score? Why?

   Show answer

3. Suppose that you want to implement deep pagination with `search_after`. Test how to implement it against the `blogs` index by getting the first 3 documents from a `match_phrase` query for "**elastic stack**" on the `content` field, and then use the result of first page's last hit for getting the second page, which is the next set of 3 documents.

   Show answer

4. The aggregation below returns how many requests were received from each of the 6 values of `status_code` for each week of log requests. We need to be sure that it is delivering a suitable level of accuracy, so enable `show_term_doc_count_error` in the `terms` query of the `status_code` field. How precise are the buckets?

```
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"
          }
        }
      }
    }
  }
}
```

Show answer

5. The aggregation below shows what are the top 20 cities that access our blogs. Is this city aggregation exact?

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

Show answer

Summary: In this lab, you have familiarized yourself with `dfs_query_then_fetch` search type, deep pagination with `search_after`, and how to check the precision of aggregations.

End of Lab 6.1

Lab 6.2: Aliases and Templates

Objective: In this lab, you will learn how to define aliases, as well as how to use index and search templates.

1. **EXAM PREP**: Your current cluster has four indices containing the web access logs. Define an alias named `access_logs` that points to all four `logs_server*` indices.

   Show answer

2. Run the following query on your `access_logs` alias to verify it points to all four indices. You should get 1,751,476 hits:

   ```
   GET access_logs/_count
   ```

3. Try to add a new document in the `access_logs` alias.

```
PUT access_logs/_doc/3
{
  "@timestamp": "2017-05-05T02:07:51.407Z",
  "host": "server1",
  "user_agent": "Amazon CloudFront",
  "originalUrl": "/blog/elasticsearch-5-4-0-released",
  "response_size": 49921,
  "input": {
    "type": "log"
  },
  "http_version": "1.1",
  "runtime_ms": 108,
  "method": "GET",
  "language": {
    "url": "/blog/elasticsearch-5-4-0-released",
    "code": "en-us"
  },
  "status_code": 200,
  "geoip": {
    "country_code3": "JP",
    "continent_code": "AS",
    "location": {
      "lon": 139.69,
      "lat": 35.69
    },
    "country_name": "Japan",
    "country_code2": "JP"
  },
  "level": "info"
}
```

What is the issue ?

Show answer

4. **EXAM PREP**: Configure `logs_server4` to be the write index in your current alias using the `is_write_index` parameter and try to index the document again.

   Show answer

5. **EXAM PREP**: Define an index template named `access_logs_template` of `order` 10, matching indices named `logs_server*` with the same mappings as your four current `logs_server*` indices.

   Show answer

6. Define a new index named `logs_server5` and verify the `access_logs_template` template was applied.

   Show answer

7. In the same request, remove `logs_server1` from the `access_logs` alias and update the same alias to write to `logs_server5` instead of `logs_server4`

   Show answer

8. Index the following document using the `access_logs` alias, assigning the `_id` to 1. Then GET the document in the `logs_server5` index to verify the alias worked successfully:

```
{
  "@timestamp": "2018-03-21T05:57:19.722Z",
  "originalUrl": "/blog/logstash-jdbc-input-plugin",
  "host": "server2",
  "response_size": 58754,
  "status_code": 200,
  "method": "GET",
  "runtime_ms": 143,
  "geoip": {
    "country_code2": "IN",
    "country_code3": "IN",
    "continent_code": "AS",
    "location": {
      "lon": 77.5833,
      "lat": 12.9833
    },
    "region_name": "Karnataka",
    "city_name": "Bengaluru",
    "country_name": "India"
  },
  "language": {
    "url": "/blog/logstash-jdbc-input-plugin",
    "code": "en-us"
  },
  "user_agent": "Amazon CloudFront",
  "http_version": "1.1",
  "level": "info"
}
```


Show answer

9. **EXAM PREP**: Now, you are going to create a search template for the following query, which finds the number of visitors to a blog on a specific day.

```
GET logs_server*/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "match": {
            "originalUrl.keyword": "/blog/elasticsearch
          }
        },
        {
          "range": {
            "@timestamp": {
              "gte": "2017-05-12",
              "lt": "2017-05-13"
            }
          }
        }
      ]
    }
  }
}
```

Make a search template for this query named `daily_hits` that uses the following parameters:

- `url` : to represent the value of `originalUrl.keyword`

- `start_date` : for the day we are searching for blogs
- `end_date` : for the upper end of our date range

Show answer

10. Test your `daily_hits` search template using the following values:

    - `url: "/blog/brewing-in-beats-postgresql-module-in-filebeat"`
    - `start_date: "2017-08-11"`
    - `end_date: "2017-08-12"`

    You should get 24 hits.

    Show answer

11. Modify `daily_hits` so that if the `end_date` parameter is not defined, then it is left off of the `range` query. In other words, if we only provide a `start_date`, then find all log events from that date onwards. Test your new search template using the same query as the previous step, but remove the `end_date` from the query. You should get 155 hits.

    Show answer

Summary: In this lab, you learned how to define aliases, as well as how to use index and search templates.

