



# Elasticsearch Engineer I

An Elastic Training Course



7.3.1

[elastic.co/training](https://elastic.co/training)

# Elasticsearch Engineer I

Course: Elasticsearch Engineer I

© 2015-2019 Elasticsearch BV. All rights reserved. Decompiling, copying, publishing and/or distribution without written consent of Elasticsearch BV is strictly prohibited.

# Welcome to This Virtual Training

- We will start momentarily
- The training will start with an audio/video test, to make sure that everyone can hear and see the instructors
- To prevent any audio/video issues, please:
  - disable any ad blockers or script blockers
  - use a supported web browser: Chrome or Firefox
- In case of problems, try the following steps in order:
  - refresh this web page
  - open this page in an "incognito" or "private" window
  - try another web browser
  - as a last resort, restarting your computer sometimes helps too

# Welcome to This Training

- Visit [training.elastic.co](https://training.elastic.co) and log in
  - follow instructions from registration email to get access
- Go to "**My Account**" and click on today's training
- Download the PDF file (this contains all the slides)
- Click on "**Virtual Link**" to access the Lab Environment
  - create an account
  - you will need an access token, which the instructor will provide

# About This Training

- Environment
- Introductions
- Code of Conduct (<https://www.elastic.co/community/codeofconduct>)
- Agenda...

# Course Agenda

- 1 Elasticsearch Fundamentals
- 2 Elasticsearch Queries
- 3 Elasticsearch Aggregations
- 4 Elasticsearch Text Analysis and Mappings
- 5 Elasticsearch Nodes and Shards
- 6 Elasticsearch Monitoring and Troubleshooting

-  Elasticsearch Fundamentals
-  Elasticsearch Queries
-  Elasticsearch Aggregations
-  Elasticsearch Text Analysis and Mappings
-  Elasticsearch Nodes and Shards
-  Elasticsearch Monitoring and Troubleshooting

Module 1

# Elasticsearch Fundamentals



# Topics

- Elastic Stack Overview
- Getting Started with Elasticsearch
- CRUD Operations
- Searching Data



Elasticsearch Fundamentals

Lesson 1

# Elastic Stack 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 it into your application
  - challenging to use
  - not originally designed for scaling

“

*“Search is something that  
any application should  
have”*

Shay Banon

# 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:
  1. *distributed from the ground up in its design*
  2. *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

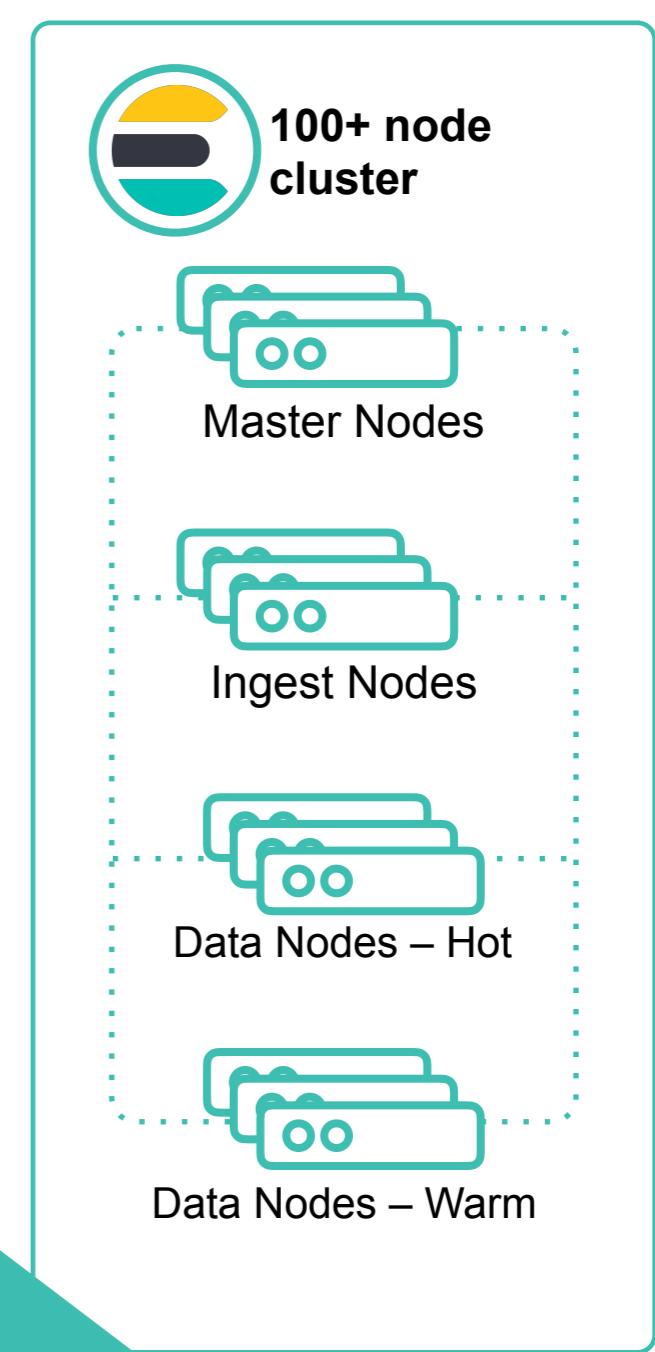
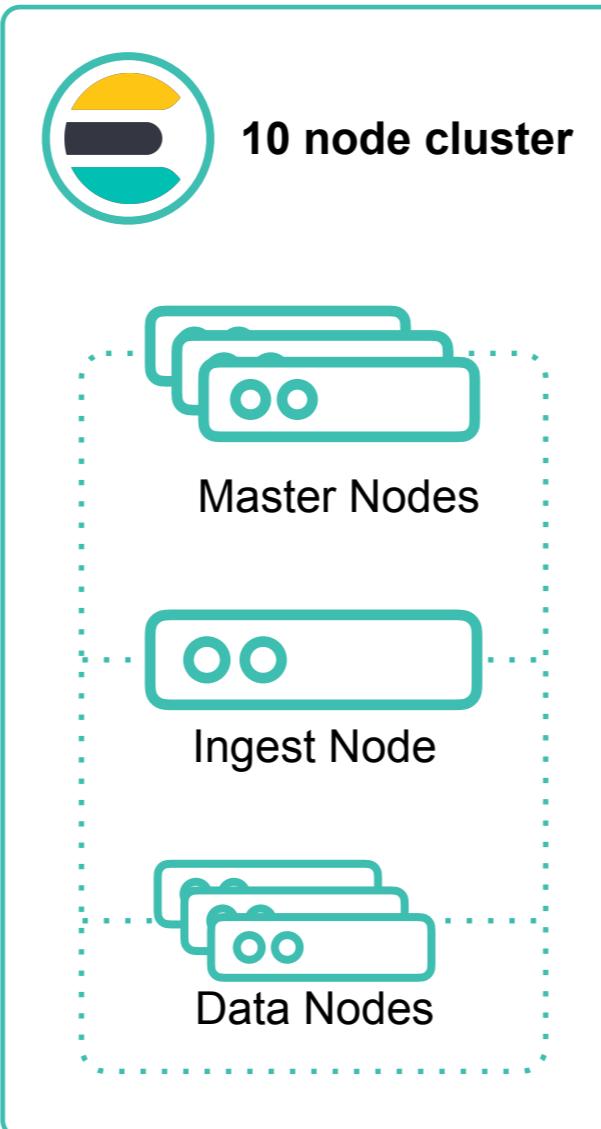
# 1. Distributed search:

- Elasticsearch is distributed and scales horizontally:



A **node** is an instance of Elasticsearch

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

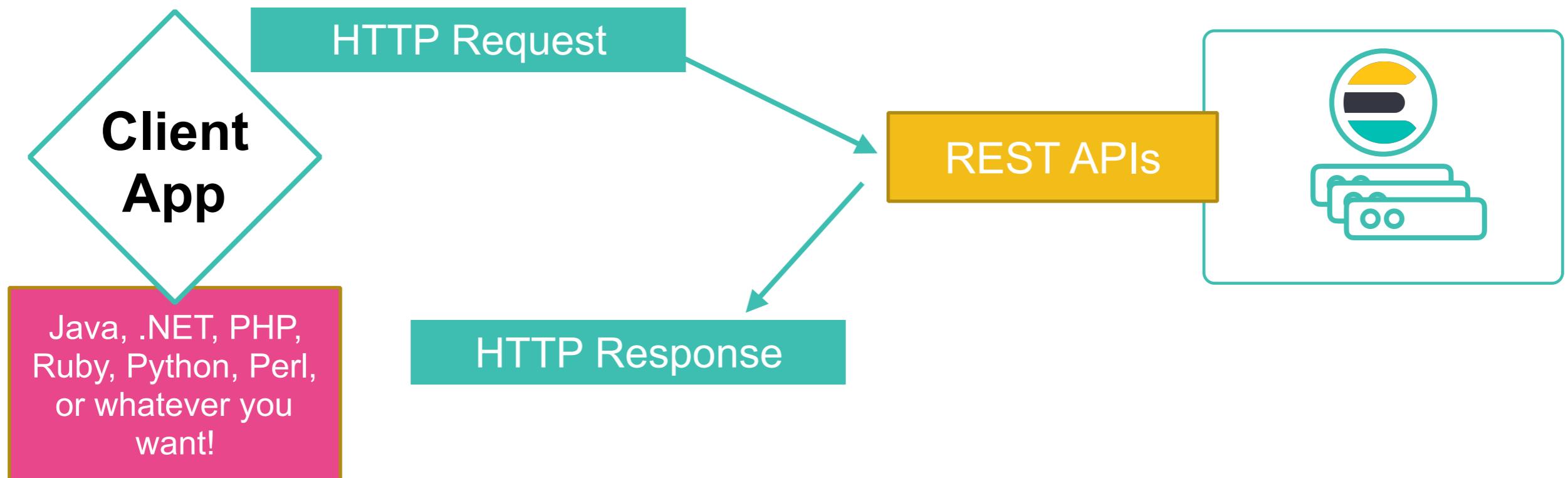


Your cluster can grow as your needs grow



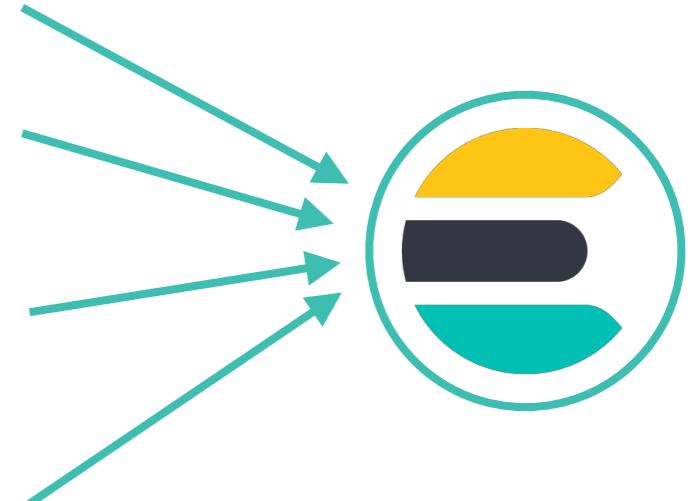
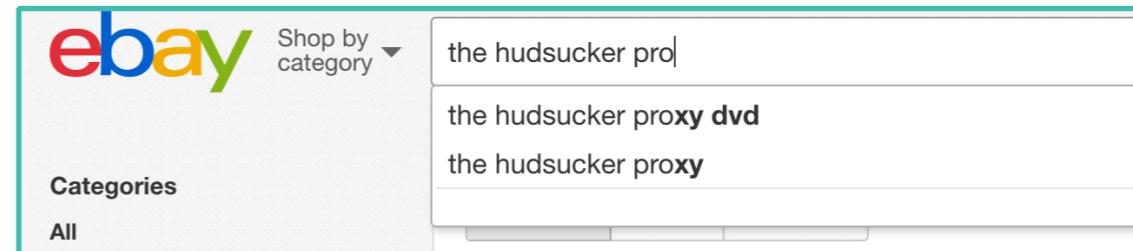
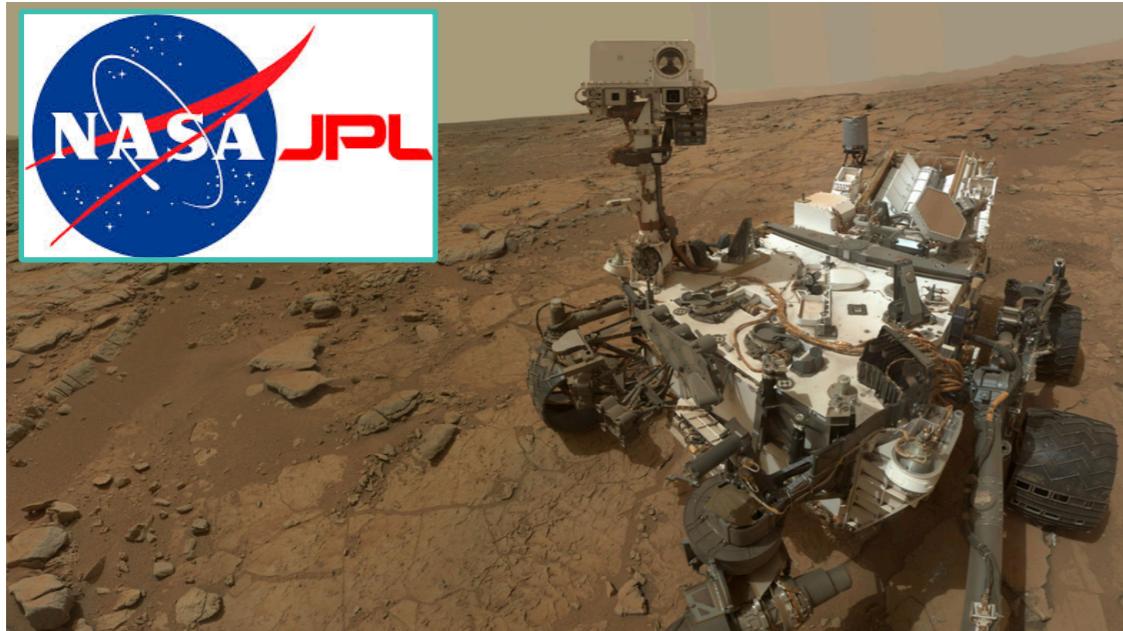
## 2. Easily used by other languages:

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



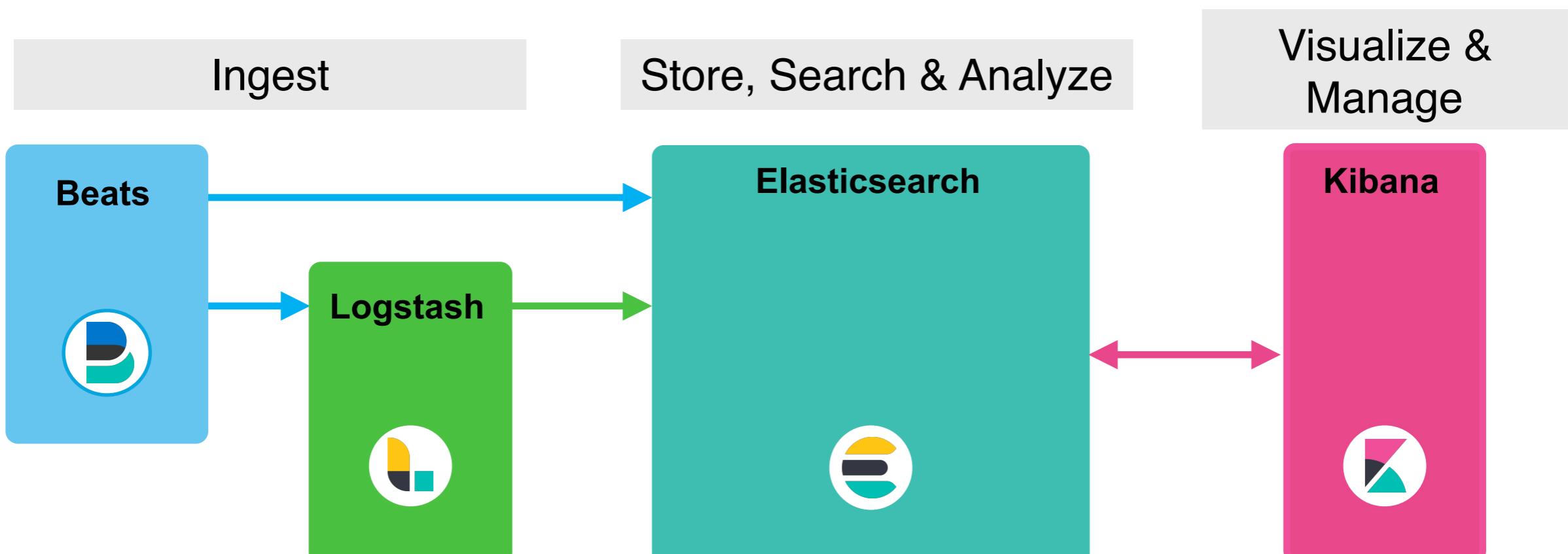
# You Know, for Search!

- ...and *logging*,
- and *metrics*,
- and *business analytics*,
- and *security analytics*,
- and lots more!



# Elastic Stack

- The Elastic Stack is a collection of products with Elasticsearch at the heart
  - reliably and securely take data from any source, in any format, as well as search, analyze, and visualize it in real time



 Standalone (self managed)

 Elastic Cloud Enterprise  
(self managed)

 Elastic Cloud (SaaS)



Elasticsearch Fundamentals

Lesson 1

# Review - Elastic Stack Overview



# Summary

- Elasticsearch is designed to be scalable and easy-to-use by any programming language
- The ***Elastic Stack*** is a collection of products with ***Elasticsearch*** at the heart
- ***Beats*** are single-purpose data shippers
- ***Logstash*** is a server-side data processing pipeline
- ***Kibana*** is an analytics and visualization platform

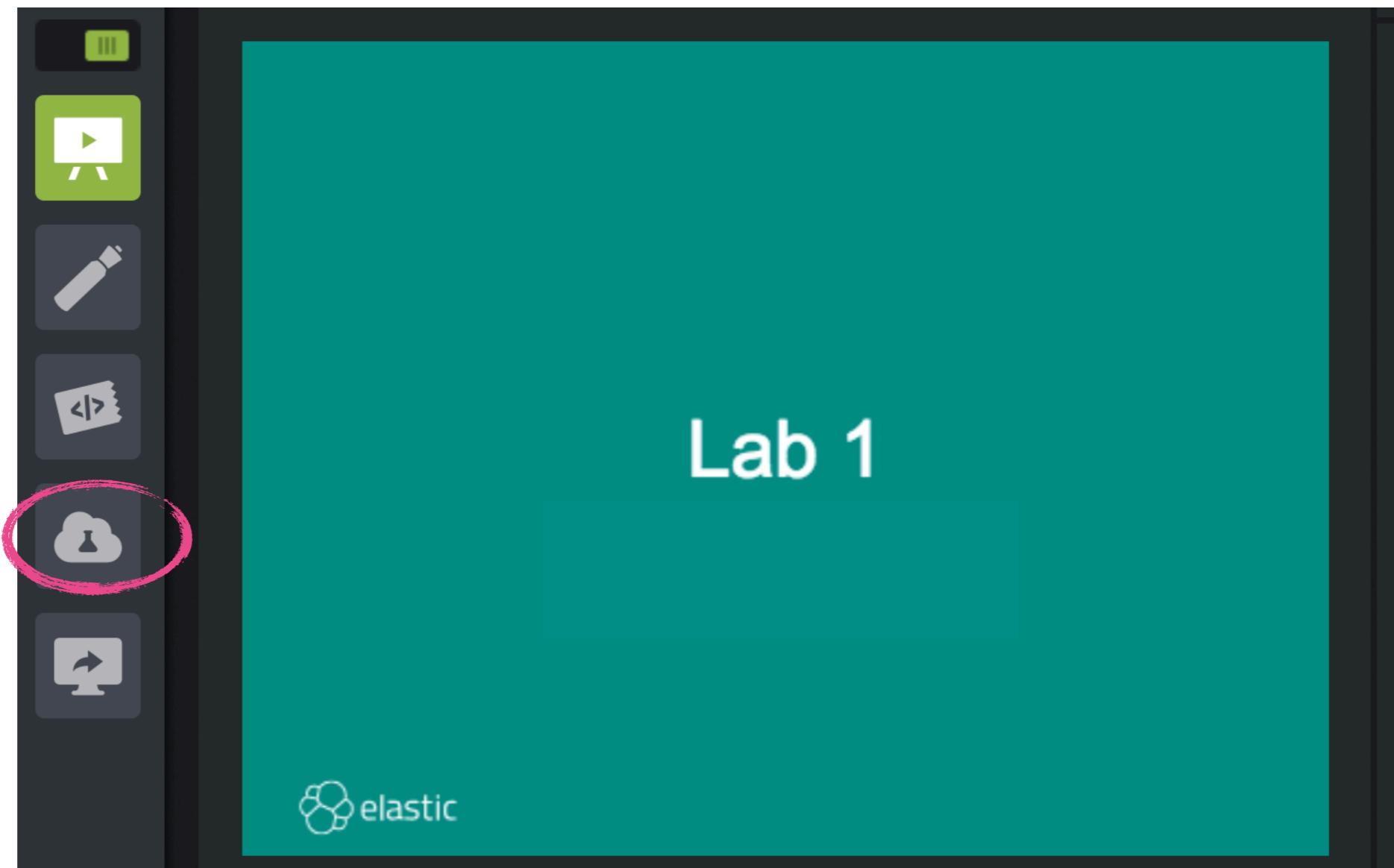
# Quiz

1. What are the 4 main components of the Elastic Stack?
2. **True or False:** Elasticsearch uses Apache Lucene behind the scenes to index and search data.
3. What were two of Shay's main goals when designing Elasticsearch?

# Lab Environment

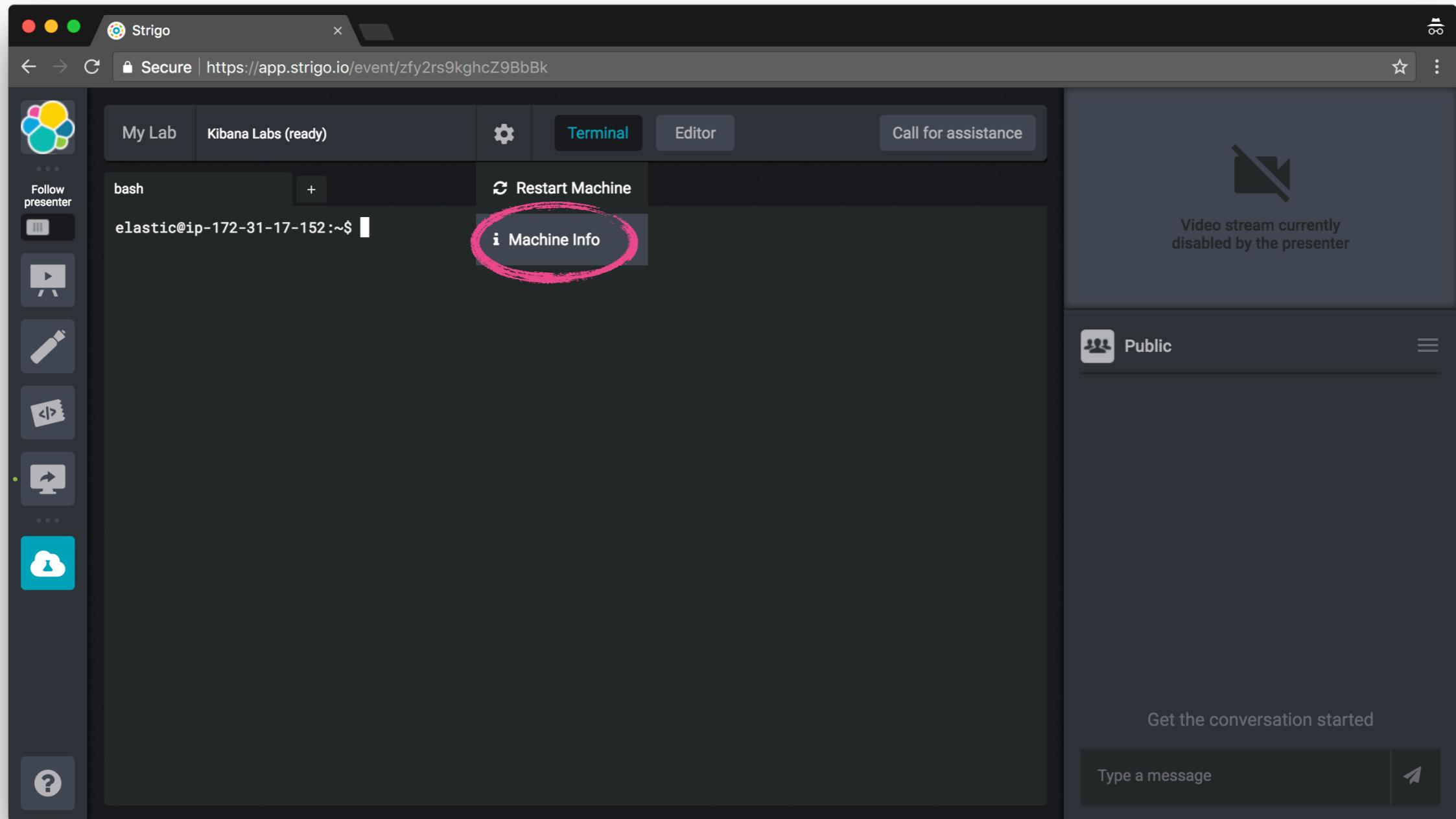
# Lab Environment

- Visit Strigo using the link that was shared with you, and log in if you haven't already done so
- Click on "**My Lab**" on the left



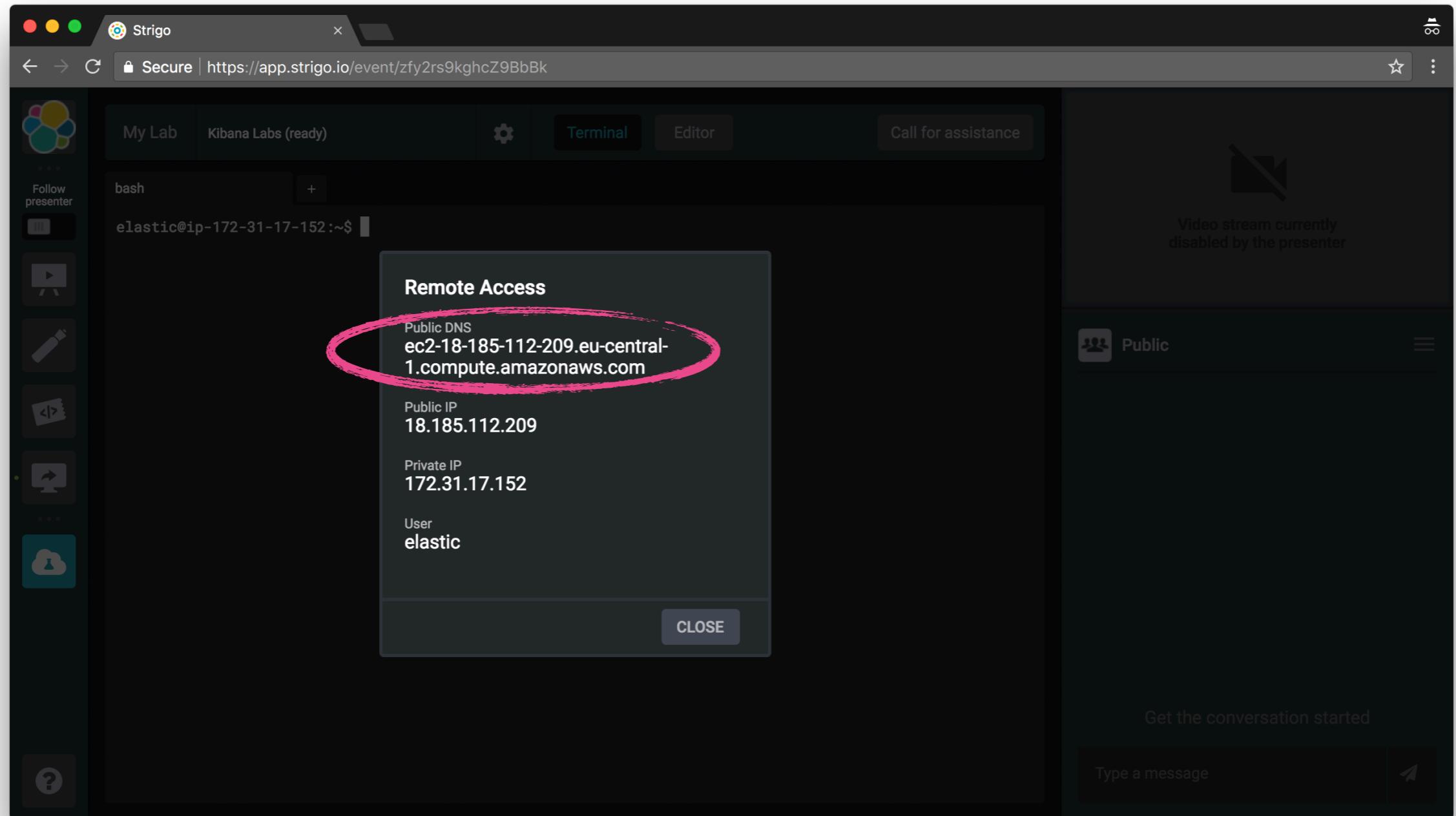
# Lab Environment

- Click on the gear icon next to "My Lab" and select "Machine Info"



# Lab Environment

- Copy the hostname that is shown under "Public DNS"



# Lab Environment

- From here you can access lab instructions and guides
  - you also have them in your .zip file, but it is easier to access and use the lab instructions from here:



The Elasticsearch logo, which consists of five overlapping colored circles (yellow, pink, blue, teal, and light green) arranged in a circular pattern.

## Welcome to Elasticsearch Engineer I

---

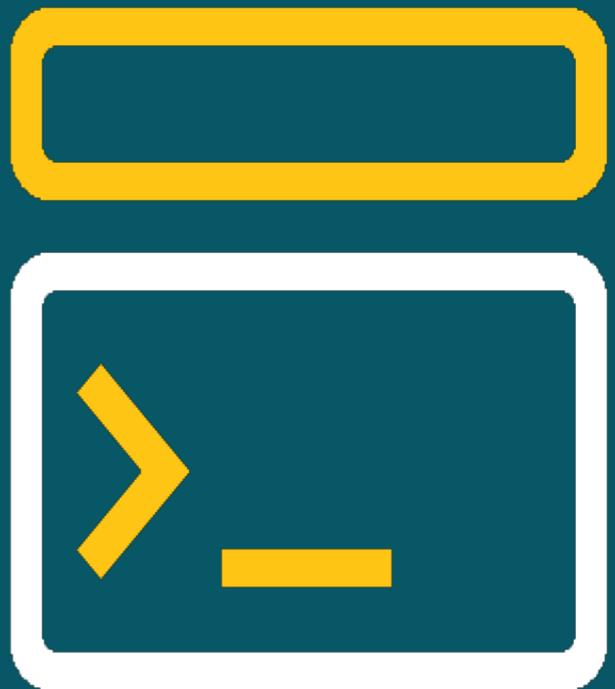
- [Lab Instructions](#)
- [Virtual Classroom User Guide](#)
- [Search Page](#)
- [Kibana](#)



Elasticsearch Fundamentals

Lesson 1

# Lab - Elastic Stack Overview





Elasticsearch Fundamentals

Lesson 2

# Getting Started with Elasticsearch



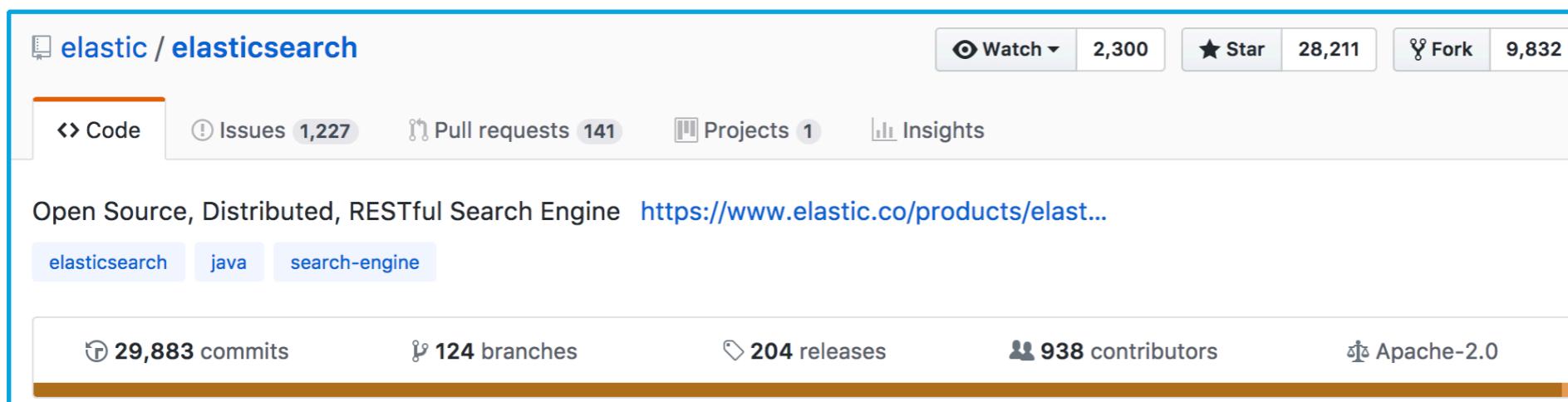
# Getting Elasticsearch

- Download the binaries at  
<https://www.elastic.co/downloads/elasticsearch>

Downloads:

<a href="#">WINDOWS sha asc</a>	<a href="#">MACOS sha asc</a>
<a href="#">LINUX sha asc</a>	<a href="#">DEB sha asc</a>
<a href="#">RPM sha asc</a>	<a href="#">MSI (BETA) sha asc</a>

- Or build your own install from  
<https://github.com/elastic/elasticsearch>
  - it is open source!



# Preparing for Installation

- Elasticsearch is a Java application
  - it requires a **JRE** or a **JDK** to run
  - it will refuse to start if a known-bad version of Java is used
- Each Elasticsearch distribution (7.0+) includes a bundled version of OpenJDK from the JDK maintainers (GPLv2+CE)
  - to use your own Java version, set the **JAVA\_HOME** variable
  - Elasticsearch 7.x requires at least Java 8
  - [https://www.elastic.co/support/matrix#matrix\\_jvm](https://www.elastic.co/support/matrix#matrix_jvm)



# Installation Options

- If you **want** to install and manage it yourself:
  - unzip the binaries (it really is that easy!)
  - MSI for Windows
  - Homebrew
  - use Linux package manager ([.deb](#), [.rpm](#))
  - CM options like Ansible, Chef, Puppet
  - [Docker](#)
  - Elastic Cloud on Kubernetes
  - Elastic Cloud Enterprise
- And if you **don't want** to install or manage it yourself:
  - [Elasticsearch Service](#)



# Elasticsearch Directories

Folder	Description	Setting
<b>bin</b>	Binary scripts including <b>elasticsearch</b> to start a node and <b>elasticsearch-plugin</b> to install plugins	
<b>config</b>	Configuration files including <b>elasticsearch.yml</b>	<b>ES_PATH_CONF</b>
<b>data</b>	The location of the data files of each index and shard allocated on the node	<b>path.data</b>
<b>jdk</b>	The bundled version of OpenJDK from the JDK maintainers (GPLv2+CE)	<b>JAVA_HOME</b>
<b>lib</b>	The Java JAR files of Elasticsearch	
<b>logs</b>	Elasticsearch log files location	<b>path.logs</b>
<b>modules</b>	Contains various Elasticsearch modules	
<b>plugins</b>	Plugin files location. Each plugin will be contained in a subdirectory	



# The Configuration Files

- Elasticsearch has three main configuration files:

- elasticsearch.yml**

```
# ----- Cluster -----  
#  
# Use a descriptive name for your cluster:  
#  
#cluster.name: my-application  
#
```

Notice by default all settings  
are commented out

- jvm.options**

```
# Xms represents the initial size of total heap space  
# Xmx represents the maximum size of total heap space  
  
-Xms1g  
-Xmx1g
```

- log4j2.properties**

```
status = error  
  
# log action execution errors for easier debugging  
logger.action.name = org.elasticsearch.action
```

# Setting Properties

- Two main options for defining configuration properties:
  - config file:
    - elasticsearch.yml: `path.data: /data/elasticsearch`
    - jvm.options: `-Xms512m`
  - command line:

```
$ ./bin/elasticsearch -E path.data=/data/elasticsearch
```

```
$ ES_JAVA_OPTS="-Xms512m" ./bin/elasticsearch
```

# Starting Elasticsearch

- When running Elasticsearch from the **.zip** installation, the **bin** folder has binaries for starting it:
  - **elasticsearch**: shell script for Mac and Linux
  - **elasticsearch.bat**: batch script for Windows

```
$ ./bin/elasticsearch
```

- Use **-d** to run as a daemon on Mac/Linux:

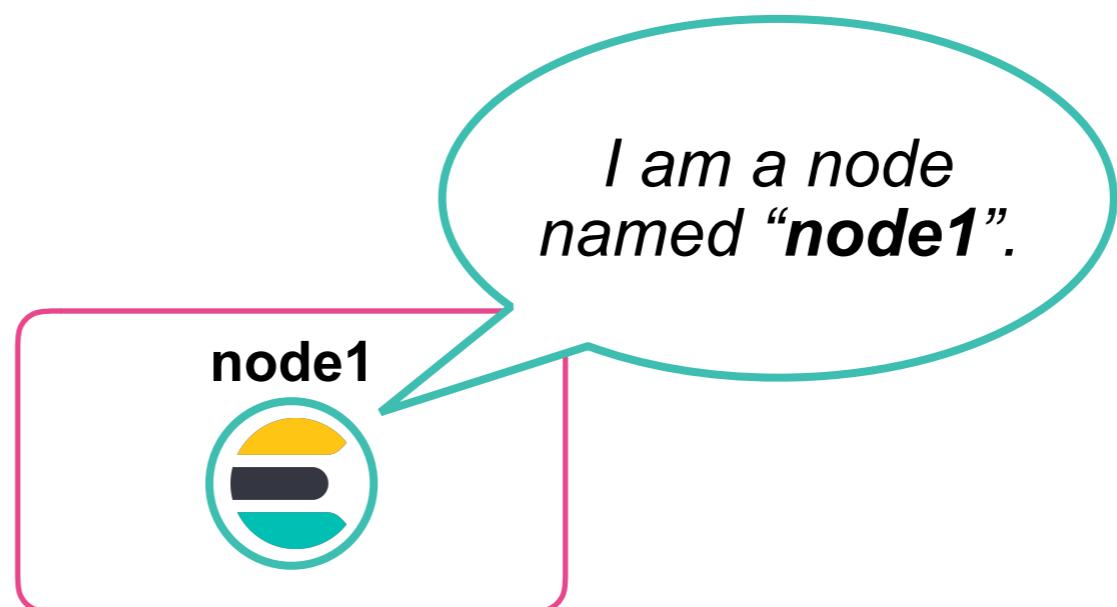
```
$ ./bin/elasticsearch -d -p elastic.pid
```

**-p** saves the process id in  
the specified file



# 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

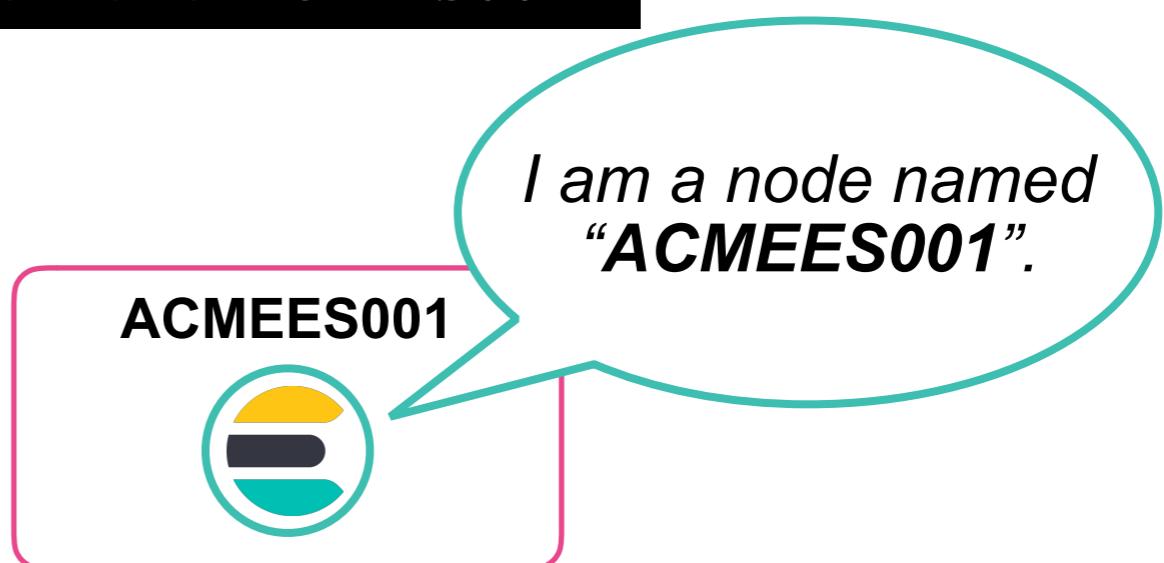


# Node Name

- Every node has an **unique id**
  - randomly generated UUID
- Every node has a name
  - default is the hostname
- Set using **node.name** (on the command line or .yml file)

```
$ ./bin/elasticsearch -E node.name=ACMEES001
```

node.name: ACMEES001



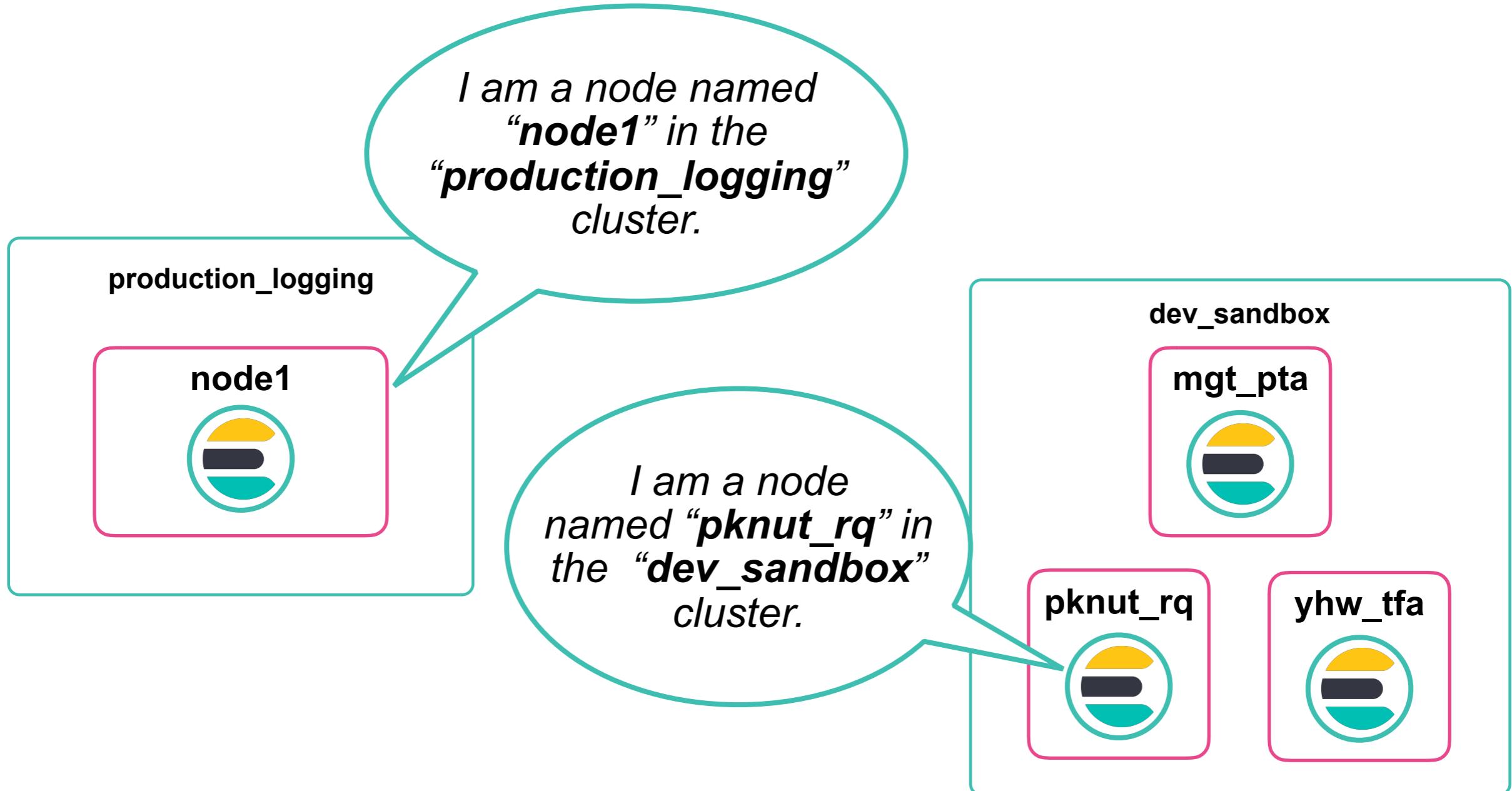
# Stopping Elasticsearch

1. **Ctrl+c** at the command prompt to kill the process, if Elasticsearch is running in the foreground
2. Or, you can kill it using the process id:

```
$ kill `cat elastic.pid`
```

# Cluster

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

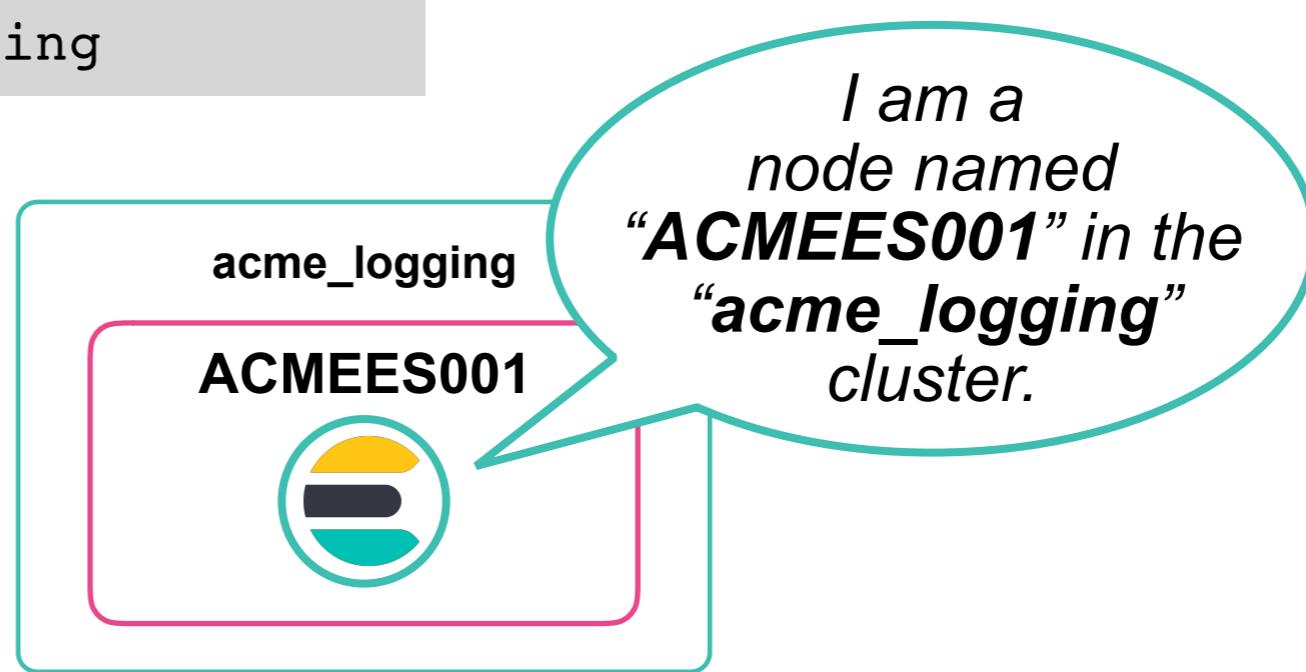


# Cluster Name

- Every cluster has a name
  - defaults to “elasticsearch”
  - change to something meaningful and unique
  - helpful to include your company name (e.g. “Acme, Inc”)
- Set using `cluster.name`:

```
$ ./bin/elasticsearch -E cluster.name=acme_logging
```

cluster.name: acme\_logging





Elasticsearch Fundamentals

Lesson 2

# Review - Getting Started with Elasticsearch



# Summary

- Installing Elasticsearch can be as easy as unzipping a file!
- A *node* is an instance of Elasticsearch
- A *cluster* is one or multiple nodes working together in a distributed manner

# Quiz

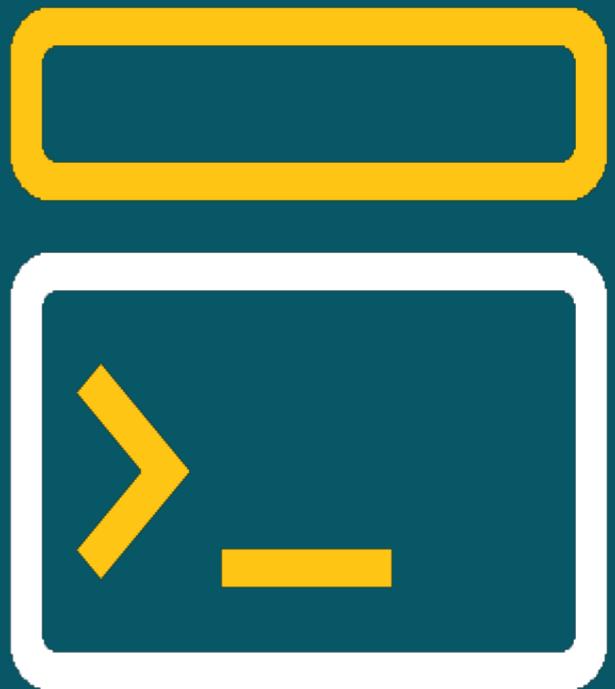
1. What are the three main configuration files you will find in the Elasticsearch config folder?
2. **True or False:** The only way to configure settings is using the config files.
3. How do you set the node name?



