

PROFINIT

Elasticsearch

B0M33BDT

Valerii Ulitin

15/12/2021

About me

PROFINIT



elasticsearch



kibana



logstash



Ing. Valerii Ulitin

Consultant



IP[y]: IPython
Interactive Computing



spaCy

✉ valerii.ulitin@profinit.eu

 [linkedin.com/in/valerii-ulitin-005b7615a](https://www.linkedin.com/in/valerii-ulitin-005b7615a)

Content

1. General overview
2. Infrastructure
 1. clusters
 2. nodes
 3. indices
 4. shards
3. CRUD operations
4. Search, queries and aggregations
5. Mapping

General overview

Once upon a time...

- › As any good story begins: “Once upon a time...”
 - more precisely: in 1999, Doug Cutting created an open-source project called **Lucene**
- › Lucene is:
 - a **search engine library** entirely written in Java
 - a top-level Apache project, as of 2005
 - great for full-text search
- › But, Lucene is also:
 - a library that you have to incorporate into your application
 - challenging to use
 - not originally designed for scaling

The Birth of Elasticsearch

- › In 2004, Shay Banon developed a product called **Compass**
 - built on top of Lucene, Shay's goal was to have search integrated into Java applications as simply as possible
- › The need for **scalability** became a top priority
- › In 2010, Shay completely rewrote Compass with two main objectives:
 - **distributed from the ground up in its design**
 - **easily used by any other programming language**
- › He called it Elasticsearch
 - ...and we all lived happily ever after!
- › Today Elasticsearch is the most popular enterprise search engine

What is Elasticsearch?

- › Elasticsearch is a **distributed** search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured
- › **Key features** of Elasticsearch:
 - Distributed
 - Scalable
 - Fast
 - Shipped with simple yet powerful REST API
 - Easily used by any other programming languages
 - Supports 34 text languages and provides analyzers for each

Are there others like Elasticsearch?

- › The closest one is the Amazon OpenSearch Service
- › **Formerly** known as Amazon Elasticsearch Service
- › Isn't it just Elasticsearch in disguise? Well... yes and no:
 - it is a fork of older version of Elasticsearch (7.10.2), when it still was under an open source license (Apache 2.0)
 - because it is a fork it shares base functionality with Elasticsearch but with every update it diverges further away from it
- › You may consider Amazon **OpenSearch** service if:
 - you already have other services running in AWS
 - you are true open source sympathizer and would like to contribute one day

How Elasticsearch looks like

PROFINIT

The screenshot displays the Elasticsearch Kibana interface. At the top, there is a navigation bar with a hamburger menu, a 'Discover' dropdown, and buttons for 'Options', 'New', 'Save', 'Open', 'Share', and 'Inspect'. Below this is a search bar containing 'jeopardy' and a date range filter from 'Sep 10, 1984 @ 00:00:00.00' to 'Jan 27, 2012 @ 00:00:00.00'. A 'Refresh' button is located to the right of the search bar.

On the left side, there is a sidebar with a search field for field names, a 'Filter by type' dropdown set to '0', and a list of 'Available fields' (12 total). Under 'Popular', the 'Category' field is selected. Other visible fields include '_id', '_index', '_score', '_type', '@timestamp', 'Air Date', 'Answer', 'Question', 'Round', 'Show Number', and 'Value'.

The main area shows a bar chart titled '216,930 hits' for the date range 'Sep 10, 1984 @ 00:00:00.000 - Jan 27, 2012 @ 00:00:00.000'. The chart has a y-axis labeled 'Count' ranging from 0 to 1,200 and an x-axis labeled '@timestamp per 30 days' with dates from 1986-01-01 to 2012-01-01. A 'Hide chart' button is in the top right of the chart area.

Below the chart, a table displays document results. The table has two columns: 'Time' and 'Document'. Three document entries are visible, all dated 'Jan 27, 2012 @ 00:00:00.000':

Time	Document
> Jan 27, 2012 @ 00:00:00.000	<pre>@timestamp: Jan 27, 2012 @ 00:00:00.000 Air Date: Jan 27, 2012 @ 01:00:00.000 Answer: Miami Category: VISITING THE CITY Category.keyword: VISITING THE CITY Question: Experience Little Havana along Calle Ocho, then get sprayed at the Seaquarium Round: Jeopardy! Show Number: 6,300 Value: \$200 _id: uALsmn0BLArv6j_yck3M _index: jeopardy _score: - _type: _doc</pre>
> Jan 27, 2012 @ 00:00:00.000	<pre>@timestamp: Jan 27, 2012 @ 00:00:00.000 Air Date: Jan 27, 2012 @ 01:00:00.000 Answer: cargo pants Category: PANTS Category.keyword: PANTS Question: A synonym for freight, or pants with large bellows pockets on the sides & 2 extra-large patch pockets in front Round: Jeopardy! Show Number: 6,300 Value: \$200 _id: uQLsmn0BLArv6j_yck3M _index: jeopardy _score: - _type: _doc</pre>
> Jan 27, 2012 @ 00:00:00.000	<pre>@timestamp: Jan 27, 2012 @ 00:00:00.000 Air Date: Jan 27, 2012 @ 01:00:00.000 Answer: Elliott Category: CHILD ACTORS Category.keyword: CHILD ACTORS Question: Henry Thomas, who played this role in "E.T.", turned 40 in 2011 Round: Jeopardy! Show Number: 6,300 Value: \$200 _id: ual_smn0RRI Arv6j_vcK3M _index: jeopardy</pre>

How Elasticsearch looks like #2

Jeopardy

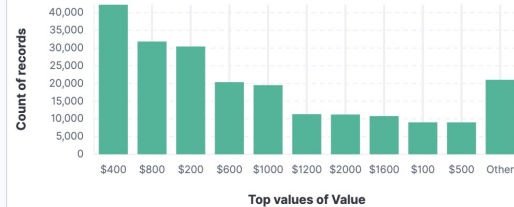
1-50 of 216930 < >

Time	Document
> Jan 27, 2012 @ 00:00:00.000	@timestamp: Jan 27, 2012 @ 00:00:00.000 Air Date: Jan 27, 2012 @ 01:00:00.000 Answer: Miami Category: VISITING THE CITY Category.keyword: VISITING THE CITY Question: Experience Little Havana along Calle Ocho, then get sprayed at the Seaquarium Round: Jeopardy! Show Number: 6,300 Value: \$200 _id: uALsmnOBLArv6j_ycK3M _index: jeopardy _score: - _type: _doc
> Jan 27, 2012 @ 00:00:00.000	@timestamp: Jan 27, 2012 @ 00:00:00.000 Air Date: Jan 27, 2012 @ 01:00:00.000 Answer: cargo pants Category: PANTS Category.keyword: PANTS Question: A synonym for freight, or pants with large bellows pockets on the sides & 2 extra-large patch pockets in front Round: Jeopardy! Show Number: 6,300 Value: \$200 _id: uQLsmnOBLArv6j_ycK3M _index: jeopardy _score: - _type: _doc
> Jan 27, 2012 @ 00:00:00.000	@timestamp: Jan 27, 2012 @ 00:00:00.000 Air Date: Jan 27, 2012 @ 01:00:00.000 Answer: Elliott Category: CHILD ACTORS Category.keyword: CHILD ACTORS Question: Henry Thomas, who played this role in "E.T.", turned 40 in 2011 Round: Jeopardy! Show Number: 6,300 Value: \$200 _id: uGLsmnOBLArv6j_ycK3M _index: jeopardy _score: - _type: _doc
> Jan 27, 2012 @ 00:00:00.000	@timestamp: Jan 27, 2012 @ 00:00:00.000 Air Date: Jan 27, 2012 @ 01:00:00.000 Answer: Central Park West Category: STUPID ANSWERS Category.keyword: STUPID ANSWERS Question: On the west, Central Park is bounded by this street Round: Jeopardy! Show Number: 6,300 Value: \$200 _id: uWLsmnOBLArv6j_ycK3M _index: jeopardy _score: - _type: _doc
> Jan 27, 2012 @ 00:00:00.000	@timestamp: Jan 27, 2012 @ 00:00:00.000 Air Date: Jan 27, 2012 @ 01:00:00.000 Answer: photosynthesis Category: LESSER-KNOWN SCIENTISTS Category.keyword: LESSER-KNOWN SCIENTISTS Question: In 1779 Dutch scientist Jan Ingenhousz published his discovery of this process in green plants Round: Jeopardy! Show Number: 6,300 Value: \$200 _id: vALsmnOBLArv6j_ycK3M _index: jeopardy _score: - _type: _doc
> Jan 27, 2012 @ 00:00:00.000	@timestamp: Jan 27, 2012 @ 00:00:00.000 Air Date: Jan 27, 2012 @ 01:00:00.000 Answer: He Category: THE TRUTH LIES THEREIN Category.keyword: THE TRUTH LIES THEREIN Question: Symbol for the second-lightest element Round: Jeopardy! Show Number: 6,300 Value: \$200 _id: vQLsmnOBLArv6j_ycK3M _index: jeopardy _score: - _type: _doc