End of Lab 6.2

Lab 6.3: Controlling Dynamic Behaviors

Objective: In this lab, you will learn how to disable dynamic indexes, define dynamic templates, and control dynamic fields.

1. First, work on the dynamic index feature. Execute the following operation:

```
PUT dynamic_test/_doc/1
{
  "my_field": "A value"
}
```

   This request will go through and a new index will be created. Delete the index.

```
DELETE dynamic_test
```

   Let's make sure that we cannot create new indices without explicitly creating a new index.

2. Disable dynamic index creation.

   Show answer

3. Execute the following indexation:

```
PUT dynamic_test/_doc/1
{
  "my_field": "A value"
}
```

What is the result of the operation?

Show answer

4. Create the index `dynamic_test` with one field `my_field` of type text. And then execute the previous indexation. The indexation should succeed.

Show answer

5. Delete the index.

```
DELETE dynamic_test
```

6. Whitelist the pattern `dynamic_test` along with all the internal indices.

Show answer

7. Do the two following operations:

```
PUT dynamic/_doc/1
{
  "my_field": "A value"
```

```
}

PUT dynamic_test/_doc/1
{
"my_field": "A value"
}
```

What will be the result of the first operation? The second?

Show answer

8. **EXAM PREP**: Next, you are going to work with dynamic templates. Create a new index named `surveys` that satisfies the following criteria:
   - The `job_title` field is mapped as `text` and `keyword`
   - The `miles_travelled` field is mapped as an `integer_range`
   - Any field name that ends in `_rating` is dynamically mapped as an `integer`
   - Any string field that is not already in the mapping is dynamically mapped as `keyword` only, and is **not** indexed

Show answer

9. Put the following survey response in your `surveys` index. You will need to modify the `miles_travelled` field so that it is in the appropriate format for an `integer_range`:

```
{
  "job_title": "Elasticsearch Engineer",
  "course_rating": 9,
  "comments": "Great class. I want to get certified nov
  "miles_travelled": "0-25"
}
```

Show answer

10. Add the following document to `surveys`. What should happen to the mapping? View the `surveys` mapping to verify your dynamic templates are working as expected.

```
POST surveys/_doc
{
  "job_title": "Software Engineer",
  "labs_rating": 10,
  "city": "Berlin",
  "miles_travelled": {
    "gt": 50,
    "lte": 100
  }
}
```

Show answer

11. Finally, you are going to work with dynamic fields. Create a new index named `surveys2` that has only two fields in its mapping:
    - A field named `feedback` of type `text`

- A field named `course_rating` of type `integer`
- In addition, configure the mapping so that it will not be changed by unexpected fields and any documents with unexpected fields will fail to be indexed

Show answer

12. Review the following **PUT** command and predict what is the result. After that run the **PUT** command to see if you are correct.

```
PUT surveys2/_doc/1
{
  "food_rating": 10
}
```

Show answer

Summary: In this lab, you learned how to disable dynamic indexes, define dynamic templates, and control dynamic fields.

End of Lab 6.3

Lab 6.4: Common Causes of Poor Search Performance

Objective: In this lab, you will review some design choices that may lead to poor performance.

1. What can you change in the following query? Why?

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "content": "security"
          }
        },
        {
          "term": {
            "category.keyword": "Engineering"
          }
        },
        {
          "range": {
            "publish_date": {
              "gte": "2015-01-01T00:00:00.000Z",
              "lte": "2018-01-01T00:00:00.000Z"
            }
          }
        }
      ]
    }
```

```
    }
  }
```

Show answer

2. The following aggregation describes a relationship between run time and response size, but it can take long to execute if you have too many documents. Limit the scope of the aggregation to documents where `status_code` is `200`.

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "statistics": {
      "matrix_stats": {
        "fields": ["runtime_ms", "response_size"]
      }
    }
  }
}
```

Show answer

3. Write an aggregation to return the top ten countries (`geoip.country_name.keyword`) whose requests to resources mentioning `elasticsearch` in their url exceeded the `685` milliseconds run time SLA in place for requests against the `logs_server*` indices. (**TIP:** Use a match query on

`originalUrl` to define the query scope and a filter bucket aggregation on `runtime_ms` to limit the aggregation scope.)

Show answer

4. Now, you are interested in analyzing the top ten countries that requested blogs mentioning `elasticsearch`. (**TIP:** Change the filter bucket aggregation from the previous exercise to a sampler aggregation with `shard_size` 100 and compare the results.)

Show answer

5. What might be a problem in the following mapping?

```
PUT livetracking
{
  "mappings": {
    "properties": {
      "relation": {
        "type": "join",
        "relations": {
          "event": "athlete",
          "athlete": "livetracking"
        }
      }
    }
  }
}
```

Show answer

6. What might a problem in the following query?

```
GET blogs/_search
{
  "query": {
    "regexp": {
      "content": ".*search"
    }
  }
}
```

Show answer

7. Suppose you really need to search the end of your content tokes. How would you index the content of your blogs?

Show answer

Summary: In this lab, you reviewed some design choices that may lead to poor performance.

End of Lab 6.4