Elasticsearch Fundamentals

Lesson 2

# Lab - Getting Started with Elasticsearch





Elasticsearch Fundamentals

Lesson 3

# CRUD Operations



# Documents must be JSON Objects

- Imagine blogs that are currently in a database table:

<b>title</b>	<b>category</b>	<b>date</b>	<b>author_first_name</b>	<b>author_last_name</b>	<b>author_company</b>
Solving the Small but Important Issues with Fix-It Fridays	Culture	December 22, 2017	Daniel	Cecil	Elastic
Fighting Ebola with Elastic	User Stories		Emily	Mosher	

- Each blog needs to be converted to a JSON object:

JSON consists  
of *fields*...

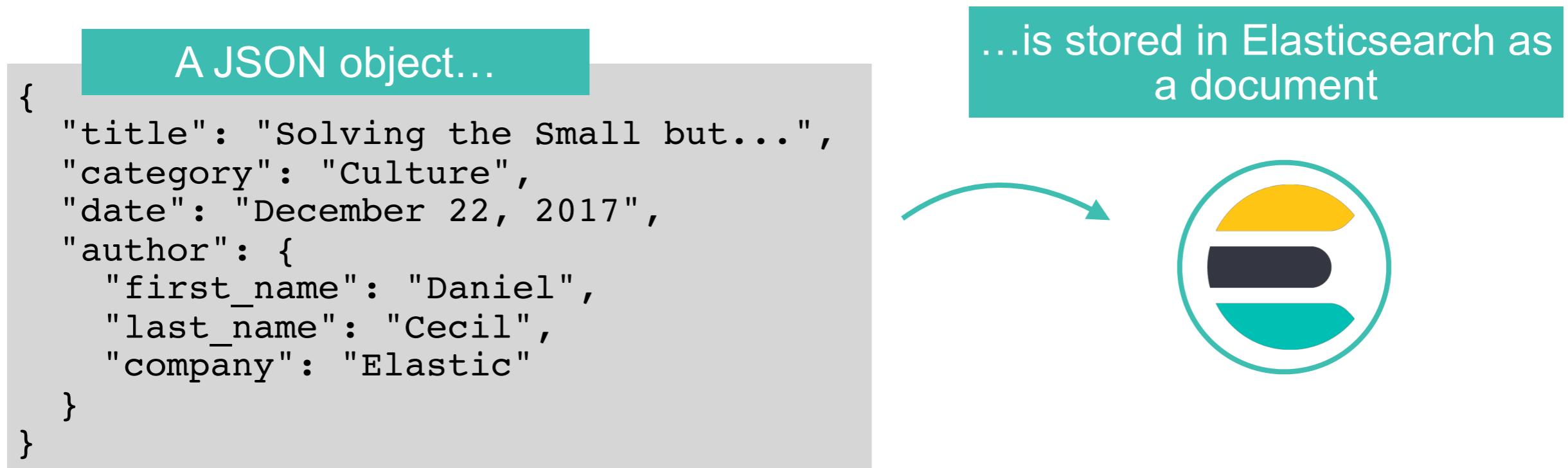
```
{  
  "title": "Fighting Ebola with Elastic",  
  "category": "User Stories",  
  "author": {  
    "first_name": "Emily",  
    "last_name": "Mosher"  
  }  
}
```

...and *values*



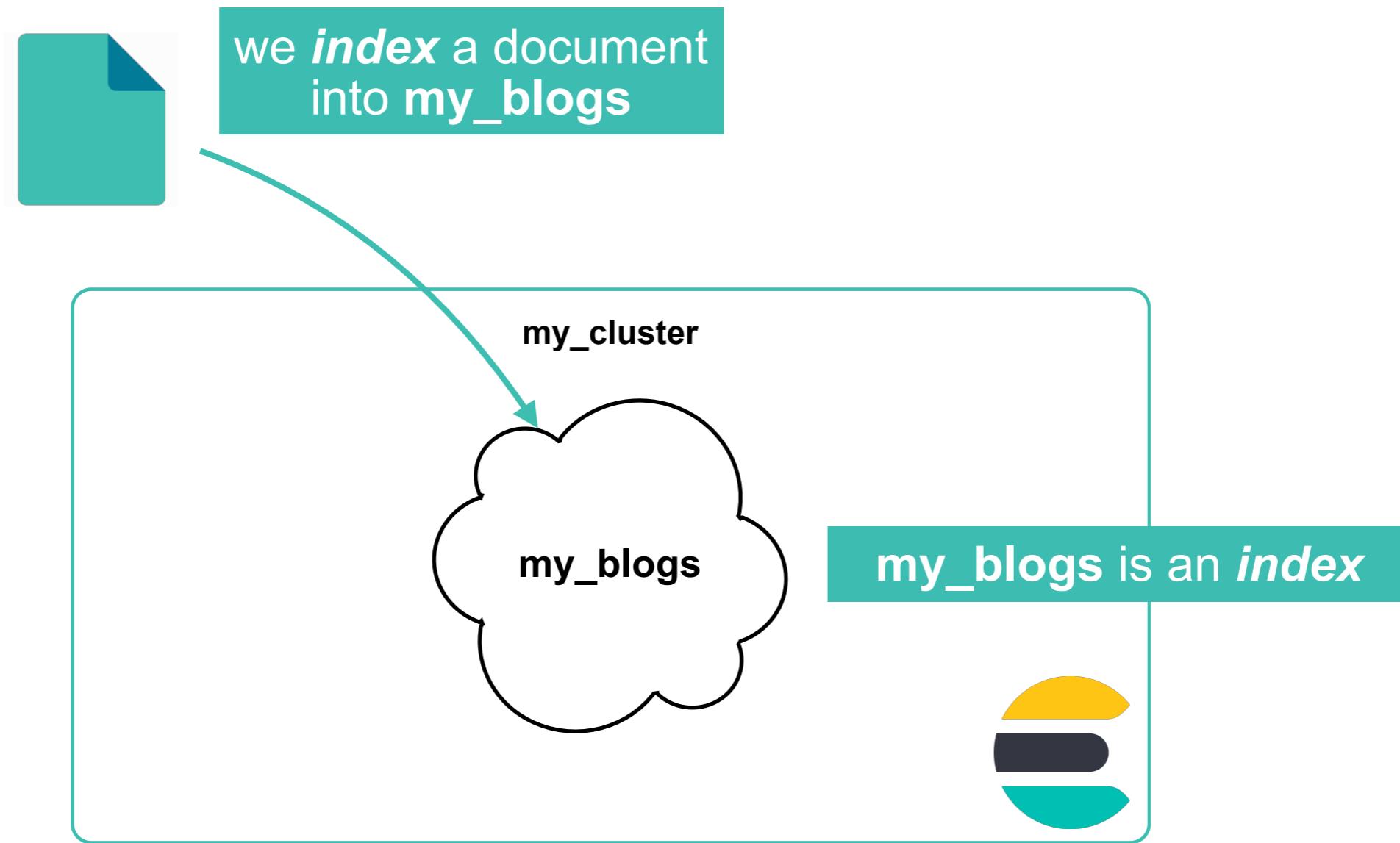
# Document Store

- 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



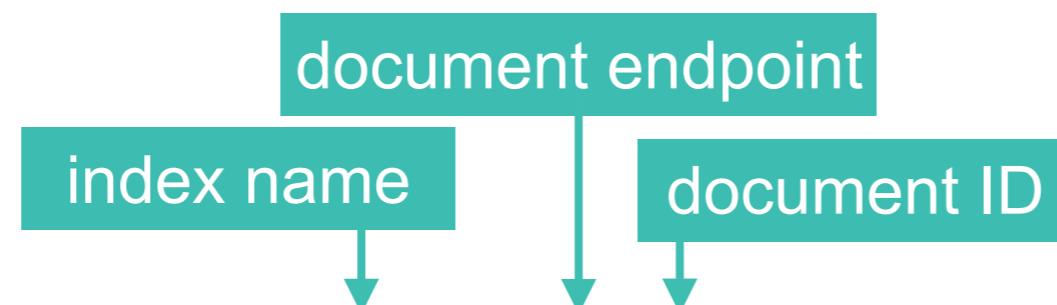
# Documents are Indexed into an Index

- In Elasticsearch, a document is *indexed* into an *index*
  - we use **index** as a verb and a noun



# 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 we specify the **index** and an **ID**
  - if no **ID** is provided, Elasticsearch generates one for you (as you will see during the labs)



```
$ curl -X PUT "localhost:9200/my_blogs/_doc/1" -H 'Content-Type: application/json' -d'
{
  "title": "Fighting Ebola with Elastic",
  "category": "User Stories",
  "author": {
    "first_name": "Emily",
    "last_name": "Mother"
  }
}
```

# Console

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

Dev Tools

History S

Console Search Profiler Grok Debugger

```
1 PUT my_blogs/_doc/1
2 {
3   "title": "Fighting Ebola with Elastic",
4   "category": "User Stories",
5   "author": {
6     "first_name": "Emily",
7     "last_name": "Mother"
8   }
9 }
10
```

The Console syntax will be used throughout this course in the examples

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

# The Response

```
{  
  "_index": "my_blogs",  
  "_type": "_doc",  
  "_id": "1",  
  "_version": 1,  
  "result": "created",  
  "_shards": {  
    "total": 2,  
    "successful": 1,  
    "failed": 0  
  },  
  "_seq_no": 0,  
  "_primary_term": 1  
}
```

201 response if successful

The ID is stored in the `_id` field

Every document has a `_version`

# But we never created the index!

- As part of the easy-to-use, out-of-the-box experience of Elasticsearch, if you index a document into a non-existing index, ***the index is created automatically***
- You will likely disable this behavior on a production cluster
  - and create your own index with your settings and mappings,
  - but that is a discussion we will have later...

```
PUT /my_blogs
{
  "settings": {
    ...
  },
  "mappings": {
    ...
  }
}
```

We typically create the index first, then index our documents

# What if the document ID already exists?

- What do you think happens if you index a document using an ID that already exists?

```
PUT my_blogs/_doc/1
{
  "title": "Elasticsearch 5.0.0-beta1 released",
  "category": "Releases",
  "date": "September 22, 2016",
  "author": {
    "first_name": "Clinton",
    "last_name": "Gormley",
    "company": "Elastic"
  }
}
```

my\_blogs already has a document with \_id = 1

# What if the document ID already exists?

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

```
PUT my_blogs/_doc/1
```

```
{  
  "title": "Elasticsearch 5.0.0-beta1 released",  
  "category": "Releases",  
  "date": "September 22, 2016",  
  "author": {  
    "first_name": "Clinton",  
    "last_name": "Gormley",  
    "company": "Elastic"  
  }  
}
```

```
{  
  "_index": "my_index",  
  "_type": "_doc",  
  "_id": "1",  
  "_version": 2,  
  "result": "updated",  
  ...  
}
```

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:

```
PUT my_blogs/_create/1
```

```
{  
  "title": "Elasticsearch 5.0.0-beta1 released",  
  "category": "Releases",  
  "date": "September 22, 2016",  
  "author": {  
    "first_name": "Clinton",  
    "last_name": "Gormley",  
    "company": "Elastic"  
  }  
}
```

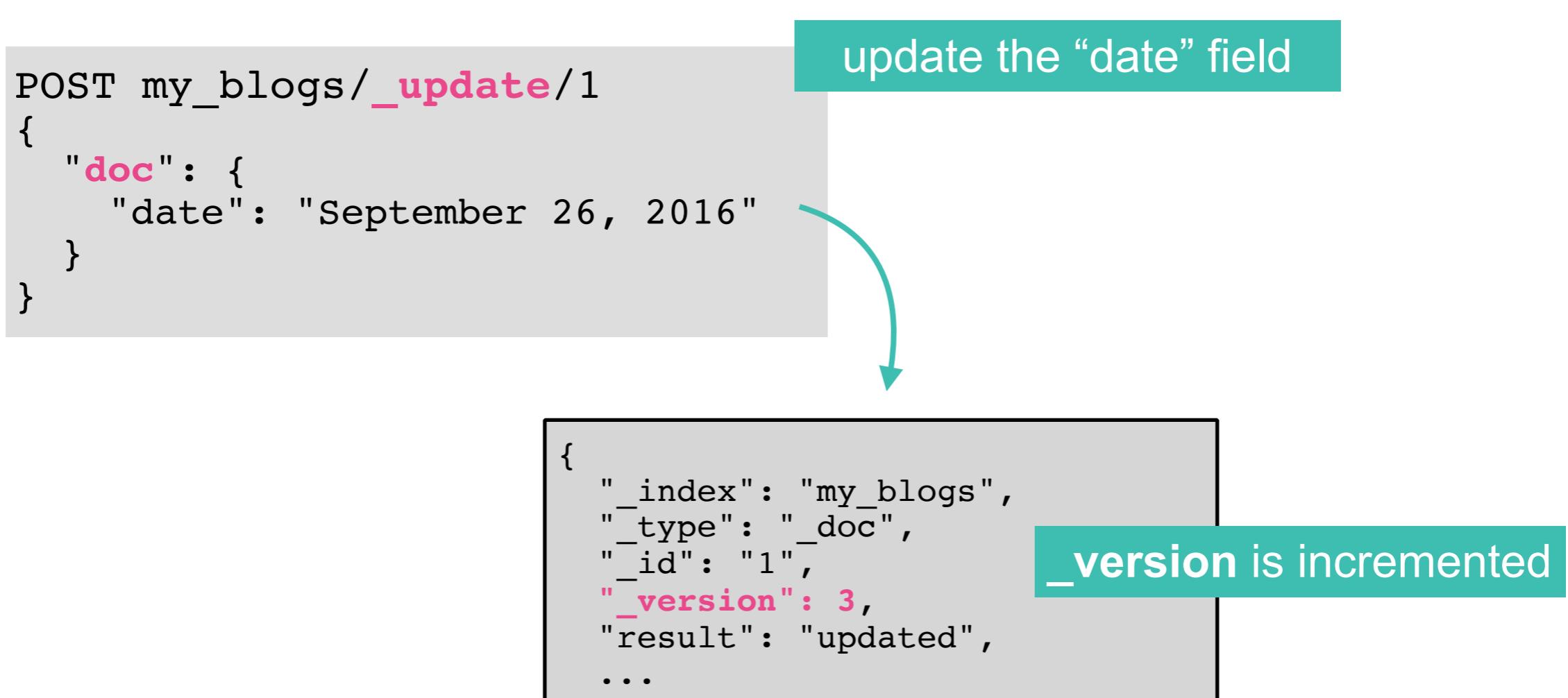
Fails if a document with  
`_id=1` is already indexed

```
{  
  "status": 409,  
  "error": {  
    "root_cause": [  
      {  
        "type": "version_conflict_engine_exception",  
        "reason": "[_doc][1]: version conflict,  
document already exists (current version [2])",  
      }  
    ]  
  }  
}
```



# The `_update` Endpoint

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



# Deleting a Document

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

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

```
{  
  "_index": "my_blogs",  
  "_type": "_doc",  
  "_id": "1",  
  "_version": 4,  
  "result": "deleted",  
  "_shards": {  
    "total": 2,  
    "successful": 1,  
    "failed": 0  
  },  
  "_seq_no": 3,  
  "_primary_term": 1  
}
```

# Cheaper in Bulk

- The **Bulk API** makes it possible to perform many write operations in a single API call
  - greatly increases the indexing speed
  - useful if you need to index a data stream such as log events
- Four actions
  - **create, index, update and delete**
- The response is a large JSON structure with the individual results of each action that was performed
  - the failure of a single action does not affect the remaining actions

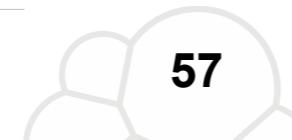
# Example of `_bulk`

- `_bulk` has a unique syntax based on lines of commands:
  - it uses the newline delimited JSON (NDJSON) structure
  - each command appears on a single line

The “`index`” action is followed by the document (on a single line)

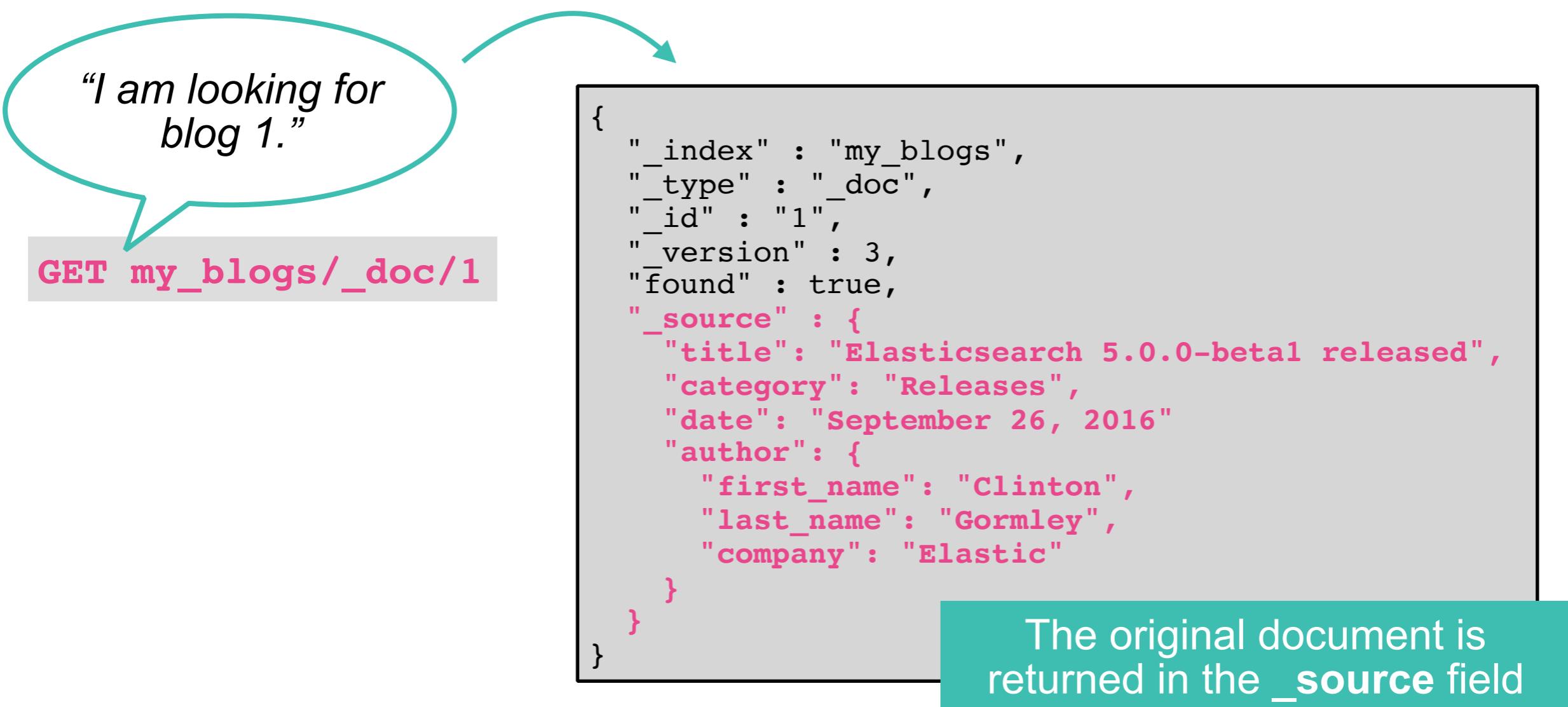
```
POST comments/_bulk
{"index" : {"_id":3}}
{"title": "Tuning Go Apps with Metricbeat", "category": "Engineering"}
{"index" : {"_id":4}}
{"title": "Elastic Stack 6.1.0 Released", "category": "Releases"}
{"index" : {"_id":5}}
{"title": "Searching for needle in", "category": "User Stories"}
{"update" : {"_id":5}}
{"doc": {"title": "Searching for needle in haystack"}}
{"delete": {"_id":4}}
```

...except for “`delete`”, which is not followed by a document



# Retrieving a Document

- Use **GET** to retrieve an indexed document
  - notice we specify the **index** and an **ID**
  - response code is **200** if the document is found, **404** if not



# Retrieving Multiple Documents

- The `_mget` endpoint allows you to **GET** multiple documents in a single request
  - Avoid multiple round trips

```
GET _mget
{
  "docs" : [
    {
      "_index" : "comments",
      "_id" : 3
    },
    {
      "_index" : "blogs",
      "_id" : "F1oSq2EBO0ytt5ZTHpaE"
    }
  ]
}
```

Use the `_mget` endpoint

“`docs`” is an array of documents to GET



Elasticsearch Fundamentals

Lesson 3

# Review - CRUD Operations



# CRUD Operations

Index

```
POST my_blogs/_doc
{
  "title" : "Elasticsearch released",
  "category": "Releases"
}
```

Create

```
PUT my_blogs/_create/4
{
  "title" : "Elasticsearch released",
  "category": "Releases"
}
```

Read

```
GET my_blogs/_doc/4
```

Update

```
POST my_blogs/_update/4
{
  "doc" : {
    "title" : "Elasticsearch 6.2 released"
  }
}
```

Delete

```
DELETE my_blogs/_doc/4
```

# Summary

- A **document** is a serialized JSON object that is stored in Elasticsearch under a unique ID
- An **index** in Elasticsearch is a **logical** way of grouping data
- The **Bulk API** makes it possible to perform many write operations in a single API call, greatly increases the indexing speed

# Quiz

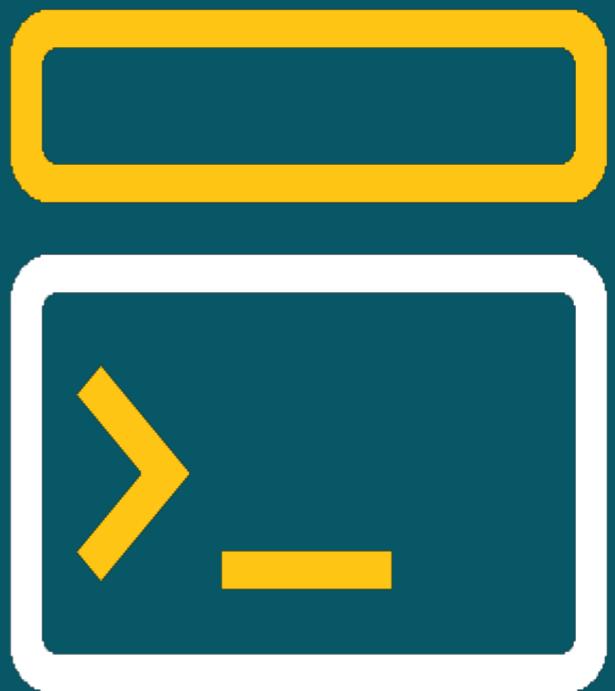
- 1. True or False:** If an index does not exist when indexing a document, Elasticsearch will automatically create the index for you.
- 2.** What happens if you index a document and the `_id` already exists in the index?
- 3. True or False:** Using the Bulk API is more efficient than sending multiple, separate requests.



Elasticsearch Fundamentals

Lesson 3

# Lab - CRUD Operations





Elasticsearch Fundamentals

Lesson 4

# Searching Data



# Datasets

# Static Data vs. Time Series Data

- In general, we can categorize most data in our users' use cases as one of the following:
  - **(relatively) static data:** a large (or small) dataset that may grow or change slowly, like a catalog or inventory of items
  - **time series data:** event data associated with a moment in time that typically grows rapidly, like log files or metrics
- Elasticsearch works great for both types of data
  - and therefore we will use two datasets in the course...

# Static Dataset

- Our static dataset is a collection of Elastic blog posts:



16 JANUARY 2018 **ENGINEERING**

**Performance Impact of Meltdown on Elasticsearch**

By Elastic Engineering

What's the impact of the kernel patches for the Meltdown vulnerability on Elasticsearch?

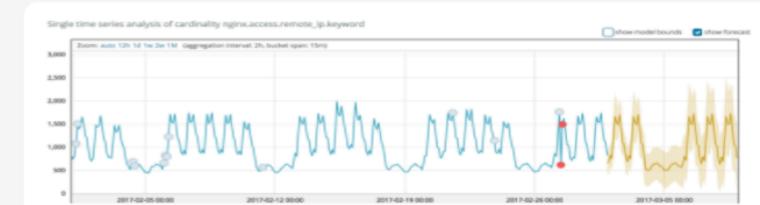


12 JANUARY 2018 **ENGINEERING**

**Document-Level Attribute-Based Access Control with X-Pack 6.1**

By Mike Barretta

Thanks to a new feature in Elasticsearch 6.1, attribute-based access control is among the X-Pack security features. Learn about what it is and why you need it!



10 JANUARY 2018 **ENGINEERING**

**On-demand forecasting with machine learning in Elasticsearch**

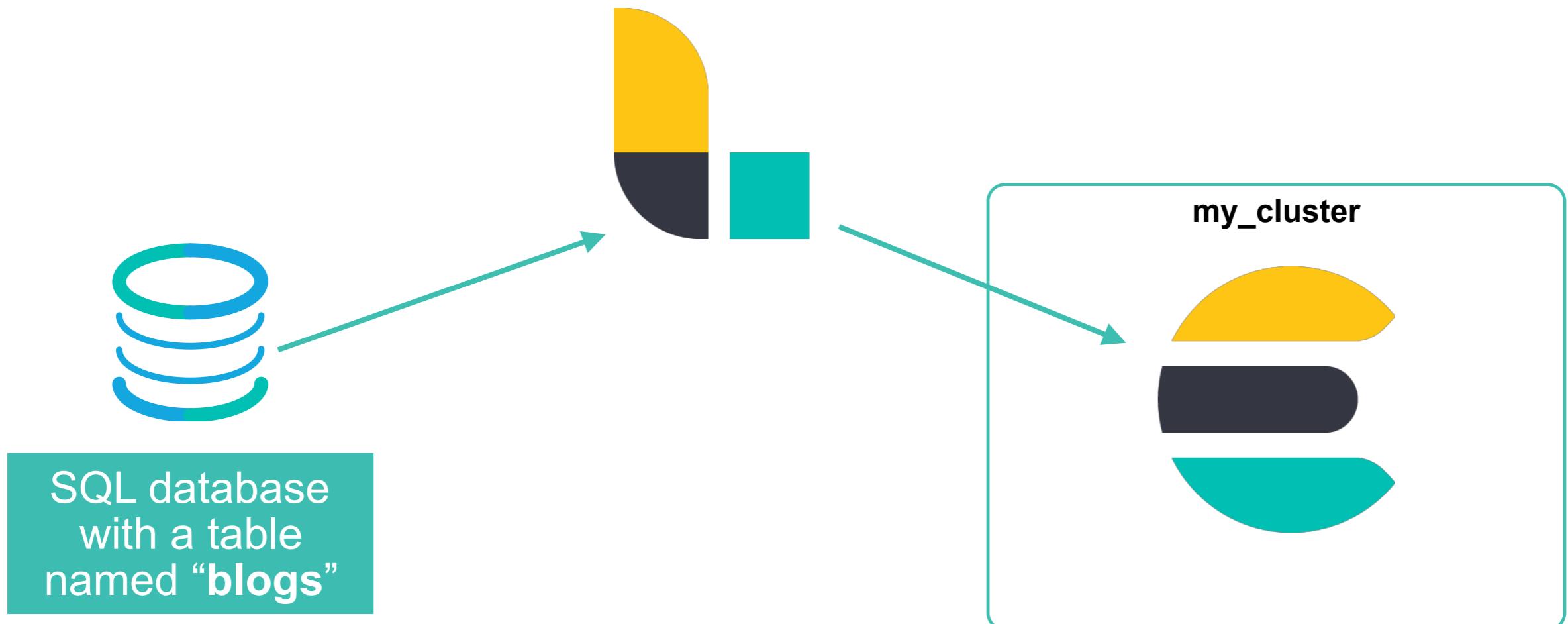
By Hendrik Muhs • Thomas Grabowski

The newest X-Pack feature in 6.1 is on-demand forecasting. Machine learning can model the data and predict multiple time intervals into the future.



# Ingesting the Blogs

- The blogs are stored in a SQL database
  - Logstash can easily index those into Elasticsearch



# What do we want to build with our data?

- We want users to be able to search our blogs
  - and get relevant and meaningful search results



# elastic

**Categories**

- (119)
- Engineering (106)
- Releases (73)
- This week in Elasticsearch and Apache Lucene (70)
- The Logstash Lines (41)

**Dates (dd/mm/yyyy)**

from:

to:

Sort by:

There are 529 results for "logstash" (13 milliseconds)

**Logstash 5.0.0 Released [ \_score: 11.896213 ]**  
October 26, 2016 [Releases]  
availability of the biggest release of Logstash yet., Previously, Logstash used /opt/logstash directory to install the binaries, whereas Elasticsearch used, Logstash 5.0 is compatible with Elasticsearch 5.x, 2.x, and even 1.x., Extracting fields from unstructured data is a popular Logstash feature., So, from the entire Logstash team, thank you to our users for using and contributing back to Logstash

**Welcome Jordan & Logstash [ \_score: 11.826626 ]**  
August 27, 2013 [ ]  
is amazing news for so many Elasticsearch and Logstash users., About Logstash Logstash, which just released version 1.2.0, is one of the most popular open source logs, have received requests to offer commercial support for Logstash ., Logstash shares the same vision., Effectively, Logstash is a generic system to process events.

**Logstash 1.4.0 beta1 [ \_score: 11.775666 ]**  
February 19, 2014 [ ]  
We are pleased to announce the beta release of Logstash 1.4.0!, Contrib plugins package Logstash has grown brilliantly over the past few years with great contributions, Now having 165 plugins, it became hard for us (the Logstash engineering team) to reliably support all, A bonus effect of this decision is that the default Logstash download size shrank by 19MB compared to, Going forward, Logstash release cycles will more closely mirror Elasticsearch's model of releases.

« 1 2 3 4 5 »

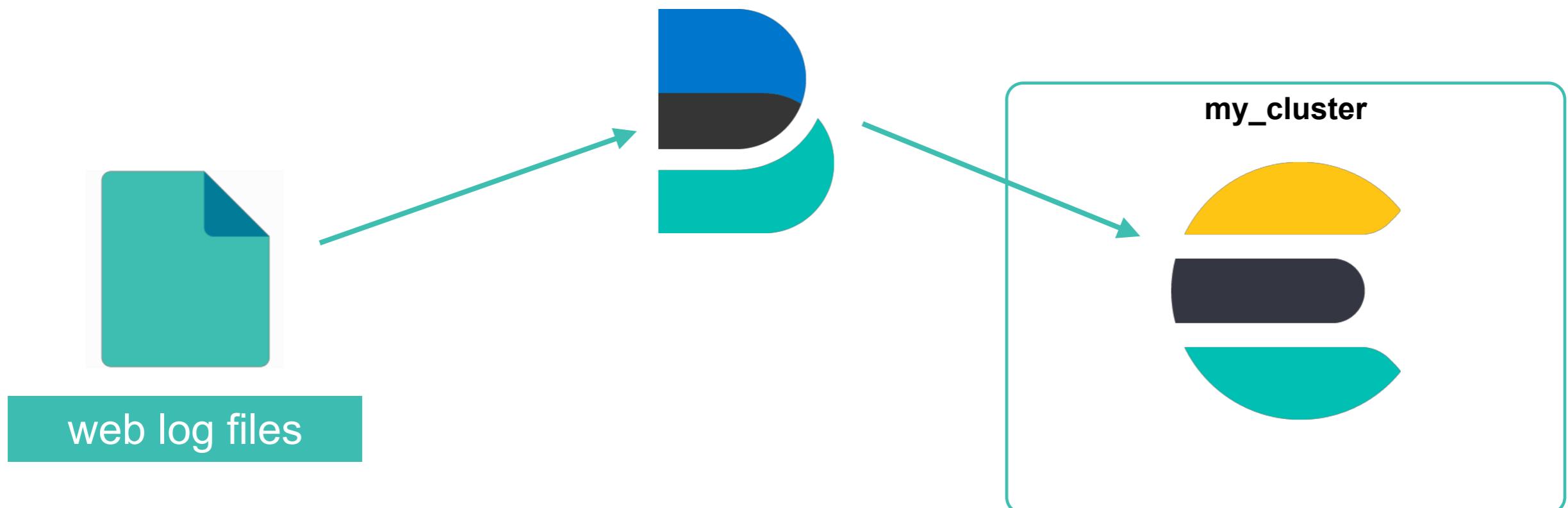
# Time Series Dataset

- For time series, we are going to use the web access log files from [elastic.co/blog](https://elastic.co/blog):

```
{  
    "@timestamp" : "2017-05-11T01:26:16.590Z",  
    "user_agent" : "Amazon CloudFront",  
    "originalUrl" : "/kr/blog/0-17-1-released",  
    "response_size" : 40534,  
    "host" : "server1",  
    "geoip" : {  
        "country_code2" : "US",  
        "country_name" : "United States",  
        "continent_code" : "NA",  
        "country_code3" : "US",  
        "location" : {  
            "lon" : -77.4728,  
            "lat" : 39.0481  
        },  
        "region_name" : "Virginia",  
        "city_name" : "Ashburn"  
    },  
    "status_code" : 200,  
    "level" : "info",  
    "method" : "GET",  
    "runtime_ms" : 224,  
    "http_version" : "1.1",  
    "language" : {  
        "url" : "/blog/0-17-1-released",  
        "code" : "ko-kr"  
    }  
}
```

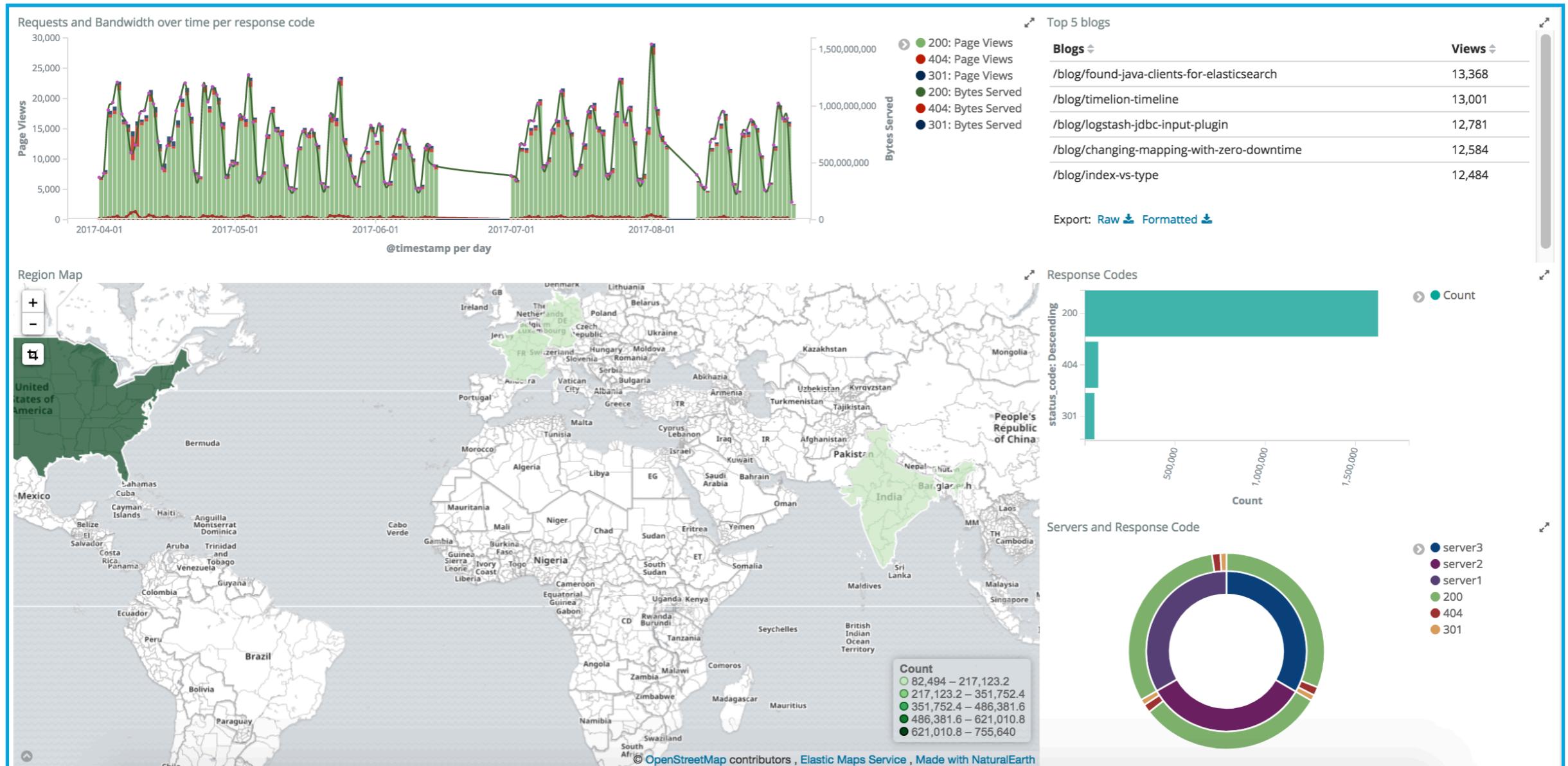
# Ingesting the Log Data

- The log events are in log files
  - can be easily ingested using Filebeat, which tails the log files



# What do we want from our log data?

- To be able to answer questions about web traffic:



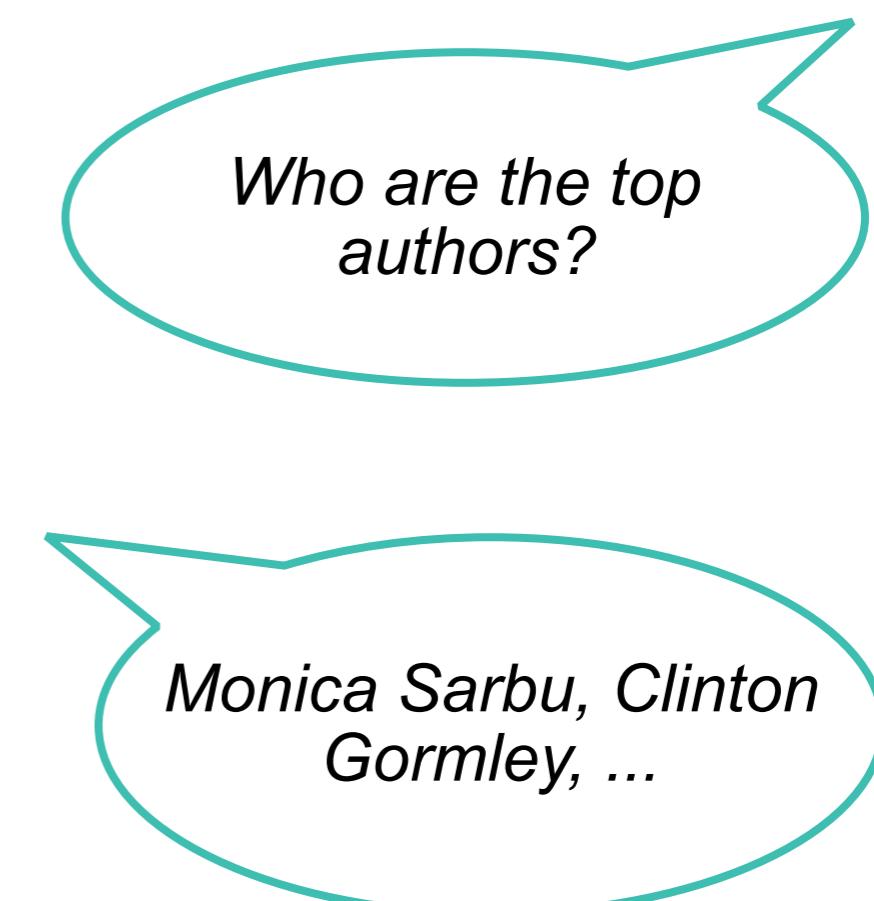
# Search

# Search

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



*What blogs have  
“community” in the title?*



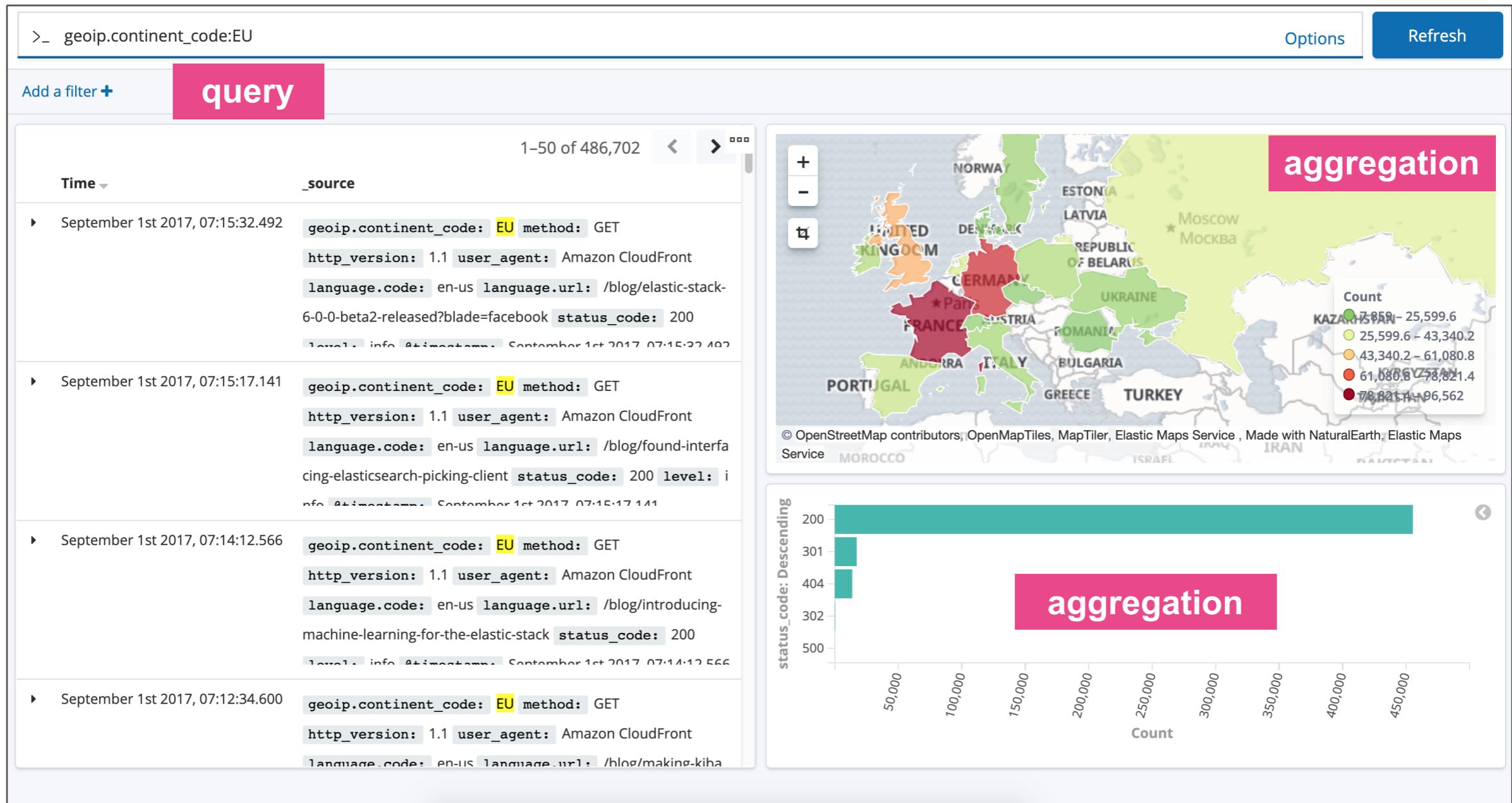
*Who are the top  
authors?*



*Community health  
metrics part one, Weekly Beats:  
more community Beats, ...*

# Queries and Aggregations

- What is the difference between queries and aggregations?