Exit full screen

216,930

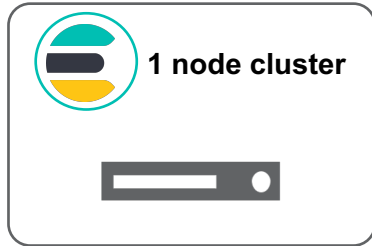
Count of records



WORD ORIGINS
POTPOURRI
SPORTS
SCIENCE
BEFORE & AFTER
LITERATURE HISTORY
AMERICAN HISTORY
WORLD HISTORY
COLLEGES & UNIVERSITIES

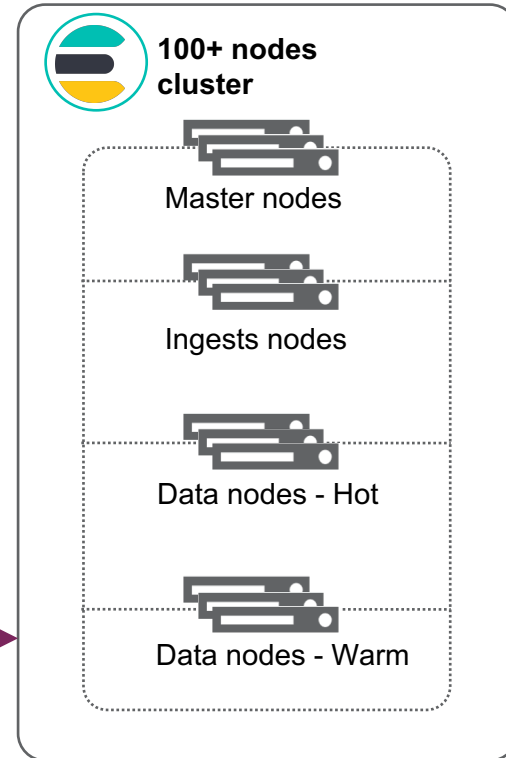
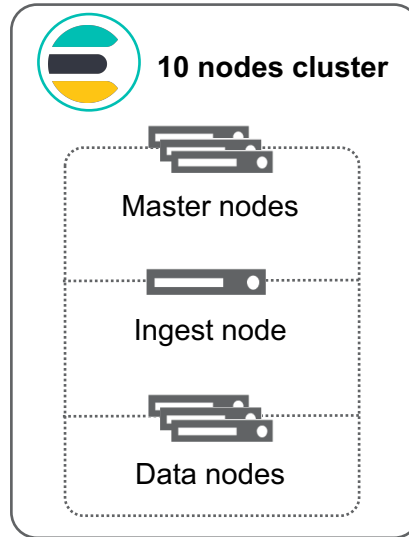
The most popular categories - Count

Distributed search



A **node** is an instance of Elasticsearch

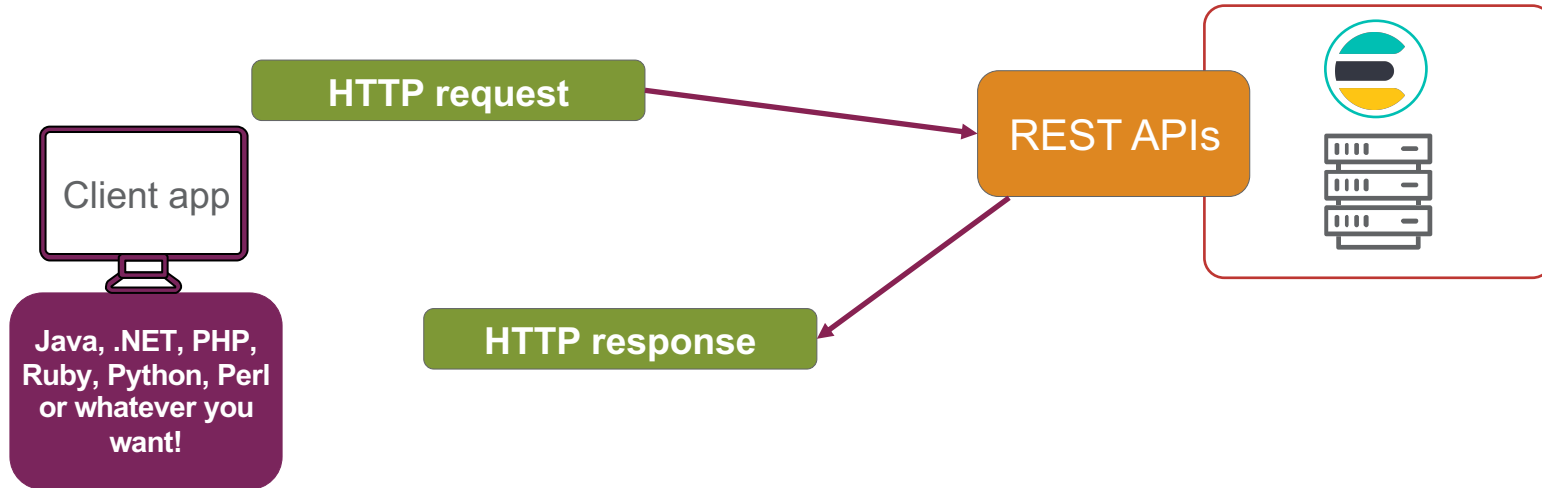
A **cluster** is a collection of Elasticsearch nodes



Your cluster can grow as your needs grow

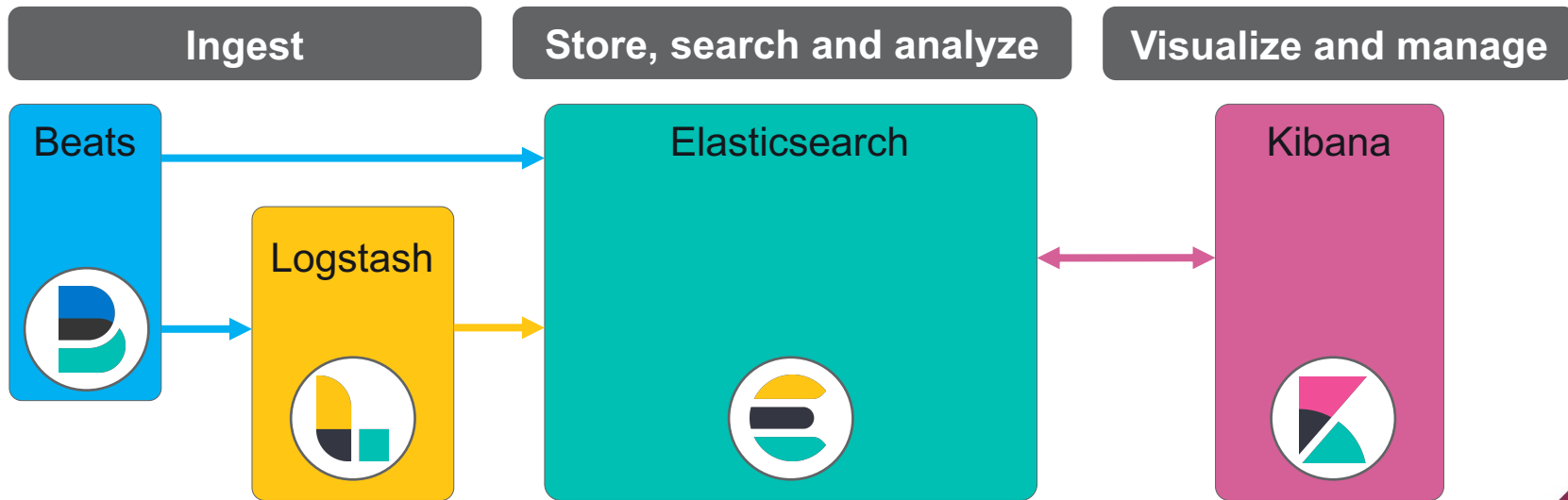
Easily used by other programming languages

- › Elasticsearch provides REST APIs for communicating with a cluster over HTTP(S)
 - allows client applications to be written in any language!