# Queries and Aggregations Request

- Kibana is a great visualization tool...
  - but it relies on Elasticsearch to execute queries and aggregations
- Send a **GET** request using the **\_search** endpoint

```
GET blogs/_search
{
  "query": {
    "match": {
      "title": "community"
    }
  },
  "aggregations": {
    "top_authors": {
      "terms": {
        "field": "author"
      }
    }
  }
}
```

**\_search** = the search endpoint

**query** = the query clauses to match documents

**aggregations** = the aggregation clauses to summarize the data

# Queries and Aggregations Response

- Kibana uses the Elasticsearch response to build visuals

```
{  
  "took" : 3,  
  ...  
  "hits" : {  
    "total" : {...},  
    "max_score" : 5.7130966,  
    "hits" : [  
      ...  
    ]  
  },  
  "aggregations" : {  
    "top_authors" : {  
      "buckets" : [  
        ...  
      ]  
    }  
  }  
}
```

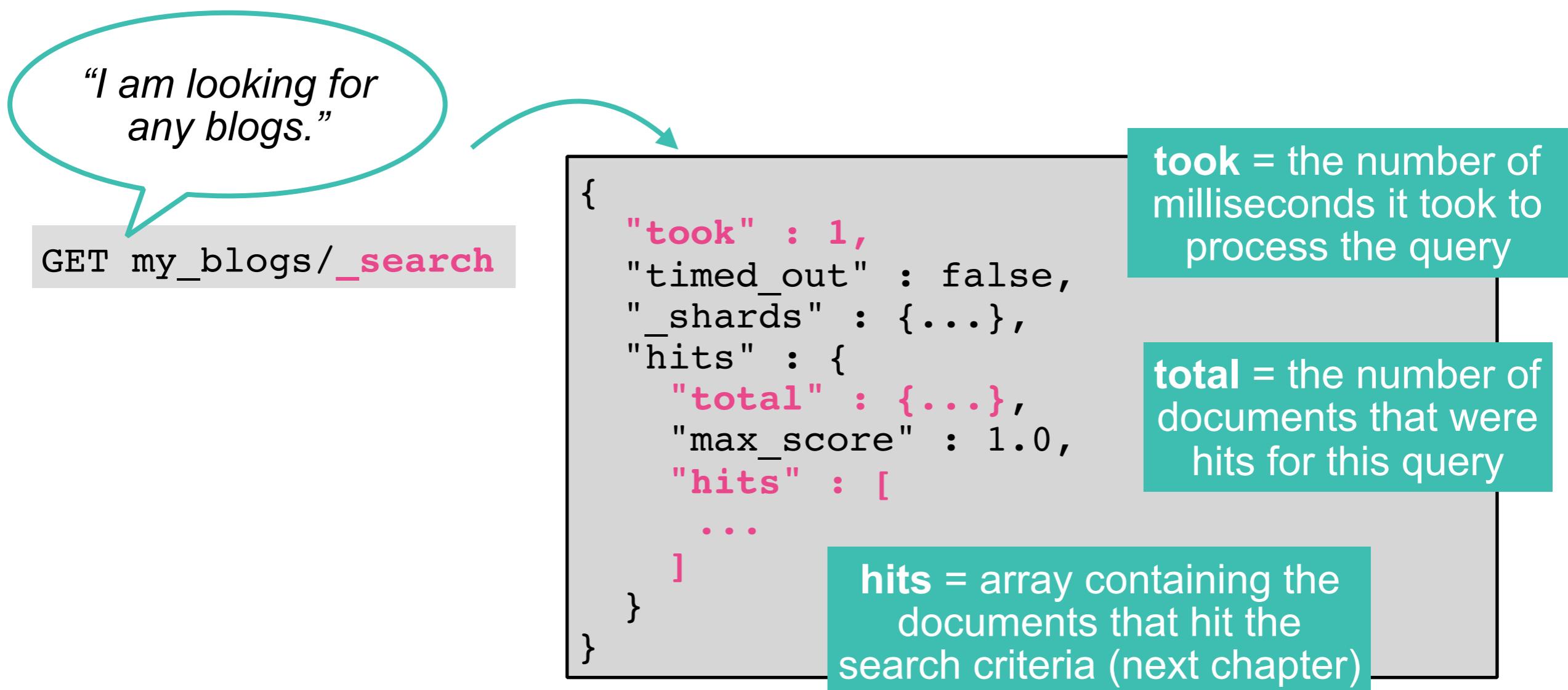
**hits** = array containing the documents that hit the search criteria

**buckets** = array containing the top authors

We will discuss the *aggregations and query* clauses in the next modules...

# 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



# Search Examples

```
GET logs_server1,logs_server2/_search
```

```
GET blogs/_search
{
  "query": {
    "match": {
      "author": "monica"
    }
  }
}
```

```
GET blogs/_search
{
  "query": {
    "range": {
      "publish_date": {
        "gte": "2018-01-01"
      }
    }
  }
}
```

```
GET logs_server*/_search
```

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        "match": {
          "author": "monica"
        }
      ],
      "filter": {
        "range": {
          "publish_date": {
            "gte": "2016-01-01"
          }
        }
      }
    }
  }
}
```



Elasticsearch Fundamentals

Lesson 4

# Review - Searching Data



# Summary

- In general, we can categorize most data in our users' use cases as one of the following:
  - **static data**
  - **time series data**
- In Elasticsearch, there are two main ways to search:
  - queries
  - aggregations
- By default, a search request only returns the first 10 hits

# Quiz

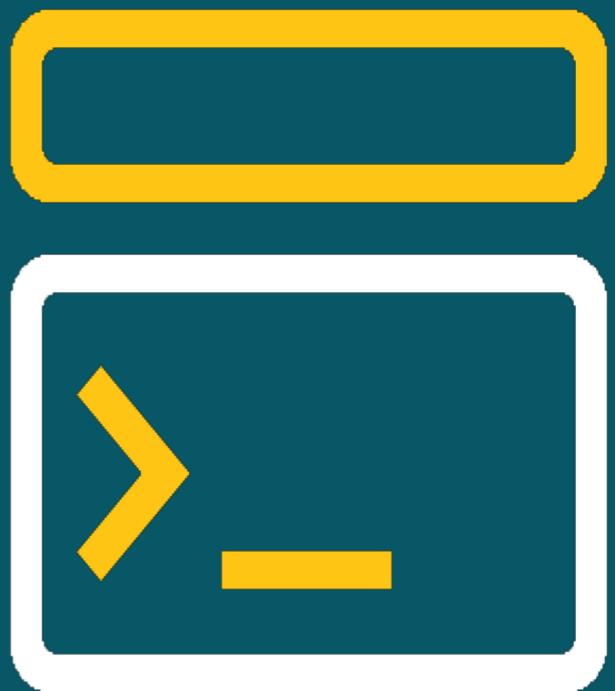
1. In general, most data in our users' use cases can be categorized into what two types of data?
2. What are the two main ways to search data in Elasticsearch?
3. What is the ***default number of hits*** that a query returns?



Elasticsearch Fundamentals

Lesson 4

# Lab - Searching Data



- Elasticsearch Fundamentals
- Elasticsearch Queries
- Elasticsearch Aggregations
- Elasticsearch Text Analysis and Mappings
- Elasticsearch Nodes and Shards
- Elasticsearch Monitoring and Troubleshooting

Module 2

# Elasticsearch Queries



# Topics

- Relevance
- Full-Text Queries
- Combining Queries
- Implementing a Search Page



Elasticsearch Queries

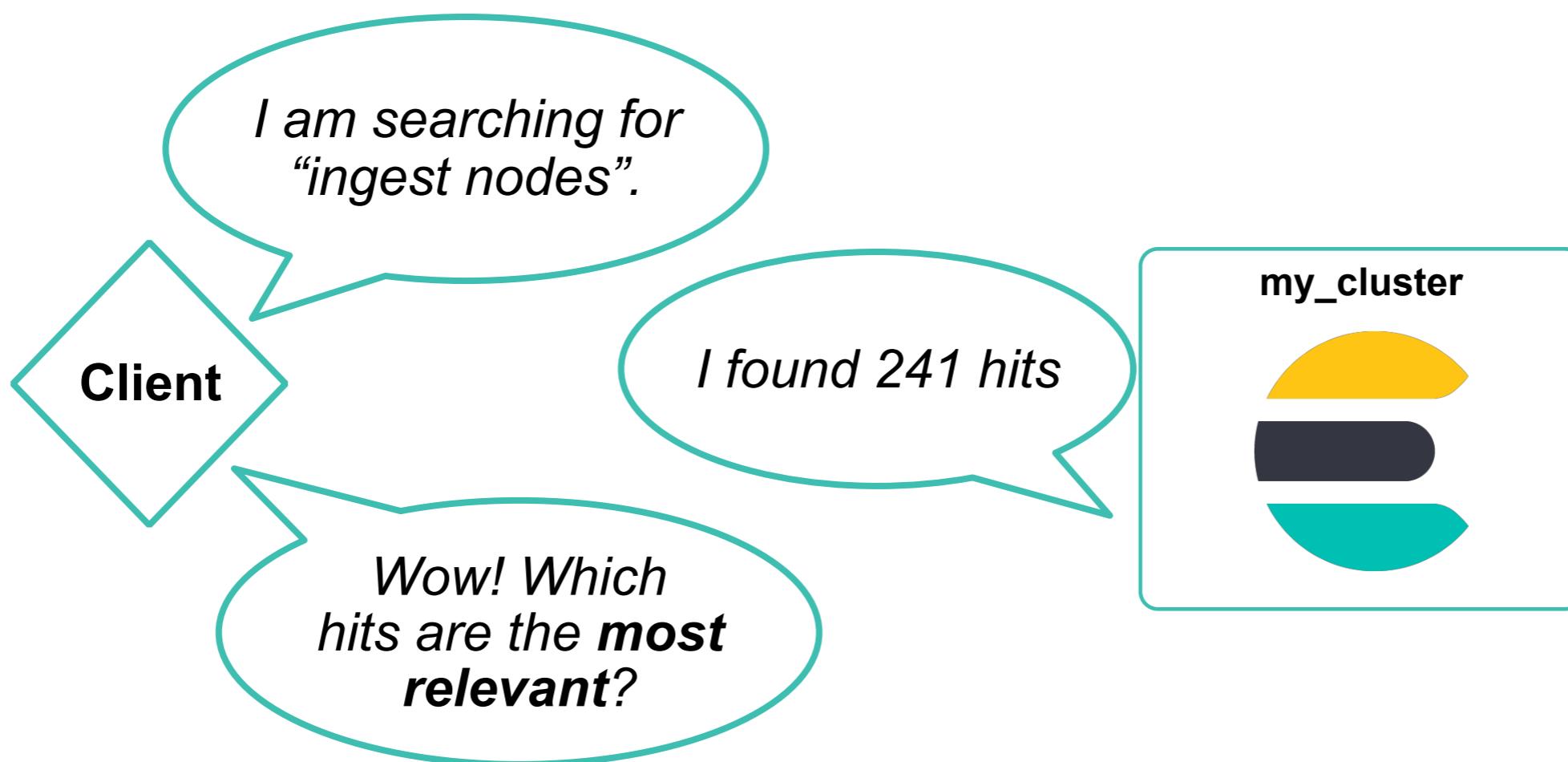
# Lesson 1

# Relevance



# Relevance

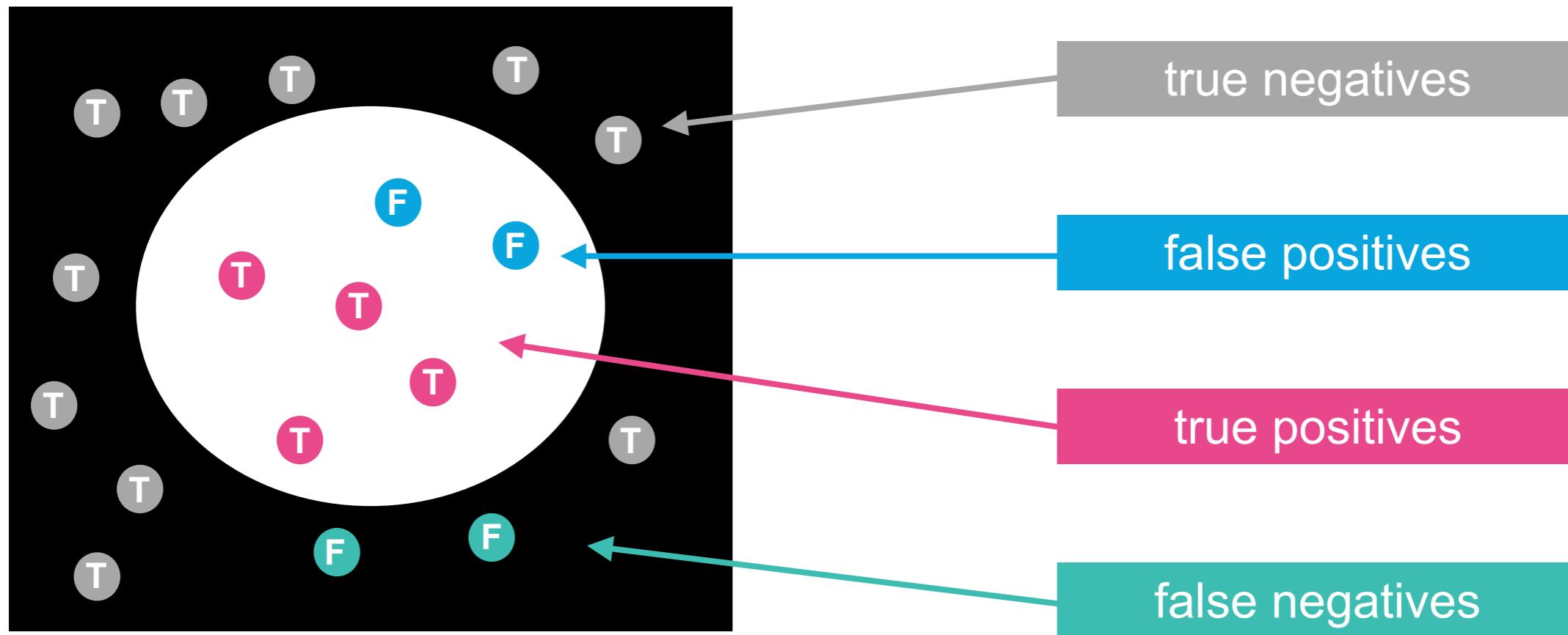
- When querying data, we are interested in the most *relevant* documents
  - are you able to find all the documents you are looking for?
  - how many irrelevant documents are returned?
  - how well are the documents ranked?



# Measuring Relevance

- **Recall**
  - did we miss any relevant results that were not returned?
- **Precision**
  - did we get irrelevant results in addition to the relevant results?
- **Ranking**
  - are the results ordered from most relevant at the top, to least relevant at the bottom?
- By default, Elasticsearch ranks the matching documents according to the score it calculates

# Recall and Precision



- **Recall** is the ratio of **true positives** versus the sum of all documents that **should have been returned** (true positives plus false negatives)
- **Precision** is the ratio of **true positives** versus the total number of docs that **were returned** (true positives plus false positives)

# Improving Recall and Precision

- **Recall** can be improved by "widening the net":
  - using queries that also return *partial* or *similar* matches
  - ... *but makes precision worse*
- **Precision** can be improved by making searches more strict:
  - using queries that only return **exact** matches
  - ... *but makes recall worse*
- We will see many queries and options that either improve recall or precision

# Let's Search for Terms

- Suppose we are interested in **learning** about
  - “*ingest nodes*”
- Let's search for it in the “**content**” field
- What do you think is required of a document to be a hit?

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

# Match Uses **or** Logic

- By default, the **match** query uses “**or**” logic if multiple terms appear in the search query
  - any document with the term “ingest” **or** “nodes” in the “**content**” field will be a hit

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

*Find blogs that mention “ingest” **or** “nodes”.*

# Query Response

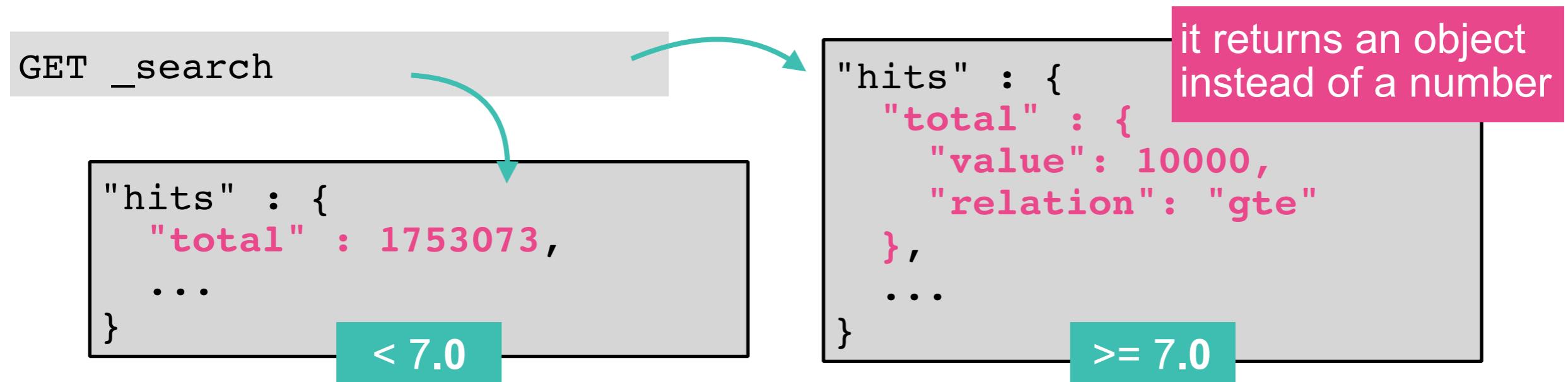
- By default, the documents are returned sorted by `_score`

```
"hits" : {  
  "total" : {  
    "value" : 241, ← 241 documents were a match  
    "relation" : "eq"  
  },  
  "max_score" : 8.739951,  
  "hits" : [  
    {  
      ...  
      "_source" : {  
        "title" : "Ingest Node: A Client's Perspective"  
      }  
    },  
    {  
      ...  
      "_source" : {  
        "title" : "A New Way To Ingest - Part 2"  
      }  
    },  
    {  
      ...  
      "_source" : {  
        "title" : "A New Way To Ingest - Part 1"  
      }  
    }  
  ]  
}
```

but only the top 10 documents will be returned by default

# Total Hits

- Since 7.0, Elasticsearch limits the total counts to 10,000
  - it improves performance



- The relation field indicates if the **value** is accurate (**eq** / **gte**)
- You can use the **track\_total\_hits** parameter to always return the accurate total value

```
GET _search
{
  "track_total_hits": true
}
```



# Analyze the Hits

- Notice the "or" logic led to some hits that we may not have been interested in:
  - out of 241 hits

Logstash 6.0.0 GA Released

This is the 4th hit, but doesn't really cover ***ingest nodes***

Elasticsearch for Apache Hadoop 1.3 M2 released

This is the 11th hit, but doesn't even mention ***ingest nodes***

Integrating the Elastic Stack with ArcSight SIEM - Part 3

This is the 12th hit, but doesn't even mention ***ingest nodes***

# How can we increase precision?

- The “or” logic can be changed to “and” in a **match** query using the **operator** parameter
  - notice the slightly different syntax for **match** (we had to add the “**query**” parameter)

```
GET blogs/_search
{
  "query": {
    "match": {
      "content": {
        "query": "ingest nodes",
        "operator": "and"
      }
    }
  }
}
```

Precision is improved  
and recall is decreased  
(only 25 hits now)

Change “operator” to “and”

# Let's Add More Terms:

- Let's try a search with three terms:

```
GET blogs/_search
{
  "query": {
    "match": {
      "content": {
        "query": "ingest nodes logstash"
      }
    }
  }
}
```

The “or” logic gives us 690 hits

# The `minimum_should_match` Property

- If a `match` query searches on multiple terms, the “`or`” or “`and`” options might be too wide or too strict
  - use the `minimum_should_match` parameter to trim the long tail of less relevant results

```
GET blogs/_search
{
  "query": {
    "match": {
      "content": {
        "query": "ingest node logstash",
        "minimum_should_match": 2
      }
    }
  }
}
```

2 of the 3 terms  
need to match

# Precision is improved

- How is the score calculated?

```
"hits" : {  
  "total" : {  
    "value" : 82, ← only 82 documents were  
    "relation" : "eq" a match (vs. 695)  
  },  
  "max_score" : 10.1513195,  
  "hits" : [  
    {  
      ...  
      "_score" : 10.1513195,  
      "_source" : {  
        "title" : "Logstash 6.0.0 GA Released"  
      }  
    },  
    {  
      ...  
      "_score" : 10.147091,  
      "_source" : {  
        "title" : "Logstash 6.0.0-beta1 released"  
      }  
    },  
    ...  
  ]  
}
```

Each hit has a `_score`

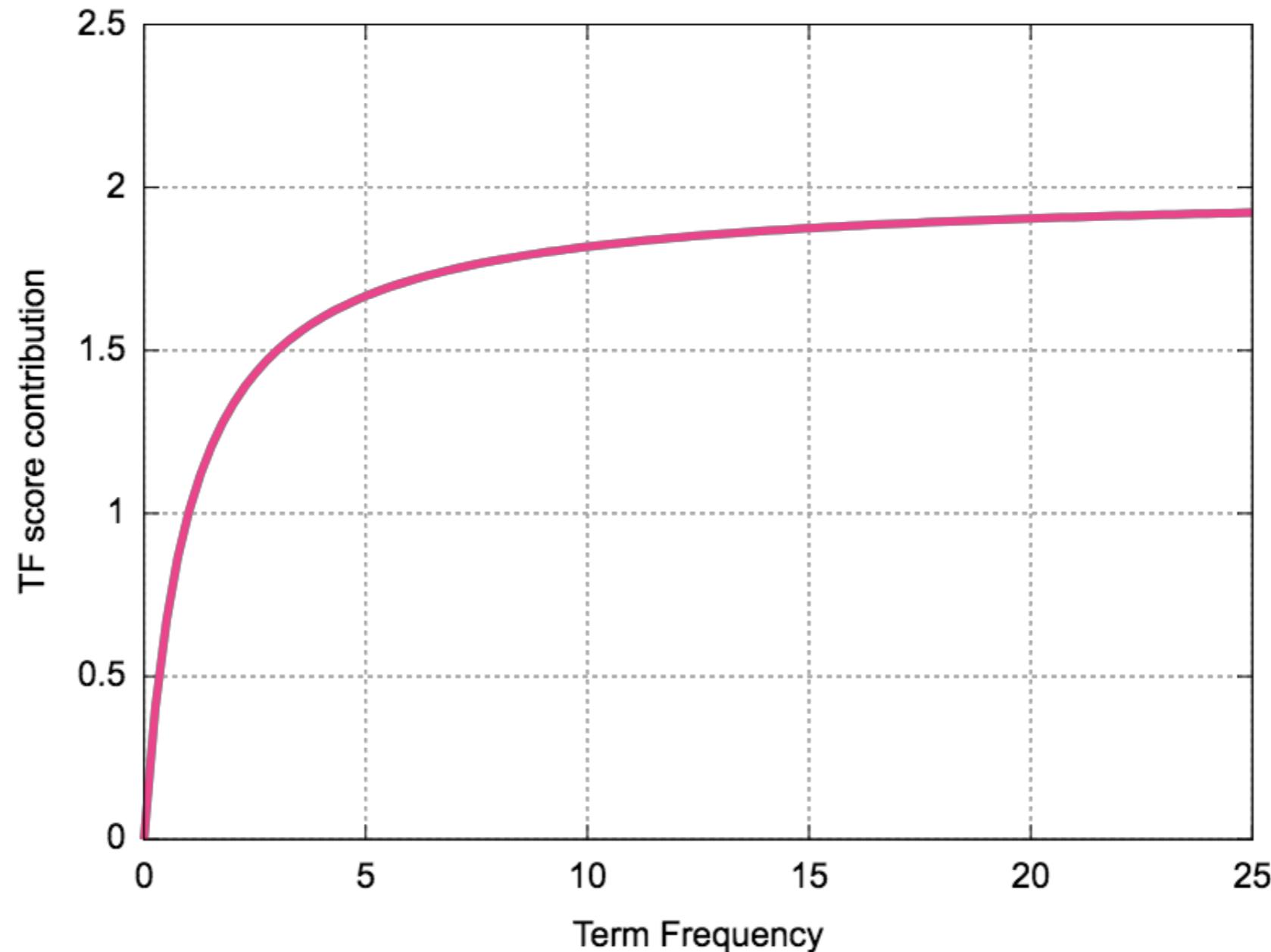
Results are sorted by  
`_score` (by default)

# Score!

- The **\_score** is a value representing how relevant the document is in regards to that specific query
  - a **\_score** is computed for each document that is a hit
- Elasticsearch's default scoring algorithm is **BM25**
- There are three main factors of a document's score:
  - **TF (term frequency)**: The more a term appears in a field, the more important it is
  - **IDF (inverse document frequency)**: The more documents that contain the term, the less important the term is
  - **Field length**: shorter fields are more likely to be relevant than longer fields

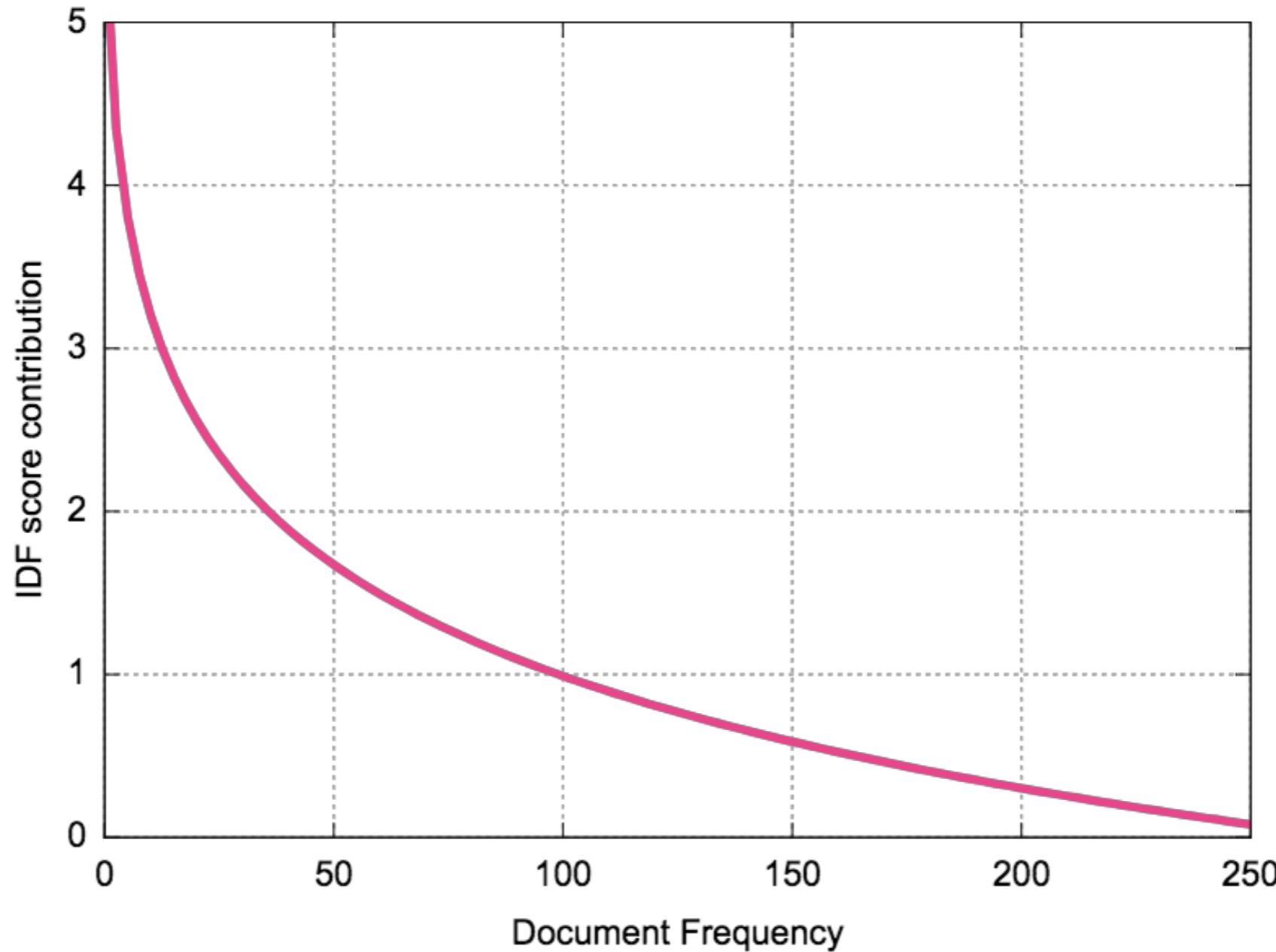
# Term Frequency

- The more a term appears in a field, the more important it is



# Inverse Document Frequency

- The more documents contain the term, the less important the term is





Elasticsearch Queries

Lesson 1

# Review - Relevance



# Summary

- **Recall** is the portion of relevant documents that are returned in the results
- **Precision** is the probability that a document in the results is relevant
- **Ranking** is an ordering of the documents in the results according to relevance
- The **match** query is a simple but powerful search

# Quiz

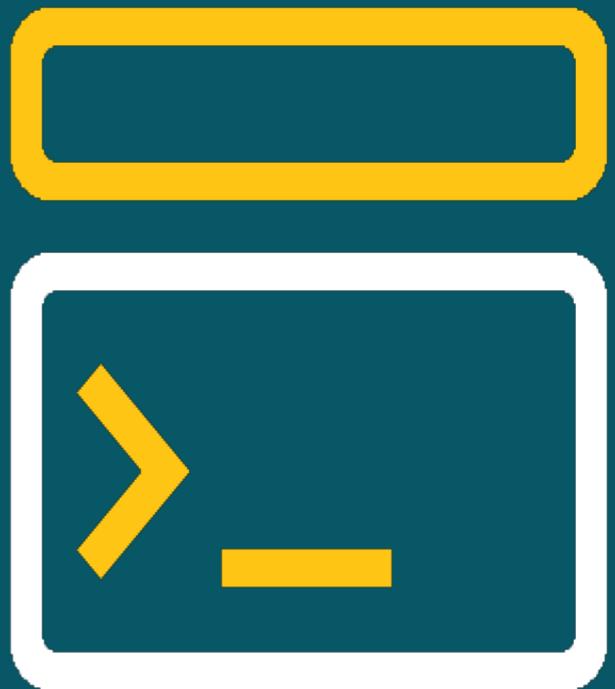
- 1. True or False:** In a match query, you should use the **AND** operator to improve relevance.
2. How could you improve the precision of a **match** query that consists of 5 terms?
- 3. True or False:** If a queried term appears 100 times in document A and 90 times in document B, then document A has a much higher score than document B.



Elasticsearch Queries

Lesson 1

# Lab - Relevance





## Lesson 2

# Full-Text Queries



# Searching for Phrases

- The match query did an OK job of searching for “**ingest nodes**”
  - but the order of terms was not taken into account:

Logstash 6.0.0 GA Released

Elasticsearch for Apache Hadoop 1.3 M2 released

Integrating the Elastic Stack with ArcSight SIEM - Part 3

Blogs that often mention “**ingest**” scored high, even though they might not mention “**ingest nodes**”

# 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

```
GET my_index/_search
{
  "query": {
    "match_phrase": {
      "FIELD": "PHRASE"
    }
  }
}
```

The field you want to search

The phrase you are searching for

# Example of match\_phrase

- Let's try the “*ingest nodes*” search using **match\_phrase** instead of **match**:
  - only 8 hits this time
  - be careful, relevant blogs are not returned
  - precision is much better but recall is much worse

```
GET blogs/_search
{
  "query": {
    "match_phrase": {
      "content": "ingest nodes"
    }
  }
}
```

Two things must happen for “**ingest nodes**” to cause a hit:

- “**ingest**” and “**nodes**” must appear in the “**content**” field
- The terms must appear in that order and next to each other

# Another Example

- Let's try a different search
  - suppose we are interested in blogs about “open data”:

```
GET blogs/_search
{
  "query": {
    "match_phrase": {
      "content": "open data"
    }
  }
}
```

Only 5 hits

“Open data at the local government level  
may ...”

“files that can be sourced from Météo  
France’s open data platform...”

“publicly available at the European Union  
Open Data Portal”

“Open data at the local government level  
may seem “smaller” ...”

“...improved your applications with (open)  
data “

# The slop Parameter

- If you want to increase the recall of a **match\_phrase**, you can introduce some flexibility into the phrase (called **slop**)
  - the **slop** parameter tells how far apart terms are allowed to be while still considering the document a match

```
GET blogs/_search
{
  "query": {
    "match_phrase": {
      "content": {
        "query": "open data",
        "slop": 1
      }
    }
  }
}
```

9 hit this time!

"Open data at the local government level  
may ..."

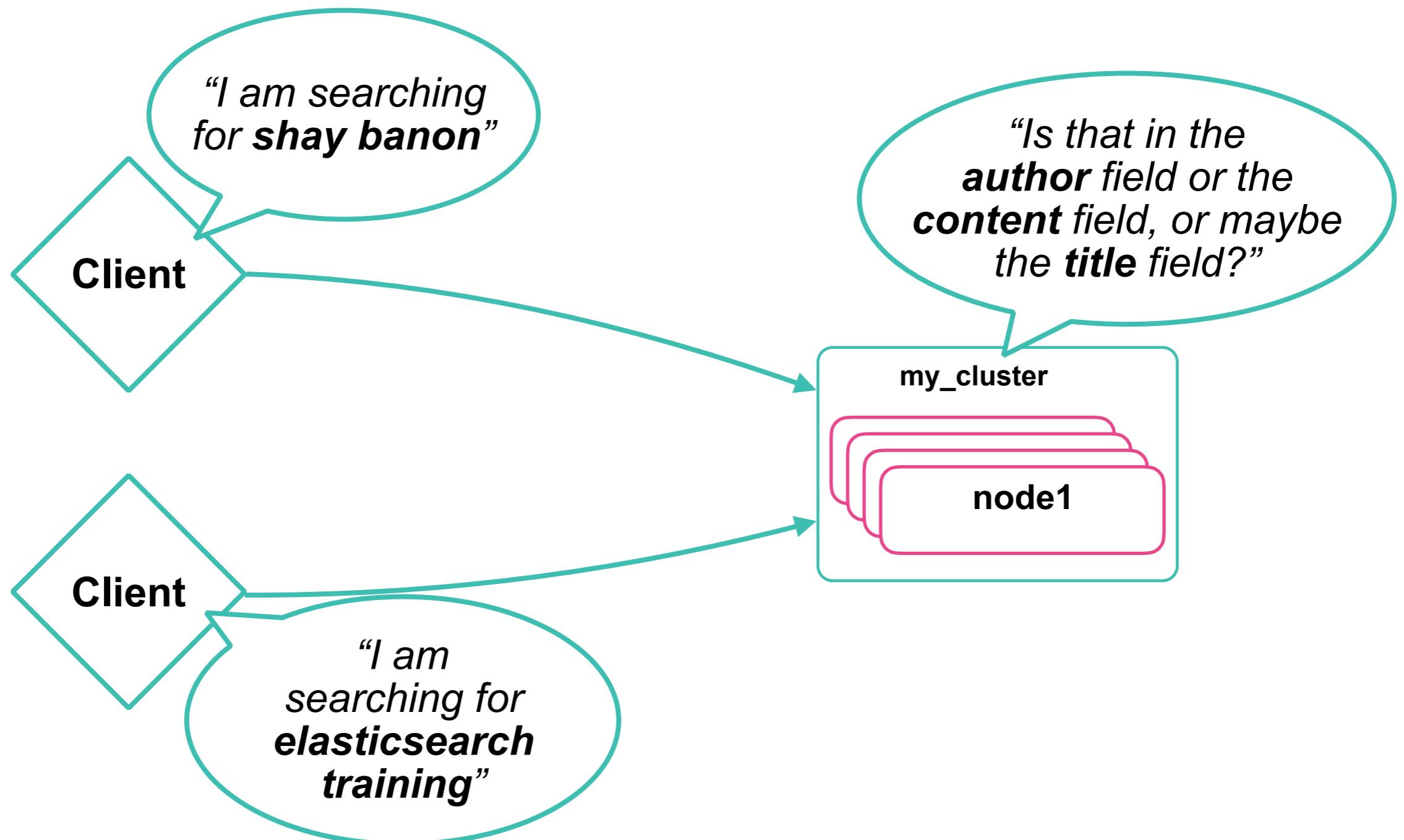
....

"building lightweight, open source data  
shippers"

Hits now include phrases  
with intervening terms

# We don't always know what a user wants

- When a user searches for something, it is not always known what *the context of the search* is:



# Searching Multiple Fields

- The solution?
  - why not query all three of those fields? (author, title **and** content)
- The **multi\_match** query provides a way to accomplish this...

# 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**)

```
GET blogs/_search
{
  "query": {
    "multi_match": {
      "query": "shay banon",
      "fields": [
        "title",
        "content",
        "author"
      ],
      "type": "best_fields"
    }
  }
}
```

3 fields are queried (which results in 3 scores) and the **best** score is used

# Let's analyze the top hits of our query...

A total of 151 hits...do the top 3 seem relevant?

"\_score": 11.740686

"author": "Shay Banon"

"title": "Alibaba Cloud to Offer Elasticsearch, Kibana, and X-Pack in China"

"content": " Heya (Ni Hao) Simon Hu, President of Alibaba Cloud and **Shay Banon**, CEO of

"\_score": 11.066599

"author": "Rashmi Kulkarni"

"title": "Second Annual Elastic{ON} Women's Breakfast: Now with Leadership Panel"

"content": " At Elastic, we take the philosophy ... Co-Founder and CTO, **Shay Banon**. ... Soup Company. **Shay Banon** is the creator of Elasticsearch, ... "

"\_score": 10.613635

"author": "Steven Schuurman"

"title": "The Next Chapter: Shay Banon to Succeed Steven Schuurman as Elastic CEO"

"content": " Rarely do you make decisions you ... founder **Shay Banon** I ... why. **Shay** understood ... soulmate, **Shay**. Even ... "

# Per-field Boosting

- It seems like a match in the blog's **title** should carry more weight than the **content** field
  - you can *boost the score of a field* using the caret (^) symbol
- We get the same 151 hits, but the top hits are different:

```
GET blogs/_search
{
  "query": {
    "multi_match": {
      "query": "shay banon",
      "fields": [
        "title^2",
        "content",
        "author"
      ]
    }
  }
}
```

boost title by 2



"_score": 21.22727
"author": "Steven Schuurman"
"title": "The Next Chapter: <b>Shay Banon</b> to Succeed Steven Schuurman as Elastic CEO"
"_score": 11.740686
"author": "Shay Banon"
"title": "Alibaba Cloud to Offer Elasticsearch, Kibana, and X-Pack in China"
"_score": 11.066599
"author": "Rashmi Kulkarni"
"title": "Second Annual Elastic{ON} Women's Breakfast: Now with Leadership Panel"



# Is there a best practice for boosting?

- Well, it depends!
  - experiment and analyze your search results to find the ideal usage
- For our blog data, search terms often appear in both the “**title**” and “**content**” fields
  - but a word in the “**title**” is probably a better hit
- “**author**” names rarely appear in the “**title**” or “**content**”
  - so boosting “**author**” is conveniently not needed because of how “**best\_fields**” works (which you can easily discover by running a few queries using author names)

# Let's try searching for a topic...

- Suppose we are looking for a blog that mentions “**elasticsearch training**”
  - notice our query contains the popular term “**elasticsearch**”, so there are a LOT of hits

```
GET blogs/_search
{
  "query": {
    "multi_match": {
      "query": "elasticsearch training",
      "fields": [
        "title^2",
        "content",
        "author"
      ]
    }
  }
}
```

1,260 hits

# Improve Precision with `match_phrase`

- The top hits are good, but the precision is not great
  - we should take into account the phrase “*elasticsearch training*”
- Let’s configure our `multi_match` query to use a `match_phrase` (instead of the default `match` behavior)

```
GET blogs/_search
{
  "query": {
    "multi_match": {
      "query": "elasticsearch training",
      "fields": [
        "title^2",
        "content",
        "author"
      ],
      "type": "phrase"
    }
  }
}
```

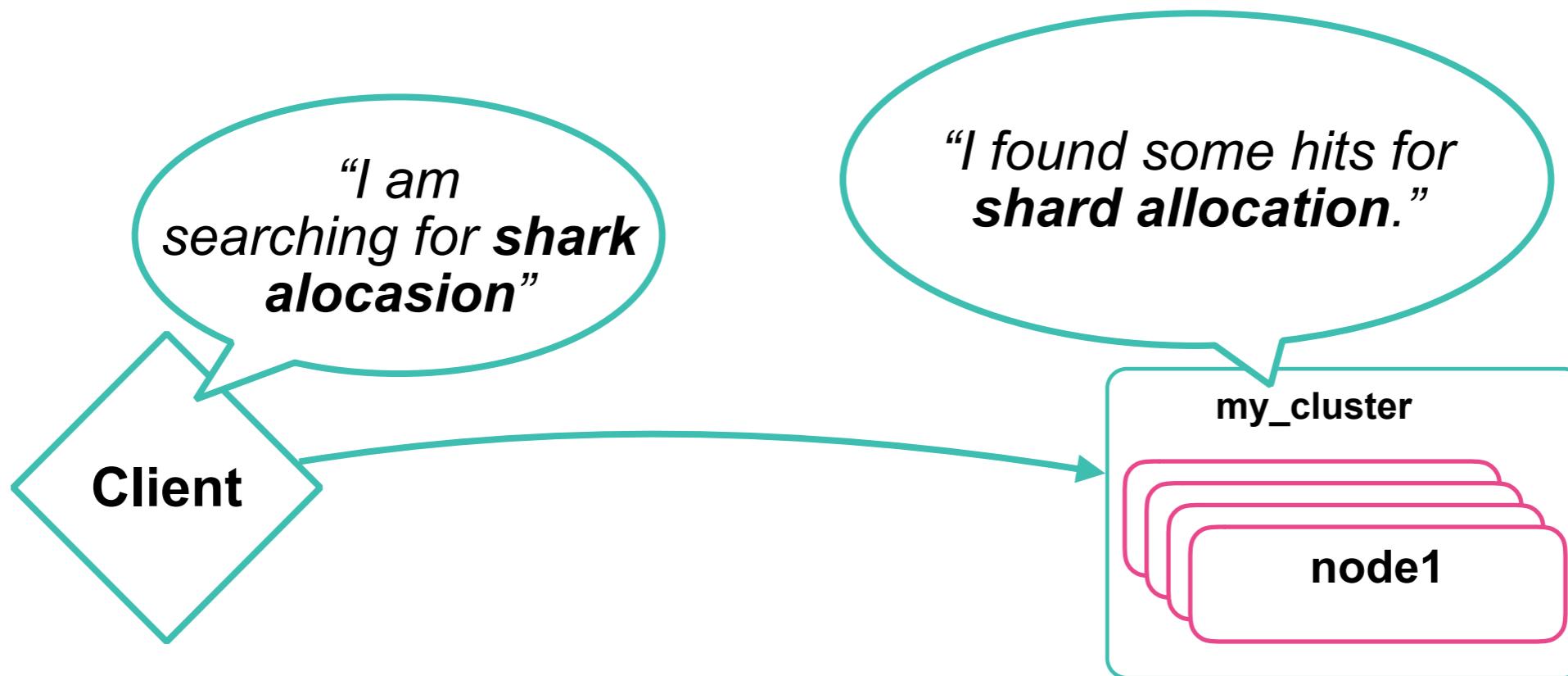
Uses `match_phrase`  
instead of `match`



Only 88 hits using  
“`phrase`”, with much  
improved precision at the  
cost of recall

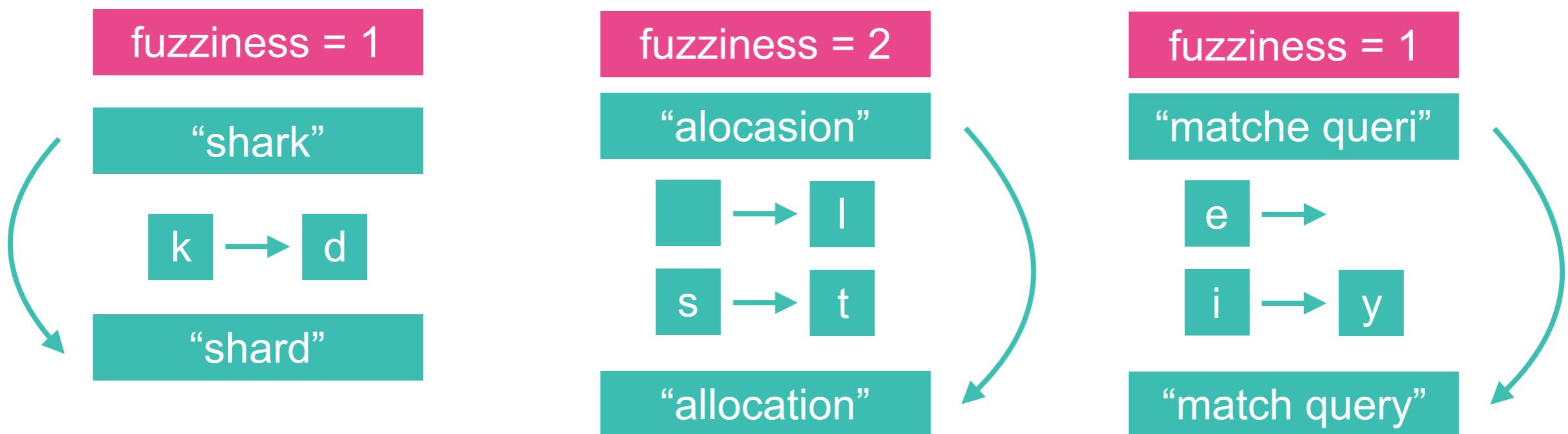
# Mispelt Werds

- Or more precisely, how to deal with “*misspelled words*”
  - in today’s world, we expect a search application to grant some leniency in terms of precisely matching
- We can add a level of *fuzziness* to our queries...



# What is Fuzziness?

- **Fuzzy matching** treats two words that are “**fuzzily**” similar as if they were the same word
  - edits are applied independently to each term in the query
  - it refers to the number of character modifications, known as **edits**, to make two words match



# Adding fuzziness to a Query

- The **match** query has a “**fuzziness**” property
  - can be set to 0, 1 or 2; or can be set to “**auto**”
- “**auto**” is the preferred way to use fuzziness
  - defines the distance based on the *length of the query terms*

```
80 hits  
GET blogs/_search  
{  
  "query": {  
    "match": {  
      "content": "shard"  
    } } }
```

```
0 hits  
GET blogs/_search  
{  
  "query": {  
    "match": {  
      "content": "shark"  
    } } }
```

```
418 hits  
GET blogs/_search  
{  
  "query": {  
    "match": {  
      "content": {  
        "query": "shark",  
        "fuzziness": 1  
      }  
    }  
  } }
```

# About Fuzziness

- Fuzziness is an easy solution to misspelling
  - but has high CPU overhead and very low precision
- Usually a good way to handle misspellings in the data
  - but a **bad way to handle misspellings in the queries**
- To properly handle user misspellings we recommend:
  - [Improving Search with Text Analysis](#)
  - [Improving Search with Suggestions](#)



Elasticsearch Queries

Lesson 2

# Review - Full-Text Queries



# Summary

- The **match\_phrase** query is for searching text when you want to find terms that are near each other
- The **multi\_match** query allows you to search the same terms in multiple fields
- Fuzziness is an easy solution to misspelling but has high CPU overhead and very low precision

# Quiz

1. Explain the difference between **match** and **match\_phrase**.
2. If you want to add some flexibility into **match\_phrase**, you can configure a \_\_\_\_\_ property.
3. Would the following query match a document that contains "**monitoring data**"?

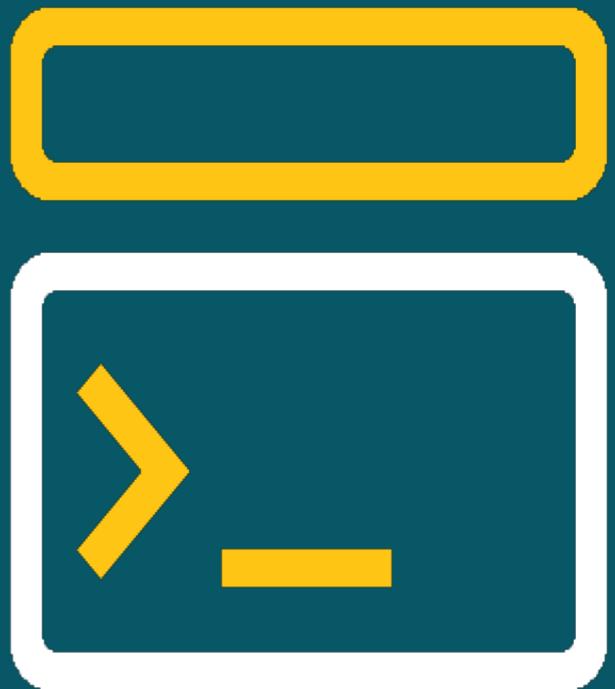
```
GET blogs/_search
{
  "query": {
    "match": {
      "content": {
        "query": "monitoring datu",
        "fuzziness": 1
      }
    }
  }
}
```



Elasticsearch Queries

Lesson 2

# Lab - Full-Text Queries





Elasticsearch Queries

Lesson 3

# Combining Queries



# Combining Searches

- Suppose we want to write the following query:
  - *find blogs about “Logstash” in the “Engineering” category*
- This search is actually a combination of two queries:
  - we need “Logstash” in the **content** or **title** field,
  - and “Engineering” 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

```
GET my_index/_search
{
  "query": {
    "bool": {
      "must": [
        {}
      ],
      "must_not": [
        {}
      ],
      "should": [
        {}
      ],
      "filter": [
        {}
      ]
    }
  }
}
```

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

# The must Clause

- We write a lot of blogs about product releases, which may not be relevant if you are looking for technical details
  - let's search for “Logstash” only in the “Engineering” category:

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "content": "logstash"
          }
        },
        {
          "match": {
            "category": "engineering"
          }
        }
      ]
    }
  }
}
```

101 hits

# The must\_not Clause

- Our previous query does not cast a wide net for “Logstash”
  - it might be better to *not* search for blogs about “Releases”

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "content": "logstash"
        }
      },
      "must_not": {
        "match": {
          "category": "releases"
        }
      }
    }
  }
}
```

449 hits

# The should Clause

- A match in a “should” clause increases the `_score` of hits
  - a very useful behavior that has a lot of use cases

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": {
        "match_phrase": {
          "content": "elastic stack"
        }
      },
      "should": {
        "match_phrase": {
          "author": "shay banon"
        }
      }
    }
  }
}
```

*“I am looking for blogs about the **Elastic Stack**, preferably from **Shay**.”*

This particular “should” clause does not add more hits, but blogs from **Shay** score the highest

# The filter Clause

- Some clauses should not change the score
  - they either don't impact ranking or you don't want them to

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": {
        "match_phrase": {
          "content": "elastic stack"
        }
      },
      "filter": {
        {
          "match": {
            "category": "engineering"
          }
        }
      }
    }
  }
}
```

The filter clause does not affect the `_score`

# Query vs. Filter

- We have seen query contexts, but there is another clause known as a *filter context*:
  - **Query context:** results in a `_score`
  - **Filter context:** results in a “yes” or “no”



# The *bool* Query

- A *bool* query is a combination of one or more of the following boolean clauses:

<i>Clause</i>	<i>Affects #hits</i>	<i>Affects _score</i>
must	Y	Y
must_not	Y	N
should	N*	Y
filter	Y	N

\* Yes in a special scenario that we will talk about later

# Improving Relevancy

- You can control a lot of precision and scoring within your **bool** queries
  - this section contains tips for how to control and improve the precision of your hits
  - as well as some clever uses of the **should** clause to cast a wider net while maintaining good rankings

# Lots of should clauses?

- Suppose we run a search for an article with “*Elastic*” in the title
  - and we want to favor articles that mention “**stack**”, “**speed**”, or “**query**”, so we add a few **should** clauses:

259 hits

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"title": "elastic"}}
      ],
      "should": [
        {"match": {"title": "stack"}},
        {"match_phrase": {"author": "shay banon"}},
        {"range": {"publish_date": {"lt": "2017-01-01"}}}
      ]
    }
  }
}
```

This will cast a wide net because no **should** clause needs to match



“F5 High Speed Logging with Elastic Stack”

“Elastic Stack 5.4.0 Released”

....