Elastic Stack (ELK)

- › Elasticsearch is a **distributed** search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured
 - reliably and securely take data from any source, in any format, as well as search, analyze, and visualize it in real time



Common use cases

PROFINIT

- › Application search
- › Website search
- › Enterprise search
- › Logging and log analytics
- › Infrastructure metrics and container monitoring
- › Application performance monitoring
- › Geospatial data analysis and visualization
- › Security analytics
- › Business analytics

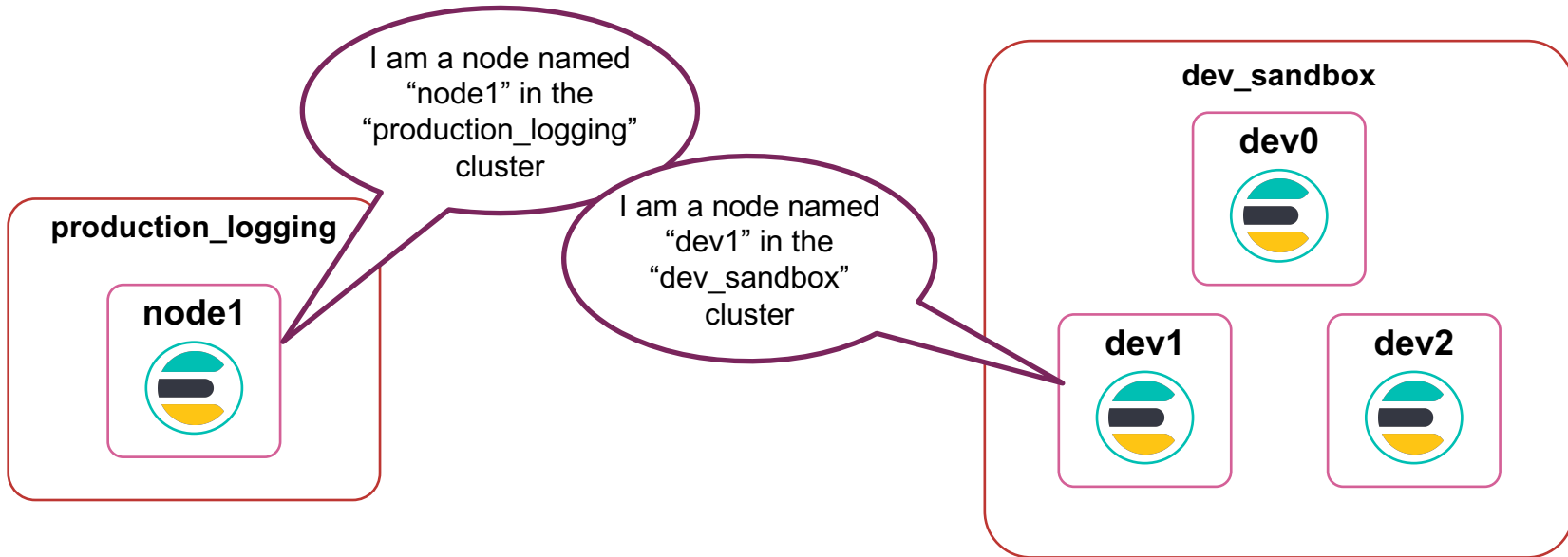


Infrastructure

- › A **cluster** is a collection of Elasticsearch nodes
- › A **node** is an instance of Elasticsearch
- › An Elasticsearch **index** is a collection of documents that are related to each other
 - an index is a **virtual namespace** that points to a number of shards
- › A **shard** is a worker unit that holds data and can be assigned to nodes
 - **primary** shards: the original shards of an index
 - **replica** shards: copies of the primary shard

Cluster

- › Every node belongs to a single **cluster**
- › A cluster is one or multiple nodes working together in a distributed manner



Node

- › A **node** is an instance of Elasticsearch
 - a Java process that runs in a JVM
- › A node is typically deployed 1-to-1 to a host

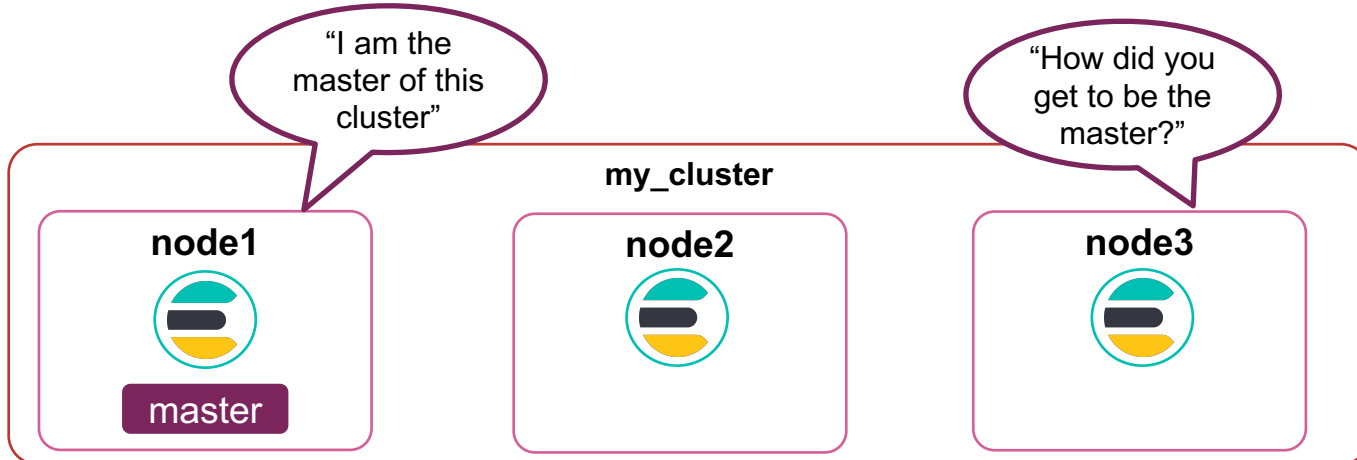


Nodes and their roles

- › There are several **roles** a node can have:
 - **master/master-eligible**
 - **data**
 - **ingest**
 - **machine learning**
- › Nodes can take on multiple roles at the same time
 - Or they can be **dedicated** nodes that only take on a single role

The Master Node

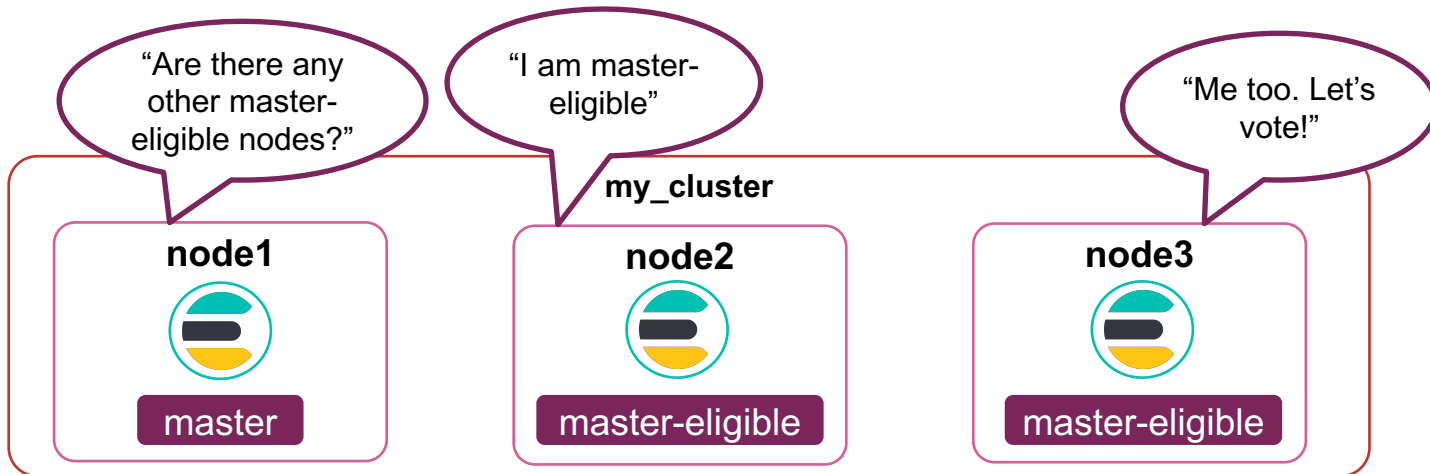
- › Every cluster has one node designated as the master
- › The master node is in charge of cluster-wide settings and changes, like:
 - creating, updating or deleting indices (incl. mappings and settings)
 - adding or removing nodes
 - allocating shards to nodes



The Master-eligible Node

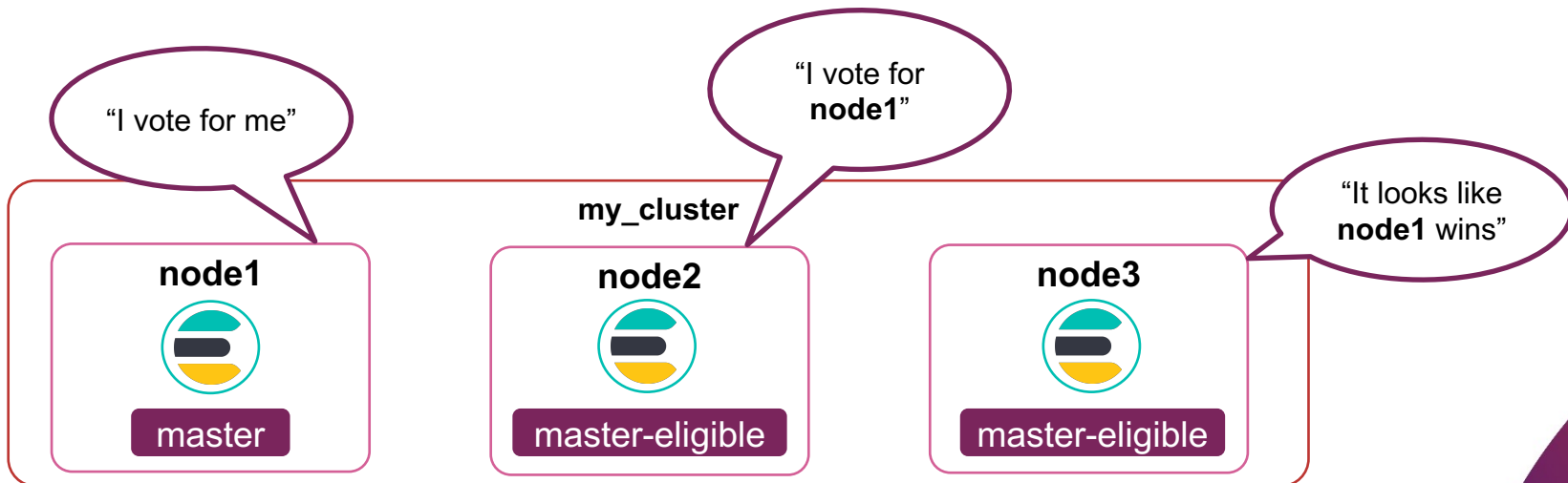
- › The master node is **elected** from the **master-eligible** nodes in the cluster
 - a node is master-eligible if **node.master*** is set to true (the default value)
 - only master-eligible nodes can **vote**

*the parameter is set in the `elasticsearch.yml`



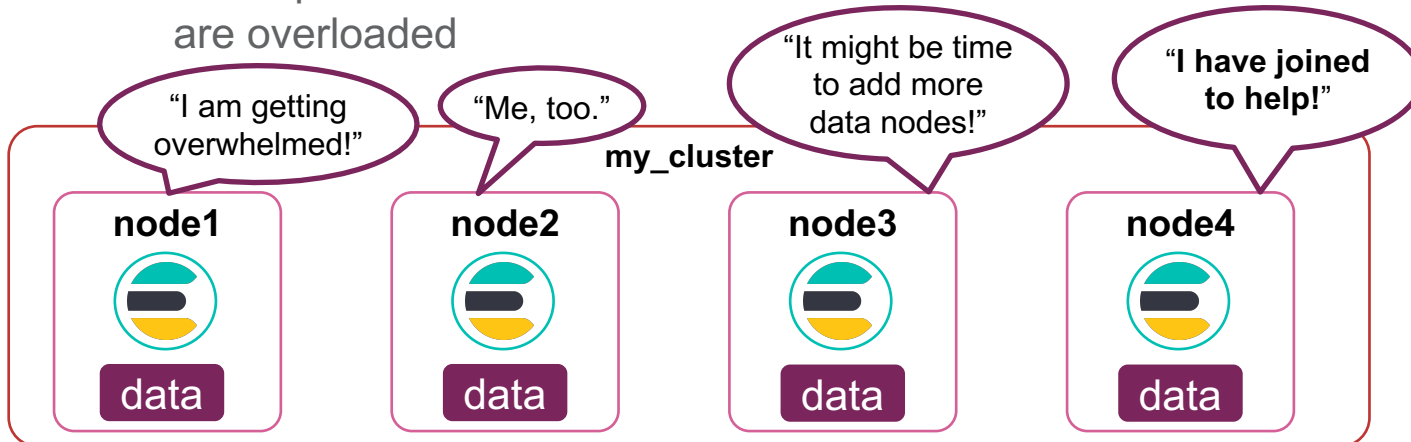
Master Elections

- › The number of votes to win the election is automatically handled by Elasticsearch to ensure a **quorum**
 - which is $\lfloor N/2 + 1 \rfloor$, where N is the number of master-eligible nodes
- › It is important to have a **quorum** to avoid a "split brain"



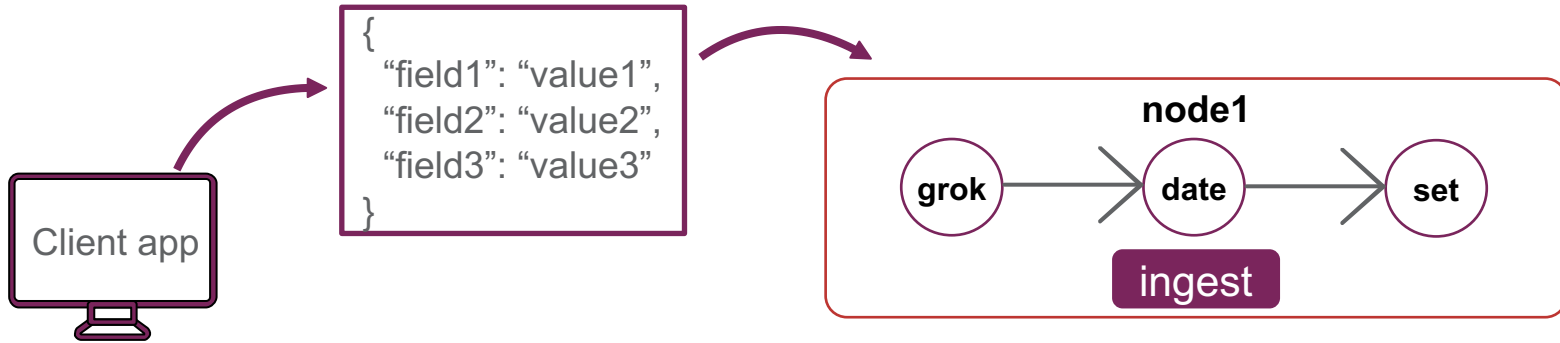
Data Nodes

- › Data nodes have two main features:
 - they hold the shards that contain the documents you have indexed
 - they execute data related operations like CRUD, search and aggregations
- › All nodes are data nodes by default
- › Data nodes are I/O, CPU, and memory-intensive
 - it is important to monitor these resources and add more data nodes if they are overloaded