“Announcing the GA of Elastic Cloud on Google Cloud Platform (GCP), More Options to Host Elasticsearch”

# Use minimum\_should\_match

- We can control how many **should** clauses need to match by specifying the **minimum\_should\_match** parameter
  - let's improve precision by requiring at least 1 of our **should** clauses to match:

Only 71 hits this time

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"title": "elastic"}}
      ],
      "should": [
        {"match": {"title": "stack"}},
        {"match_phrase": {"author": "shay banon"}},
        {"range": {"publish_date": {"lt": "2017-01-01"}}}
      ],
      "minimum_should_match": 1
    }
  }
}
```



"_score": 9.721097	"F5 High Speed Logging with Elastic Stack"
"_score": 6.346445	"Elastic Stack 5.4.0 Released"
"_score": ...	....
"_score": 2.9527168	"Process transparency and error categorization at 1&1 by using the Elastic Stack and ETL"

# Only “should” Clauses

- If you have a **bool** query with no **must** or **filter** clauses, one or more **should** clauses must match a document:
  - a special scenario where **minimum\_should\_match** is at least 1

```
GET blogs/_search
{
  "query": {
    "bool": {
      "should": [
        {"match": {"title": "stack"}},
        {"match_phrase": {"author": "shay banon"}},
        {"range": {"publish_date": {"lt": "2017-01-01"}}}
      ]
    }
  }
}
```

At least one clause must match for a hit

# A Search Tip for Phrases

- Suppose we search for “***open data***”
  - there are a lot of hits, since “***open***” and “***data***” are common terms

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



852 hits

- we could use **match\_phrase** to improve the scoring
- we could use “**and**” instead of “**or**” to improve precision
- But both of those options narrow our search results...



# A Search Tip for Phrases

- We could improve the ranking, while at the same casting a wide net, by adding a **match\_phrase** in a **should** clause
  - so documents that actually have the phrase will score higher

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": {
        "match": {"content": "open data"}
      },
      "should": {
        "match_phrase": {"content": "open data"}
      }
    }
  }
}
```

We get the same 852 hits, but the scoring is higher if the phrase “open data” appears

# Other Ways to Query Elasticsearch

# Do I always need to use the query DSL?

- No, there are some simplified ways to query Elasticsearch
- Query String
  - simple and easy-to-use
  - unforgiving for small typos (quotes, parentheses, ...)
  - great for incident analysis

```
GET logs_server*/_search?q=geoip.city_name:(san jose)
```

```
GET logs_server*/_search?q=geoip.city_name:(san AND jose)
```

```
GET logs_server*/_search
{
  "query": {
    "query_string": {
      "query": "admin geoip.city_name:(\"san jose\") host:server1",
      "default_operator": "AND"
    }
  }
}
```

it is part of the Query DSL,  
but can be used in the URL

# What about Kibana?

- Query String supported (default <7.0)
  - no auto-completion
- Kibana Query Language (default 7.0+)
  - similar syntax
  - auto-completion (Basic)

The image contains four separate screenshots of the Kibana interface, each showing a different type of query:

- Top Screenshot (Lucene):** Shows a query string "geoip.city\_name:(san jose)" in the search bar. The interface is labeled "Lucene".
- Second Screenshot (KQL):** Shows a query string "city" in the search bar. Below it, two filter options are listed: "geoip.city\_name.keyword" and "geoip.city\_name". A tooltip for "geoip.city\_name.keyword" says "Filter results that contain geo:". A tooltip for "geoip.city\_name" says "Filter results that contain geo:".
- Third Screenshot (KQL):** Shows a query string "geoip.city\_name:\"san jose\"" in the search bar. The interface is labeled "KQL".
- Bottom Screenshot (KQL):** Shows a query string "geoip.city\_name:san or geoip.city\_name:jose" in the search bar. The interface is labeled "KQL".

# SQL Access

- Execute SQL queries against Elasticsearch indices
  - get results back in tabular format
- Basic license (free)
- REST API, `_sql`
- Translate API, `_sql/translate`
- Aggregate and Full-Text Search Functions and more...

```
POST /_sql?format=txt
{
  "query": """
    SELECT * FROM blogs
    WHERE publish_date >= CAST('2018-01-01' AS DATETIME)
    LIMIT 5
  """
}
```

SQL is a good option to easily integrate an Elasticsearch backend with SQL-base reporting and visualization tools.  
**The `_search` API is the preferred access method.**



Elasticsearch Queries

Lesson 3

# Review - Combining Queries



# Summary

- The ***bool*** query allows you to combine queries using Boolean logic
- Any clause in a “**should**” clause increases the **\_score** of hits
- Any clause inside a “**filter**” clause has no contribution to score

# Quiz

1. A user searches our blogs for “***scripting***” and checks the box that only displays blogs from the “**Engineering**” category. How would you implement this query?
2. **True or False:** If a document matches a **filter** clause, the score is increased by a factor of 2.
3. **True or False:** The following query will return all blogs, and documents that contain “***performance***” in the ***title*** field will be on top.

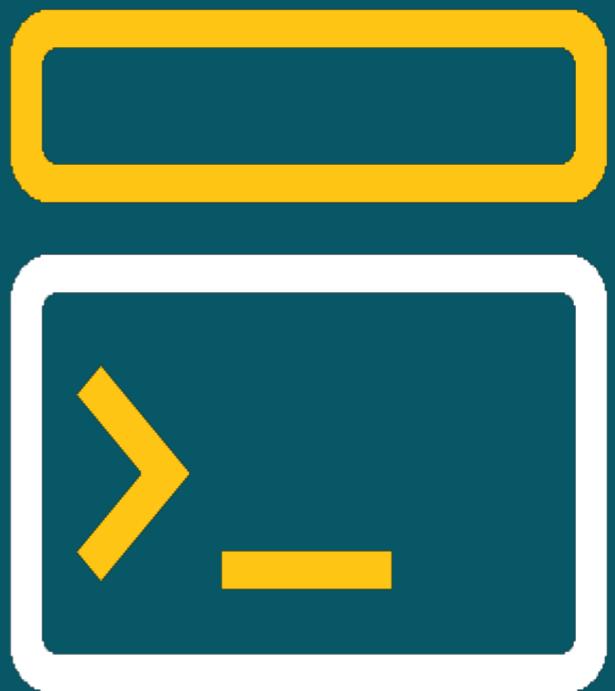
```
GET blogs/_search
{
  "query": {
    "bool": {
      "should": [
        {"match": {"title": "performance"}}
      ]
    }
  }
}
```



Elasticsearch Queries

Lesson 3

# Lab - Combining Queries





Elasticsearch Queries

Lesson 4

# Implementing a Search Page



# Building a Search Page



elastic

logstash **Search**

**Filters**

Categories

- (119)
- Engineering (106)
- Releases (73)
- This week in Elasticsearch and Apache Lucene (70)
- The Logstash Lines (41)

Dates (dd/mm/yyyy)

from: \_\_\_\_\_

to: \_\_\_\_\_

Sort by: Score ▾

**Sorting**

Q

There are 529 results for "logstash" (13 milliseconds)

**Highlighting**

**Logstash 5.0.0 Released** [\_score: 11.896213]

October 26, 2016 [Releases]

availability of the biggest release of Logstash yet., Previously, Logstash used /opt/logstash directory to install the binaries, whereas Elasticsearch used, Logstash 5.0 is compatible with Elasticsearch 5.x, 2.x, and even 1.x., Extracting fields from unstructured data is a popular Logstash feature., So, from the entire Logstash team, thank you to our users for using and contributing back to Logstash

**Welcome Jordan & Logstash** [\_score: 11.826626]

August 27, 2013 [ ]

is amazing news for so many Elasticsearch and Logstash users., About Logstash Logstash, which just released version 1.2.0, is one of the most popular open source logs, have received requests to offer commercial support for Logstash ., Logstash shares the same vision., Effectively, Logstash is a generic system to process events.

**Logstash 1.4.0 beta1** [\_score: 11.775666]

February 19, 2014 [ ]

We are pleased to announce the beta release of Logstash 1.4.0!, Contrib plugins package Logstash has grown brilliantly over the past few years with great contributions, Now having 165 plugins, it became hard for us (the Logstash engineering team) to reliably support all, A bonus effect of this decision is that the default Logstash download size shrank by 19MB compared to, Going forward, Logstash release cycles will more closely mirror Elasticsearch's model of releases.

« 1 2 3 4 5 »

**Pagination**

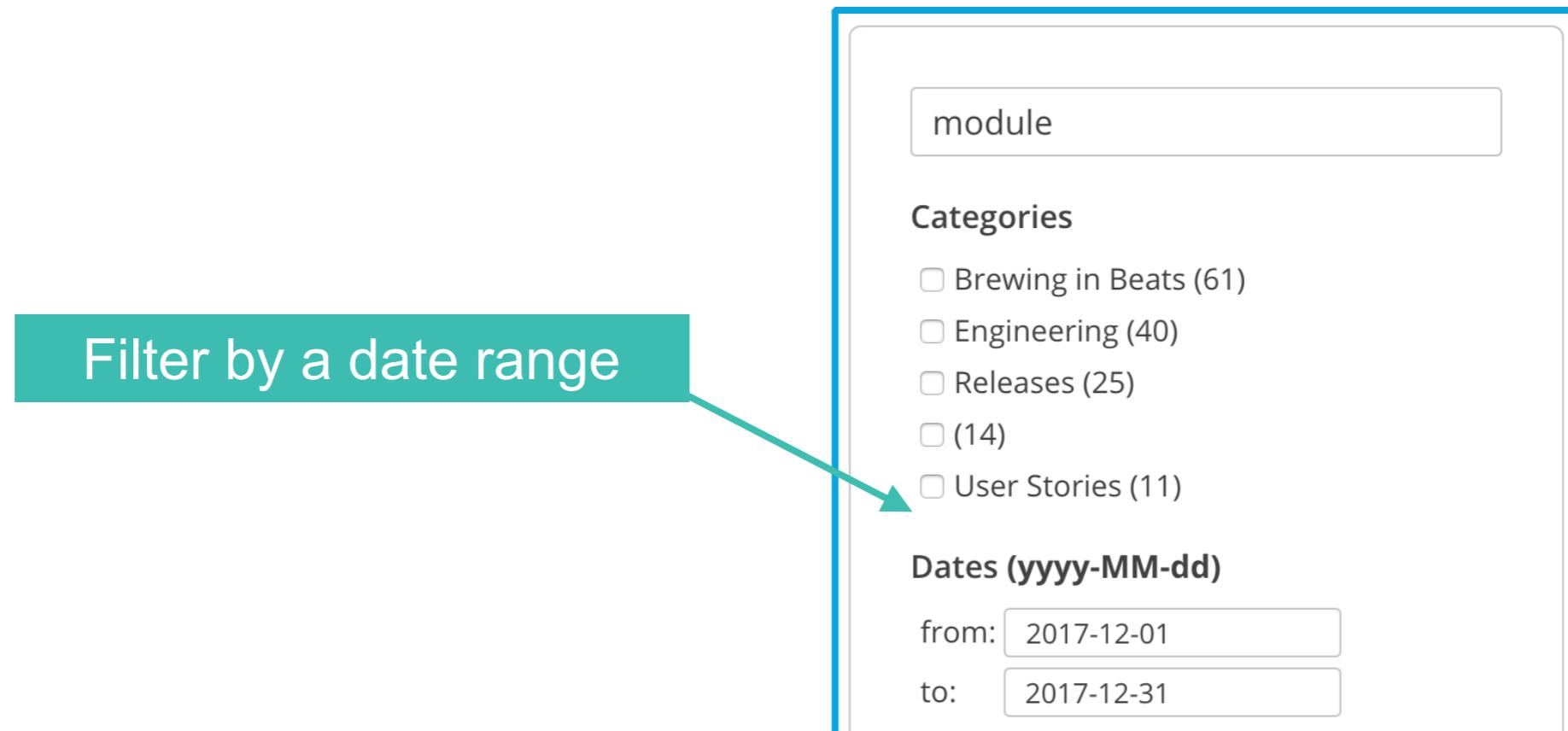


# Filters

# Specifying Date Ranges

- Suppose we want to allow users to search for a blog over a specific date range:

Filter by a date range



module

Categories

- Brewing in Beats (61)
- Engineering (40)
- Releases (25)
- (14)
- User Stories (11)

Dates (yyyy-MM-dd)

from: 2017-12-01

to: 2017-12-31

# The range Query

- Range queries work on **numerics** and **date** fields
  - you can specify the dates as a string
  - or use a special syntax called **date math**

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "content": "module" } }
      ],
      "filter": {
        "range": {
          "publish_date": {
            "gte": "2017-12-01",
            "lt": "2018-01-01"
          }
        }
      }
    }
  }
}
```

*"I am looking for all blogs about module from December, 2017"*

# Date Math

- Elasticsearch has a user-friendly way to express date ranges by using **date math**

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "content": "module" } }
      ],
      "filter": [
        {
          "range": {
            "publish_date": {
              "gte": "now-3M"
            }
          }
        }
      ]
    }
  }
}
```

*"I am looking for blogs about module **from the last three***

The date math expression for  
*"right now minus three months"*

# Date Math Expressions

- Here are the supported time units for date math and a few examples

<b>y</b>	<b>years</b>
<b>M</b>	<b>months</b>
<b>w</b>	<b>weeks</b>
<b>d</b>	<b>days</b>
<b>h or H</b>	<b>hours</b>
<b>m</b>	<b>minutes</b>
<b>s</b>	<b>seconds</b>

<b>If now = 2017-10-19T11:56:22</b>	
now-1h	2017-10-19T10:56:22
now+1h+30m	2017-10-19T13:26:22
now/d+1d	2017-10-20T00:00:00
2018-01-15    +1M	2018-02-15T00:00:00

# Searching for Exact Terms

- Suppose we want users to be able to find blogs by a specific author, or within a specific category
  - we would need hits that are exact matches



elastic

module

Categories

- Brewing in Beats (61)
- Engineering (40)
- Releases (25)
- (14)
- User Stories (11)

A “**category**” filter will require exact matches

# Searching for a Specific Category

- Let's search for blogs belonging to the “*Brewing in Beats*” category
  - using a **match** query on “category” gets us close, but it also picks up all other categories containing “*in*”

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "content": "module" } }
      ],
      "filter": [
        {
          "match": {
            "category": "Brewing in Beats"
          }
        }
      ]
    }
  }
}
```

We get back all categories containing “*in*”

“Brewing in Beats”

“This week in  
Elasticsearch and  
Apache Lucene”

“Currently in Kibana”

# Using the keyword Field

- We didn't discuss yet, but strings are usually analyzed
- By default, every string has an optional field that can be used for exact matches
  - they are the **keyword** fields
  - you will learn more about them in the **Text Analysis** module

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "content": "module" } }
      ],
      "filter": [
        { "match": { "category.keyword": "Brewing in Beats" } }
      ]
    }
  }
}
```

Only blogs in the  
“*Brewing in Beats*”  
category are hits

# Filtering on Exact Terms

- We often use exact matches in a **filter** clause
  - filters can be cached and reused, so they are faster
  - plus it keeps the **filter** part of the logic out of the scoring

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "content": "module"
        }
      },
      "filter": {
        "match": {
          "category.keyword": "Brewing in Beats"
        }
      }
    }
  }
}
```

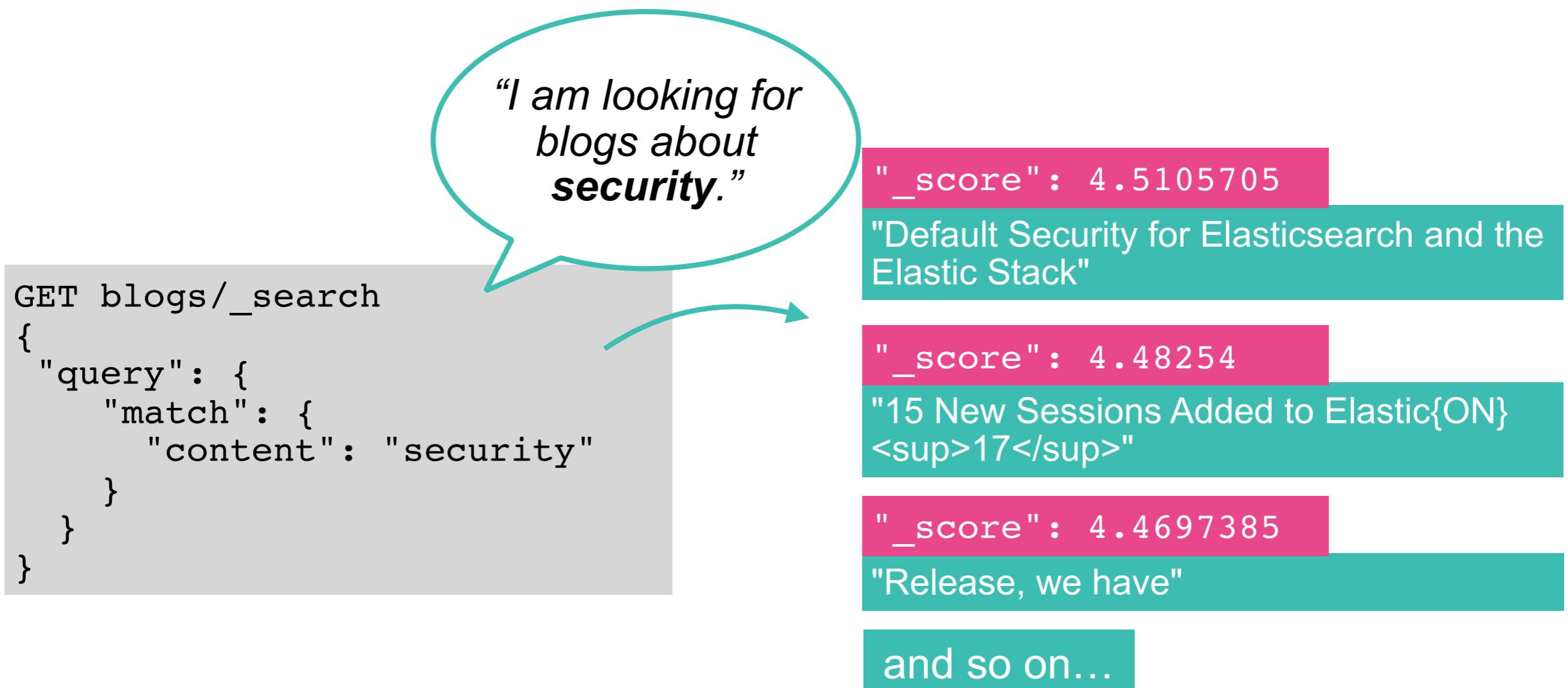
*"I am looking for blogs about **module** in the **Brewing in Beat** category."*

No need for category to contribute to the `_score`

# Sorting

# Sorting Results

- Up until now, our queries have returned hits in order of relevancy
  - \_score descending is the default sorting for a query



# Other Sort Options

- We want users to be able to sort their blog search results by other values
  - like *category* or *publish date*

Our blog app has a drop-down for sort options

The screenshot shows a search interface for 'logstash'. It includes a 'Categories' section with links like '(119)', 'Engineering (106)', 'Releases (73)', 'This week in Elasticsearch and Apache Lucene (70)', and 'The Logstash Lines (41)'. Below that is a 'Dates (dd/mm/yyyy)' section with 'from:' and 'to:' input fields. At the bottom is a 'Sort by' dropdown menu with the following options:

- Score (selected)
- Date Asc
- Date Desc** (highlighted in blue)
- Category Asc
- Category Desc

A teal arrow points from the text 'Our blog app has a drop-down for sort options' to the 'Sort by' dropdown.

# The sort Clause

- A query can contain a **sort** clause that specifies one or more fields to sort on
  - as well as the order (**asc** or **desc**)

*"I want the  
most recent blogs on  
security."*

```
GET blogs/_search
{
  "query": {
    "match": {
      "content": "security"
    }
  },
  "sort": [
    {
      "publish_date": {
        "order": "desc"
      }
    }
  ]
}
```

"publish\_date": "2017-12-19"  
"Kibana 6.1.1 released"

"publish\_date": "2017-12-18"  
"Default Password Removal in Elasticsearch  
and X-Pack 6.0"

"publish\_date": "2017-12-12"  
"Custom Region Maps in Kibana 6.0"

and so on...

# If `_score` is not a field in the sort clause...

- ...then scores are not calculated for hits
  - there is no need to waste the resources of computing scores

```
GET blogs/_search
{
  "query": {
    "match": {
      "content": "security"
    }
  },
  "sort": [
    {
      "publish_date": {
        "order": "desc"
      }
    }
  ]
}
```

```
"hits": [
  {
    "_index": "blogs",
    "_type": "doc",
    "_id": "14",
    "_score": null,
    "_source": {
      ...
    }
  }
]
```

**`_score` is null because it was not calculated**

# Sort on Multiple Fields

- When adding multiple sort fields, they are applied in the order listed in the “sort” array:

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": {"match": {"content": "security"}},
      "must_not": {"match": {"author.keyword": ""}}
    }
  },
  "sort": [
    {
      "author.keyword": {
        "order": "asc"
      }
    },
    {
      "publish_date": {
        "order": "desc"
      }
    }
  ]
}
```

Sort by **author**, then by  
**publish\_date**

# The sort Response

- Each hit contains a “sort” section in the response with the values used for sorting that hit

The **author** and  
**publish\_date** values  
appear in the “sort”  
clause of the response

```
{  
  "_index": "blogs",  
  "_type": "_doc",  
  "_id": "631",  
  "_score": null,  
  "_source": {...},  
  "sort": [  
    "Aaron Mildenstein",  
    1463356800000  
  ],  
  {  
    "_index": "blogs",  
    "_type": "_doc",  
    "_id": "1204",  
    "_score": null,  
    "_source": {...},  
    "sort": [  
      "Aaron Mildenstein",  
      1415577600000  
    ]  
  },  
}
```

# Pagination

# Pagination

- Let's take a look at how to use the “from” and “size” parameters to add pagination to our search results

**Logstash 1.4.0 beta1** [\_score: 11.775666]

February 19, 2014 []

We are pleased to announce the beta release of Logstash 1.4.0! Contrib plugins package Logstash has grown brilliantly over the past few years with great contributions. Now having 165 plugins, it became hard for us (the Logstash engineering team) to reliably support all. A bonus effect of this decision is that the default Logstash download size shrank by 19MB compared to. Going forward, Logstash release cycles will more closely mirror Elasticsearch's model of releases.

« 1 2 3 4 5 »

Let's see how to add  
paging...

# Pagination

- We have seen how the **size** parameter is used to specify the number of hits
  - which is really just the first “*page*” of results

*“I want the **first page** of hits.”*

```
GET blogs/_search
{
  "size": 10,
  "query": {
    "match": {
      "content": "elasticsearch"
    }
  }
}
```

# The from Parameter

- You can add the **from** parameter to a query to specify the offset from the first result you want to fetch
  - **from** defaults to 0

*"I want the **second** page of hits."*

```
GET blogs/_search
{
  "from": 10,
  "size": 10,
  "query": {
    "match": {
      "content": "elasticsearch"
    }
  }
}
```

# Highlighting

# Highlighting

- A common use case for search results is to *highlight* the matched terms
  - Elasticsearch makes it easy to do this

Highlight the search term  
in the response

logstash

Categories

- Engineering (3)
- Releases (3)
- The Logstash Lines (2)
- Brewing in Beats (1)

Dates (dd/mm/yyyy)

from: 01/12/2017

to: 31/12/2017

Sort by: Score ▾

There are 9 results for "logstash" (23 milliseconds)

**Logstash 6.1.0 Released** [\_score: 10.546306 ]

December 13, 2017 [Releases]

Logstash 6.1.0 has launched!,Read on for what's new in Logstash 6.1.0.,We're proud to announce a great new way to extend Logstash functionality in 6.1.0.,scripting via the Logstash Ruby filter.,We've been working on a full rewrite of the internal execution engine in Logstash.

**Logstash Lines: Update for December 12, 2017** [\_score: 8.693237 ]

December 12, 2017 [The Logstash Lines]

We've now for resetting logging settings changed via the Logstash web API back to their defaults. ,The Logstash HTTP input previously exhibited poor behavior when the queue was blocked.,This plugin will now either return a 429 (busy) error when Logstash is backlogged, or it will time out,backing off exponentially with some random jitter, then retry their request.This plugin will block if the Logstash



# The highlight Clause

- Add the fields you want highlighted to a “highlight” clause:

Highlight “**kibana**” if it appears  
in “**title**”

```
GET blogs/_search
{
  "query": {
    "match_phrase": {
      "title": "kibana"
    }
  },
  "highlight": {
    "fields": {
      "title": {}
    }
  }
}
```



```
_source": {
  "url": "/blog/kibana-5-6-1-released",
  "title": "Kibana 5.6.1 released",
  ...
},
"highlight": {
  "title": [
    "<em>Kibana</em> 5.6.1 released"
  ]
}
```

The response will contain a  
“highlight” section

# Changing the Tags

- Use “`pre_tags`” and “`post_tags`” to change the highlighting:

```
GET blogs/_search
{
  "query": {
    "match_phrase": {
      "title": "kibana"
    }
  },
  "highlight": {
    "fields": {
      "title": {}
    },
    "pre_tags": ["<es-hit>"],
    "post_tags": ["</es-hit>"]
  }
}
```

Results highlighted with your  
custom tags

```
"highlight": {
  "title": [
    "<es-hit>Kibana</es-hit> 4.1.1 Released"
  ]
}
```



Elasticsearch Queries

Lesson 4

# Review - Implementing a Search Page



# Summary

- If you need searches for ***exact text***, you typically use the **.keyword** field
- Use the **sort** clause for sorting by a field, **\_score** or **\_doc**
- Use “**from**” and “**size**” parameters to implement pagination
- A common use case for search results is to ***highlight*** the matched terms, which can be accomplished by adding a “**highlight**” clause to a search body

# Quiz

1. What are the two parameters that you can use to implement paging of search results?
2. True or False: Search results are always sorted by score.
3. How do the following two queries behave differently?

```
GET blogs/_search
{
  "query": {
    "match": {
      "category": "User Stories"
    }
  }
}
```

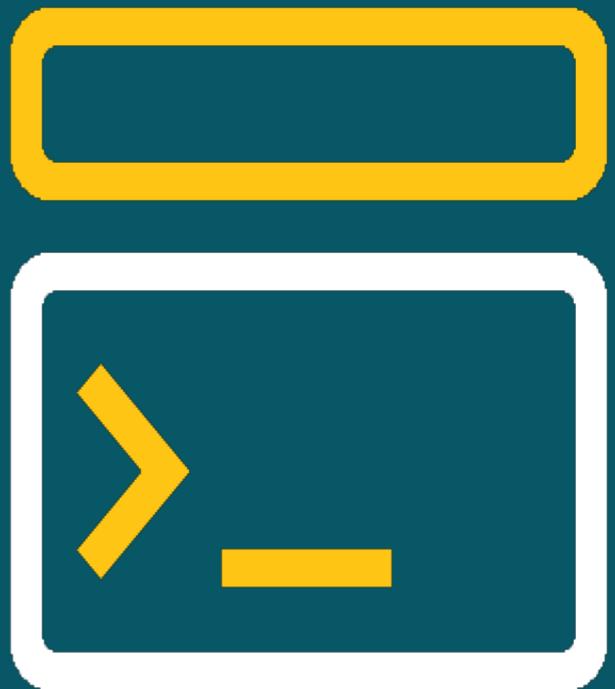
```
GET blogs/_search
{
  "query": {
    "match": {
      "category.keyword": "User Stories"
    }
  }
}
```



Elasticsearch Queries

Lesson 4

# Lab - Implementing a Search Page



- Elasticsearch Fundamentals
- Elasticsearch Queries
- Elasticsearch Aggregations
- Elasticsearch Text Analysis and Mappings
- Elasticsearch Nodes and Shards
- Elasticsearch Monitoring and Troubleshooting

Module 3

# Elasticsearch Aggregations



# Topics

- Metrics Aggregations
- Bucket Aggregations
- Combining Aggregations



Elasticsearch Aggregations

Lesson 1

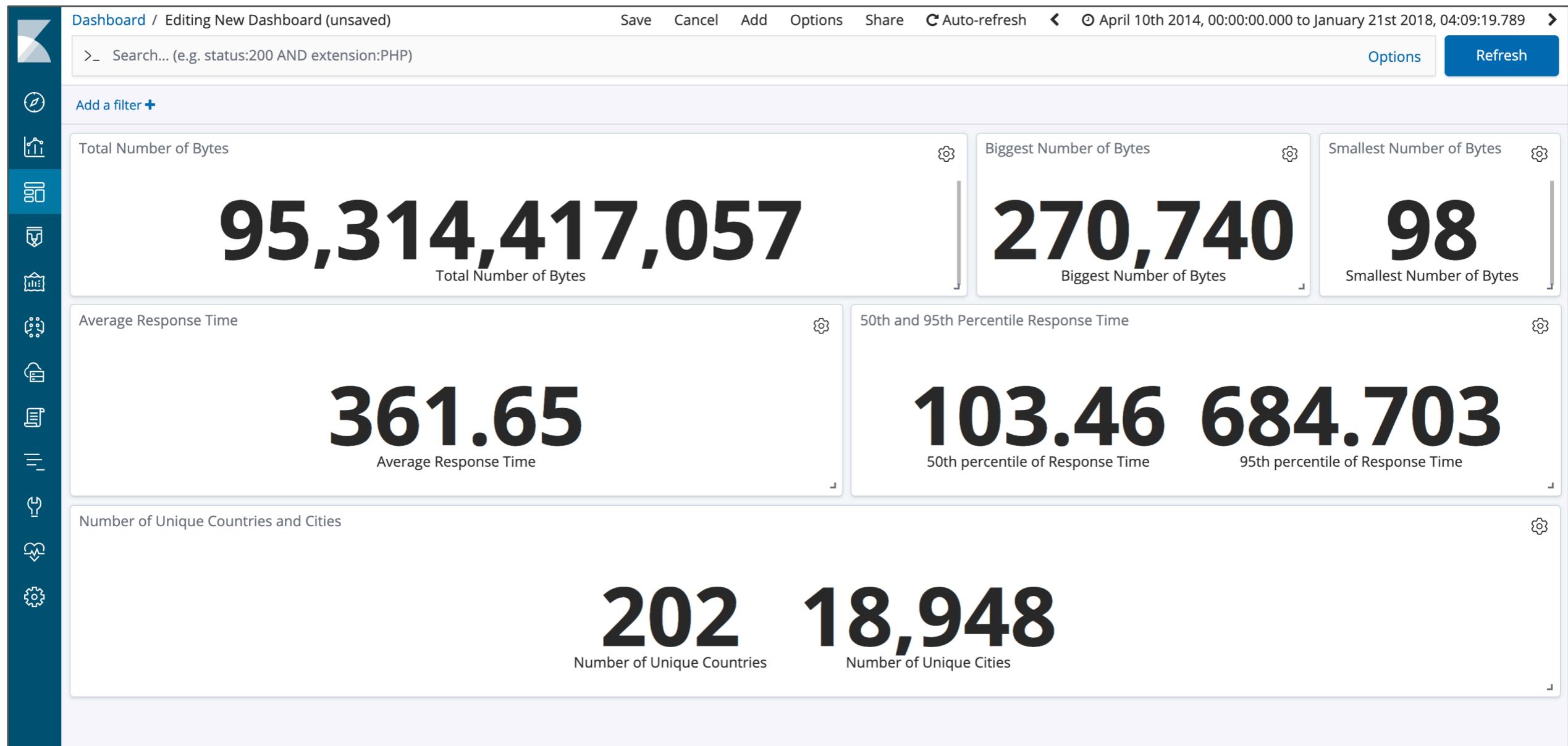
# Metrics Aggregations



# Metrics: Questions

- What is the number of bytes served from all blogs?
- What is the number of bytes served from blogs in French?
- What is the average response time?
- What is the median response time?
  - and the 95 percentile?
- What is the smallest number of bytes in a response?
- What is the biggest number of bytes in a response?
- How many unique countries are accessing the blogs?
  - and cities?

# Metrics: Answers



# Kibana Visualizations under the Hood

The screenshot displays two Kibana visualizations side-by-side, each with its corresponding Elasticsearch request details.

**Top Visualization:**

- Title:** Total Number of Bytes
- Value:** 95,314,417,0
- Subtext:** Total Number of Bytes
- Request Statistics:** 1 request was made, Request took 75ms
- Request Body:**

```
{  
  "aggs": {  
    "1": {  
      "sum": {  
        "field": "response_size"  
      }  
    }  
  },  
}
```

**Bottom Visualization:**

- Title:** Total Number of Bytes
- Value:** 95,314,417,0
- Subtext:** Total Number of Bytes
- Average Response Time:** 361.65
- Subtext:** Average Response Time
- Number of Unique Countries and Cities:** 202
- Request Statistics:** 1 request was made, Request took 75ms
- Request Body:**

```
{  
  "took": 2,  
  "timed_out": false,  
  "_shards": {  
    "total": 15,  
    "successful": 15,  
    "skipped": 0,  
    "failed": 0  
  },  
  "hits": {  
    "total": 1751458,  
    "max_score": 0,  
    "hits": []  
  },  
  " aggregations": {  
    "1": {  
      "value": 95314417057  
    }  
  },  
  "status": 200  
}
```

# 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**”

```
GET my_index/_search
{
  "aggs": {
    "my_aggregation": {
      "AGG_TYPE": {
        ...
      }
    }
  }
}
```

The name you choose comes back in the results

Lots of different aggregation types

# Example of an Aggregation

*What is the number of bytes served from all blogs?*

```
GET logs_server*/_search
{
  "aggs": {
    "total_sum_bytes": {
      "sum": {
        "field": "response_size"
      }
    }
  }
}
```

The “**sum**” aggregation computes the sum of a numeric field

In this agg, the sum of “**response\_size**” is calculated over all documents

# Aggregation Results

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

```
"hits": {  
  ...  
  "hits": [  
    ...  
  ]  
,  
  "aggregations": {  
    "total_sum_bytes": {  
      "value" : 9.5314417057E10  
    }  
  }  
}
```

We get the top 10 hits...

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

The sum of response size over all docs

# Aggregation Results Only

- If you are not interested in the hits and only want the values of the aggregations, then set “**size**” to 0
  - this will speed up the query

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "total_sum_bytes": {
      "sum": {
        "field": "response_size"
      }
    }
  }
}
```

```
{
  "took": 7,
  "timed_out": false,
  "_shards": {
    "total": 15,
    "successful": 15,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 1751476,
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "total_sum_bytes": {
      "value": 9.5314417057E10
    }
  }
}
```

# The Scope of an Aggregation

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

*What is the **sum** of bytes where the language code is French?*

```
GET logs_server*/_search
{
  "size": 0,
  "query": {
    "match": {
      "language.code": "fr-fr"
    }
  },
  "aggs": {
    "french_sum_bytes": {
      "sum": {
        "field": "response_size"
      }
    }
  }
}
```

```
"hits": {
  "total": 99671,
  "max_score": 0,
  "hits": []
},
"aggregations": {
  "french_sum_bytes": {
    "value": 5.095259666E9
  }
}
```

The sum of bytes is computed over 99,671 docs (instead of all docs)

# Questions

What is the **smallest** number of bytes in a response?

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "min_response_size": {
      "min": {
        "field": "response_size"
      }
    }
  }
}
```

What is the **biggest** number of bytes in a response?

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

What is the **average** response time?

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

What is the **median** response time?

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



# What is the median response time?

- You can specify the **percentiles** to be computed
  - for example, the following aggregation computes the quartiles (25% intervals) for the **runtime** of the log events.

```
GET logs_server*/_search
{
  "size" : 0,
  "aggs": {
    "runtime_quartiles": {
      "percentiles": {
        "field": "runtime_ms",
        "percents": [
          25,
          50,
          75
        ]
      }
    }
  }
}
```



The 50th percentile is also called the median

```
"aggregations" : {
  "runtime_quartiles" : {
    "values" : {
      "25.0" : 95.0,
      "50.0" : 103.49048422066971,
      "75.0" : 159.86132183257953
    }
  }
}
```

50% of the responses have of runtime less than 103.5 ms



# The cardinality Aggregation

- The result may not be exactly precise for large datasets
  - based on the HyperLogLog++ algorithm
  - trades accuracy over speed

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "number_of_countries": {
      "cardinality": {
        "field": "geoip.country_name.keyword"
      }
    }
  }
}
```

*How many unique countries did we receive requests from?*