Ingest Nodes

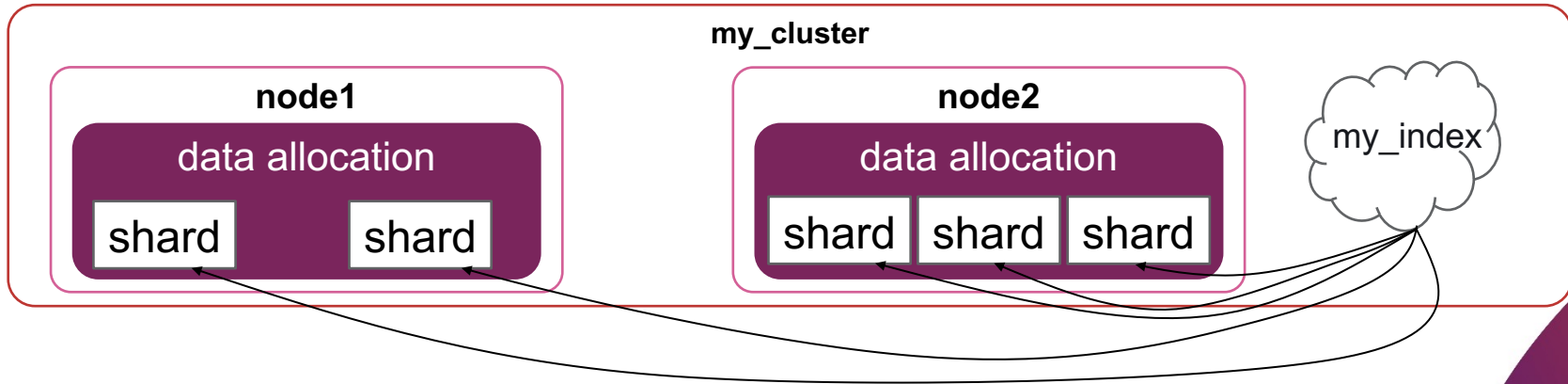
- › **Ingest nodes** provide the ability to
 - pre-process a document right before it gets indexed
- › All nodes are ingest nodes by default



When indexing a doc, you can specify a **pipeline**

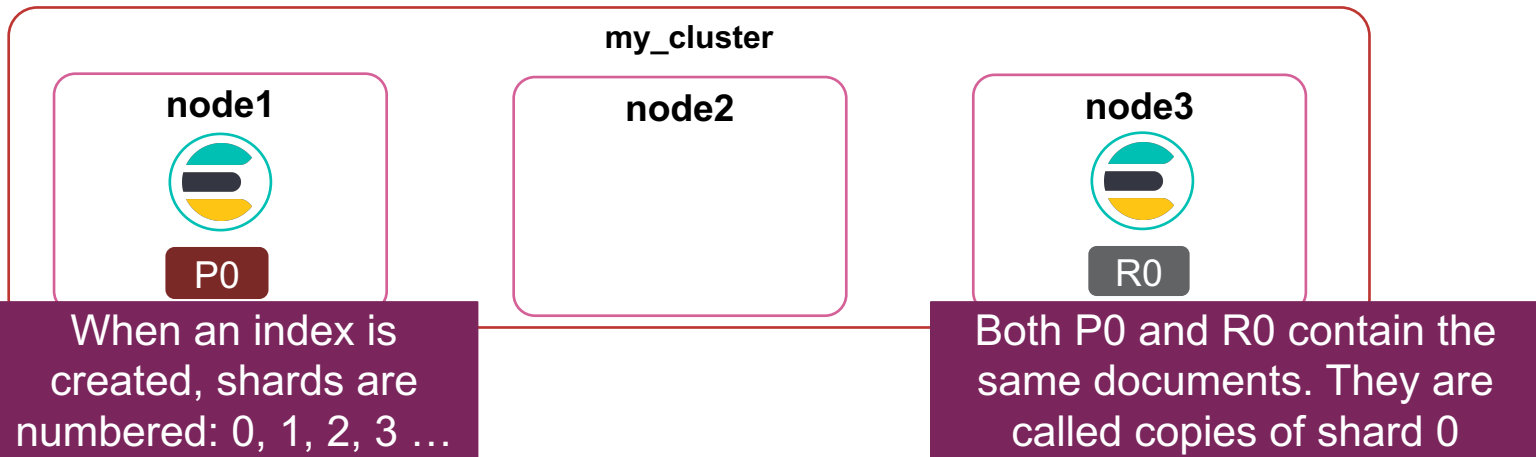
Shards and index relationship

- › A **shard** is a worker unit that holds data and can be assigned to nodes
- › An index is a **virtual namespace** which points to a number of shards
 - an index is “split” into **shards** before any documents are indexed



Primary vs. Replica

- › There are two types of shards
 - **primary** shards: the original shards of an index
 - **replica** shards: copies of the primary shard
- › Documents are replicated between a primary and its replicas
 - a primary and all replicas are guaranteed to be on different nodes



CRUD operations



dataset link: https://drive.google.com/file/d/0BwT5wj_P7BKXUI9tOUJWYzVvUjA/view?usp=sharing&resourcekey=0-uFrn8bQkUfSCvJlmtKGCdQ

source:

<https://www.jeopardy.com/jbuzz/behind-scenes/what-are-some-questions-about-jeopardy>

<https://g.cz/porad-riskuj-legendarni-televizni-soutez-ve-ktere-jste-mohli-vyhrat-pracku-i-felicii-v-kombiku/>

Documents must be JSON Objects

› Imagine records that are currently in a database table:

Show Number	Category	Air Date	Question	Answer	Value	Round
6298	20th CENTURY WORDS & PHRASES	2012-01-25	This word for a large self-service store that sells household goods as well as groceries hit the shelves in 1933	a supermarket	\$600	Jeopardy!
3746	SECOND-MOST POPULOUS CITIES	2000-12-11	Brno	Czech Republic	\$1000	Double Jeopardy!

› Each record needs to be converted to a **JSON** object:

```
{  
  "Show Number": 3746,  
  "Category": "SECOND-MOST POPULOUS CITIES",  
  "Air Date": "2000-12-11",  
  "Question": "Brno",  
  "Answer": "Czech Republic",  
  "Value": "$1000",  
  "Round": "Double Jeopardy!"  
}
```

JSON consists of *fields*...

...and *values*

Document Store

- › Elasticsearch is a distributed **document store**
 - it can store and return complex data structures that are represented as JSON objects
- › A **document** is a serialized JSON object that is stored in Elasticsearch under a unique ID

A JSON object...

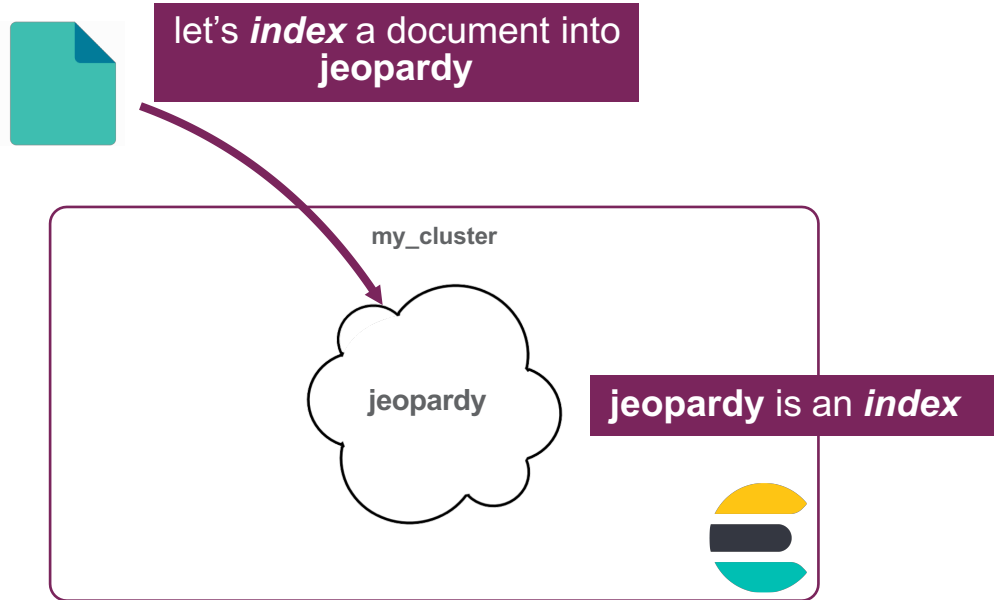
```
{  
  "Show Number": 3746,  
  "Category": "SECOND-MOST POPULOUS CITIES",  
  "Air Date": "2000-12-11",  
  "Question": "Brno",  
  "Answer": "Czech Republic",  
  "Value": "$1000",  
  "Round": "Double Jeopardy!"  
}
```

...is stored in Elasticsearch as a document



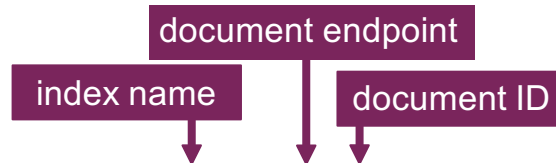
Documents are indexed into an index

- › In Elasticsearch, a document is *indexed* into an *index*
 - yes, **index** is used as both a noun and a verb



Index a document

- › The *Index API* is used to index a document
 - use a **PUT** or a **POST** and add the document in the body request
 - notice that the **index** and an **ID** were specified
 - if no **ID** is provided, Elasticsearch generates one for you



```
$ curl -X PUT "localhost:9200/jeopardy/_doc/2" -H 'Content-Type: application/json' -d '{
  "Show Number": 3746,
  "Category": "SECOND-MOST POPULOUS CITIES",
  "Air Date": "2000-12-11",
  "Question": "Brno",
  "Answer": "Czech Republic",
  "Value": "$1000",
  "Round": "Double Jeopardy!"
}'
```

Kibana Dev Tools

- › Using curl all the time can be a bit tedious and convoluted
- › **Kibana** has a developer tool named **Console** for creating and submitting Elasticsearch requests in a simpler fashion



The screenshot shows the Kibana Dev Tools Console interface. At the top, there are navigation tabs: "Console" (selected), "Search Profiler", "Grok Debugger", "Painless Lab", and a "BETA" badge. Below the tabs are links for "History", "Settings", and "Help". The main area is split into two panes. The left pane shows a PUT request to the endpoint `jeopardy/_doc/2` with a JSON body containing fields like "Show Number", "Category", "Air Date", "Question", "Answer", "Value", and "Round". The right pane shows the corresponding JSON response from Elasticsearch, including fields like "_index", "_type", "_id", "_version", "result", "_shards", "_seq_no", and "_primary_term".

```
1 PUT jeopardy/_doc/2
2 {
3   "Show Number": 3746,
4   "Category": "SECOND-MOST POPULOUS
5     CITIES",
6   "Air Date": "2000-12-11",
7   "Question": "Brno",
8   "Answer": "Czech Republic",
9   "Value": "$1000",
10  "Round": "Double Jeopardy!"
11 }
12
13
```

```
1 {
2   "_index" : "jeopardy",
3   "_type" : "_doc",
4   "_id" : "2",
5   "_version" : 1,
6   "result" : "created",
7   "_shards" : {
8     "total" : 1,
9     "successful" : 1,
10    "failed" : 0
11  },
12   "_seq_no" : 216934,
13   "_primary_term" : 1
14 }
```

The **Console** syntax will be used throughout this presentation in examples

The response

201 response if successful

```
1 {
2   "_index" : "jeopardy",
3   "_type" : "_doc",
4   "_id" : "2",
5   "_version" : 1,
6   "result" : "created",
7   "_shards" : {
8     "total" : 1,
9     "successful" : 1,
10    "failed" : 0
11  },
12   "_seq_no" : 216934,
13   "_primary_term" : 1
14 }
```

The ID is stored in the `_id` field

Every document has a `_version`

What if the document ID already exists?

- › The document gets *reindexed*
 - the new document *overwrites* the existing one

```
1 PUT jeopardy/_doc/2
2 {
3   "Show Number": 3746,
4   "Category": "SECOND-MOST POPULOUS
5     CITIES",
6   "Air Date": "2000-12-11",
7   "Question": "Brno",
8   "Answer": "Czech Republic",
9   "Value": "$1000",
10  "Round": "Double Jeopardy!"
11 }
```

```
1 {
2   "_index" : "jeopardy",
3   "_type" : "_doc",
4   "_id" : "2",
5   "_version" : 2,
6   "result" : "updated",
7   "_shards" : {
8     "total" : 1,
9     "successful" : 1,
10    "failed" : 0
11  },
12   "_seq_no" : 216935,
13   "_primary_term" : 1
14 }
```

200 response
(instead of 201)

`_version` is incremented
"updated" instead of "created"

The `_create` endpoint

- › If you do *not* want a document to be overwritten if it already exists, use the `_create` endpoint
 - no indexing occurs and returns a **409** error message

```
1 PUT jeopardy/_create/2
2 {
3   "Show Number": 3746,
4   "Category": "SECOND-MOST POPULOUS
5     CITIES",
6   "Air Date": "2000-12-11",
7   "Question": "Brno",
8   "Answer": "Czech Republic",
9   "Value": "$1000",
10  "Round": "Double Jeopardy!"
11 }
```

```
1 {
2   "error" : {
3     "root_cause" : [
4       {
5         "type" : "version_conflict_engine_exception",
6         "reason" : "[2]: version conflict, document already exists (current version [2])",
7         "index_uuid" : "NZ8_38tASn0p3tY80UQZAQ",
8         "shard" : "0",
9         "index" : "jeopardy"
10      }
11     ],
12     "type" : "version_conflict_engine_exception",
13     "reason" : "[2]: version conflict, document already exists (current version [2])",
14     "index_uuid" : "NZ8_38tASn0p3tY80UQZAQ",
15     "shard" : "0",
16     "index" : "jeopardy"
17   },
18   "status" : 409
19 }
```

Fails if a document with `_id=2` is already indexed

Retrieving a document

- › Use **GET** to retrieve an indexed document
 - note that both the **index** and **ID** are specified
 - response code is **200** if the document is found, **404** if not

"I am looking for
the record with
id=2"

```
1 GET jeopardy/_doc/2
```

```
1 {  
2   "_index" : "jeopardy",  
3   "_type" : "_doc",  
4   "_id" : "2",  
5   "_version" : 2,  
6   "_seq_no" : 216935,  
7   "_primary_term" : 1,  
8   "found" : true,  
9   "_source" : {  
10    "Show Number" : 3746,  
11    "Category" : "SECOND-MOST POPULOUS CITIES",  
12    "Air Date" : "2000-12-11",  
13    "Question" : "Brno",  
14    "Answer" : "Czech Republic",  
15    "Value" : "$1000",  
16    "Round" : "Double Jeopardy!"  
17  }  
18 }
```