```
"aggregations": {
  "number_of_countries": {
    "value": 202
  }
}
```



Elasticsearch Aggregations

Lesson 1

# Review - Metrics Aggregations



# Summary

- **Metrics aggregations** compute numeric values based on your dataset
  - such as **min**, **max**, **avg**, and **stats**
- Use the **percentiles** aggregations to calculate different percentiles, such as the 50% (median)
- Use the **cardinality** aggregation to calculate unique counts

# Quiz

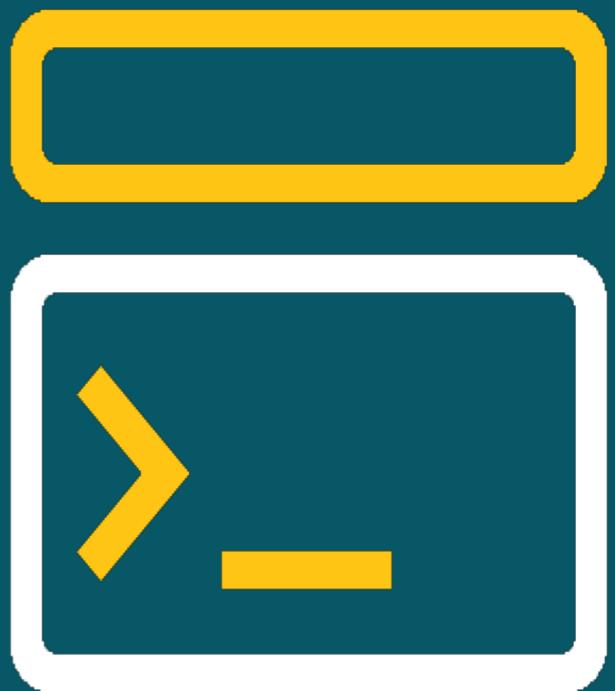
1. **True or False:** You can choose arbitrary names to identify aggregation clauses.
2. In our **logs\_server\*** indices, how could you verify that 95% of web requests are executed in less than 100ms?
3. What aggregation(s) would you use to answer the following question: “How many unique visitors came to our website today?”



Elasticsearch Aggregations

Lesson 1

# Lab - Metrics Aggregations





Elasticsearch Aggregations

Lesson 2

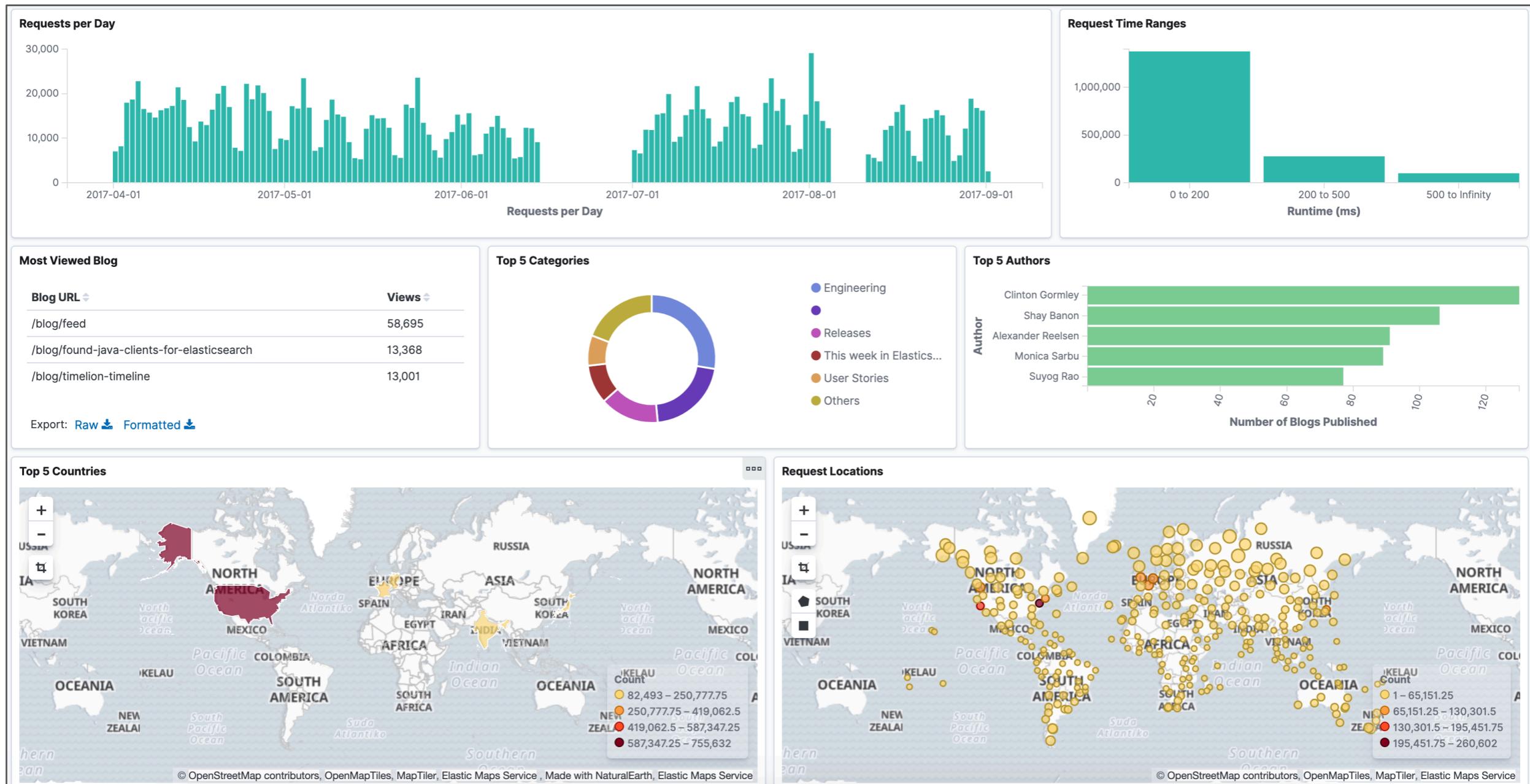
# Bucket Aggregations



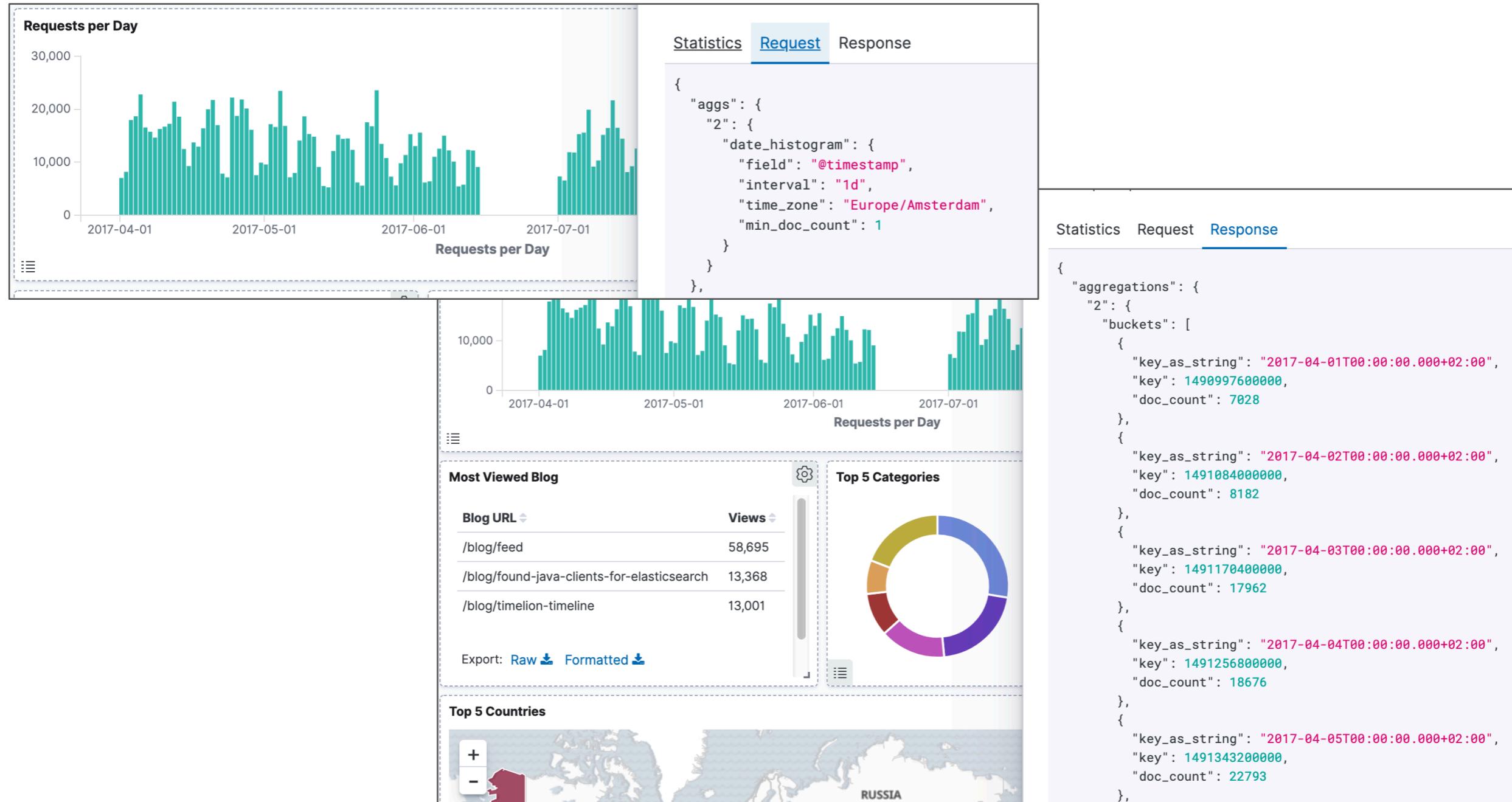
# Buckets Questions

- How many requests reach our system every day?
- How many requests took between 0-200, 200-500, 500-\* milliseconds?
- What are the most viewed blogs on our website?
- Which are the 5 most popular blog categories?
- How many blogs got published by our top authors?
- What are the top 5 countries with respect to the number of requests?
- Where are my requests coming from?

# Buckets: Answers

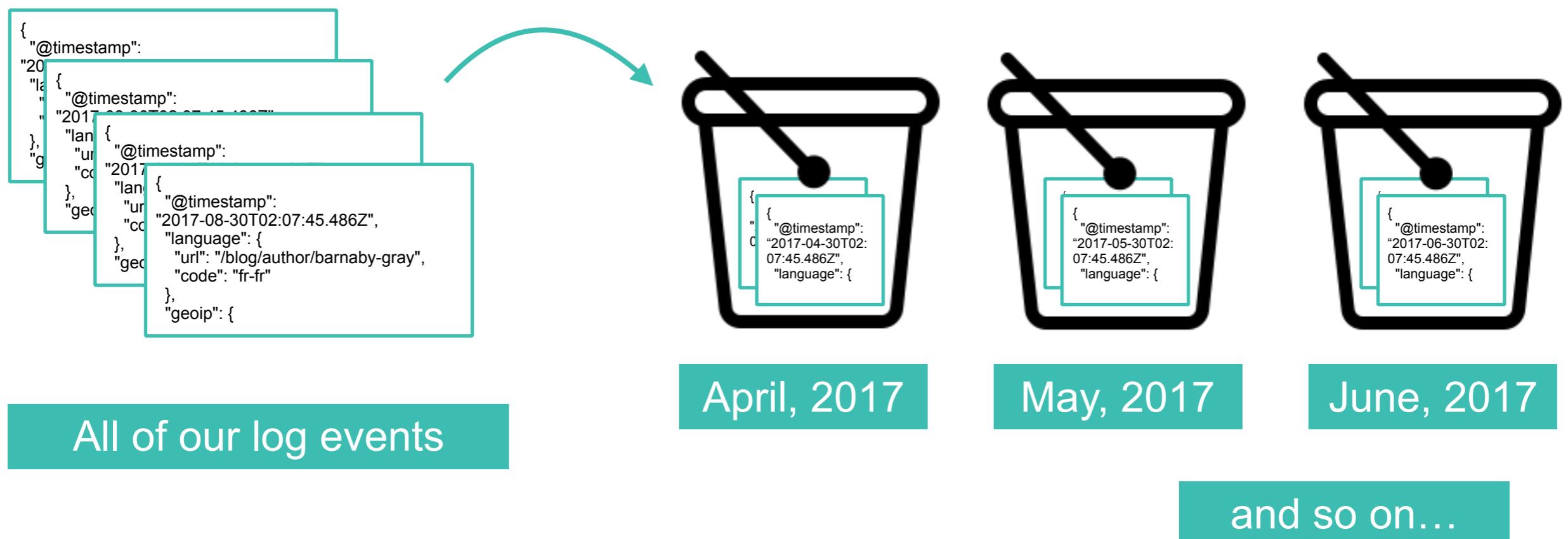


# Kibana Visualizations under the Hood



# Buckets

- A *bucket* represents a collection of documents that share common criteria
  - buckets are a key element of aggregations
- For example, suppose we want to analyze log traffic by month:

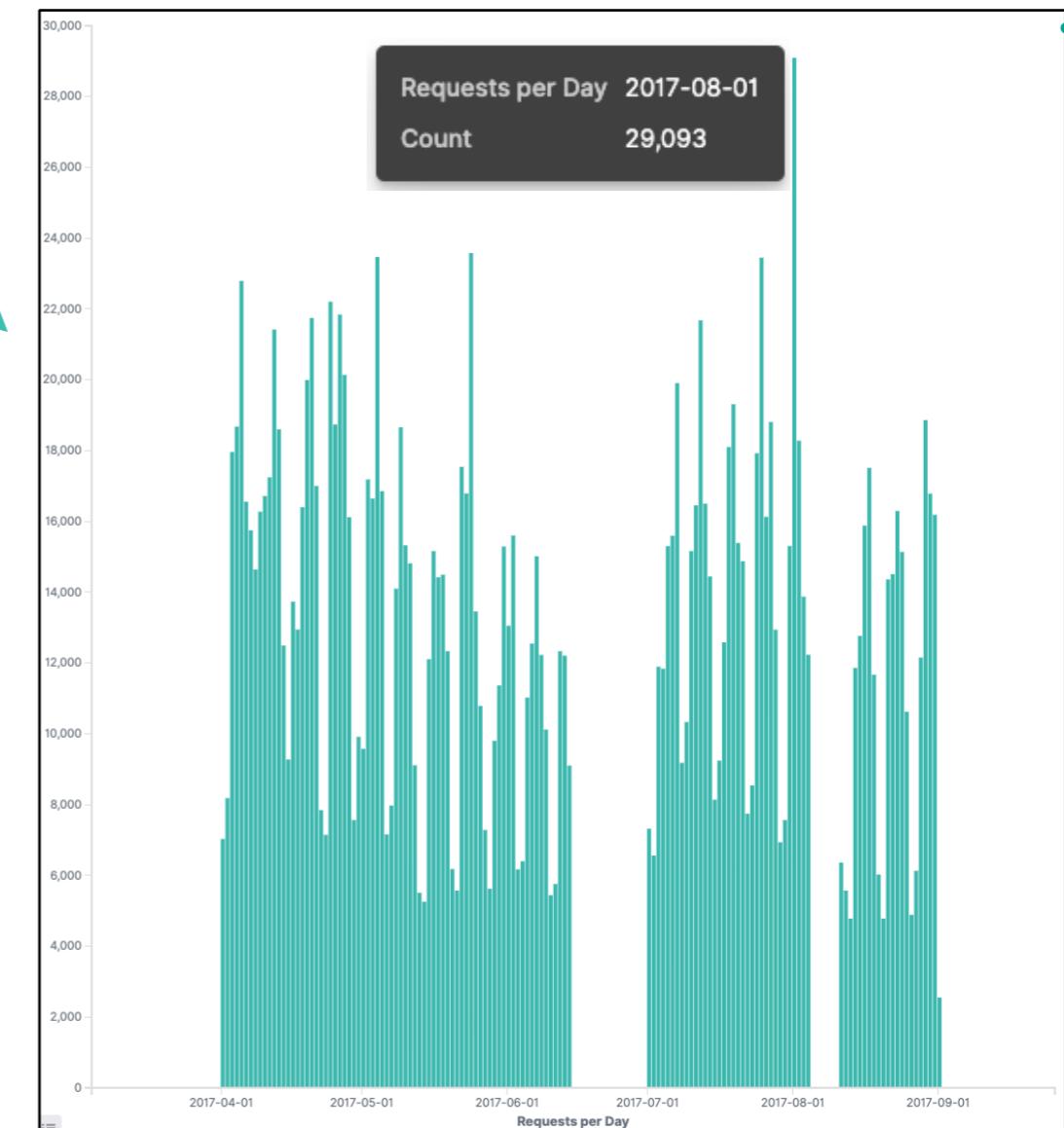


# How many requests reach our system every day?

- The *date\_histogram* puts documents in **buckets**
  - based on a given interval
- It provides a good overview of the data

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

**Be careful with the interval.**  
Creating too many buckets  
can harm the cluster.



# The date\_histogram Response

- The aggs response contains a **buckets** array
  - each entry is a day with the number of requests

```
"aggregations" : {  
  "logs_by_day" : {  
    "buckets" : [  
      {  
        "key_as_string" : "2017-03-31T00:00:00.000Z",  
        "key" : 1490918400000,  
        "doc_count" : 252  
      },  
      {  
        "key_as_string" : "2017-04-01T00:00:00.000Z",  
        "key" : 1491004800000,  
        "doc_count" : 7250  
      },  
      {  
        "key_as_string" : "2017-04-02T00:00:00.000Z",  
        "key" : 1491091200000,  
        "doc_count" : 8163  
      },  
      ...  
    ]  
  }  
}
```

*“There were **8163** requests on **April 2nd 2017**.”*

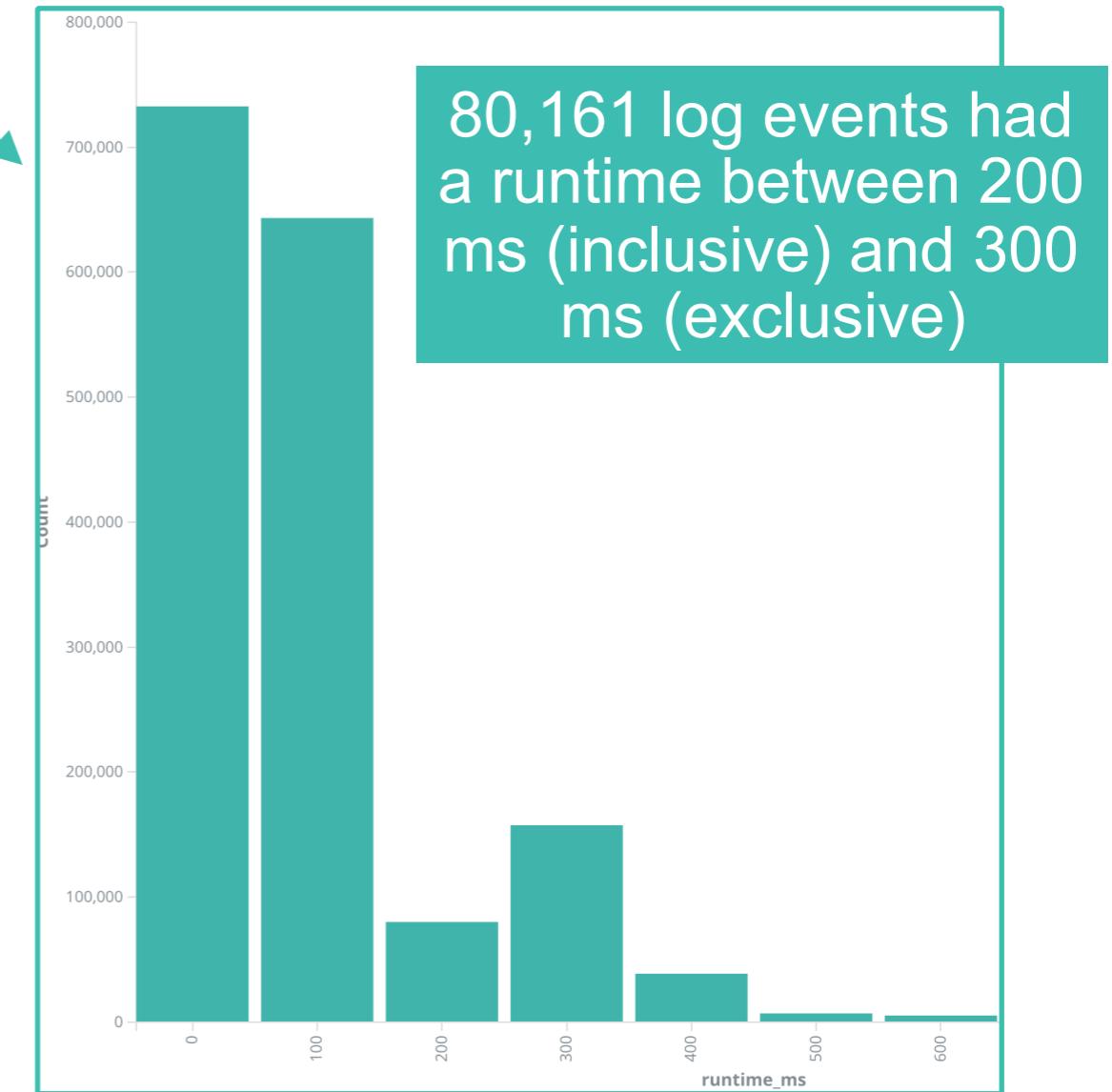
The maximum number of buckets that can be calculated defaults to **10000**

# The histogram Aggregation

- The *histogram* bucket aggregation builds a histogram on a given “field” using a specified “interval”:
  - the number of buckets is dynamic and depends on the data and the interval

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "runtime_histogram": {
      "histogram": {
        "field": "runtime_ms",
        "interval": 100
      }
    }
  }
}
```

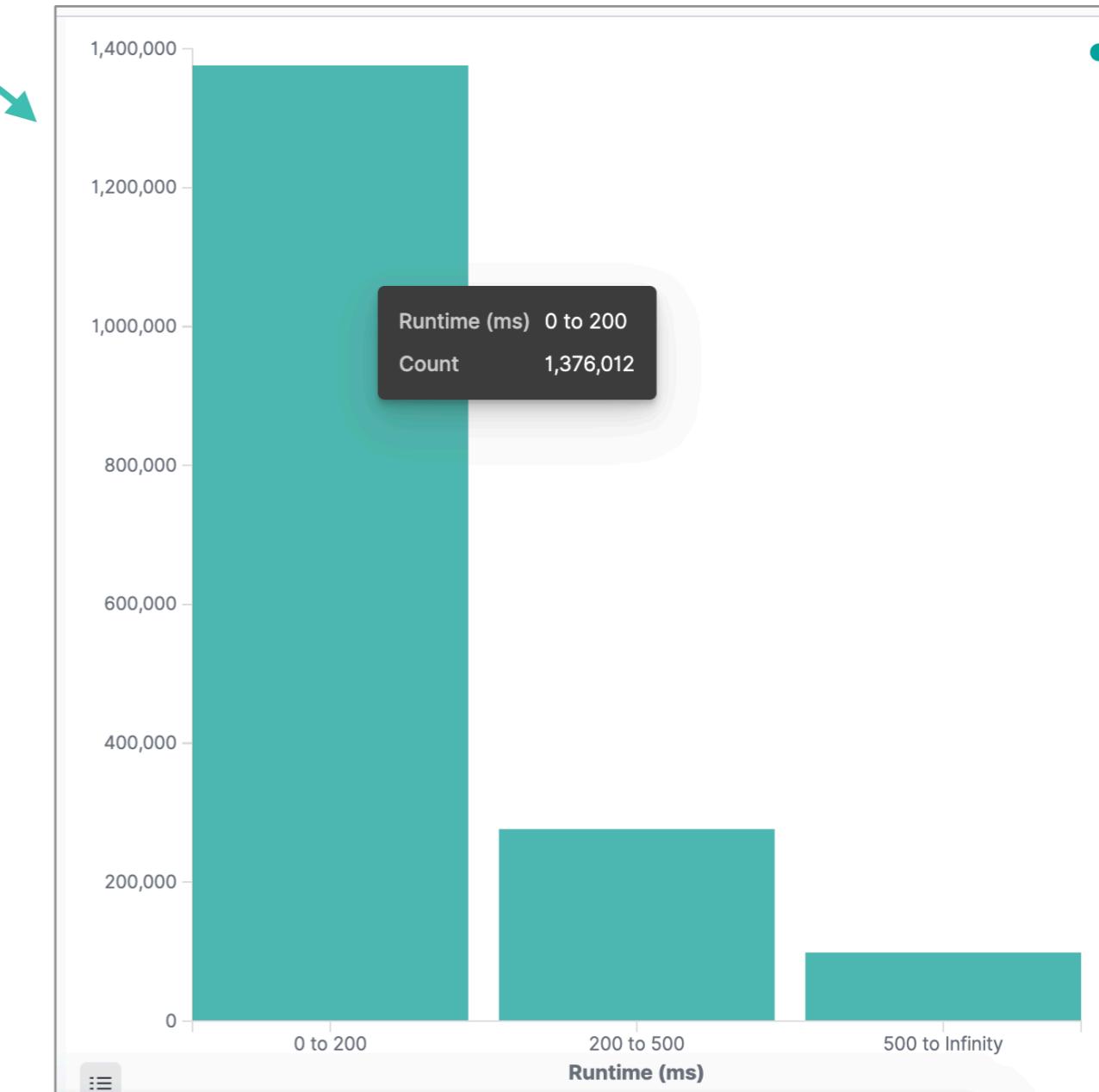
A bucket is created for every 100 milliseconds



# The range Aggregation

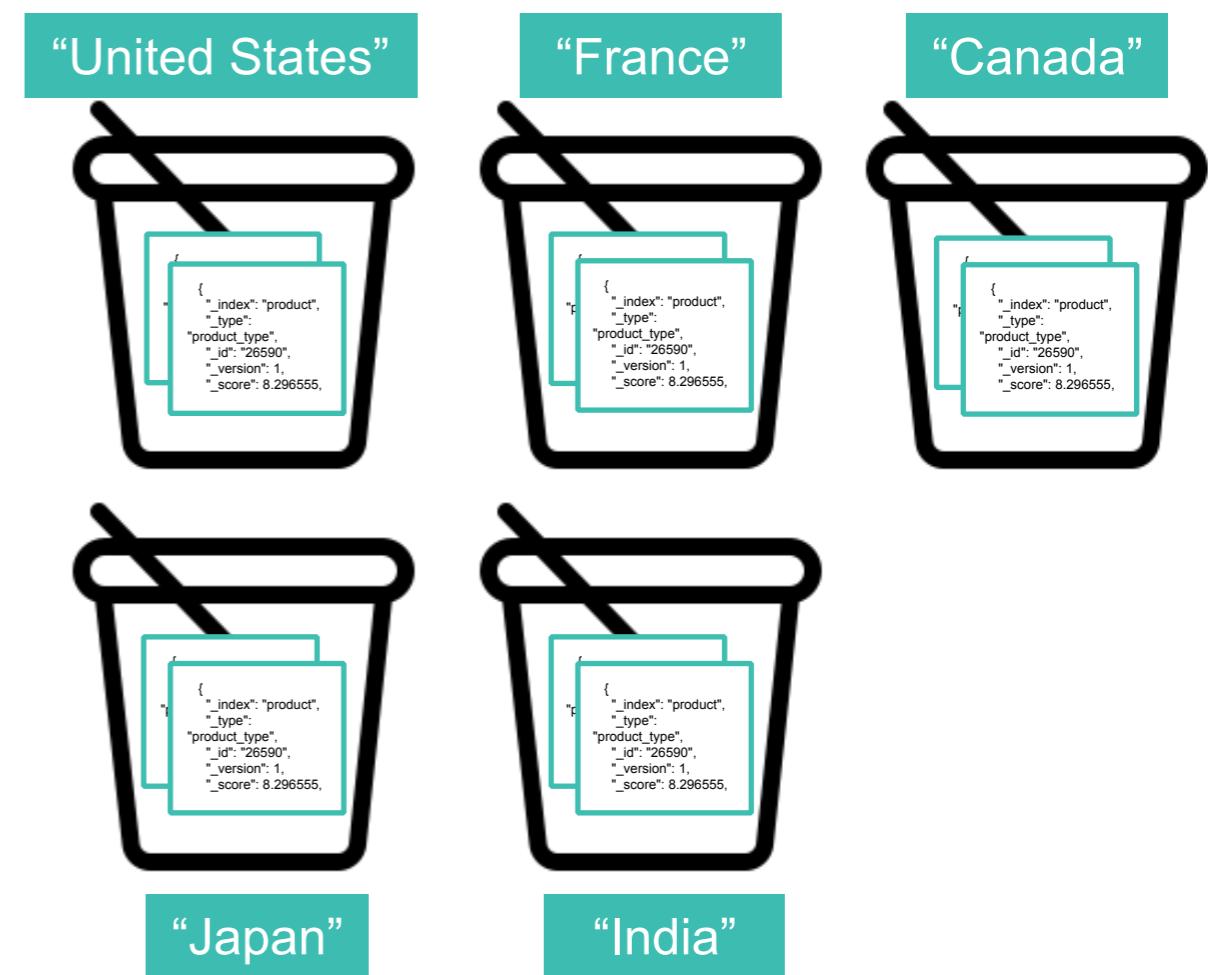
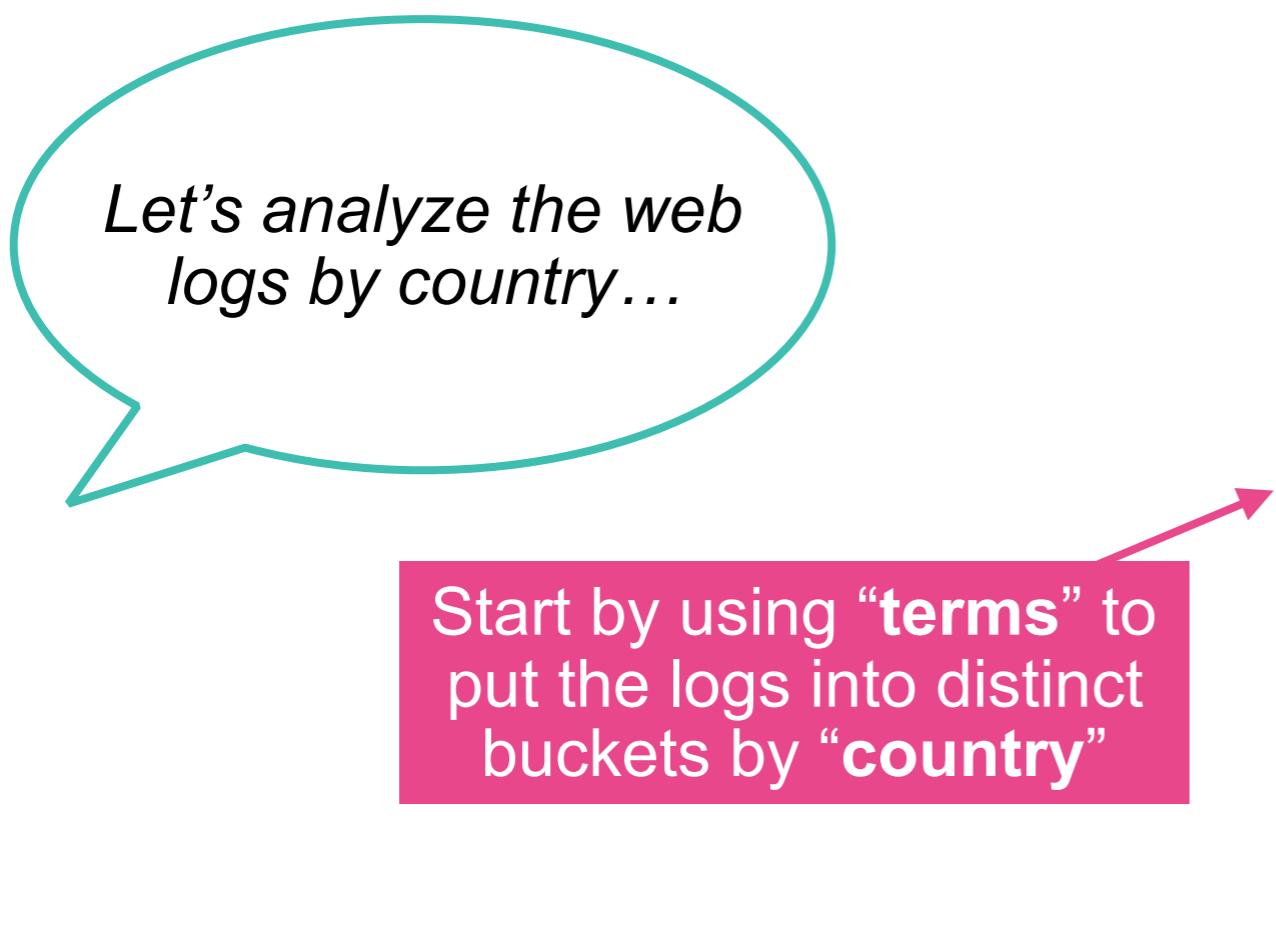
- Allows you to define your own intervals
  - how many requests took between 0-200, 200-500, 500-\* ms?

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "runtime_breakdown": {
      "range": {
        "field": "runtime_ms",
        "ranges": [
          {
            "from": 0,
            "to": 200
          },
          {
            "from": 200,
            "to": 500
          },
          {
            "from": 500
          }
        ]
      }
    }
  }
}
```



# The terms Aggregation

- The **terms** aggregation will dynamically create a new bucket for every unique term it encounters for the specified field
  - in other words, each distinct value of the field will have its own bucket of documents



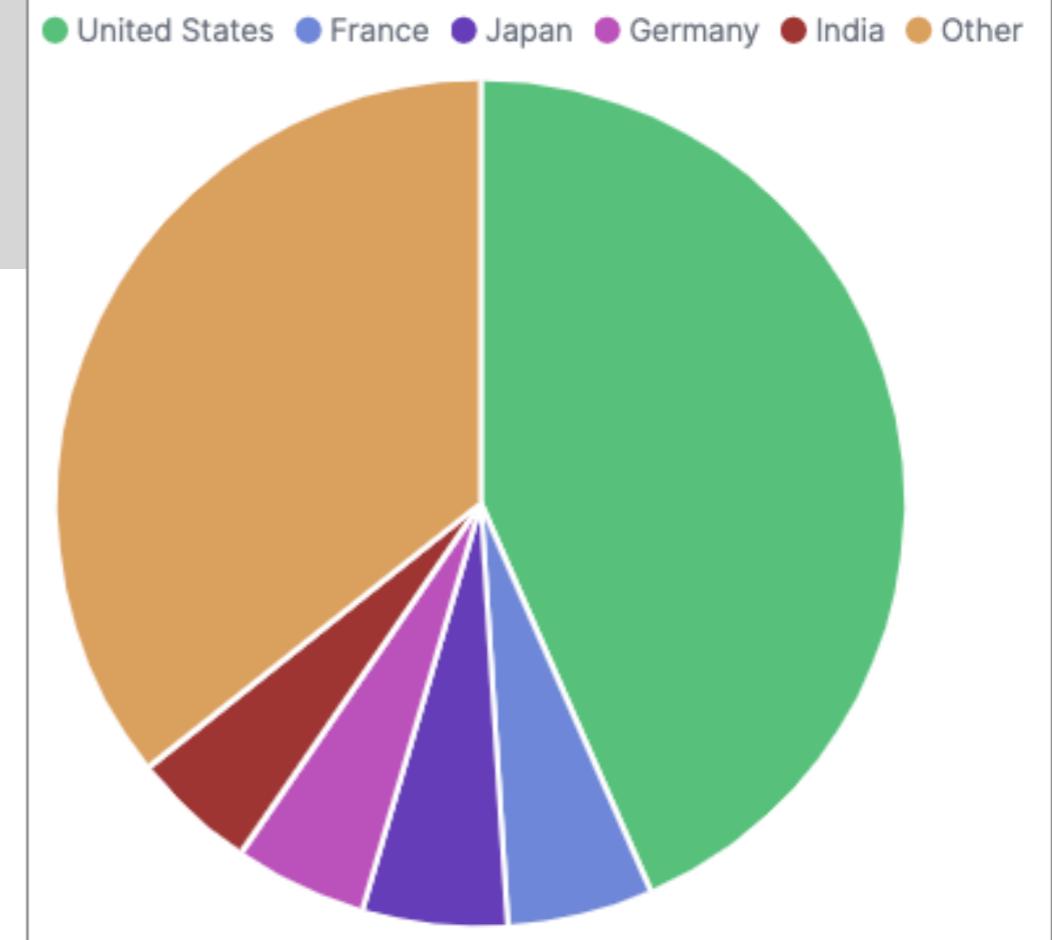
# An Example of a terms Aggregation

- You just need to specify a “**field**” to bucket the terms by

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "country_name_terms": {
      "terms": {
        "field": "geoip.country_name.keyword",
        "size": 5
      }
    }
  }
}
```

“**size**” = number of buckets  
to create (default is 10)

*“What are the  
**top 5 countries** that  
visited our blogs?”*



# The terms Aggregation Response

“United States” has the most requests, followed by “France”

- **key** represents the distinct value of field
- **doc\_count** is the number of docs in the bucket
- **sum\_other\_doc\_count** is the sum of all other buckets

```
"aggregations": {  
  "country_name_terms": {  
    ...  
    "sum_other_doc_count": 620884,  
    "buckets": [  
      {  
        "key": "United States",  
        "doc_count": 755643  
      },  
      {  
        "key": "France",  
        "doc_count": 96562  
      },  
      {  
        "key": "Japan",  
        "doc_count": 95944  
      },  
      {  
        "key": "Germany",  
        "doc_count": 87806  
      },  
      {  
        "key": "India",  
        "doc_count": 82494  
      }  
    ]  
  }  
}
```

# The terms Aggregation for quick Insights

- What are the most viewed blogs on our website?

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

- Which are the 5 most popular blog categories?

```
"field": "category.keyword",
"size": 5
```

On the **blogs** index

- How many blog articles got published by our top authors?

```
"field": "author.keyword"
```

On the **blogs** index

# Bucket Sorting

- Some aggs allow you to specify the sorting “order”
  - e.g. **terms**, **histogram**, and **date\_histogram**:
  - **\_count** sorts by their **doc\_count** (default in **terms**)
  - **\_key** sorts alphabetically (default in **histogram** and **date\_histogram**)

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "logs_by_day": {
      "date_histogram": {
        "field": "@timestamp",
        "interval": "day",
        "order": {
          "_key": "desc"
        }
      }
    }
  }
}
```

The default sorting order for **date\_histogram** is by ascending **\_key**



Elasticsearch Aggregations

Lesson 2

# Review - Bucket Aggregations



# Summary

- A **bucket** represents a collection of documents that share common criteria
- The ***terms*** aggregation dynamically creates buckets for every unique term it encounters for a specified field
- Some bucket aggs allow you to specify the sorting “**order**”

# Quiz

1. Which aggregation would you use to put logging events into buckets by log level (“error”, “warn”, “info”, etc.)?
2. Which question does the following request answer?

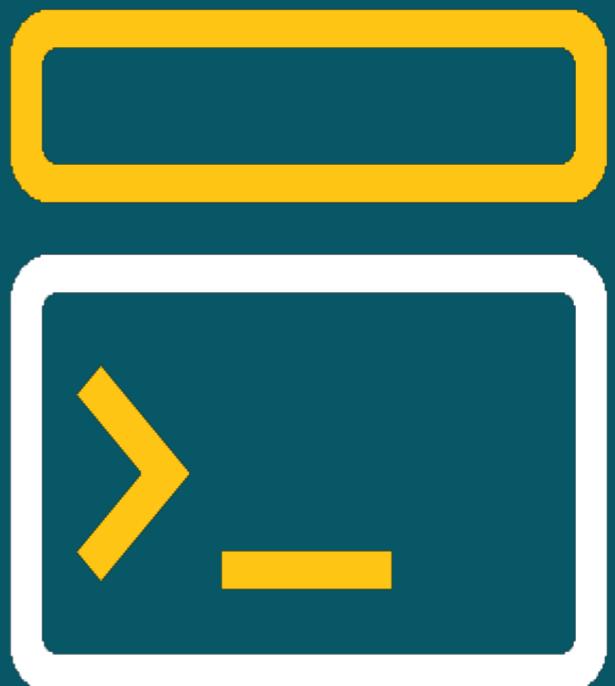
```
GET blogs/_search
{
  "size": 0,
  "query": {
    "range": {
      "publish_date": {
        "gte": "2017-01-01",
        "lt": "2018-01-01"
      }
    }
  },
  "aggs": {
    "NAME": {
      "date_histogram": {
        "field": "publish_date",
        "interval": "month"
      }
    }
  }
}
```



Elasticsearch Aggregations

Lesson 2

# Lab - Bucket Aggregations





Elasticsearch Aggregations

Lesson 3

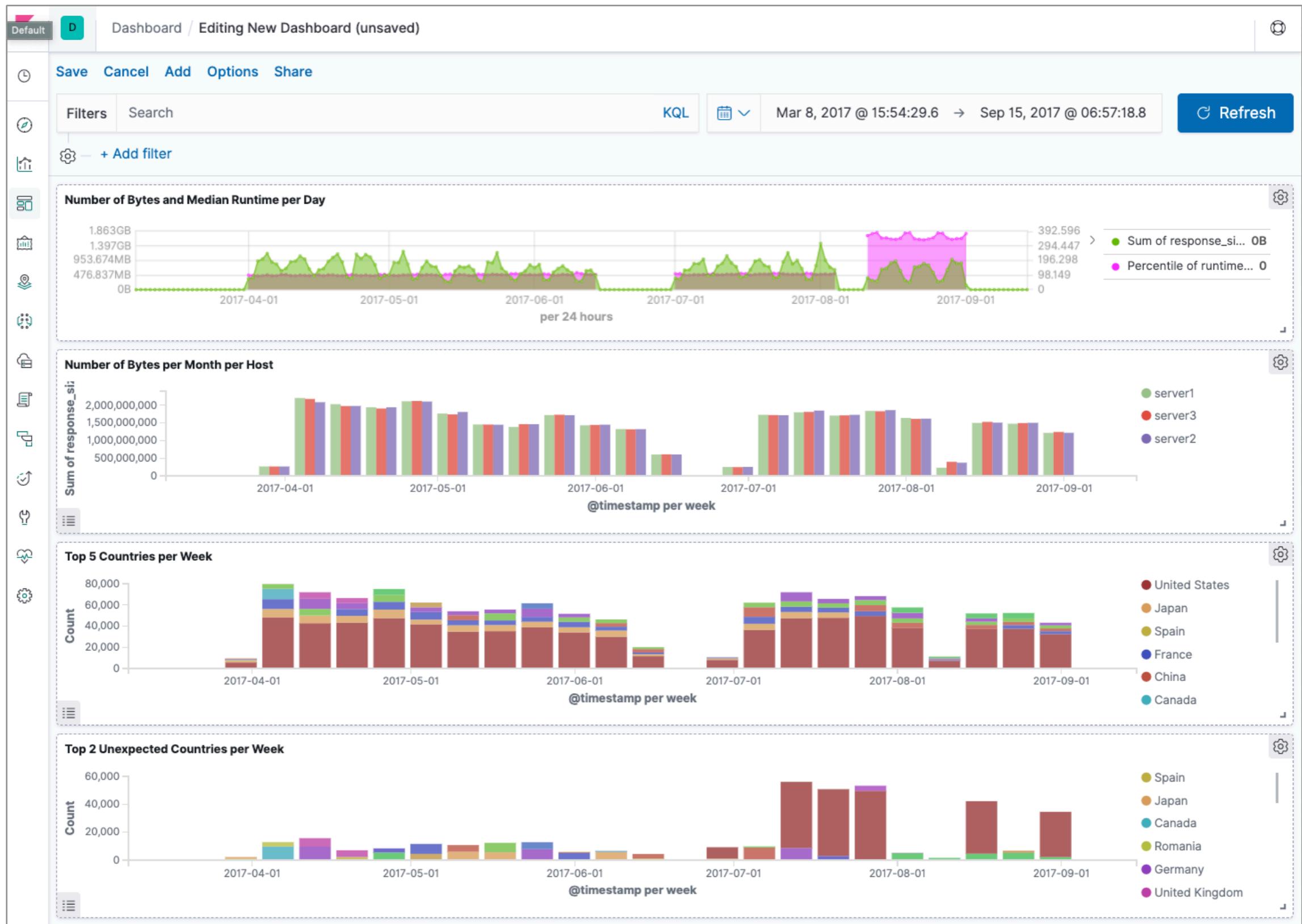
# Combining Aggregations



# Combined Aggregations: Questions

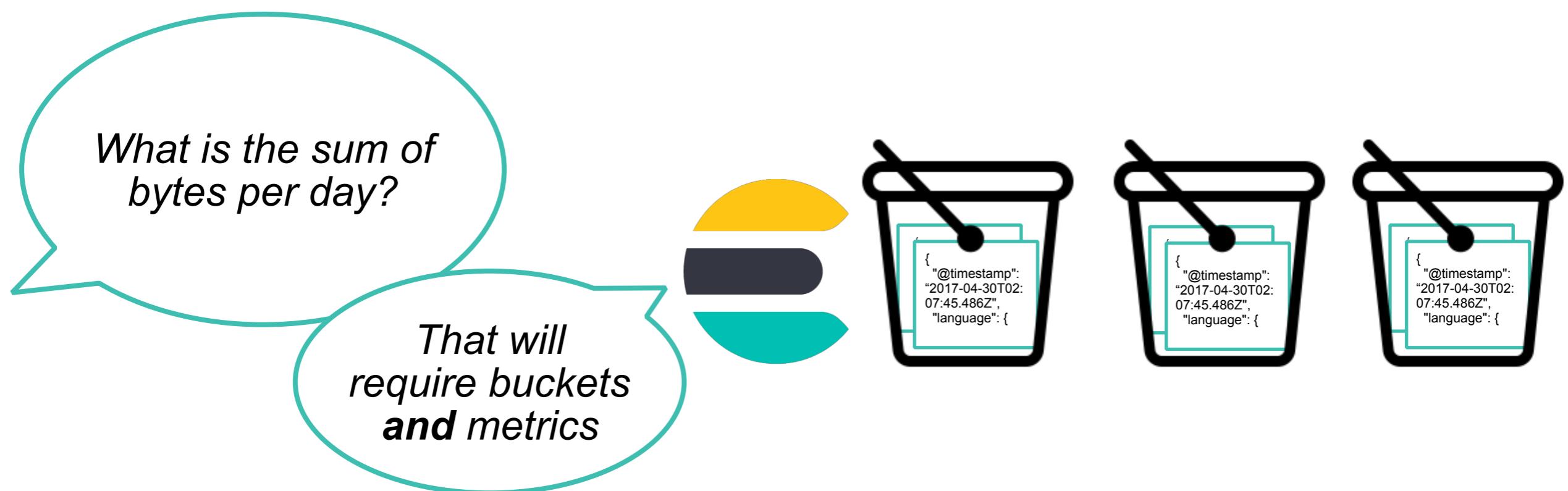
- What is the sum of bytes per day?
- What is the number of bytes served daily and median response time?
- What is the number of bytes served monthly per host?
- What are the weekly top 5 countries with respect to the number requests?
- What are the two most unexpected countries with respect to the number of requests per week?

# Combined Aggregations: Answers



# Combining Bucket and Metrics Aggregations

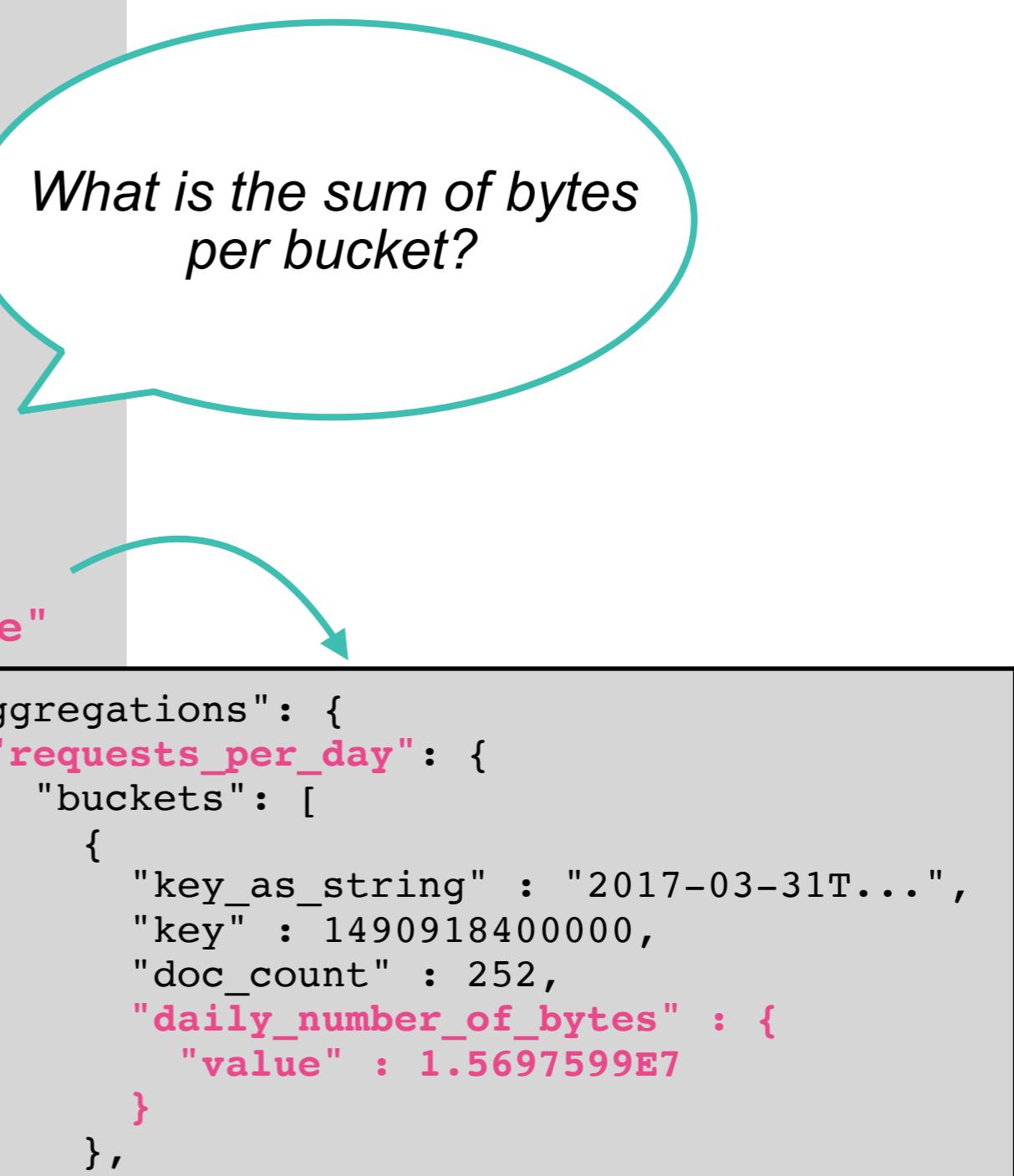
- An *aggregation* can be a combination of *buckets* and *metrics* aggregations
  - this is why aggregations are so powerful - they can be combined in any number of combinations



# Calculating a single Metric per Bucket

- Let's compute a metric in a bucket (besides `doc_count`):

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "requests_per_day": {
      "date_histogram": {
        "field": "@timestamp",
        "interval": "day"
      },
      "aggs": {
        "daily_number_of_bytes": {
          "sum": {
            "field": "response_size"
          }
        }
      }
    }
  }
}
```



# Calculating Multiple Metrics per Bucket

- Just like any bucket, you can calculate multiple metrics on a date histogram

```
"aggs": {  
  "requests_per_day": {  
    "date_histogram": {  
      "field": "@timestamp",  
      "interval": "day"  
    },  
    "aggs": {  
      "daily_number_of_bytes": {  
        "sum": {  
          "field": "response_size"  
        },  
        "median_runtime": {  
          "percentiles": {  
            "field": "runtime_ms",  
            "percents": [50]  
          }  
        }  
      }  
    }  
  }  
}
```

*What is the  
number of bytes served  
daily and median  
response time?*

```
"requests_per_day" : {  
  "buckets" : [  
    {  
      "key_as_string" : "2017-03-31T...",  
      "key" : 1490918400000,  
      "doc_count" : 252,  
      "daily_number_of_bytes" : {  
        "value" : 1.5697599E7  
      },  
      "median_runtime" : {  
        "values" : {  
          "50.0" : 96.0  
        }  
      },  
    },  
  ]  
},
```

# Sorting by a Metric

- You can also sort by a metric value in a sub-aggregation

```
"aggs": {  
  "requests_per_day": {  
    "date_histogram": {  
      "field": "@timestamp",  
      "interval": "day",  
      "order": {  
        "daily_number_of_bytes": "desc"  
      }  
    },  
    "aggs": {  
      "daily_number_of_bytes": {  
        "sum": {  
          "field": "response_size"  
        }  
      },  
      "median_runtime": {  
        "percentiles": {  
          "field": "runtime_ms",  
          "percents": [50]  
        } } } } }
```

*“Find the day with the maximum bytes served.”*

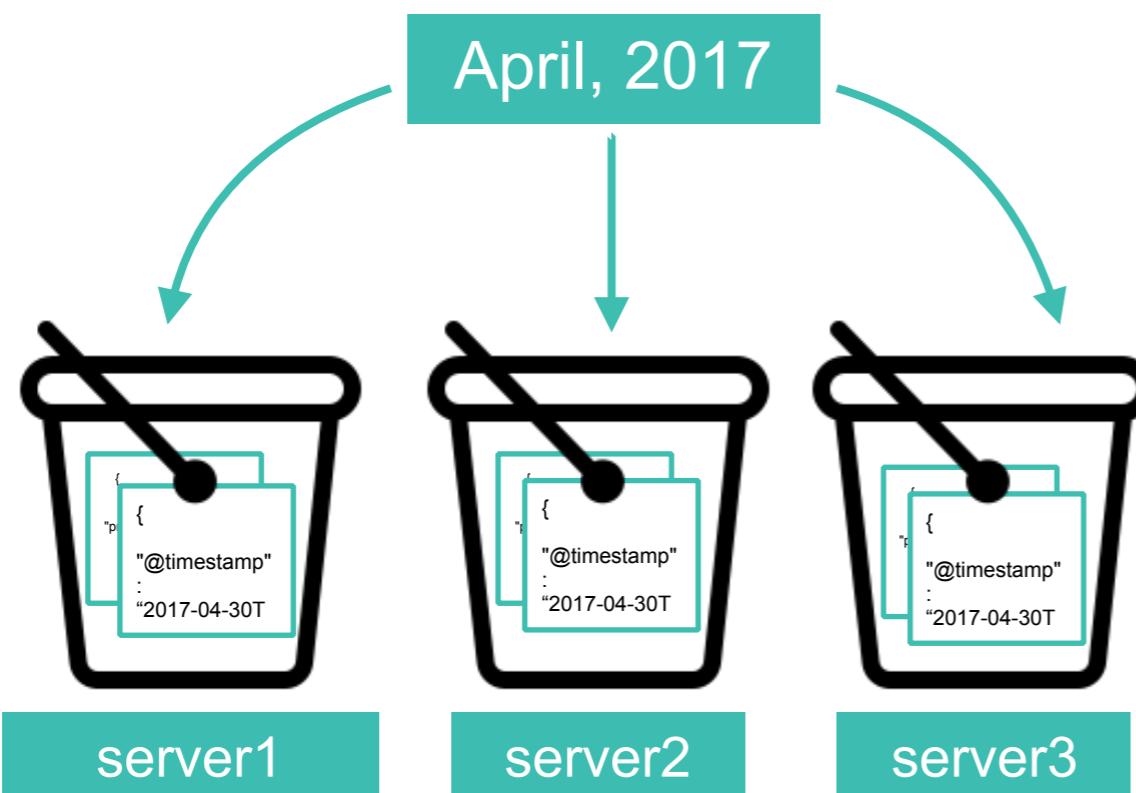
This output will be sorted by the results of the sub-aggregated metric

# Buckets can be Sub-Divided

- Suppose we want to analyze logs by month and also by host:



A bucket can consist of sub-buckets within it



Logs from April, 2017,  
bucketed by host

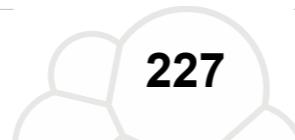


# Sub-Buckets

- What question gets answered by the following **terms** aggregation nested inside a **date\_histogram** aggregation?

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "logs_by_month": {
      "date_histogram": {
        "field": "@timestamp",
        "interval": "month"
      },
      "aggs": {
        "host_name": {
          "terms": {
            "field": "host.keyword",
            "size": 10
          }
        }
      }
    }
  }
}
```

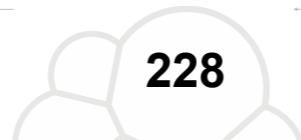
The “**terms**” agg is sub-bucketed inside the “**date\_histogram**”



# Sub-Buckets

- The log events are bucketed *by month*,
  - then within each month the events are further bucketed *by host*

```
"aggregations": {  
    "logs_by_month": {  
        "buckets": [  
            {  
                "key_as_string" : "2017-03-01T00:00:00.000Z",  
                "key" : 1488326400000,  
                "doc_count" : 252,  
                "host_name" : {  
                    "doc_count_error_upper_bound" : 0,  
                    "sum_other_doc_count" : 0,  
                    "buckets" : [  
                        {  
                            "key" : "server2",  
                            "doc_count" : 147  
                        },  
                        {  
                            "key" : "server3",  
                            "doc_count" : 91  
                        },  
                        {  
                            "key" : "server1",  
                            "doc_count" : 14  
                        }  
                    ]  
                }  
            }  
        ]  
    }  
}
```



# Motivation for top\_hits Aggregation

- Suppose we do a search for blogs about Logstash filters, and we want to bucket them by author:

```
GET blogs/_search
{
  "size": 0,
  "query": {
    "match": {
      "content": "logstash filters"
    }
  },
  "aggs": {
    "blogs_by_author": {
      "terms": {
        "field": "author.keyword"
      }
    }
  }
}
```

Suyog wrote 69 blogs about Logstash filters, but which blogs are the most relevant?

```
"buckets": [
  {
    "key": "Suyog Rao",
    "doc_count": 69
  },
  {
    "key": "Alexander Reelsen",
    "doc_count": 67
  },
  {
    "key": "Megan Wieling",
    "doc_count": 31
  },
  {
    "key": "Leslie Hawthorn",
    "doc_count": 29
  },
  {
    "key": "Jesper Mikkelsen",
    "doc_count": 25
  }
]
```

# Example of top\_hits Aggregation

```
GET blogs/_search
{
  "size": 0,
  "query": {
    "match": {
      "content": "logstash filters"
    }
  },
  "aggs": {
    "blogs_by_author": {
      "terms": {
        "field": "author.keyword"
      },
      "aggs": {
        "logstash_top_hits": {
          "top_hits": {
            "size": 5
          }
        }
      }
    }
  }
}
```

Returns the top 5 blogs from each author (based on the `_score` from the “`match`” query)

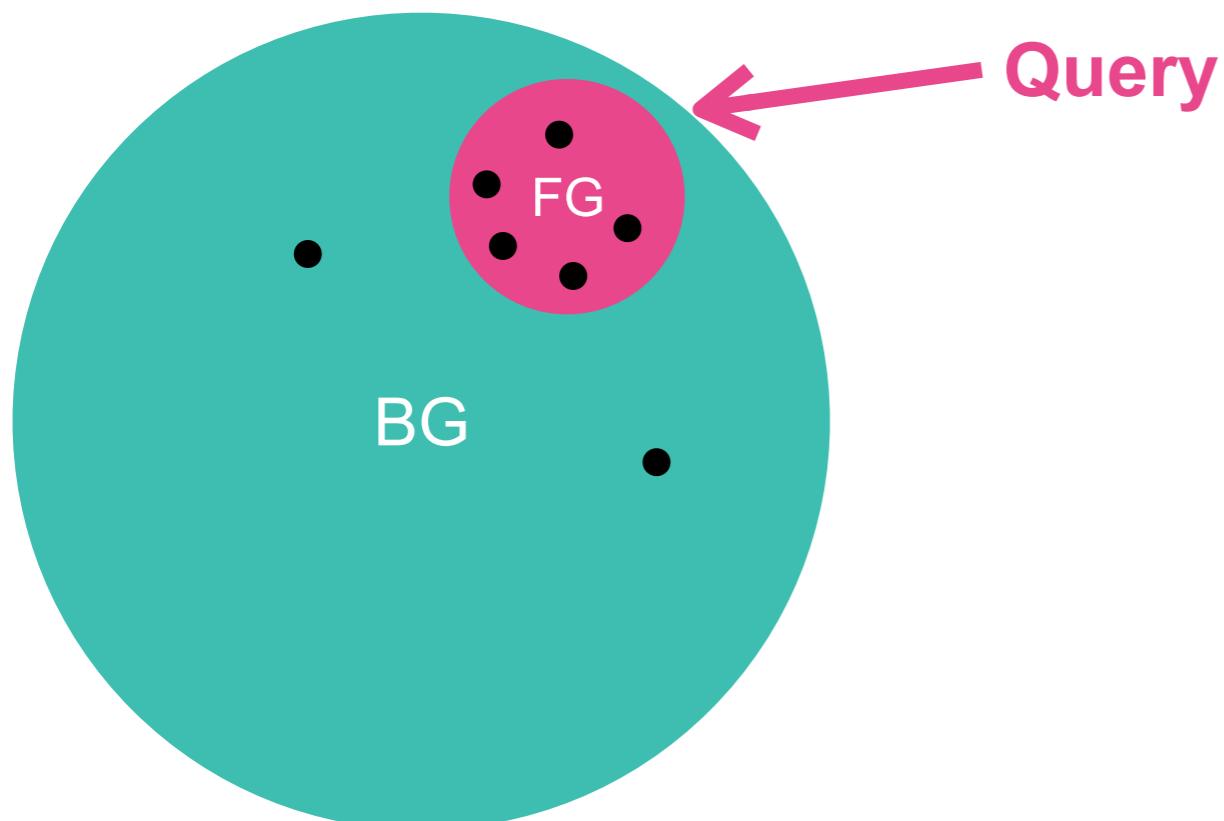
# The top\_hits Aggregation Response

- Notice the top 5 hits from each bucket are returned in the “aggregations” clause of the response

```
"buckets": [
  {
    "key": "Suyog Rao",
    "doc_count": 69,
    "logstash_top_hits": {
      "hits": {
        ...
        "hits": [
          {
            "_index": "blogs",
            "_type": "doc",
            "_id": "TM1CKmIBCLh5xF6i7Y2b",
            "_score": 6.6510196,
            "_source": {
              "publish_date": "2016-06-27T06:00:00.000Z",
              "seo_title": "",
              "category": "The Logstash Lines",
              "locales": "",
              "title": "Logstash Lines: More Monitoring
Info, Beats Input Improvements",
            }
          }
        ]
      }
    }
  }
]
```

# The Significant Aggregations

- Terms Aggregation + Noise Filter
  - discards **commonly common** terms that the **terms** aggregation would return
- Low frequency **terms** in the background data pop out as high frequency **terms** in the foreground data
  - finds **uncommonly common** terms in your dataset



## Some Use Cases:

- Recommendation
- Fraud Detection
- Defect Detection

And more...



# Let's Start with a terms Aggregation

```
GET blogs/_search
{
  "size": 0,
  "aggs": {
    "author_buckets": {
      "terms": {
        "field": "author.keyword",
        "size": 10
      },
      "aggs": {
        "content_terms": {
          "terms": {
            "field": "content",
            "size": 10
          }
        }
      }
    }
  }
}
```

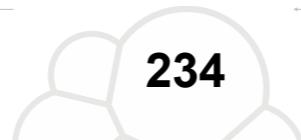
*Let's look for a relationship between **authors** and **content**.*

# The terms Aggregation Response

- These would be considered “***commonly common***” terms that our authors use in their blogs:

```
{  
  "key": "Monica Sarbu",  
  "doc_count": 89,  
  "content_terms": {  
    "doc_count_error_upper_bound": 66,  
    "sum_other_doc_count": 15096,  
    "buckets": [  
      {  
        "key": "and",  
        "doc_count": 89  
      },  
      {  
        "key": "the",  
        "doc_count": 89  
      },  
      {  
        "key": "to",  
        "doc_count": 88  
      },  
      {  
        "key": "in",  
        "doc_count": 86  
      },  
      {  
        "key": "is",  
        "doc_count": 86  
      }  
    ]  
  }  
}
```

Monica likes to blog about “**and**”,  
“**the**”, “**to**”, “**in**” and so on.



# Switching to significant\_text Aggregation

- Try the same aggregation with **significant\_text**:

```
GET blogs/_search
{
  "size": 0,
  "aggs": {
    "author_buckets": {
      "terms": {
        "field": "author.keyword",
        "size": 10
      },
      "aggs": {
        "content_significant_text": {
          "significant_text": {
            "field": "content",
            "size": 10
          }
        }
      }
    }
  }
}
```

*Let's look for the  
“uncommonly common”  
relationship between  
**author** and **content**.*

# The significant\_text Aggregation Response

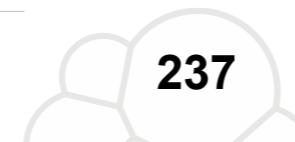
```
"key" : "Monica Sarbu",
"doc_count" : 89,
"content_significant_text" : {
  "doc_count" : 89,
  "bg_count" : 1729,
  "buckets" : [
    {
      "key" : "metricbeat",
      "doc_count" : 66,
      "score" : 9.060797540662497,
      "bg_count" : 97
    },
    {
      "key" : "filebeat",
      "doc_count" : 66,
      "score" : 6.40752430248706,
      "bg_count" : 133
    },
    {
      "key" : "beat",
      "doc_count" : 56,
      "score" : 5.890098051592813,
      "bg_count" : 105
    },
    {
      "key" : "beats",
      "doc_count" : 84,
      "score" : 5.14387681117837,
      "bg_count" : 253
    },
    ...
  ]
}
```

It appears Monica is an expert on Beats!



# There are Two Significant Aggregations

- There are two different significant aggregations
  - **significant\_terms** (should be used with **keyword** fields)
  - **significant\_text** (should be used with **text** fields)
- It is recommended that you use a significant aggregation as an **inner** aggregation of the **sampler aggregation**
  - it limits the analysis to a small selection of top-matching documents (e.g 200)
  - it will typically improve speed, memory use and quality of results
- **This blog** is a good reference





Elasticsearch Aggregations

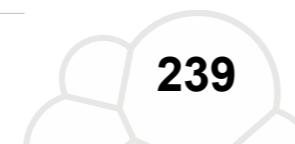
Lesson 3

# Review - Combining Aggregations



# Summary

- An aggregation can be a combination of bucket and metrics aggregations
- The **top\_hits** aggregation returns the most relevant documents for each bucket
- The **significant\_text** aggregation finds uncommonly common terms in your dataset



# Quiz

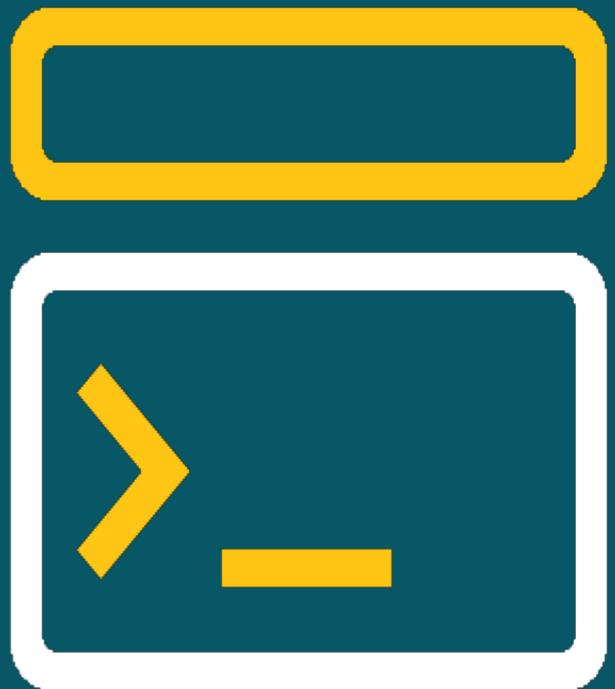
- Explain which aggregations answer the questions below:
  1. On which days did we not meet our SLAs (95% of the requests took less than 500ms)?
  2. How many requests per day by response code?
  3. What are the most significant terms of the top 3 authors?



Elasticsearch Aggregations

Lesson 3

# Lab - Combining Aggregations



- Elasticsearch Fundamentals
- Elasticsearch Queries
- Elasticsearch Aggregations
- Elasticsearch Text Analysis and Mappings
- Elasticsearch Nodes and Shards
- Elasticsearch Monitoring and Troubleshooting

Module 4

# Elasticsearch Text Analysis and Mappings



# Topics

- What is a Mapping?
- Text and Keyword Strings
- The Inverted Index and Doc Values
- Custom Mappings



Elasticsearch Text Analysis and Mappings

Lesson 1

# What is a Mapping?

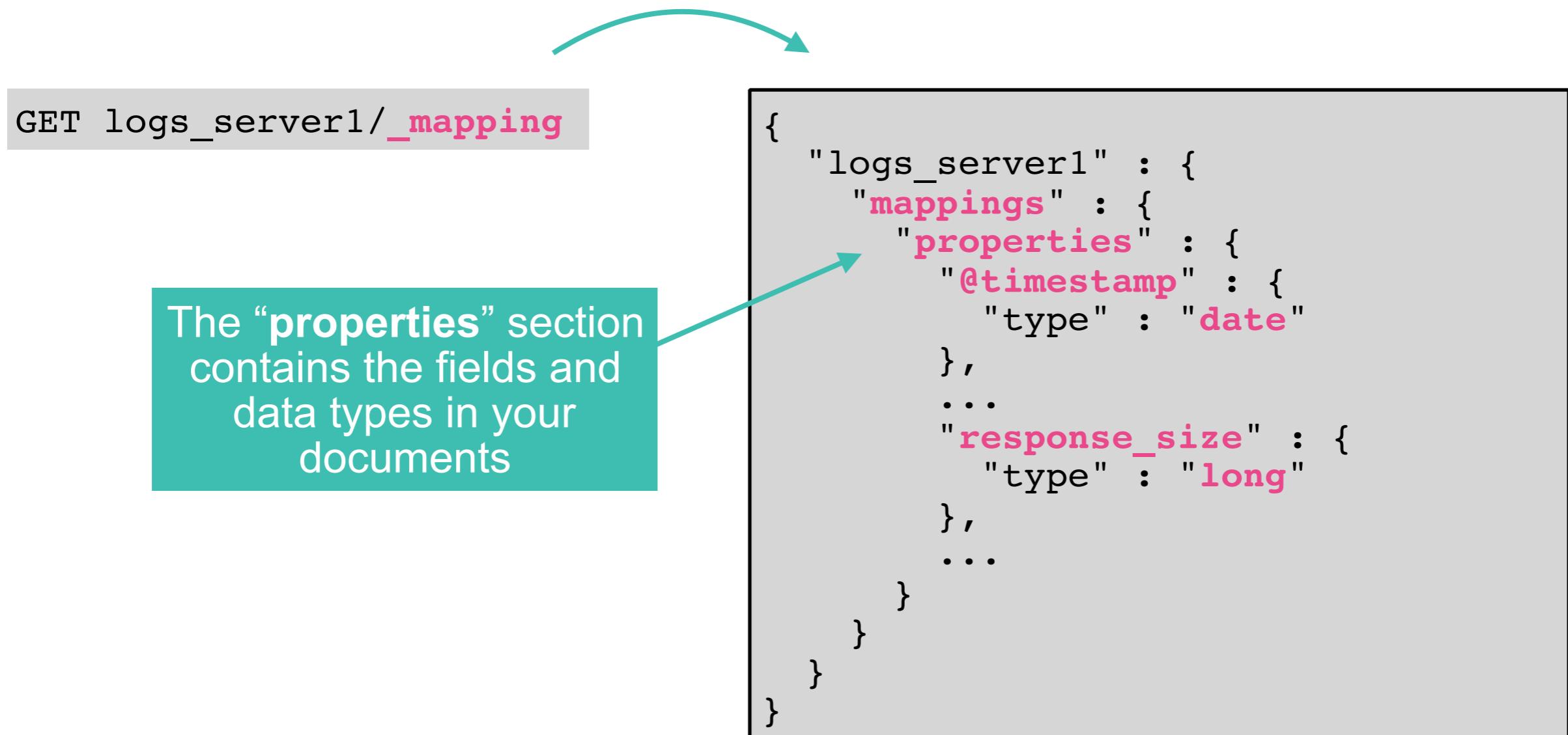


# What is a Mapping?

- 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:
  - names of fields
  - data types of fields
  - how the field should be indexed and stored by Lucene
- Mappings map your complex JSON documents into the simple flat documents that Lucene expects

# Let's View the Mapping of the Logs:

- Mappings are defined per index



# Elasticsearch Data Types for Fields

- **Simple Types, including:**
  - **text**: for full text (analyzed) strings
  - **keyword**: for exact value strings
  - **date** and **date\_nanos**: string formatted as dates, or numeric dates
  - integer types: like **byte**, **short**, **integer**, **long**
  - floating-point numbers: **float**, **double**, **half\_float**, **scaled\_float**
  - **boolean**
  - **ip**: for IPv4 or IPv6 addresses
- **Hierarchical Types**: like **object** and **nested**
- **Specialized Types**: **geo\_point**, **geo\_shape**, **percolator**, **range** types and more

we will talk about  
these later

# Defining 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:

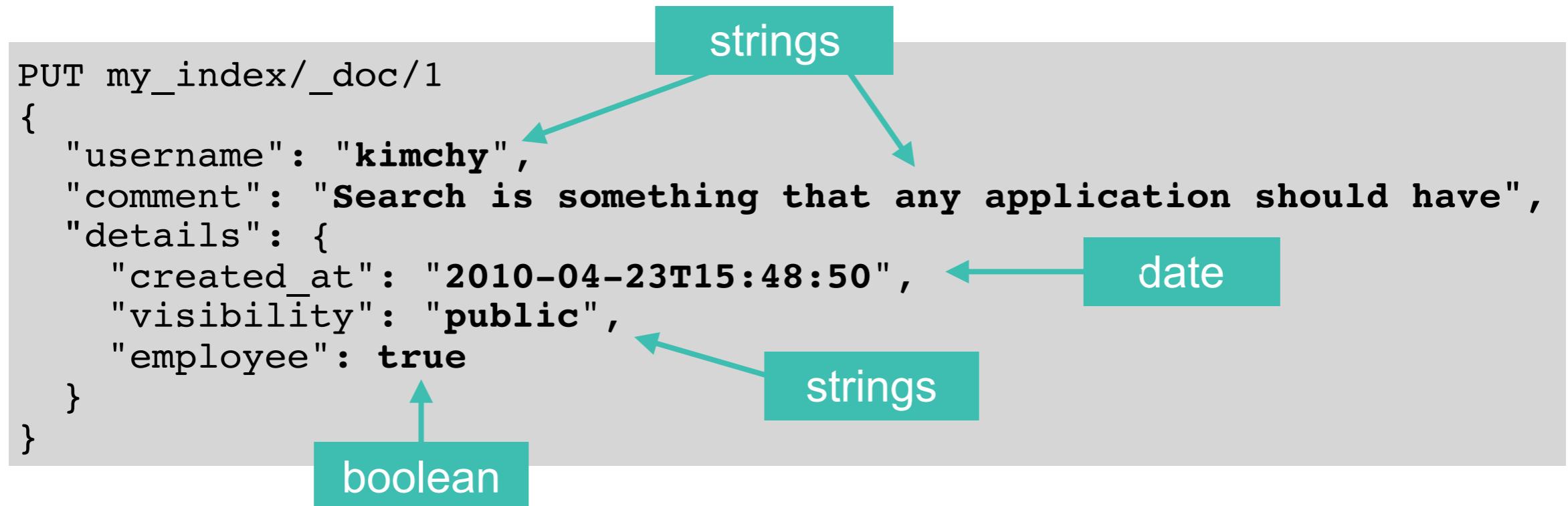
```
PUT my_index
{
  "mappings": {
    define mappings here
  }
}
```

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

```
PUT my_index/_mapping
{
  additional mappings here
}
```

# Why have we not defined any mappings yet?

- You are not required to manually define mappings
  - when you index a document, Elasticsearch *dynamically creates or updates the mapping*
- By default:
  - a new mapping is defined if one does not exist
  - fields not already defined in a mapping are added



# Elasticsearch “Guesses” Data Types:

Sometimes it conveniently  
guesses what you want...

```
PUT my_index/_doc
{
  "status_code": 404
}
```

```
"status_code": {
  "type": "long"
},
```

# Elasticsearch “Guesses” Data Types:

Dates in the default format are also recognized

```
PUT my_index/_doc
{
  "my_date": "2019-07-08"
```

```
"my_date": {
  "type": "date"
}
```

# Custom Mappings

# Dynamic Mappings are not Always Optimal

- The HTTP `status_code` field was dynamically mapped to a `long` in the logs indices
  - Do you need 64 bits to index a number that will never be higher than 511?

```
GET logs_server1/_mapping
```



```
{  
  "logs_server1" : {  
    "mappings" : {  
      "properties" : {  
        [ ... ]  
        "status_code" : {  
          "type" : "long"  
        }  
        [ ... ]  
      }  
    }  
  }  
}
```



# Defining Your Own Mappings

- By defining your own mappings, indexing and searching can be more efficient

```
PUT my_logs
{
  "mappings": {
    "properties": {
      "status_code": {
        "type": "short"
      }
    }
  }
}
```

A 16 bit short is  
more efficient than  
the default 64 bit  
long

# Dynamic Mappings are not Always Optimal

- The same is true for floating point numbers

```
POST my_index/_doc
{
  "price": 1.99
}
```

Do you need a 32 bit float to index a number when you are only interested in two digits after the decimal point?

```
{
  "my_index" : {
    "mappings" : {
      "properties" : {
        "price" : {
          "type" : "float"
        }
      }
    }
  }
}
```

# Use Scaled Floats Instead

- **Scaled floats** are backed by a **long**
  - long values can be efficiently compressed by Lucene, saving disk space
  - configure the precision you need with a **scaling\_factor**

```
PUT my_index2
{
  "mappings": {
    "properties": {
      "price": {
        "type": "scaled_float",
        "scaling_factor": 100
      }
    }
  }
}
```

The **price** field is mapped as a **scaled float**

The **scaling\_factor** of **100** gives this field a precision of two digits after the decimal point

# Can you change a mapping?

- **No** - not without reindexing your documents
  - adding fields is possible
  - ... all other mapping changes require reindexing
- **Why not?**
  - if you switch a field's data type, all existing documents with that field already indexed would become unsearchable on that field
- **Why can I add fields without reindexing?**
  - adding a field has no effect on the existing indexed fields or documents
  - the index can freely grow, but indexed fields can not change data types



# A Word on Types

# What were types?

- Before 7.0 each document had a type
  - but types were a design mistake...
  - ...so we are moving away from types
- Since 7.0 specifying types in requests is **deprecated**
  - `_doc` is now an endpoint and not the type anymore
- In 8.0 specifying types in requests is **not supported**
- <https://www.elastic.co/guide/en/elasticsearch/reference/current/removal-of-types.html>



Elasticsearch Text Analysis and Mappings

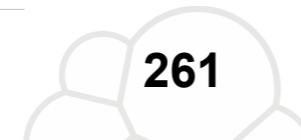
Lesson 1

# Review - What is a Mapping?



# Summary

- Mappings are Elasticsearch' ***data schema***
- Mappings are defined ***per index***
- If you do not define an explicit mapping, Elasticsearch will ***dynamically*** map the fields in your documents
- You ***cannot change*** the mapping of a field, after the index has been created
- You ***can add new things*** to a mapping
- You should optimize your mappings so Elasticsearch can index and query your data most efficiently



# Quiz

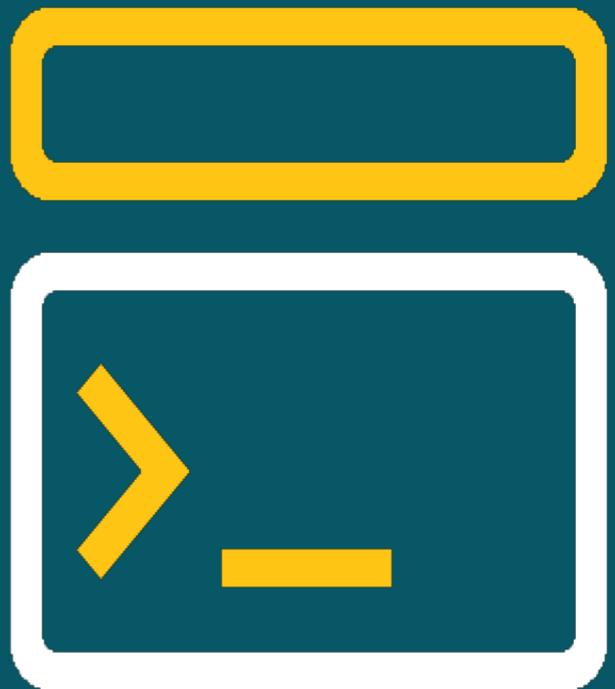
- 1. True or False:** Mappings are defined per index.
- 2. What data type would "my\_field" : 300 get mapped to dynamically?**
- 3. True or False:** You can change a field's data type from "integer" to "long" because those two types are compatible.



Elasticsearch Text Analysis and Mappings

Lesson 1

# Lab - What is a Mapping?





Elasticsearch Text Analysis and Mappings

Lesson 2

# Text and Keyword Strings



# Have you noticed...

- ...that your text searches seem to be case-insensitive? Or that punctuation does not seem to matter?
  - this is due to a process referred to as *text analysis* which occurs when your fields are indexed

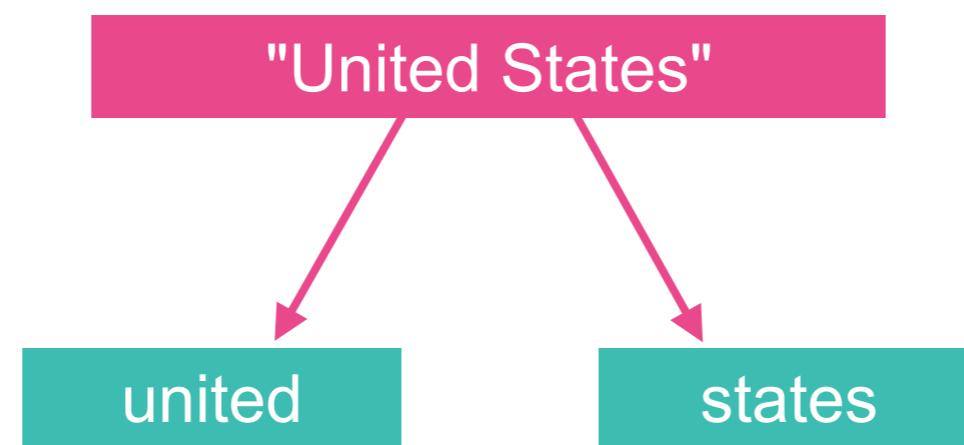
```
GET logs_server*/_search
{
  "query": {
    "match": {
      "geoip.country_name": "united states"
    }
  }
}
```

These two queries return the same hits with the same scoring

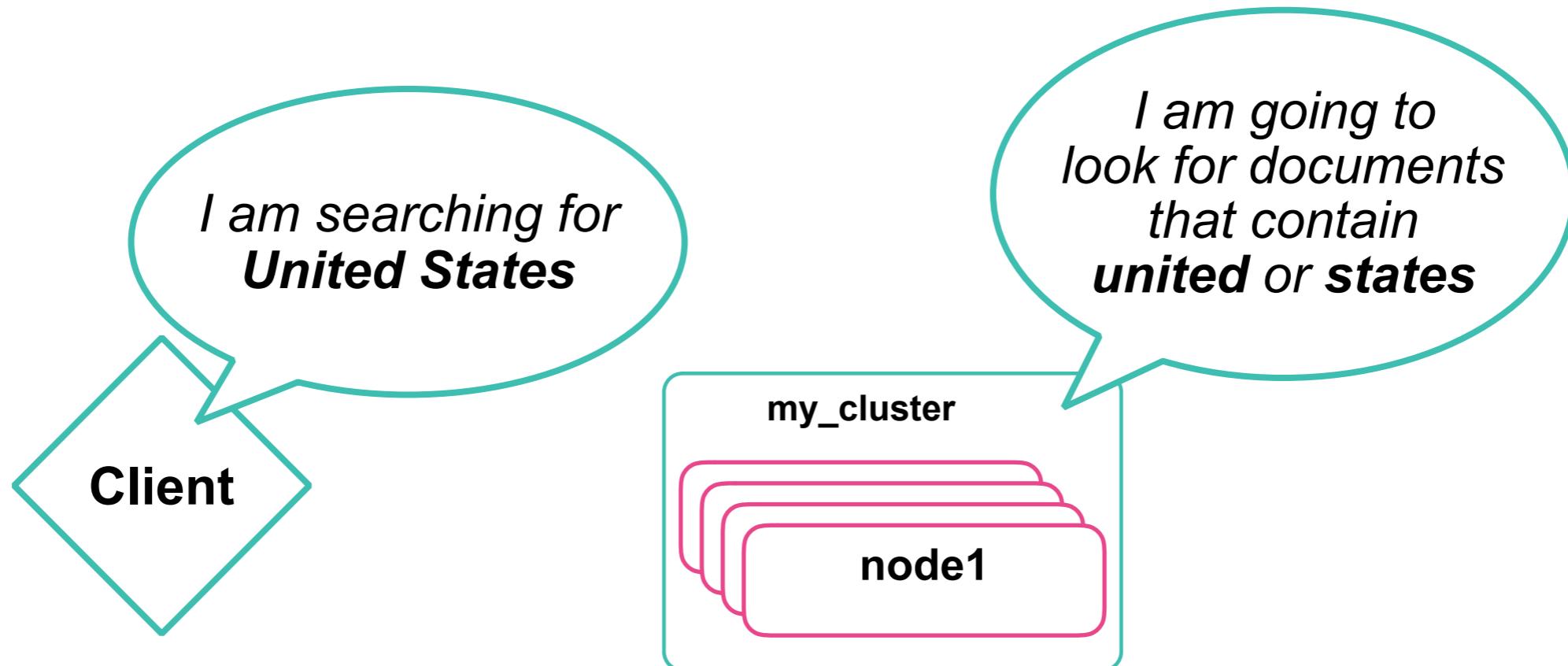
```
GET logs_server*/_search
{
  "query": {
    "match": {
      "geoip.country_name": "United States"
    }
  }
}
```

# Analysis Makes Full Text Searchable

- At index time, text strings get *analyzed*
  - by default, text analysis breaks up a text string into individual words (tokens) and lowercases those words



# Your Query String Gets Analyzed Too



# Analyzers

- Text analysis is done by an **analyzer**
- By default, Elasticsearch applies the **standard** analyzer
- There are many other analyzers, including:
  - **whitespace, stop, pattern, language-specific** analyzers, and more are described in the docs at  
<https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html>
- The built-in analyzers can work great for many use cases
  - but you may need to define your own custom analyzers
  - *we cover that in Elasticsearch Engineer II...*

# Analyzer

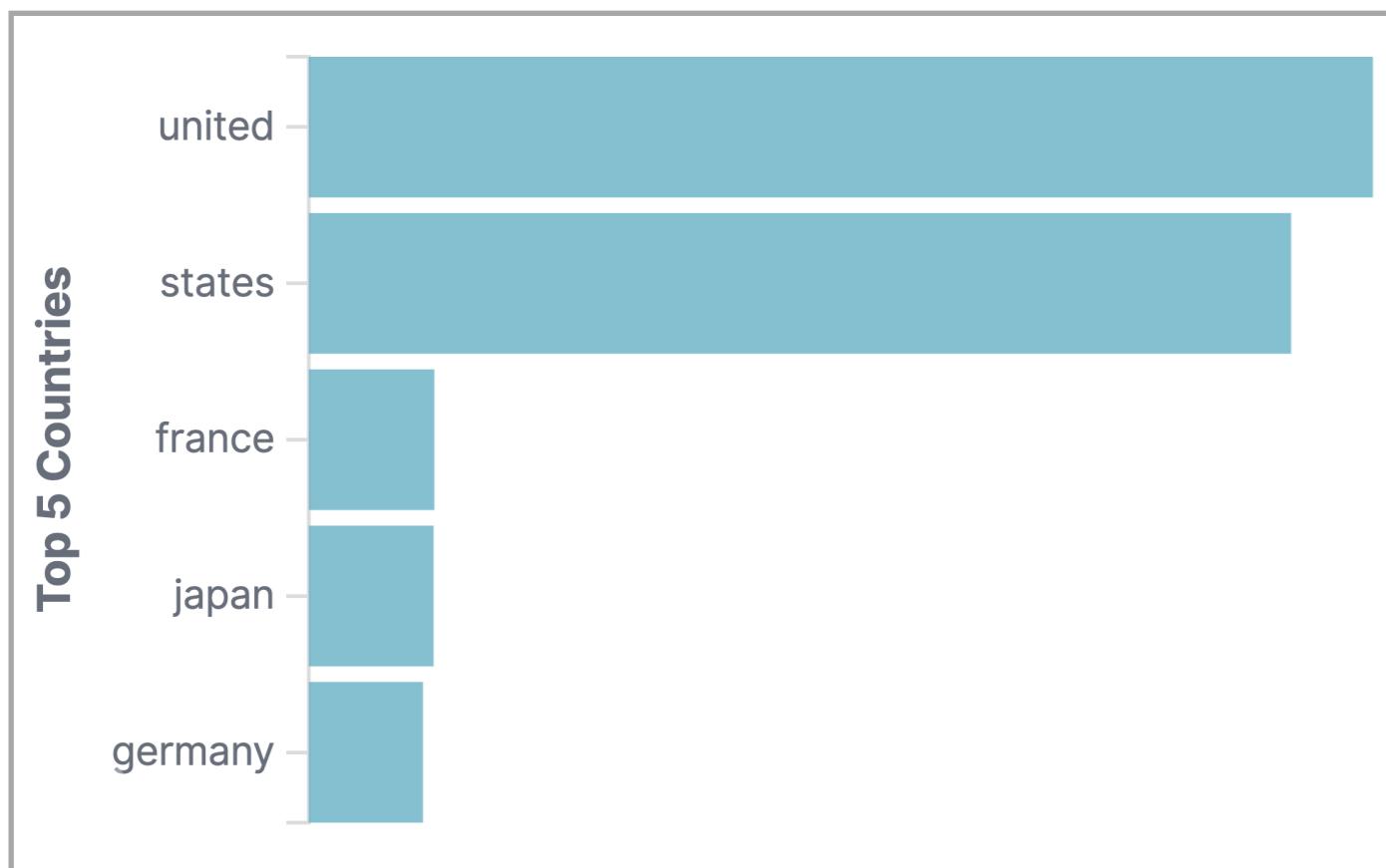
- If you want to apply a different analyzer than the standard analyzer, you can do so in the mappings:

```
{  
  "my_blogs" : {  
    "mappings" : {  
      "properties" : {  
        ...  
        "content" : {  
          "type" : "text",  
          "analyzer": "english"  
        }  
        ...  
      }  
    }  
  }  
}
```

Analyze the **content** field using the **english** analyzer instead of the default **standard** analyzer

# Some strings do not need to be analyzed

- Text Analysis is great for full text search
  - it allows you to search for individual words in a case-insensitive manner
- However, it is not great for things like aggregations
  - when you often want to see the original strings



Instead of "**united**" and "**states**" as separate values, you may want to show "**United States**" in a Kibana chart like this

# Text and Keyword fields

- Elasticsearch has two kinds of string datatypes:
  - **text fields**, for **full text search**
    - text fields are *analyzed*
  - **keyword fields**, for **aggregations, sorting and exact searches**
    - keyword fields are *not analyzed*
    - it stores the original strings, as they occur in the documents



# Multi-Fields

# Understanding Multi-Fields

- Sometimes strings need to be analyzed, sometimes not
  - a string that is used for ***full-text search*** needs to be analyzed
  - a string used for ***sorting*** or ***aggregating*** typically does not
- Elasticsearch does not know what do you want to use a string for, and will give you both by default:
  - it will analyze any string as type **text**
  - and it will create a **keyword multi-field**



# Understanding Multi-Fields

- Multi-fields allow a field to be indexed in multiple ways

```
PUT my_index/_doc/20
{
  "country_name": "United States"
}
```

“country\_name” is analyzed

united

states

“country\_name.keyword” is not analyzed

United States

# Multi-Fields in the Mapping

- An inverted index is built for **each text or keyword field in a mapping**

GET my\_index/\_mapping

In addition to the **country\_name** field of type **text**, Elasticsearch also created the **country\_name.keyword** multi-field of type **keyword**

```
{  
  "my_index" : {  
    "mappings" : {  
      "properties" : {  
        ...  
        "country_name" : {  
          "type" : "text",  
          "fields" : {  
            "keyword" : {  
              "type" : "keyword",  
              "ignore_above" : 256  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

It is a bit confusing because “**keyword**” is the name of the multi-field **and** the data type!

# Do you always need Text and Keyword?

- Probably not! Indexing every string twice:
  - slows down indexing
  - takes up *more disk space*
- Think what do you want to do with each string field:
  - some strings need to be full text searchable
  - some other strings are only going to be used in aggregations
  - some strings need to support both use cases
- ***Optimize*** the mapping to support your use case

# Optimized Mappings

```
PUT my_logs
{
  "mappings": {
    "properties": {
      "message": {
        "type": "text"
      },
      "http_version": {
        "type": "keyword"
      },
      "country_name": {
        "type": "text",
        "fields": {
          "keyword": {
            "type": "keyword",
            "ignore_above": 256
          }
        }
      }
    }
  }
}
```

The **message** field is a **text** field. You can use the field to search for individual words

The **http\_version** is a **keyword** field. It can be used for exact searches and aggregations

The **country\_name** field has been indexed twice, so there is full flexibility for that field



Elasticsearch Text Analysis and Mappings

Lesson 2

# Review - Text and Keyword Strings



# Summary

- Elasticsearch has two kinds of string datatypes: ***text*** and ***keyword***
- Text fields are for ***full text search***
- Keyword fields are for ***exact searches, aggregations*** and ***sorting***
- By default, every string gets dynamically mapped twice: as a ***text*** field and as a ***keyword multi-field***
- You can ***optimize*** your mapping by choosing either text or keyword explicitly (or both!)

# Quiz

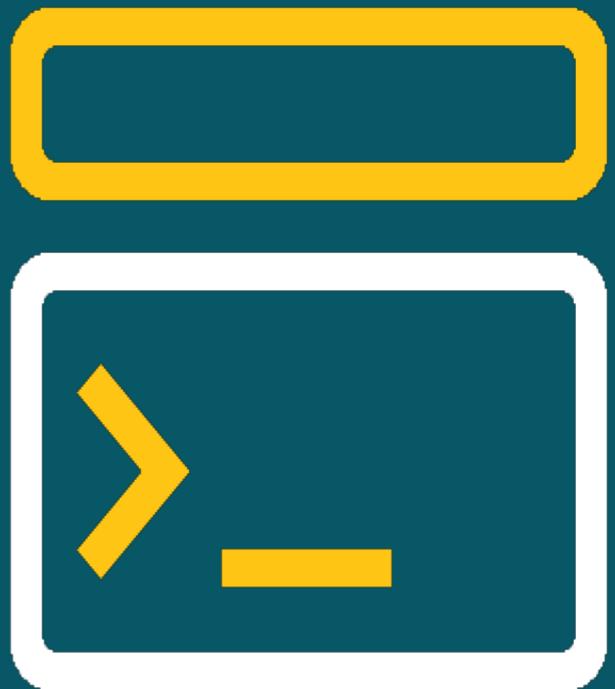
- 1. True or False:** By default, full text searches in Elasticsearch are **case-sensitive**.
- 2.** How many times does a **dynamically mapped** string field get indexed by default?
- 3. True or False:** Aggregations often give the best results when performed against a **keyword** field.



Elasticsearch Text Analysis and Mappings

Lesson 2

# Lab - Text and Keyword Strings





Elasticsearch Text Analysis and Mappings

Lesson 3

# The Inverted Index and Doc Values



# What is an Inverted Index?

- You are familiar with the index in the back of a book
  - useful terms from the book are sorted, and page numbers tell you where to find that term
- Lucene creates a similar *inverted index* for your fields

match\_all query, 103

isolated aggregations in scope of, 446

score as neutral 1, 111

match\_all query clause, 98, 175

match\_mapping\_type setting, 149

match\_phrase query, 242

documents matching a phrase, 243

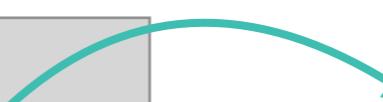
on multivalue fields, 245

Page 242 contains information about the **match\_phrase** query

# Building an Inverted Index

- Let's look at how a field like `country_name` is analyzed and added to an inverted index:
  - text is broken into tokens
  - tokens are lowercased
  - indexed into a sorted list
  - with the IDs of the documents that the tokens occur in

```
PUT my_index/_doc/1
{
  "country_name": "United States"
}
PUT my_index/_doc/2
{
  "country_name": "United States"
}
PUT my_index/_doc/3
{
  "country_name": "United Kingdom"
}
```

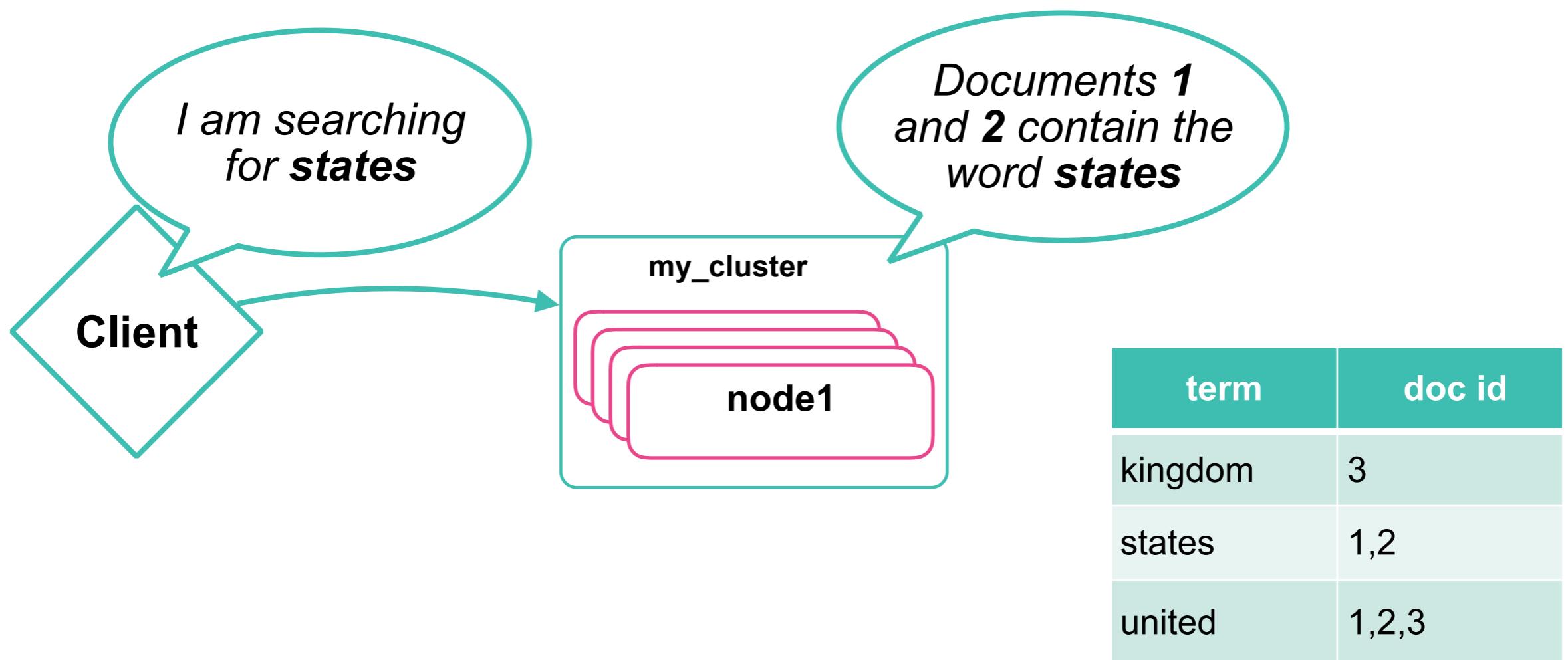


term	doc id
kingdom	3
states	1,2
united	1,2,3



# The Inverted Index Makes Searching Fast

- Thanks to the inverted index, Elasticsearch can immediately return matching document IDs
  - instead of having to evaluate every document



# Limitations of the Inverted Index

- Let's try something simple, like *sorting* logs by country name:

```
GET my_index/_search
{
  "query": {
    "match": {
      "geoip.country_name": "united"
    }
  },
  "sort": {
    "geoip.country_name": {
      "order": "asc"
    }
  }
}
```

This simple-looking query  
actually fails. Why?