The original document is
returned in the **_source** field

The `_update` endpoint

- › If you want to update fields in a document, use the `_update` endpoint
 - make sure to add the `"doc"` context

```
1 POST jeopardy/_update/2
2 {
3   "doc": {
4     "Air Date": "2000-12-10"
5   }
6 }
```

update the "Air Date" field



```
1 {
2   "_index" : "jeopardy",
3   "_type" : "_doc",
4   "_id" : "2",
5   "_version" : 3,
6   "result" : "updated",
7   "_shards" : {
8     "total" : 1,
9     "successful" : 1,
10    "failed" : 0
11  },
12   "_seq_no" : 216936,
13   "_primary_term" : 1
14 }
```

`_version` is incremented

Deleting a document

- › Use **DELETE** to delete an indexed document
 - response code is **200** if the document is found, **404** if not

```
1 DELETE jeopardy/_doc/2
```



```
1 {  
2   "_index" : "jeopardy",  
3   "_type" : "_doc",  
4   "_id" : "2",  
5   "_version" : 4,  
6   "result" : "deleted",  
7   "_shards" : {  
8     "total" : 1,  
9     "successful" : 1,  
10    "failed" : 0  
11  },  
12   "_seq_no" : 216937,  
13   "_primary_term" : 1
```

CRUD operations overview

Index	POST <code>faq/_doc</code> { "question" : "Who writes Wikipedia?", "answer": "Wikipedia is written and edited by volunteers from ..." }
	PUT <code>faq/_doc/4</code> { "question" : "Who writes Wikipedia?", "answer": "Wikipedia is written and edited by volunteers from ..." }
Create	PUT <code>faq/_create/4</code> { "question" : "Who writes Wikipedia?", "answer": "Wikipedia is written and edited by volunteers from ..." }
Read	GET <code>faq/_doc/4</code>
Update	POST <code>faq/_update/4</code> { "doc" : { "answer" : "Wikipedia is written and edited by people" } }
Delete	DELETE <code>faq/_doc/4</code>

Search, queries and aggregations

Search

- › Search is asking questions and getting answers
- › There are two main ways to search:
 - queries
 - aggregations

What documents have a word "cat" in the question?

Character who sang "Smelly cat, smelly cat, what are they feeding you?"; proverbially speaking, it "killed the cat"



What are categories with the most questions?

Before & after, science

Queries and aggregations

> What is the difference between queries and aggregations?

The screenshot shows a search interface with a query 'Question: cat' and a 'query' button. The results are displayed in a table with columns for 'Time' and 'Document'. The table shows three rows of search results. To the right of the table, there are three aggregation panels:

- aggregation 408**: A large number '408' with the text 'Count of records' below it.
- aggregation**: A bar chart titled 'Top values of Value'. The y-axis is 'Count of records' (0 to 70) and the x-axis is 'Top values of Value' (\$400, \$800, \$200, \$1000, \$600, \$1200, \$2000, \$500, \$100, \$1600, Other). The bars show the following approximate counts: \$400 (70), \$800 (65), \$200 (50), \$1000 (40), \$600 (35), \$1200 (30), \$2000 (25), \$500 (20), \$100 (15), \$1600 (10), Other (25).
- aggregation**: A text-based aggregation with the text: 'LOOK WHAT THE CAT SCAN DRAGGED IN KITTY LIT CATS & DOGS ENTERTAINING CATS CATS FELINE FOLLIES'.

Queries and aggregations request

- › How to create an aggregation request?
 - simply send a **GET** request using the **_search** endpoint

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "match": {
5       "Question": "beer"
6     }
7   },
8   "aggs": {
9     "num_questions_per_value": {
10      "terms": {
11        "field": "Category.keyword"
12      }
13    }
14  }
15 }
```

_search = the search endpoint

query = the query clauses to match documents

aggs = the aggregation clauses to summarize the data

Don't worry about the "keyword" here. We'll cover it a bit later.

Queries and aggregations response

```
1 ▾ {
2   "took" : 2,
3   "timed_out" : false,
4 ▾  "_shards" : { [↔] },
10 ▾ "hits" : {
11 ▾   "total" : { [↔] },
15   "max_score" : 11.084997,
16 ▾   "hits" : [ [↔] ]
178 ▾ },
179 ▾ "aggregations" : {
180 ▾   "num_questions_per_value" : {
181     "doc_count_error_upper_bound" : 0,
182     "sum_other_doc_count" : 263,
183 ▾     "buckets" : [ [↔] ]
225 ▾   }
226 ▾ }
227 ▾ }
```

hits = array containing the documents that hit the search criteria

buckets = array containing the top categories

A simple search

- › Use a **GET** request sent to the **_search** endpoint
 - every document is a **hit** for this search
 - by default, Elasticsearch returns 10 hits

"I am looking for any records"

```
1 GET jeopardy/_search
```

```
1 {
2   "took" : 68,
3   "timed_out" : false,
4   "_shards" : {↔},
10  "hits" : {
11   | "total" : {↔},
15   | "max_score" : 1.0,
16   | "hits" : [↔]
178 }
179 }
```

took = the number of milliseconds it took to process the query

total = the number of documents that were hits for this query

hits = array containing the documents that hit the search criteria

Search examples

```
1 GET log_server1,log_server2/_search
```

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "match": {
5       "Question": "math"
6     }
7   }
8 }
```

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "range": {
5       "Air Date": {
6         "gte": "2012-01-01"
7       }
8     }
9   }
10 }
```

```
1 GET log_server*/_search
```

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {
7           "match": {
8             "Question": "math"
9           }
10        }
11      ],
12     "filter": [
13       {
14         "range": {
15           "Air Date": {
16             "gte": "2012-01-01"
17           }
18         }
19       }
20     ]
21   }
22 }
23 }
```

Query types

Types

- › There are plenty of different query types for the **_search** endpoint
- › We will cover just some of them:
 - **match** query
 - **match_phrase** query
 - **multi_match** query
 - **bool** query
- › Although, other types can be found in the official documentation
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html>

Match query

- › Suppose we are interested in all the questions that mentioned
 - “Czech republic”
- › Let’s search for it in the “**Question**” field
- › What do you think is required of a document to be a hit?

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "match": {
5       "Question": "Czech republic"
6     }
7   }
8 }
```

Match query

- › Suppose we are interested in all the questions that mentioned
 - “Czech republic”
- › Let’s search for it in the “**Question**” field
- › By default, the **match** query uses “**or**” logic if multiple terms appear in the search query
 - any document with the term “Czech republic”, “Czech” **or** “*republic*” in the “**Question**” field will be a hit

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "match": {
5       "Question": "Czech republic"
6     }
7   }
8 }
```

The `match_phrase` query

- › The *match_phrase* query is for searching text when you want to find terms that are near each other
 - all the terms in the phrase must be in the document
 - the position of the terms must be in the same relative order

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "match_phrase": {
5       "FIELD": "PHRASE"
6     }
7   }
8 }
```

The field you want to search

The phrase you are searching for

Example of match_phrase

- › Let's try the “**Czech republic**” search using `match_phrase` instead of `match`:
 - only 31 hits (instead of 448 with `match`)
 - be careful, relevant questions may be omitted, e.g. “*The Romantic nationalist composer Bedrich Smetana was born in 1824 in what's now this republic*”
 - we got improved precision but recall is much worse now

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "match_phrase": {
5       "Question": "Czech republic"
6     }
7   }
8 }
```

Two things must happen for “**Czech republic**” to cause a hit:

1. “**Czech**” and “**republic**” must appear in the “**Question**” field
2. The terms must appear in that order and next to each other

The multi_match query

- › The **multi_match** query provides a convenient shorthand for running a **match** query against multiple fields
 - by default, Elasticsearch only considers the best scoring field when calculating the `_score` (**best_fields**)

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "multi_match": {
5       "query": "Czech republic",
6       "fields": ["Question", "Answer"],
7       "type": "best_fields"
8     }
9   }
```

Combining searches

- › Suppose we want to write the following query:
 - find questions about “beer” in the “FOOD & DRINK” category
- › This search is actually a combination of two queries:
 - we need “*beer*” in the **Question** field,
 - and “FOOD & DRINK” in the **Category** field
- › How can we combine these two queries?
 - by using Boolean logic and the **bool** query...

Bool query

- › Each of the following clauses is possible (but optional) in a **bool** query
 - and they can appear in any order

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {}
7       ],
8       "must_not": [
9         {}
10      ],
11      "should": [
12        {}
13      ],
14      "filter": [
15        {}
16      ]
17    }
18  }
19 }
```

Notice the JSON array syntax. You can specify multiple queries in each clause if desired.