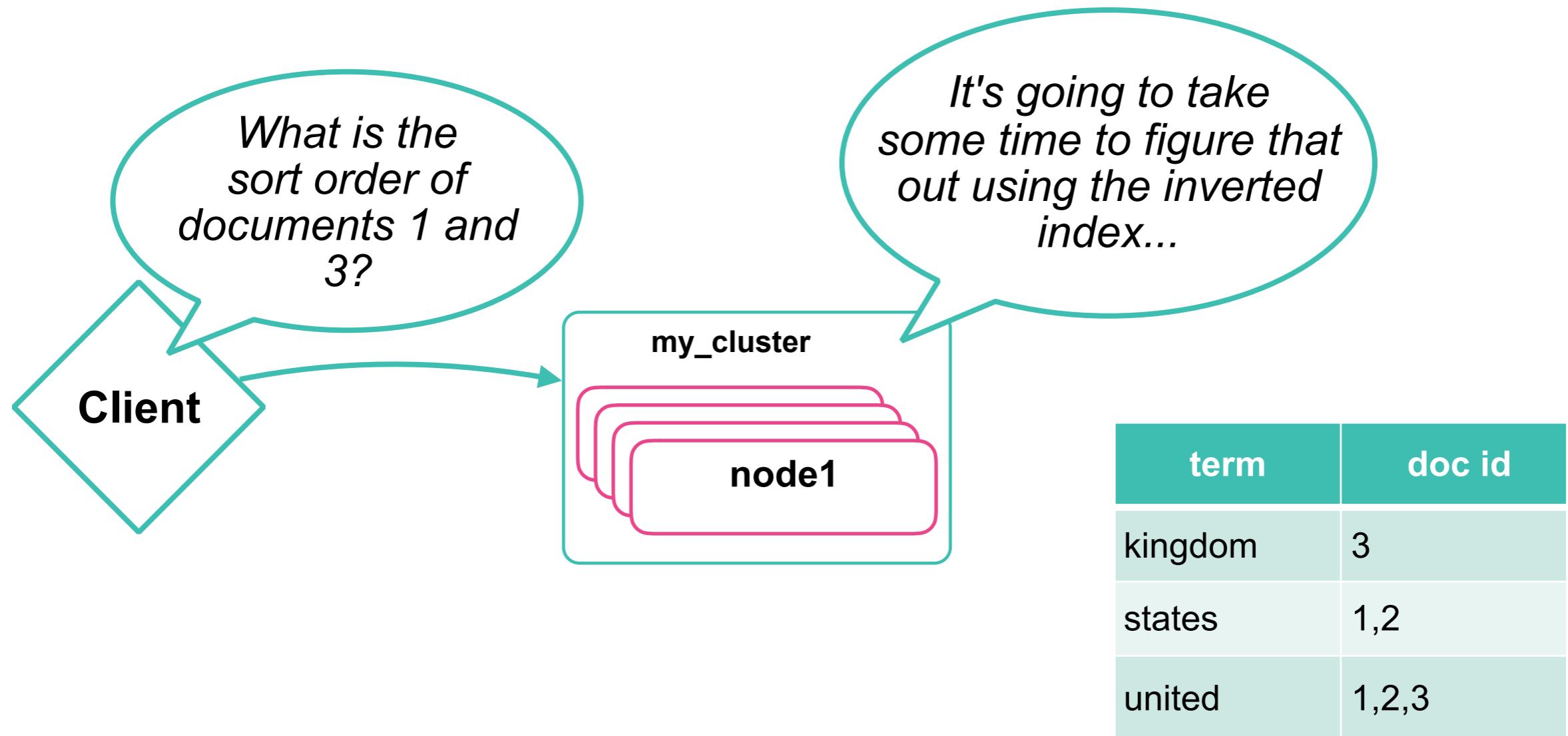
# The Error Mentions Keyword Fields

- Here is the error message from the previous query
  - we are being advised to use a **keyword** field instead

```
{  
  "error": {  
    "root_cause": [  
      {  
        "type": "illegal_argument_exception",  
        "reason": "Fielddata is disabled on text fields by  
default. Set fielddata=true on [geoip.country_name] in order  
to load fielddata in memory by uninverting the inverted  
index. Note that this can however use significant memory.  
Alternatively use a keyword field instead."  
      }  
    ],  
    ...  
  }  
}
```

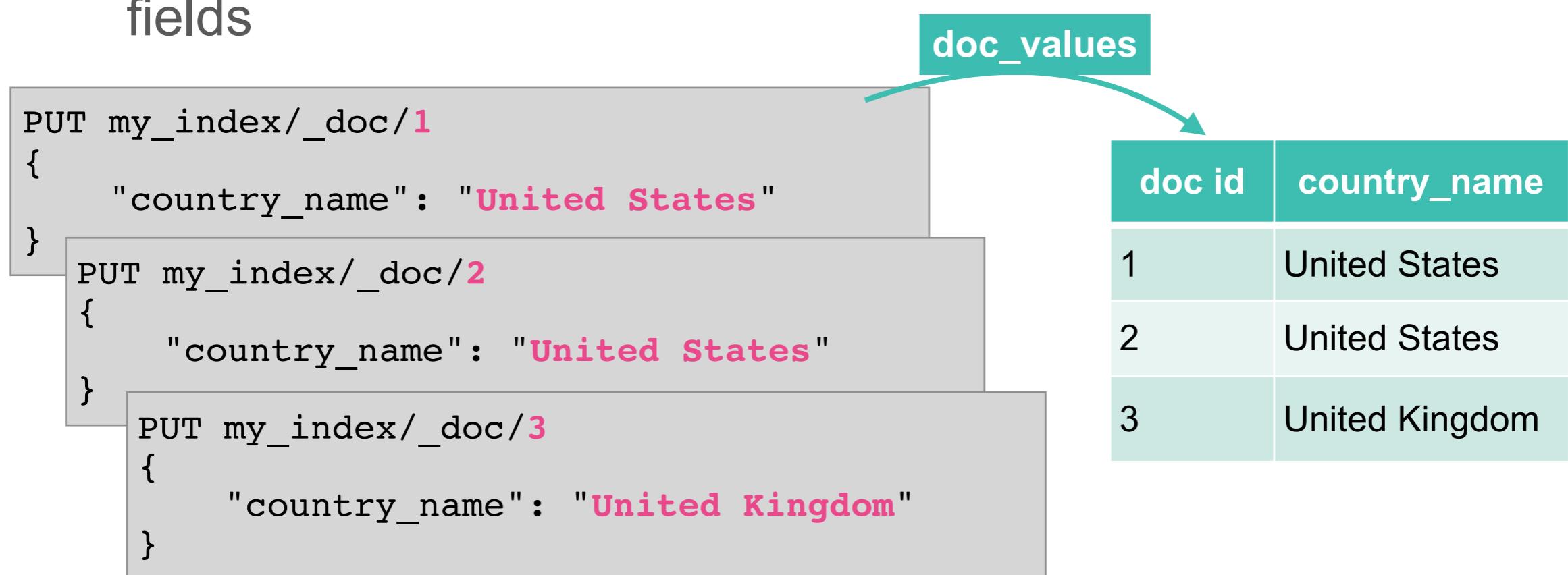
# Limitations of the Inverted Index

- The inverted index is great for searching, but not for things like sorting



# Doc Values

- Lucene can create a second data structure out of your documents, called **doc values**
- Doc values are like a columnar store
  - telling you what the value of a field is, for a given document ID
  - doc values do not exist for text fields, but they exist for keyword fields



# So how do I sort by analyzed text?

- ...or perform an aggregation or use it in a script?
  - well, you should ***avoid that scenario*** when possible
- Consider using the **keyword** multi-field, that Elasticsearch will create for you by default
  - keyword fields have doc values enabled by default

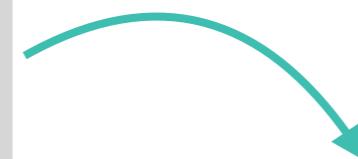
```
"country_name" : {  
    "type" : "text",  
    "fields" : {  
        "keyword" : {  
            "type" : "keyword",  
            "ignore_above" : 256  
        }  
    }  
}
```

In the mappings of the logs-server\* indices, Elasticsearch has dynamically created a **keyword multi-field** for the **country\_name** field

# Sorting by Keyword Fields

- Instead of sorting by `geoip.country_name` you need to sort by its **keyword** multi-field
  - keyword fields have **doc values** enabled by default

```
GET my_index/_search
{
  "query": {
    "match": {
      "geoip.country_name": "united"
    }
  },
  "sort": {
    "geoip.country_name.keyword": {
      "order": "asc"
    }
  }
}
```



```
"country_name": "United Kingdom",
```

```
"country_name": "United States",
```



# Optimizing the Mappings

# Not Indexing a Field

- Suppose you know that you will *not run any queries* on the `http_version` field of the access logs
  - it is possible to not create an inverted index for that field
  - the field is still returned in the `_source` and can be used in aggregations, but the field will not be queryable
  - less disk space, faster indexing

```
"mappings": {  
    "properties": {  
        "http_version": {  
            "type": "keyword",  
            "index": false  
        }  
    ...  
    }
```

“`http_version`” will  
not be indexed

# Disabling Doc Values

- Or maybe you will *not run any aggregations* on the **http\_version** field of the access logs
  - it is also possible to not have doc values for a specific field
  - the field is still returned in the **\_source** and can be used in queries, but the field cannot be used in aggregations

```
"mappings": {  
    "properties": {  
        "http_version": {  
            "type": "keyword",  
            "doc_values": false  
        }  
    ...  
    }
```

“**http\_version**” will  
not have **doc\_values**

# Disabling a Field

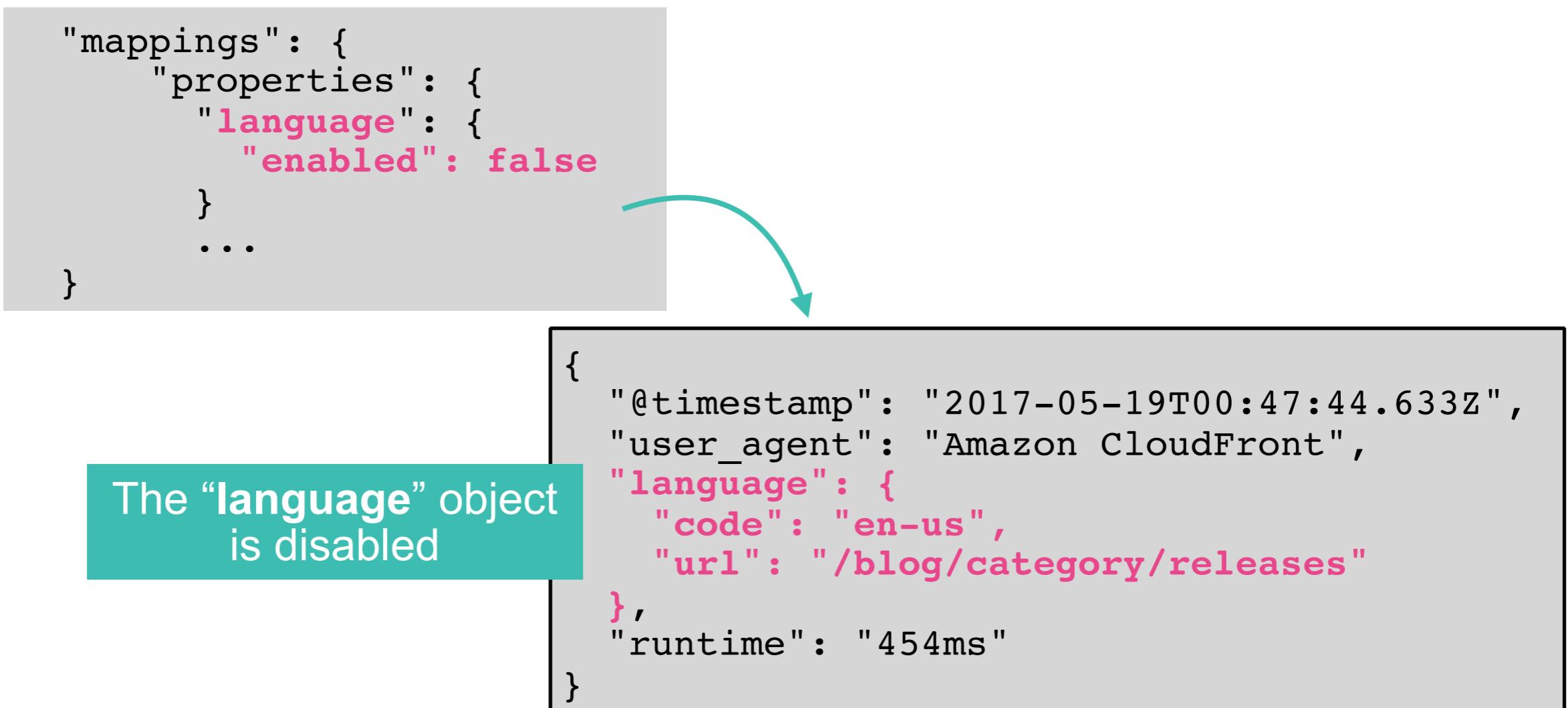
- Another option is to ***completely disable a field:***
  - *you cannot query or aggregate this field*
  - but this field will still be returned in the `_source`

```
PUT my_logs/_mapping
{
  "properties": {
    "http_version": {
      "enabled": false
    }
  }
}
```

“`http_version`” will not be indexed nor stored in `doc_values`. It will still be stored in `_source`

# Disabling an Object

- Setting “enable” to `false` is useful when you want to skip the indexing of an **entire JSON object** in your document
  - suppose you will never perform any queries or aggregations on the fields of the “language” object in the access logs:





Elasticsearch Text Analysis and Mappings

Lesson 3

# Review - Inverted Index and Doc Values



# Summary

- Lucene builds multiple data structures out of your documents: **inverted indices** and **doc values**
- The **inverted index** make searching fast
- **Doc values** allow you to **aggregate** and **sort** on values
- You can **disable** the inverted index or doc values for individual fields in the **mapping**, to optimize Elasticsearch

# Quiz

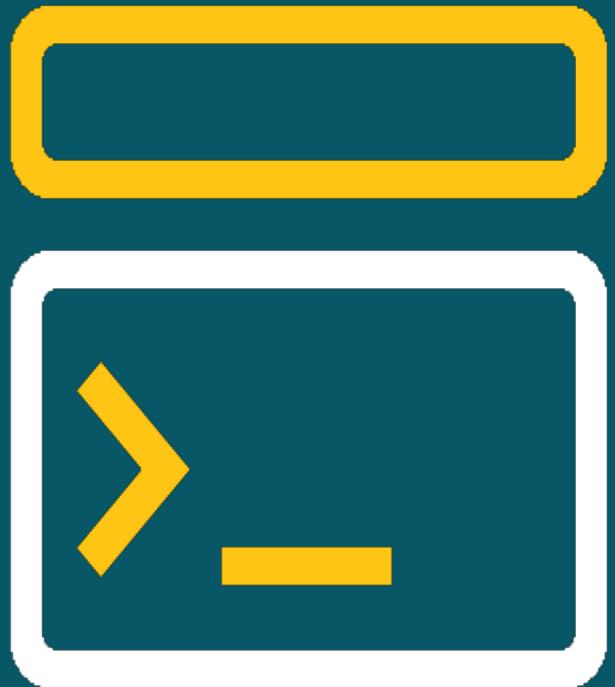
- 1. True or False:** You should **sort** by a **text** field instead of a **keyword** field.
- 2. True or False:** If you set "**index**": **false** for a field, you can no longer query on that field.
- 3. True or False:** When you set "**enabled**": **false** for a field, that field will no longer be returned in the **\_source** with your hits.



Elasticsearch Text Analysis and Mappings

Lesson 3

# Lab - Inverted Index and Doc Values





Elasticsearch Text Analysis and Mappings

Lesson 4

# Custom Mappings



# Mapping Parameters

- There are various parameters that can be applied to fields when defining your mappings
  - for example, you saw the **index** parameter in the last lesson:

```
"mappings": {  
    "properties": {  
        ...  
        "http_version": {  
            "type": "keyword",  
            "index": false  
        }  
        ...  
    }  
}
```

An example of a mapping parameter

- We will discuss a few more mapping parameters now. See the docs for a complete list:
  - <https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-params.html>

# Specifying a Date Format

- If your **date** format is different than standard ISO, you will need a custom mapping
  - choose from dozens of built-in date formats, or define your own
  - <https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-date-format.html>

```
"properties": {  
    "last_viewed" : {  
        "type": "date_nanos",  
        "format": "strict_date_optional_time"  
    },  
    "comment_time" : {  
        "type": "date",  
        "format": "dd/MM/yyyy||epoch_millis"  
    }  
}
```

View the documentation for a complete list of built-in date formats

Use || for multiple formats

# The Use Case for `copy_to`

- The access logs dataset has several fields representing the location of the event:

```
"region_name": "Victoria",
"country_name": "Australia",
"city_name": "Surrey Hills"
```

- Suppose we want to frequently search all three of these fields:
  - we could run a **bool** query with **should** clauses in a **must** clause
  - or we could copy all three values to a single field during indexing using **copy\_to**...

# The copy\_to Parameter

```
"mappings": {  
    "properties": {  
        "region_name": {  
            "type": "keyword",  
            "copy_to": "locations_combined"  
        },  
        "country_name": {  
            "type": "keyword",  
            "copy_to": "locations_combined"  
        },  
        "city_name": {  
            "type": "keyword",  
            "copy_to": "locations_combined"  
        },  
        "locations_combined": {  
            "type": "text"  
        }  
    ...  
}
```

During indexing, the values will be copied to the **“locations\_combined”** field

# The copy\_to Parameter

- The `locations_combined` field is not a part of `_source`, but it is indexed
  - so you can query on it:

```
GET weblogs/_search
{
  "query": {
    "match": {
      "locations_combined": "victoria australia"
    }
  }
}
```

*I am searching for events in “Victoria Australia”*

```
"hits": [
  {
    "_index": "weblogs",
    "_type": "_doc",
    "_id": "1",
    "_score": 0.5753642,
    "_source": {
      "region_name": "Victoria",
      "country_name": "Australia",
      "city_name": "Surrey Hills"
    }
  }
]
```

# Null Values

- When a field is set to **null**, it is treated as if that field has no value:

```
PUT ratings/_doc/1
{
  "rating": null
}
```

```
PUT ratings/_doc/2
{
  "rating": 5.0
}
```

```
GET ratings/_search?size=0
{
  "aggs": {
    "average_rating": {
      "avg": {
        "field": "rating"
      }
    }
  }
}
```



```
"aggregations": {
  "average_rating": {
    "value": 5
  }
}
```

# Specifying a Default Value for nulls

- Use the `null_value` parameter to assign a value to a field if it is `null`
  - the `_source` is not altered, but the value of `null_value` is indexed

```
PUT ratings
{
  "mappings": {
    "properties": {
      "rating": {
        "type": "float",
        "null_value": 1.0
      }
    }
  }
}
```

If “`rating`” is `null`, then  
1.0 will be indexed for  
that field

# Specifying a Default Value for nulls

- Notice the average changes now:

```
PUT ratings/_doc/1
{
  "rating": null
}
```

```
PUT ratings/_doc/2
{
  "rating": 5.0
}
```

```
GET ratings/_search?size=0
{
  "aggs": {
    "average_rating": {
      "avg": {
        "field": "rating"
      }
    }
  }
}
```



```
"aggregations": {
  "average_rating": {
    "value": 3
  }
}
```



# Coercing Data

- By default, Elasticsearch attempts to **coerce** data to match the data type of the field
  - for example, suppose the “rating” field is a “long”:

```
PUT ratings/_doc/1
{
  "rating": 4
}
```

All three PUT  
commands work fine

```
PUT ratings/_doc/2
{
  "rating": "3"
}
```

```
PUT ratings/_doc/3
{
  "rating": 4.5
}
```

A “sum” aggregation on  
“rating” returns “11”

# Disabling Coercion

- You can ***disable coercion*** if you do not want Elasticsearch to try and clean up your dirty fields:

```
"mappings": {  
    "properties": {  
        "rating": {  
            "type": "long",  
            "coerce": false  
        }  
    }  
}
```

Set the “coerce” parameter to **false**

```
PUT ratings/_doc/1  
{  
    "rating": 4  
}  
PUT ratings/_doc/2  
{  
    "rating": "3"  
}  
PUT ratings/_doc/3  
{  
    "rating": 4.5  
}
```

Works fine

Fails

Fails

# Defining Your Own Mappings

# Defining Your Own Mappings

- If you need to define an explicit mapping, we typically follow these steps:
  1. Index a sample document that contains the fields you want defined in the mapping (using a temporary index)
  2. Get the dynamic mapping that was created automatically by Elasticsearch in Step 1
  3. Edit the mapping definition
  4. Create your index, using your explicit mappings

# 1. Index a Sample Document

- Start by indexing a document into a temporary index

```
PUT blogs_temp/_doc/1
{
  "date": "December 22, 2017",
  "author": "Firstname Lastname",
  "title": "Elastic Advent Calendar 2017, Week 3",
  "seo_title": "A Good SEO Title",
  "url": "/blog/some-url",
  "content": "blog content",
  "locales": "ja-jp",
  "@timestamp": "2017-12-22T07:00:00.000Z",
  "category": "Engineering"
}
```

Use values that will map closely to the data types you want

## 2. Get the Dynamic Mapping

- GET the mapping, then copy-and-paste it into the **Console**:

GET **blogs\_temp/\_mapping**

```
{  
  "blogs_temp": {  
    "mappings": {  
      "properties": {  
        "@timestamp": {  
          "type": "date"  
        },  
        "author": {  
          "type": "text",  
          "fields": {  
            "keyword": {  
              "type": "keyword",  
              "ignore_above": 256  
            }  
          }  
        },  
        "category": {  
          "type": "text",  
          "fields": {  
            "keyword": {  
              "type": "keyword",  
              "ignore_above": 256  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

### 3. Edit the Mapping

- It seems like “**keyword**” might work well for “**category**”
- and “**content**” only needs to be “**text**”

```
"category": {  
    "type": "text",  
    "fields": {  
        "keyword": {  
            "type": "keyword",  
            "ignore_above": 256  
        }  
    }  
},  
"content": {  
    "type": "text",  
    "fields": {  
        "keyword": {  
            "type": "keyword",  
            "ignore_above": 256  
        }  
    }  
},  
...  
}  
  
"category": {  
    "type": "keyword"  
},  
"content": {  
    "type": "text"  
},  
"date": {  
    "type": "date",  
    "format": "MMM dd, yyyy"  
},  
"locales": {  
    "type": "keyword"  
},
```

# 4. Define a New Index with the Mapping

- When you are done editing

```
PUT blogs ←  
{  
  "mappings": {  
    "properties": {  
      "@timestamp": {  
        "type": "date"  
      },  
      "author": {  
        "type": "text",  
        "fields": {  
          "keyword": {  
            "type": "keyword",  
            "ignore_above": 256  
          }  
        }  
      },  
      "category": {  
        "type": "keyword"  
      },  
      "content": {  
        "type": "text"  
      },  
      ...  
    }  
  }  
}
```

“blogs” is a new index with our explicit mapping...

...and now you are ready to start indexing!

# Dynamic Templates

# Use Case for Dynamic Templates

- Suppose you have documents with a *large number of fields*,
- or documents with *dynamic field names* not known at the time of your mapping definition
- Using *dynamic templates*, you can define a field's mapping based on:
  - the field's **datatype**,
  - the **name** of the field, or
  - the **path** to the field

# Defining a Dynamic Template

- Suppose you want any unmapped string fields to be mapped as type “**keyword**” by default:

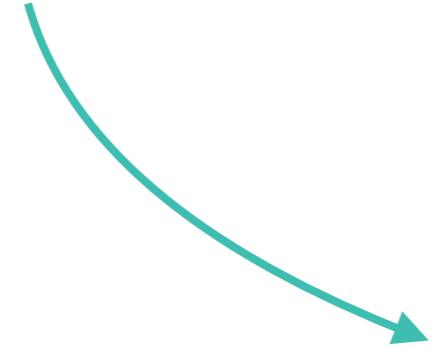
```
PUT test2
{
  "mappings": {
    "dynamic_templates": [
      {
        "my_string_fields": {
          "match_mapping_type": "string",
          "mapping": {
            "type": "keyword"
          }
        }
      }
    ]
  }
}
```

If the field is a JSON string in the document, map it as “**keyword**”

# Test Our Template

```
POST test2/_doc
{
  "blog_reaction": ":thumbsup:"
}

GET test2/_mapping
```



```
"properties": {
  "blog_reaction": {
    "type": "keyword"
  }
}
```



Elasticsearch Text Analysis and Mappings

Lesson 4

# Review - Custom Mappings



# Summary

- **Mapping parameters** allow you to influence how Elasticsearch will index the fields in your documents
- **Dynamic templates** make it easier to set up your own mappings by defining **defaults** for fields, based on their **JSON type, name or path**

# Quiz

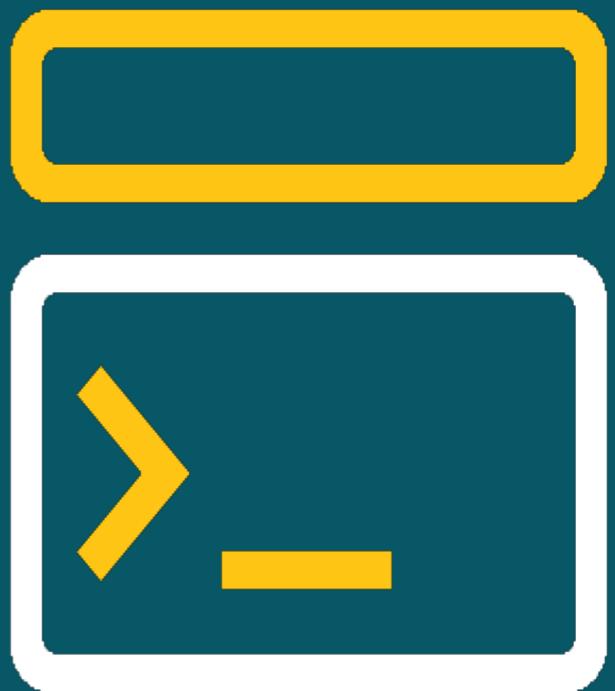
1. What **mapping parameter** would you use to change a field's date format?
2. **True or False:** Using `copy_to` you are adding a new field to the `_source` that gets returned with the hits.
3. **True or False:** It is a common practice to use dynamically generated mapping to create a custom mapping.



Elasticsearch Text Analysis and Mappings

Lesson 4

# Lab - Custom Mappings



- Elasticsearch Fundamentals
- Elasticsearch Queries
- Elasticsearch Aggregations
- Elasticsearch Text Analysis and Mappings
- Elasticsearch Nodes and Shards
- Elasticsearch Monitoring and Troubleshooting

Module 5

# Elasticsearch Nodes and Shards



# Topics

- Master Nodes
- Node Roles
- Understanding Shards
- Distributed Operations



Elasticsearch Nodes and Shards

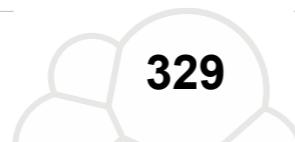
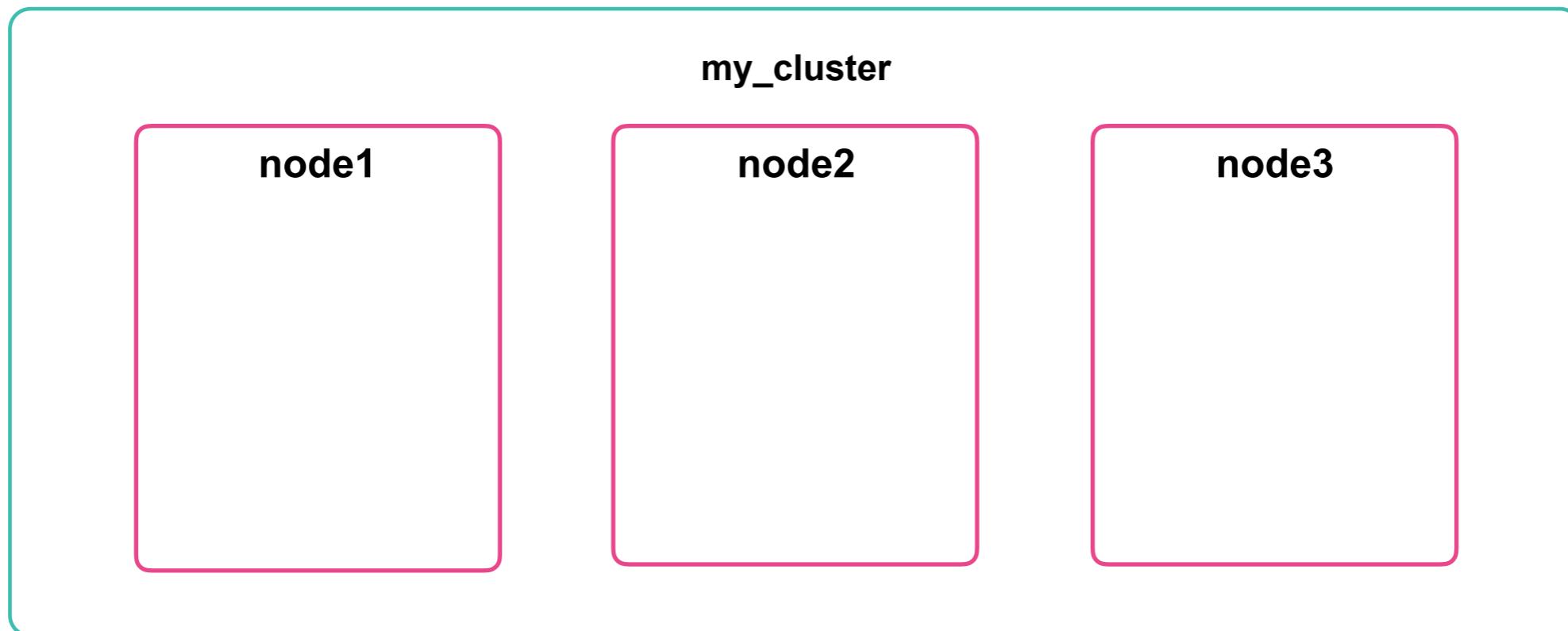
Lesson 1

# Master Nodes



# Creating an Elasticsearch Cluster

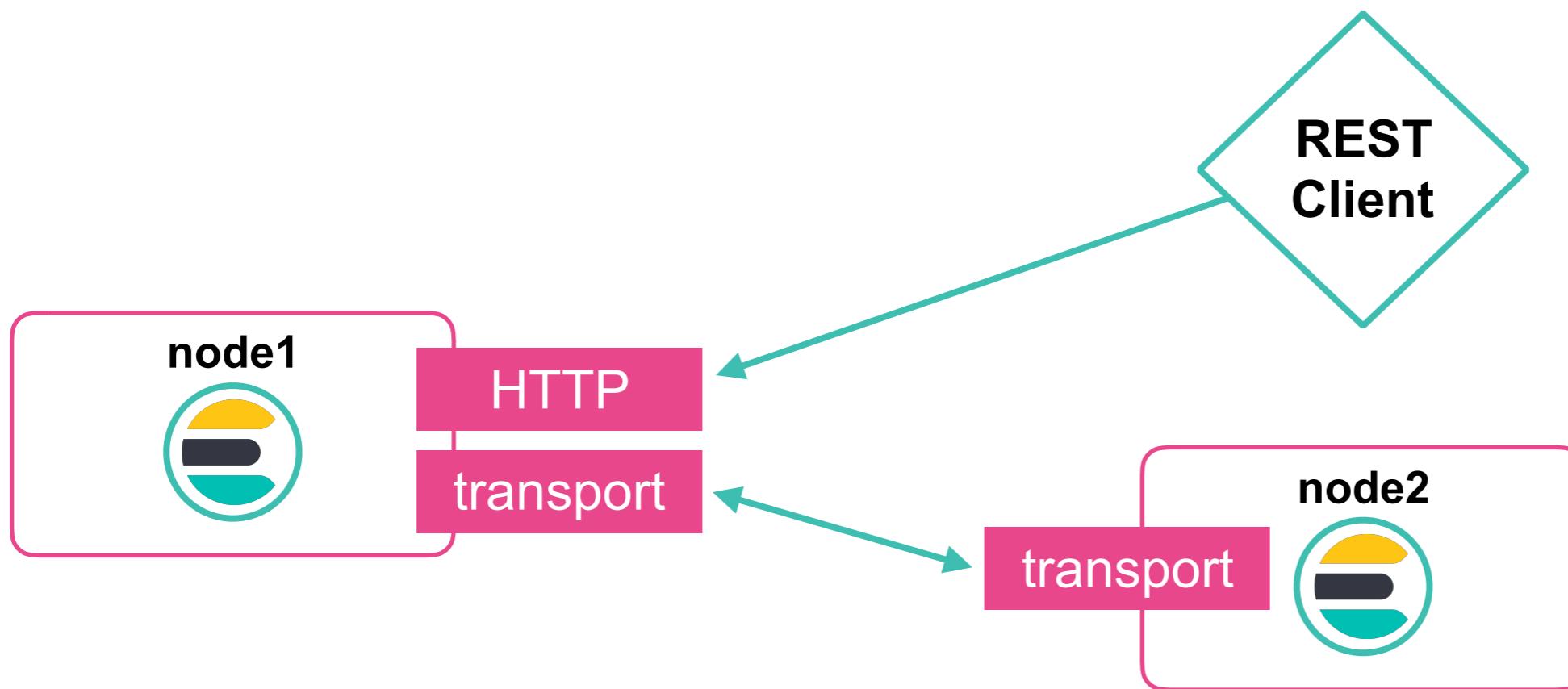
- How can we add more nodes to our cluster?
- How nodes communicate among them?



# Elasticsearch Communication

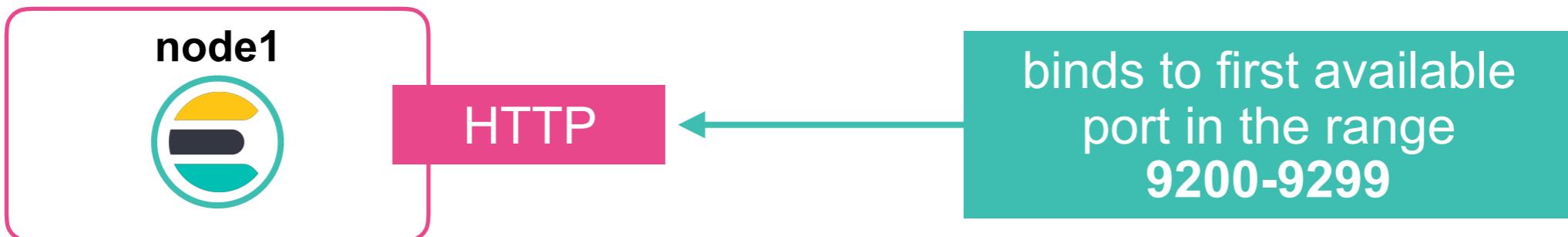
# HTTP vs. Transport

- There are two important network communication mechanisms in Elasticsearch to understand:
  - **HTTP**: which is how the Elasticsearch REST APIs are exposed
  - **transport**: used for internal communication between nodes within the cluster



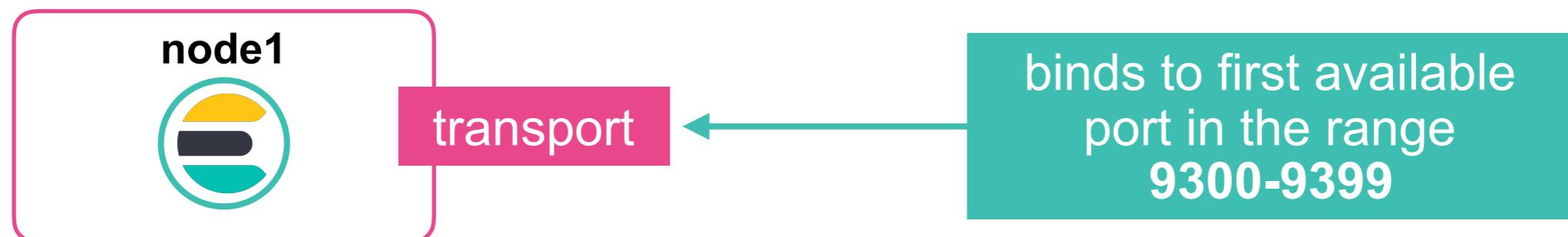
# HTTP Communication

- The REST APIs of Elasticsearch are exposed over **HTTP**
- The HTTP module binds to **localhost** by default
  - configure with **http.host**
- Default port is the first available between **9200-9299**
  - configure with **http.port**



# Transport Communication

- Each call that goes from one node to another uses the ***transport*** module
- Transport binds to **localhost** by default
  - configure with **transport.host**
- Default port is the first available between **9300-9399**
  - configure with **transport.tcp.port**



# Configuring Network Settings

- Three ways to configure protocol portion:

<b>transport.*</b>	specify settings for the transport protocol
<b>http.*</b>	specify settings for the http protocol
<b>network.*</b>	specify settings for both protocols in one setting

- ...and three ways to configure the interface portion:

<b>*.bind_host</b>	interface to bind the specified protocol to
<b>*.publish_host</b>	interface used to advertise for other nodes to connect to
<b>*.host</b>	specify both bind and publish in one setting

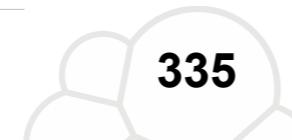
# Special Values for Network Settings

- The following special values may be used as the values of the various network settings:
  - and help to avoid hard-coding IP addresses in your config files

Value	Description
<code>_local_</code>	Any loopback addresses on the system (e.g. 127.0.0.1)
<code>_site_</code>	Any site-local addresses on the system (e.g. 192.168.0.1)
<code>_global_</code>	Any globally-scoped addresses on the system (e.g. 8.8.8.8)
<code>_networkInterface_</code>	Addresses of a network interface (e.g. <code>_en0_</code> )

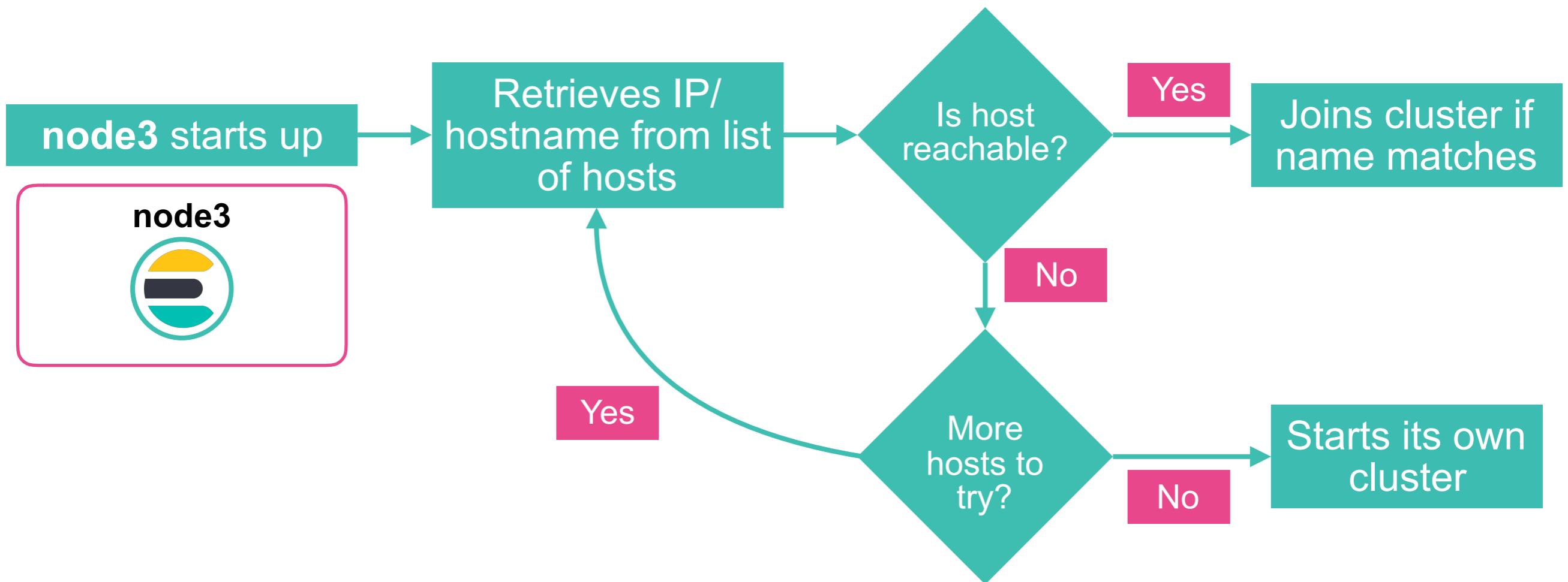
```
network.host: _site_
http.publish_host: _global_
transport.bind_host: _local_
```

A common setting



# Discovery Module

- The discovery module is responsible for discovering nodes within a cluster
- To join a cluster, given a list of **seed hosts** a node issues ping requests to find other nodes on the same network



# Seed Hosts Providers

- The list of seed addresses is generated from one or more seed hosts providers

- Static (default)

```
discovery.seed_hosts: ["server1", "server2", "server3"]
```

- Dynamic

```
discovery.seed_providers: file
```

```
server1  
server2  
server3
```

\$ES\_PATH\_CONF/unicast\_hosts.txt

- Plugins

- cloud services: EC2, GCE and Azure
- community: Kubernetes

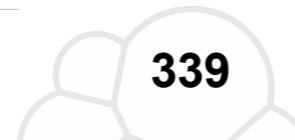
# Cluster State and Master Nodes

# Cluster State

- The details of a cluster are maintained in the ***cluster state***
  - includes the nodes in the cluster, indices, mappings, settings, shard allocation, and so on
- Use the **\_cluster** endpoint to view a cluster's state:

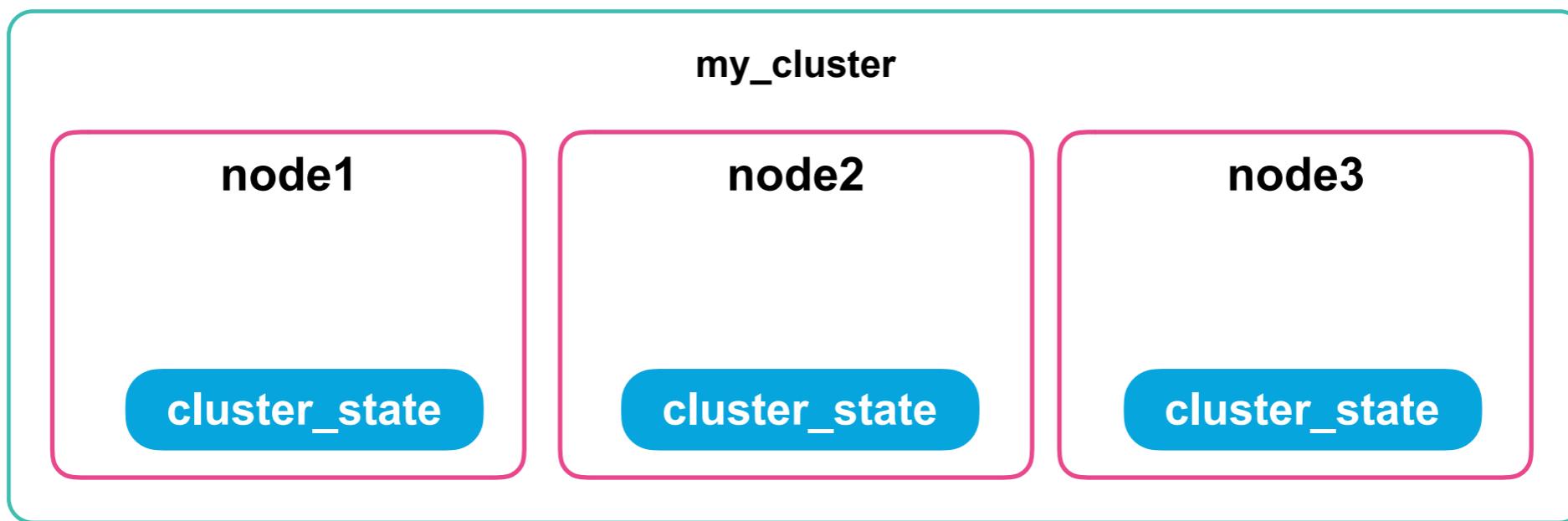
GET **\_cluster/state**