The must clause

```
1 GET jeopardy/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {
7           "match": {
8             "Question": "beer"
9           }
10        },
11        {
12          "match": {
13            "Category.keyword": "FOOD &
14              DRINK"
15          }
16        }
17      ]
18    }
19  }
```

- > The clause (query) must appear in matching documents and will contribute to the score
- > Let's go back to our search
 - we are looking for questions with “**beer**” that reside in the “**FOOD & DRINK**” category

Other clauses

- › “**must_not**”
 - the clause (query) must not appear in the matching documents
 - scoring is ignored, a score of 0 for all documents is returned
- › “**should**”
 - the clause (query) should appear in the matching document
- › “**filter**”
 - the clause (query) must appear in matching documents. However unlike **must** the score of the query will be ignored.
- › For more information reach out to the official Elastic documentation about the **bool** query
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-bool-query.html>

Aggregations

Aggregation types

- › Metrics aggregations
 - What is the total amount of prize money in the “Science” category?
- › Bucket aggregations
 - What are the top 5 most popular categories?
- › Combining aggregations
 - What is the total amount of prize money per each category?
- › More about aggregation in the official documentation
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html>

Aggregation syntax

- › An aggregation request is a part of the Search API
 - with or without a “**query**” clause

The “**aggs**” clause can be spelled out “**aggregations**”

```
1 GET jeopardy/_search
2 {
3   "aggs": {
4     "NAME": {
5       "AGG_TYPE": {}
6     }
7   }
8 }
```

The **name** you choose comes back in the results

Lots of **different** aggregation types

Aggregation example. Terms

- › What is the total number of documents per each **Value**?

The “**terms**” aggregation put all the documents into buckets defined by a field value

```
1 GET jeopardy/_search
2 {
3   "aggs": {
4     "num_questions_per_value": {
5       "terms": {
6         "field": "Value"
7       }
8     }
9   }
10 }
```

In this agg the number of documents are collected per each unique **Value**

Aggregation results

- › The response has an “**aggregations**” section that contains the results of all the “**aggs**” in the search request

A list of top 10 buckets.
One bucket corresponds to
a one unique **Value** with
the total number of
documents in it

```
1- {
2  "took" : 8,
3  "timed_out" : false,
4  "_shards" : { [redacted] },
10 "hits" : { [redacted] },
179- "aggregations" : {
180-   "num_questions_per_value" : {
181-     "doc_count_error_upper_bound" : 0,
182-     "sum_other_doc_count" : 21035,
183-     "buckets" : [
184-       {
185-         "key" : "$400",
186-         "doc_count" : 42244
187-       },
188-       {
189-         "key" : "$800",
190-         "doc_count" : 31860
191-       },
192-       { [redacted] },
196-       { [redacted] },
200-       { [redacted] },
204-       { [redacted] },
208-       { [redacted] },
212-       { [redacted] },
216-       { [redacted] },
220-       { [redacted] }
224-     ]
225-   }
226- }
227- }
```

We get the top 10 hits ...

... and an “aggregations”
section in the response

The scope of an aggregation

- › You can add a **query** clause to an aggregation to limit the scope

What are the top 3 categories with questions about the Czech republic

```
1 GET jeopardy/_search
2 {
3   "size": 0,
4   "query": {
5     "match": {
6       "Question": "Czech republic"
7     }
8   },
9   "aggs": {
10    "top_categories": {
11      "terms": {
12        "field": "Category.keyword",
13        "size": 3
14      }
15    }
16  }
17 }
```

```
1 {
2   "took" : 1,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 1,
6     "successful" : 1,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : 1,
12    "max_score" : 1.0,
13    "hits" : [
14      {
15        "_type" : "question",
16        "_source" : {
17          "Question": "Czech republic",
18          "Category.keyword": "WORLD CAPITALS"
19        },
20        "_id" : "1",
21        "score" : 1.0
22      }
23    ]
24  },
25  "aggregations" : {
26    "top_categories" : {
27      "doc_count_error_upper_bound" : 0,
28      "sum_other_doc_count" : 462,
29      "buckets" : [
30        {
31          "key" : "WORLD CAPITALS",
32          "doc_count" : 7
33        },
34        {
35          "key" : "\\\"C\\\"OUNTRIES",
36          "doc_count" : 4
37        },
38        {
39          "key" : "CANCELED CZECHS",
40          "doc_count" : 4
41        }
42      ]
43    }
44  }
45 }
```

buckets field consists of category names and number of documents in it that satisfy the query

Mapping

What is Mapping?

- › Elasticsearch will happily index any document without knowing its details (number of fields, their data types, etc.)
 - however, behind-the-scenes Elasticsearch assigns data types to your fields in a *mapping*
- › A **mapping** is a **schema definition** that contains:
 - name of fields
 - data types of fields
 - how the fields should be indexed and stored by Lucene
- › Mappings map complex JSON documents into the simple flat documents that Lucene expects

Remember the odd “Category.keyword”?

- > The one on the slide number 44
- > Let's have a closer look where does it come from

```
8 > "aggs": {
9 >   "num_questions_per_value": {
10 >     "terms": {
11 >       "field": "Category.keyword"
12 >     }
13 >   }
14 > }
15 > }
```

```
1 > {
2 >   "jeopardy": {
3 >     "mappings": {
4 >       "_meta": { },
7 >       "properties": {
8 >         "@timestamp": { },
11 >         "Air Date": {
12 >           "type": "date",
13 >           "format": "iso8601"
14 >         },
15 >         "Answer": {
16 >           "type": "text"
17 >         },
18 >         "Category": {
19 >           "type": "text",
20 >           "fields": {
21 >             "keyword": {
22 >               "type": "keyword",
23 >               "ignore_above": 256
24 >             }
25 >           }
26 >         },
27 >         "Question": {
28 >           "type": "text"
29 >         },
30 >         "Round": { },
33 >         "Show Number": { },
36 >         "Value": { }
39 >       }
40 >     }
41 >   }
42 > }
```

1 GET jeopardy/_mapping

Define a mapping

- › In many use cases, you will need to define your own mappings
- › Mappings are defined in the “**mappings**” section of an index:
 - you can define mappings at index creation:

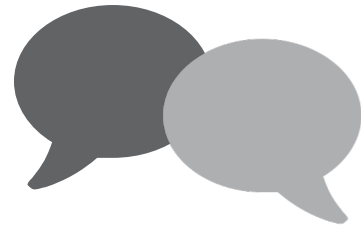
```
1 PUT jeopardy
2 {
3   "mappings": {
4     |   define your mapping here
5   }
6 }
```

- or, add to a mapping of an existing index:

```
1 PUT jeopardy/_mapping
2 {
3   additional mapping here
4 }
```

- › More about mapping can be found in the documentation
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html>

- › Elasticsearch documentation
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>
- › Kibana documentation
 - <https://www.elastic.co/guide/en/kibana/current/index.html>
- › Information provided on slides was inspired by materials from Elasticsearch Engineer training
 - <https://www.elastic.co/training/elasticsearch-engineer>



Questions

Thank you

PROFINIT

Profinit EU, s.r.o., Tychonova 2, 160 00 Praha 6
Tel.: + 420 224 316 016, web: www.profinit.eu



LinkedIn
linkedin.com/company/profinit



Twitter
twitter.com/Profinit_EU



Facebook
facebook.com/Profinit.EU



Youtube
Profinit EU