```
{  
  "cluster_name": "elasticsearch",  
  "compressed_size_in_bytes": 1920,  
  "version": 10,  
  "state_uuid": "rPiUZXbURICvkP18GxQXUA",  
  "master_node": "O4cNlHDuTyWdDhq7vhJE7g",  
  "blocks": {},  
  "nodes": {...},  
  "metadata": {...},  
  "routing_table": {...},  
  "routing_nodes": {...},  
  "snapshots": {...},  
  "restore": {...},  
  "snapshot_deletions": {...}  
}
```



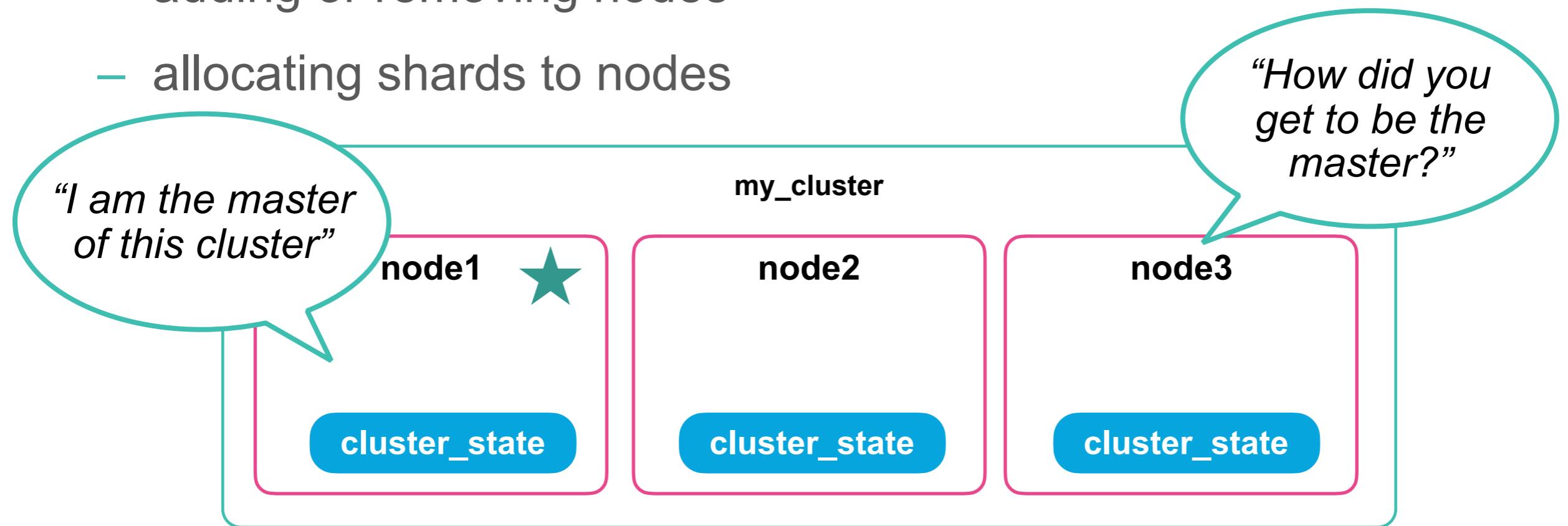
# Cluster State

- The cluster state is stored on each node
  - but it *can only be changed by the master node...*



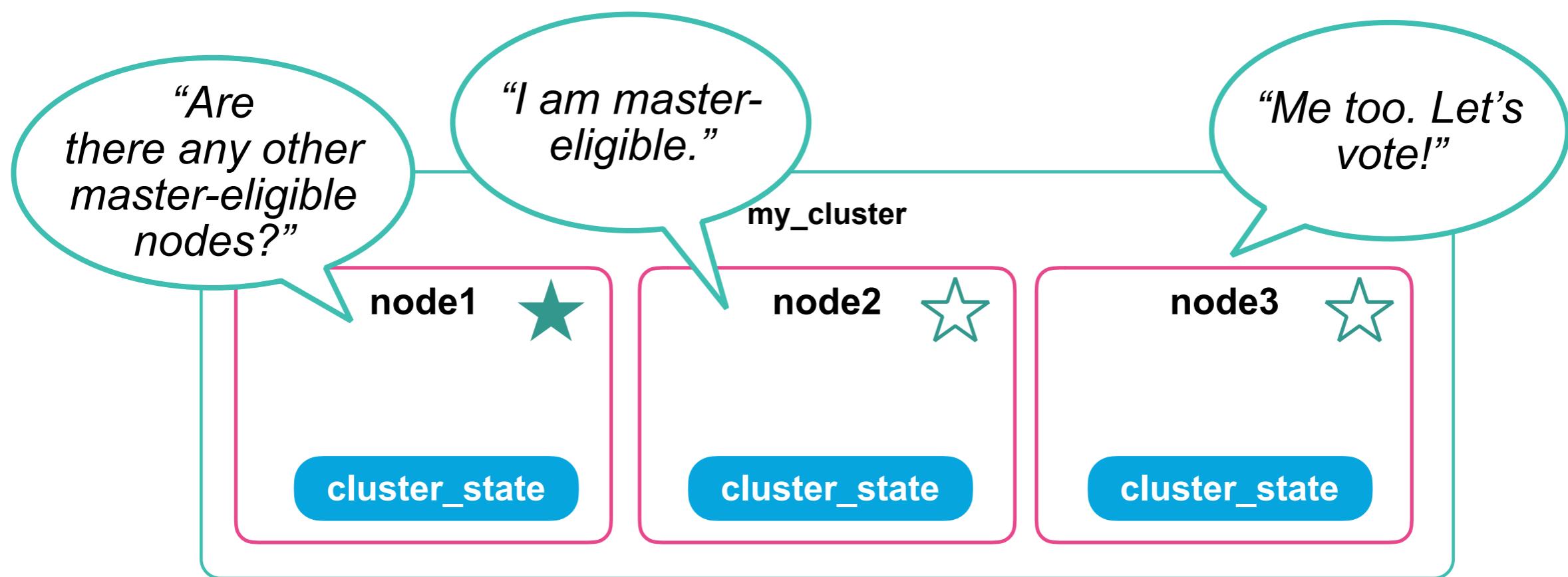
# The Master Node

- 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



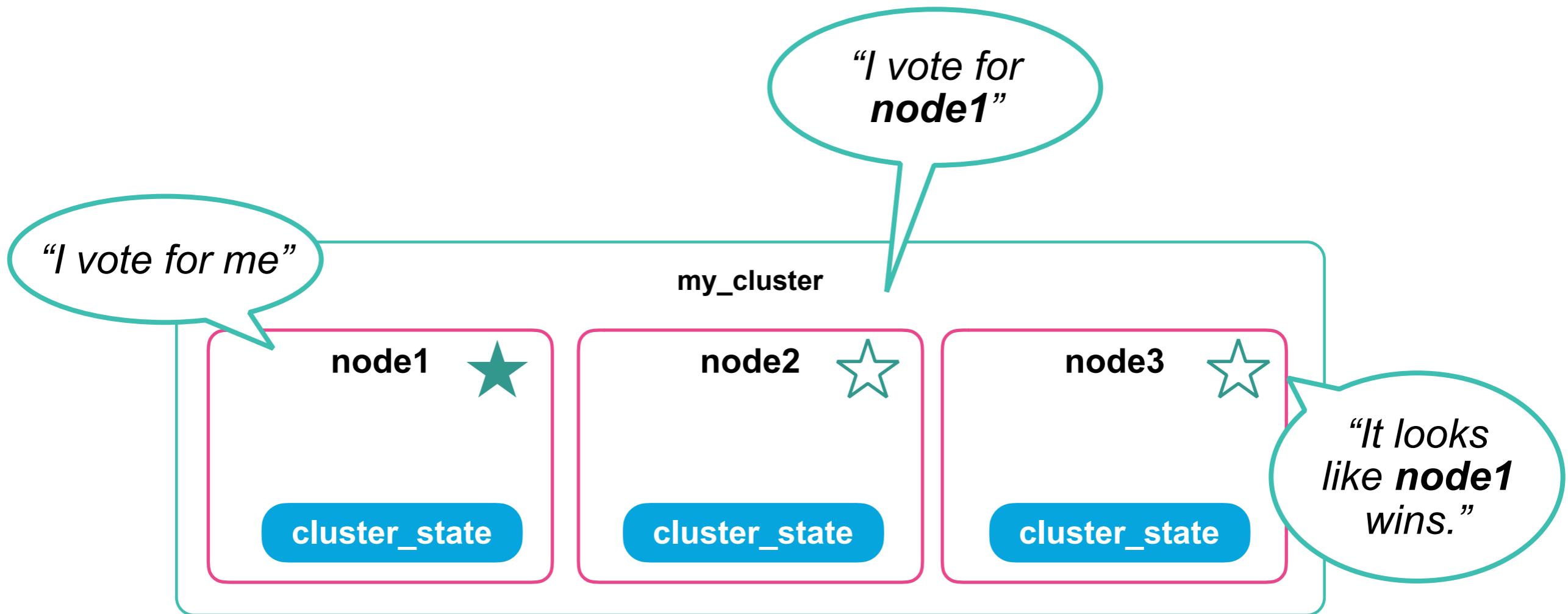
# Master-eligible Nodes

- 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 vote



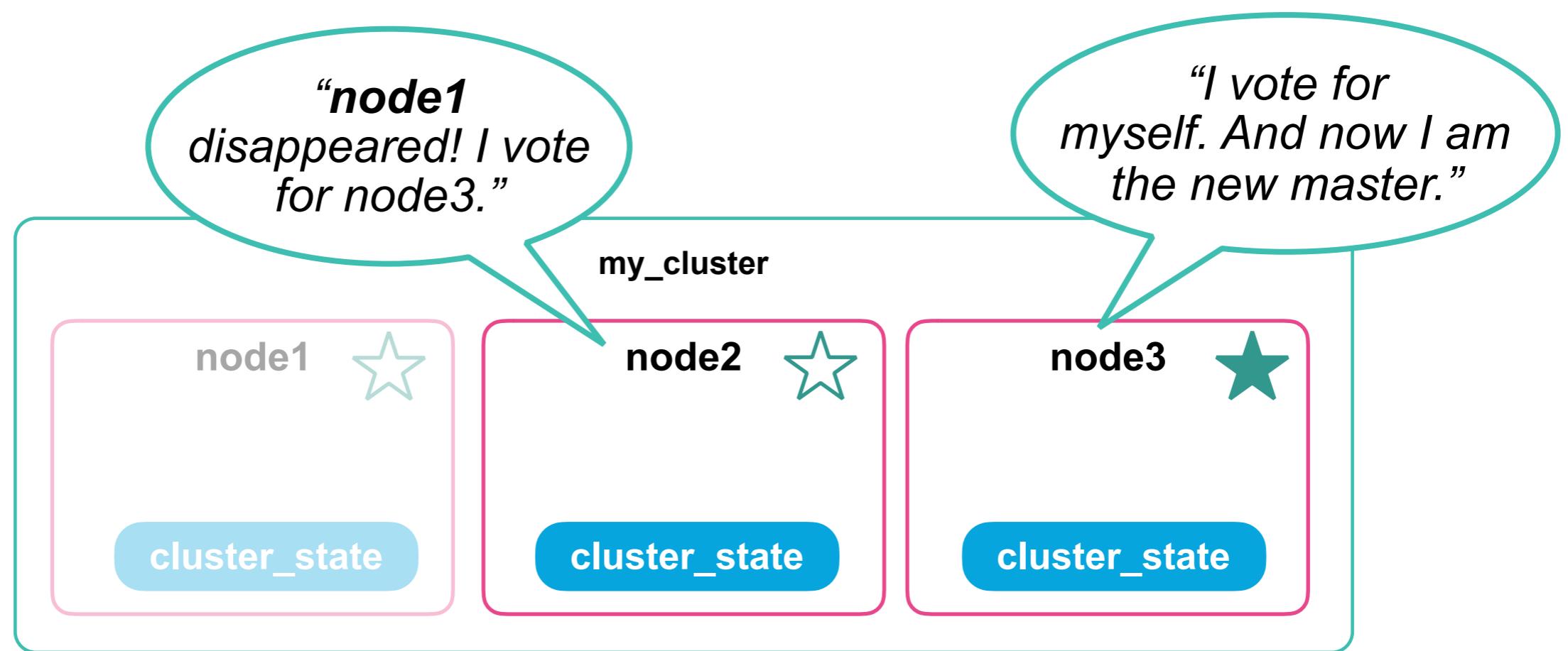
# Master Elections

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



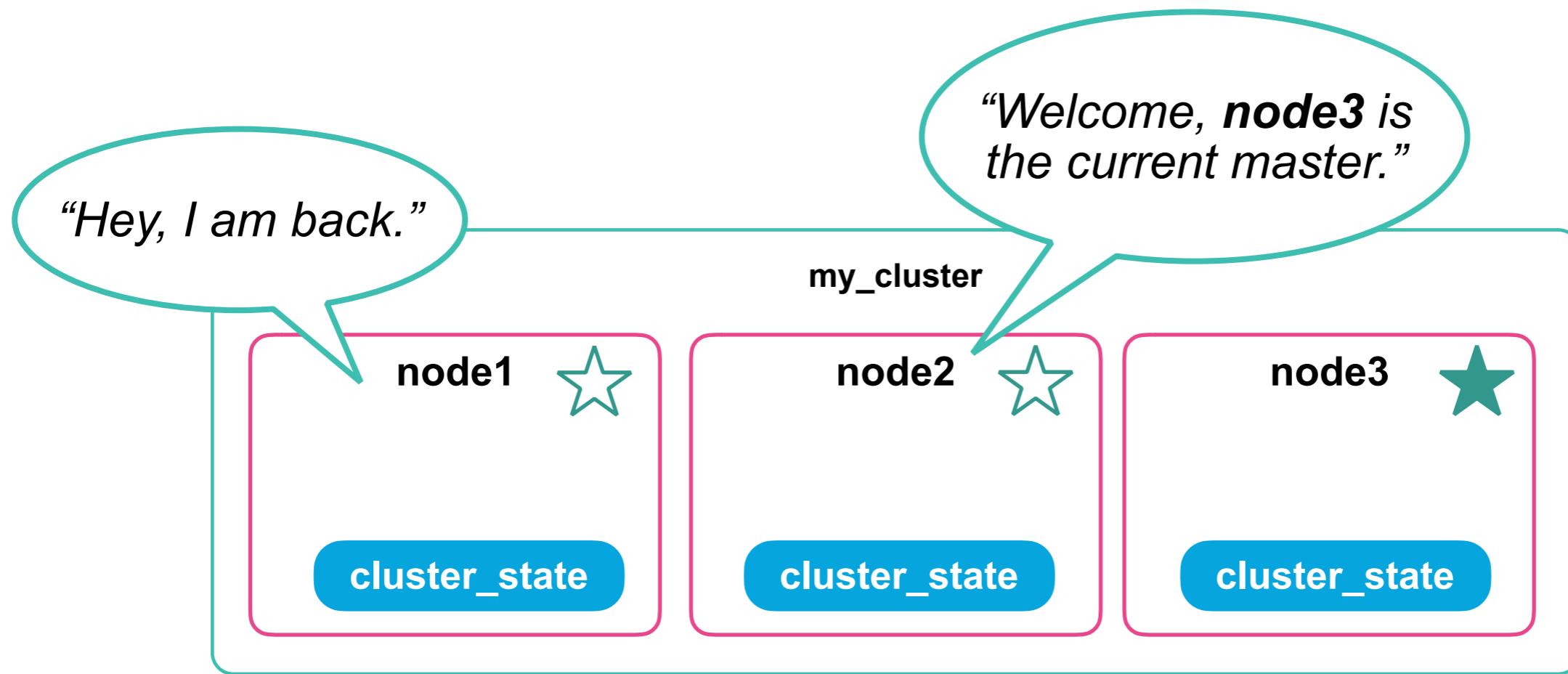
# High Availability of the Master Node

- The purpose of having 3 master-eligible nodes is so that if the master node fails, there are 2 backups
  - just enough master-eligible nodes so that a new master can be elected



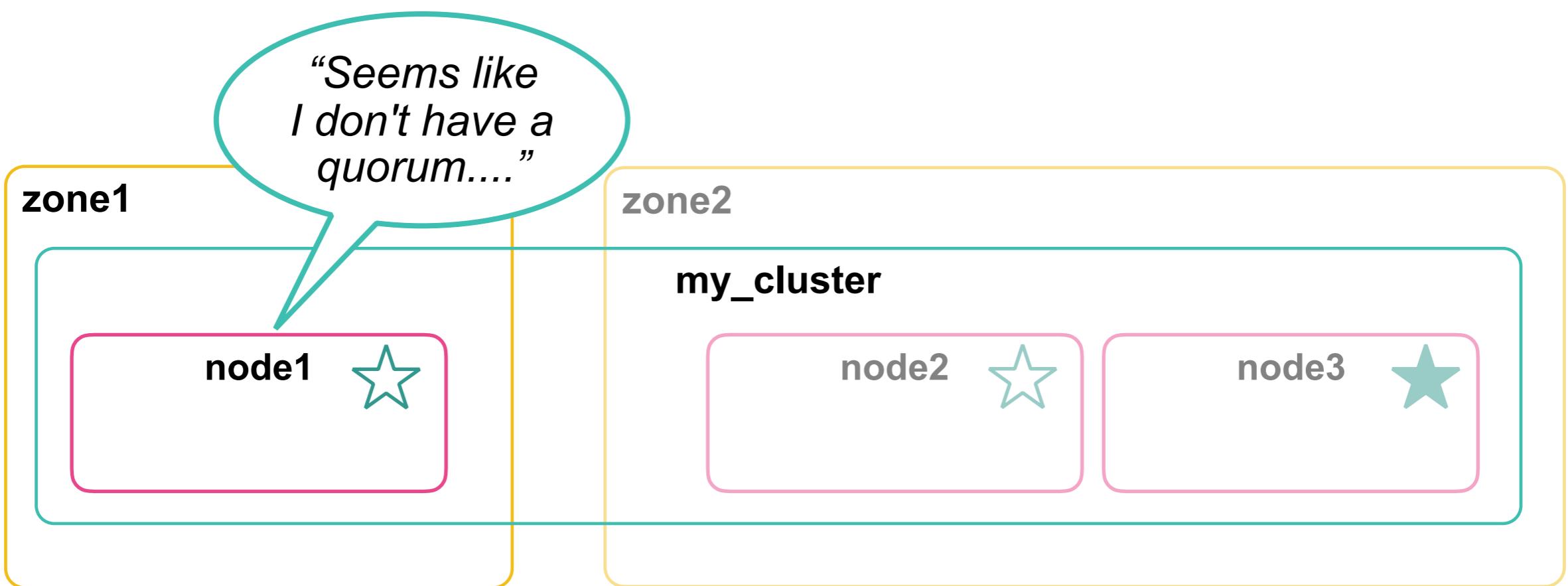
# High Availability of the Master Node

- When **node1** comes back online, one of the other master-eligible nodes will be the master
  - so it will simply re-join the cluster as a master-eligible node



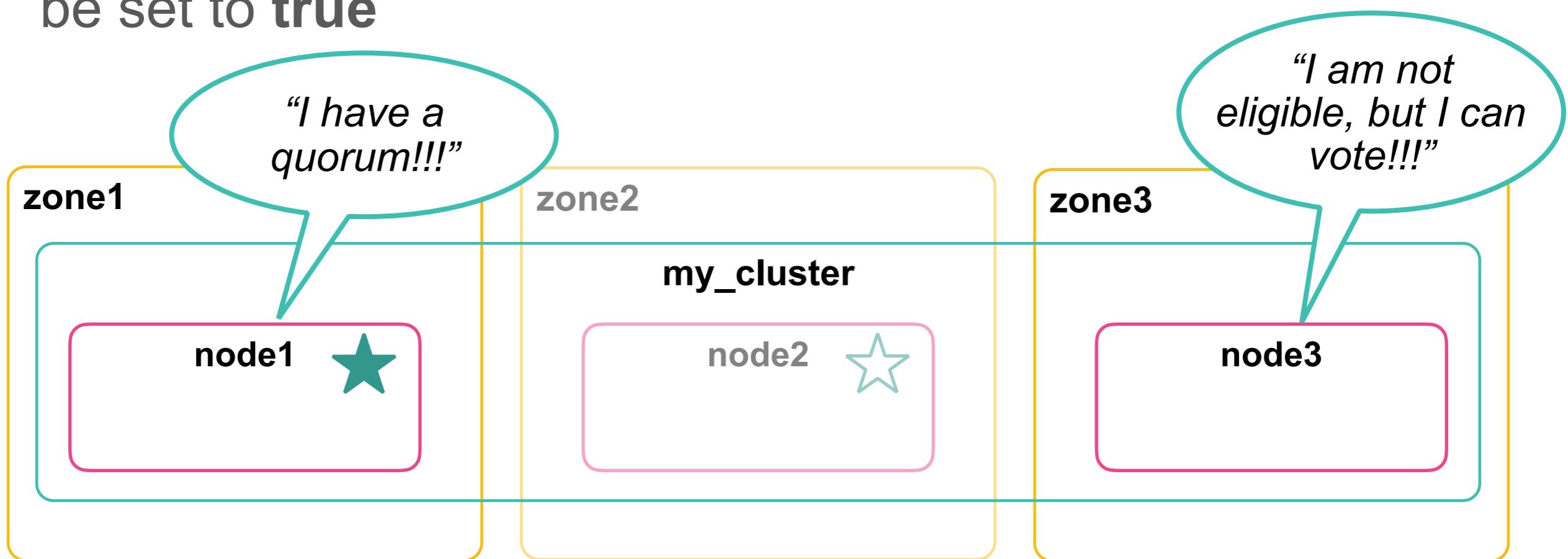
# Multiple Availability Zones

- Nodes can be distributed across multiple availability zones
- ... but that could introduce a single point of failure if the **zone2** goes down



# Voting Only Nodes

- A voting only node is a tie breaker, it cannot be elected master
- It requires less CPU and less heap than a "normal" master eligible nodes (if it doesn't have other roles)
- The parameters **node.master** and **node.voting\_only** must be set to **true**

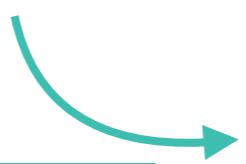


# Voting Configuration

- Elasticsearch handles the voting configuration automatically
- If there is an even number of master-eligible nodes
  - Elasticsearch leaves one of them out of the voting configuration
- It is stored in the cluster state

```
GET /_cluster/state?filter_path=metadata.cluster_coordination.last_committed_config
```

Checking the current  
vote configuration



```
{  
  "metadata" : {  
    "cluster_coordination" : {  
      "last_committed_config" : [  
        "OvD79L11Qme1hi060uiu7Q",  
        "goWbX4DAT068J3i6jGr8Aw",  
        "N7SB1AGdS9ioU5r67LJNRg"  
      ]  
    }  
  }  
}
```

node ids



# Bootstrapping a Cluster

- When you Start an Elasticsearch cluster for the **very** first time
  - you must provide the initial set of master-eligible nodes

```
cluster.initial_master_nodes: [ "node1", "node2", "node3" ]
```
- This should be set on each initial master-eligible node
- After the cluster has formed
  - this setting is no longer required and is ignored
- If you stop half or more voting nodes at the same time
  - your cluster will become unavailable



Elasticsearch Nodes and Shards

Lesson 1

# Review - Master Nodes



# Summary

- Elasticsearch uses two network communication mechanisms: *HTTP* for REST clients; and *transport* for inter-node communication
- The details of a cluster are maintained in the *cluster state*
- Every cluster has *one node* designated as the *master*
- You need to bootstrap the Elasticsearch cluster when you start it for the very first time

# Quiz

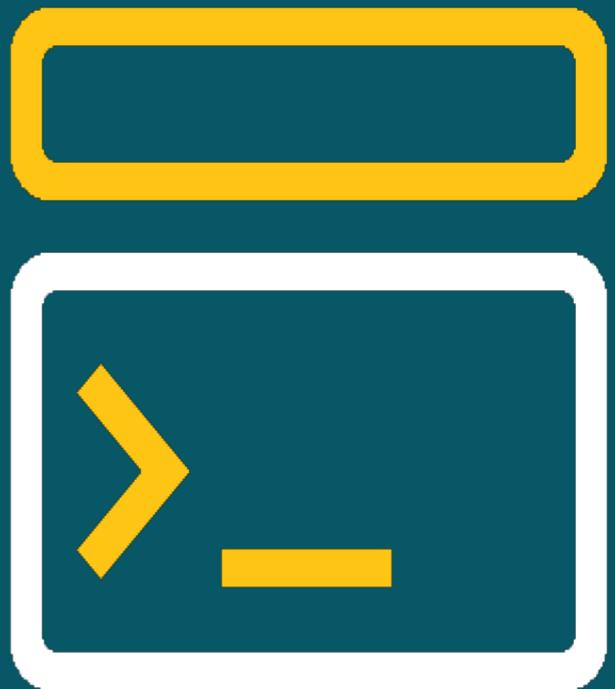
1. How does a new node joining a cluster find the cluster?
2. What do you need to do for bootstrapping a cluster?
3. **True or False:** Every Elasticsearch cluster should run an even number of nodes.



Elasticsearch Nodes and Shards

Lesson 1

# Lab - Master Nodes





Elasticsearch Nodes and Shards

## Lesson 2

# Node Roles



# Node Roles

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

# Configuring Node Roles

- By default, a node is a master-eligible, data, ingest, and machine learning node:
  - defined on the command line or config file (`elasticsearch.yml`)

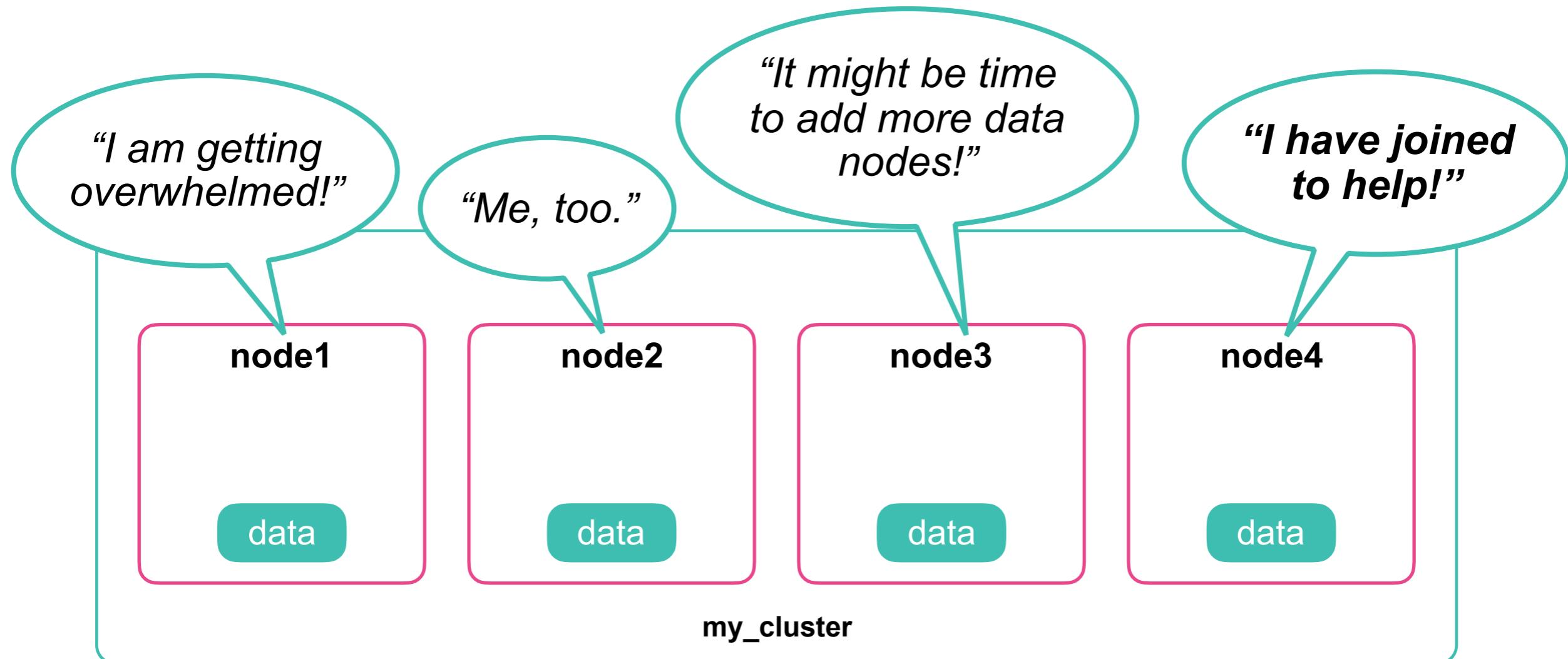
Node type	Configuration parameter	Default value
master eligible	<b>node.master</b>	true
data	<b>node.data</b>	true
ingest	<b>node.ingest</b>	true
machine learning	<b>node.ml</b>	true

# 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
  - configured using the **node.data** property

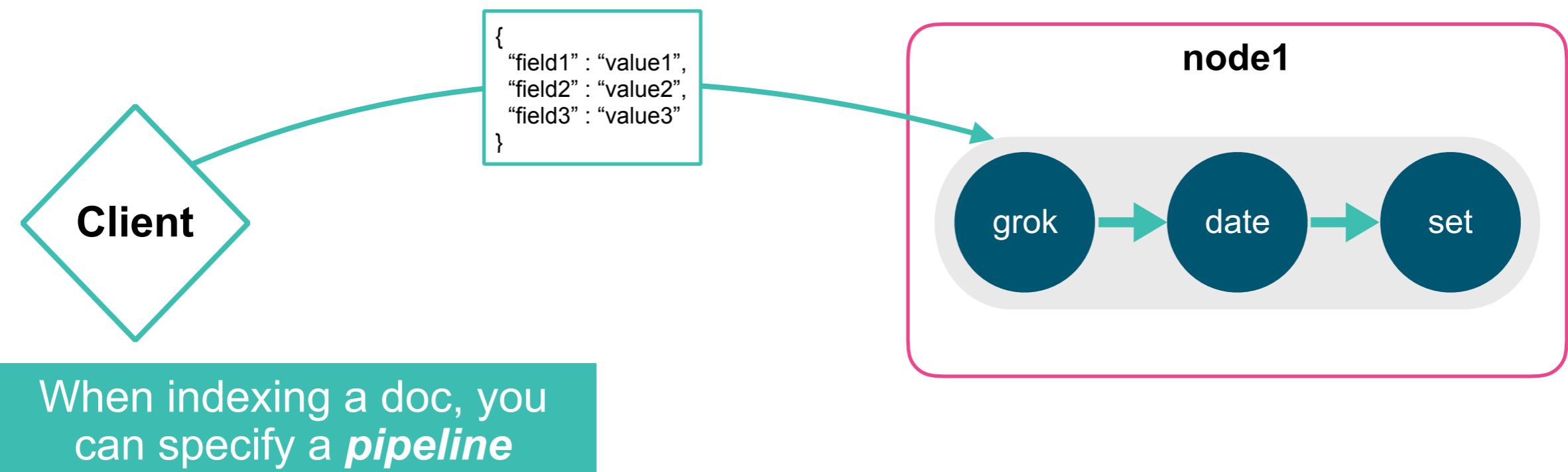
# Data Nodes Scale

- 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



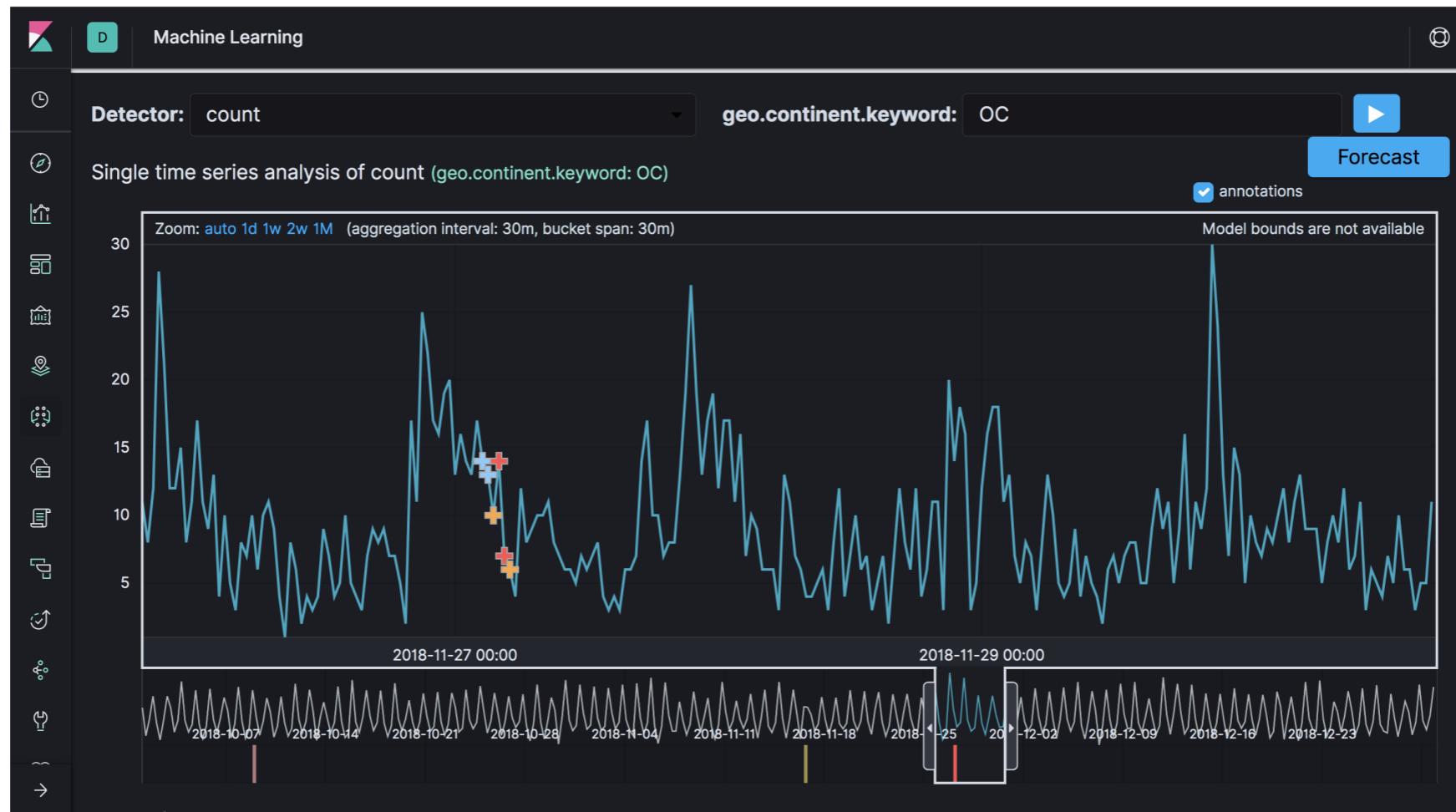
# Ingest Node

- ***Ingest nodes*** provide the ability to
  - pre-process a document right before it gets indexed
- All nodes are ingest nodes by default
  - configured using the **node.ingest** property
  - ingest nodes are covered in detail in the ***Engineer II*** course



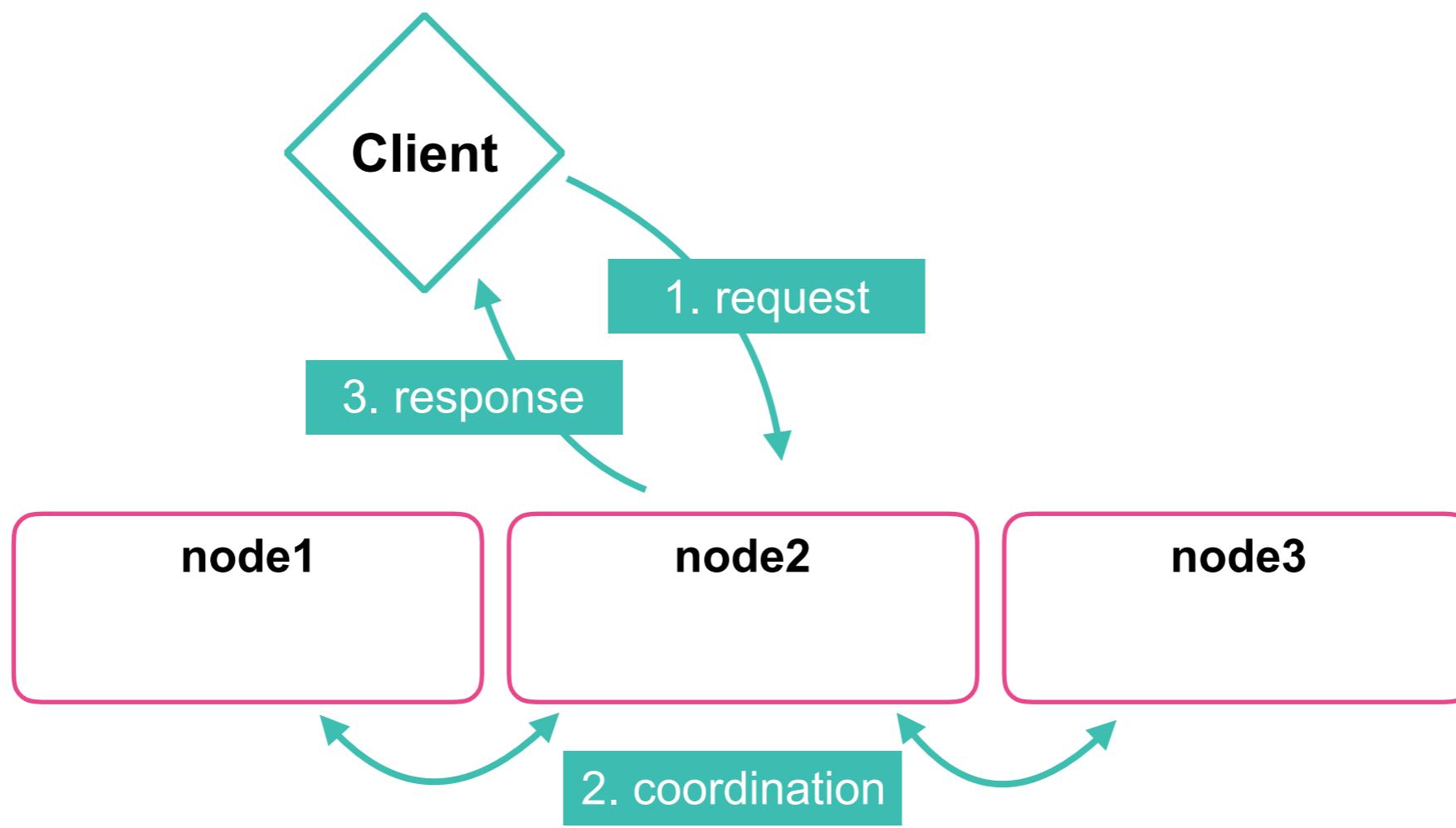
# Machine Learning Nodes

- *Machine Learning nodes* provide the ability to
  - run machine learning jobs
  - handle machine learning API requests
- Configured using the **node.ml** property



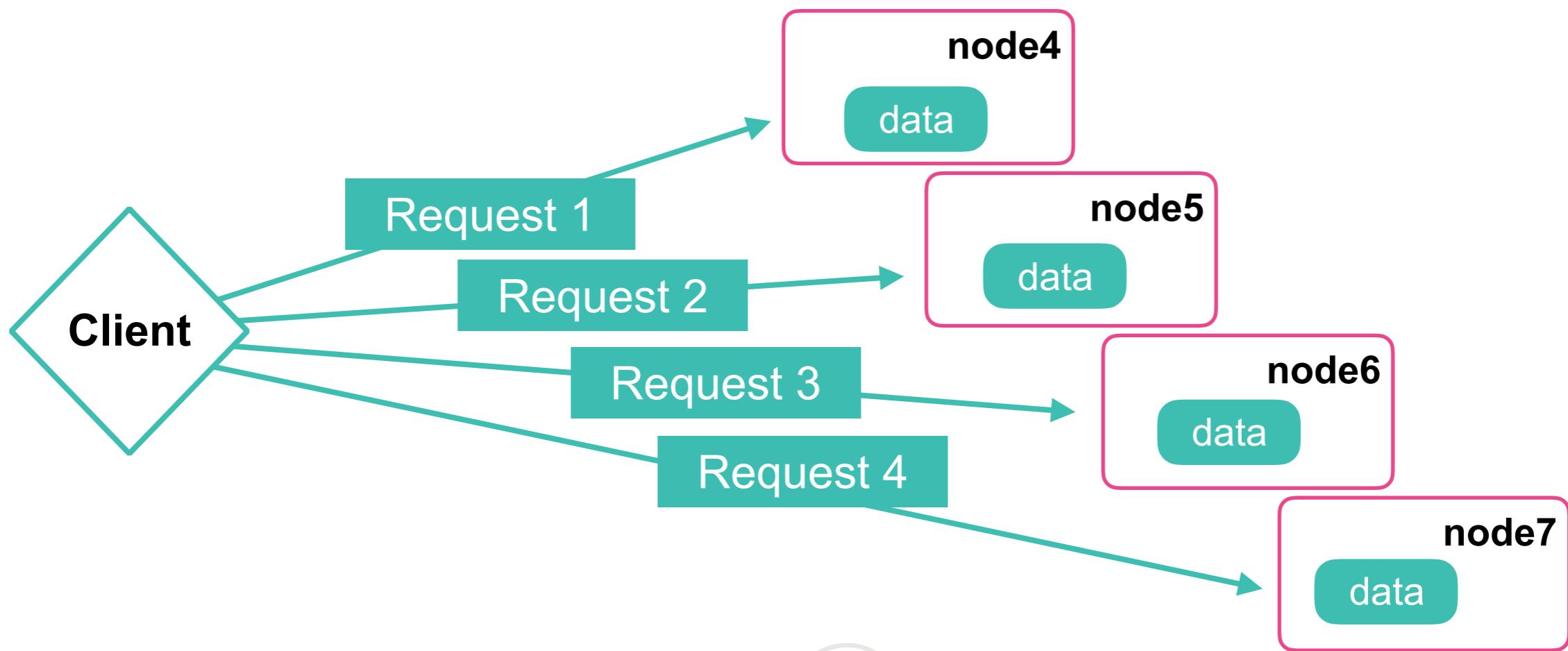
# Coordinating Node

- A *coordinating node* is the node that receives and handles a specific client request
  - every node is implicitly a coordinating node
  - forwards the request to other relevant nodes, then combines those results into a single result



# Coordinating Node

- Configure your client to round-robin between multiple nodes
  - avoid having single points of failure/creating hotspots
- Data nodes are great coordinating nodes
  - do **NOT** send requests to dedicated master nodes



# Sniffing

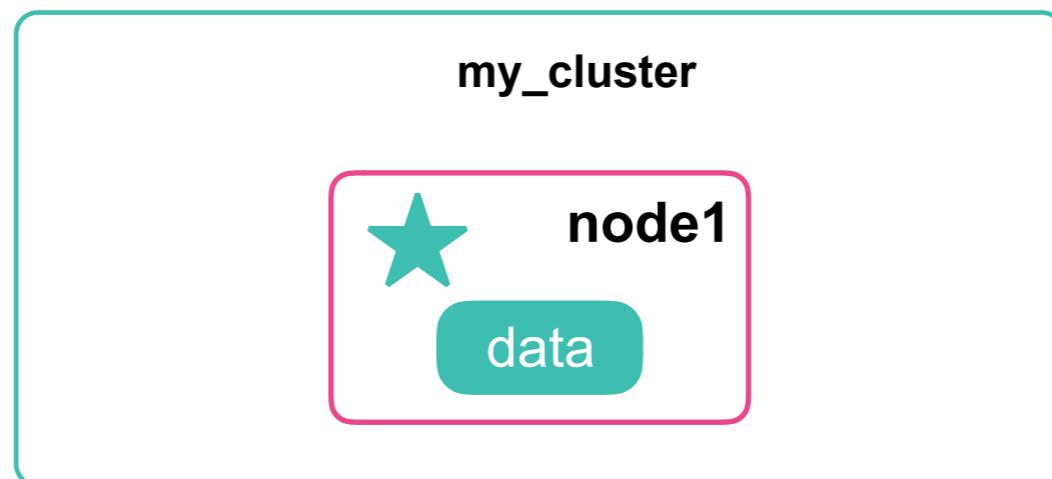
- Many language clients and tools can be configured with an array of hosts. For example, Kibana
  - client applications will round-robin between nodes
  - retry requests if a node is down
- Alternatively, you can use ***sniffing***
  - connect to one of the configured seed nodes to retrieve the cluster state
  - determine a list of nodes suitable for coordination
  - use those nodes for the list of hosts
- Enables you to rearrange your cluster without having to reconfigure the client



# Sample Architectures

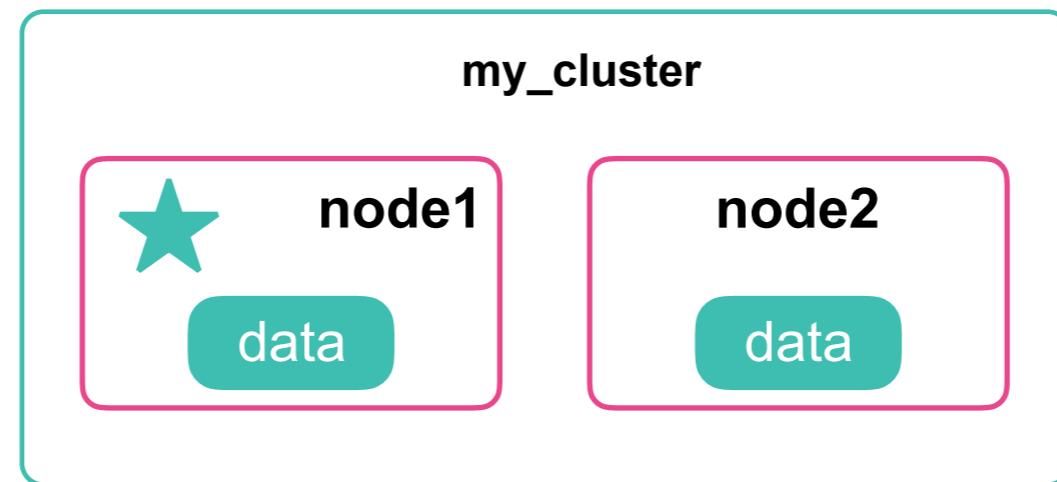
# 1-Node Clusters

- A single node is still a cluster
  - just without any high-availability
  - useful for development and testing



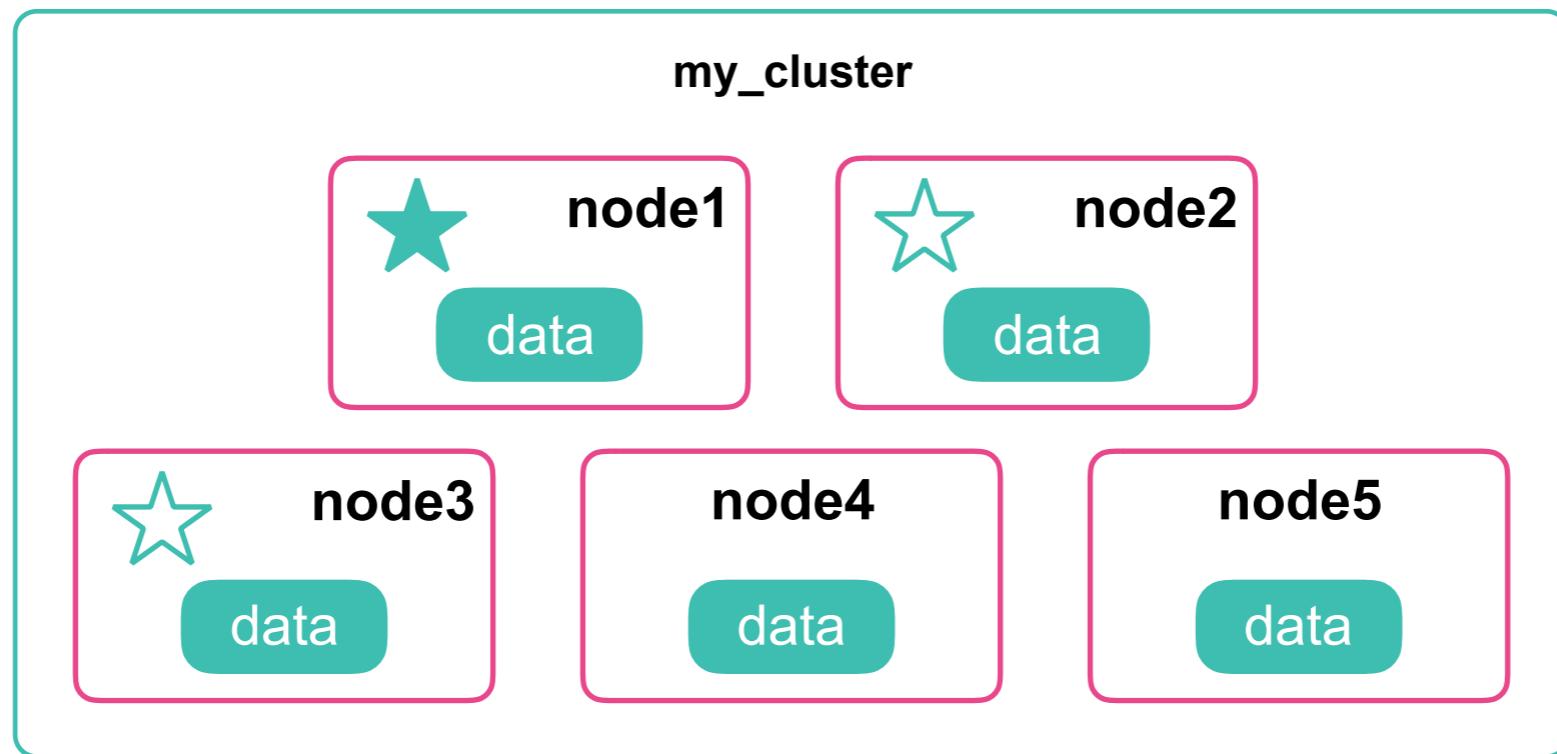
# 2-Node Clusters

- 2-node clusters have high risk of being unavailable
  - there is no quorum even if you set 2 master-eligible nodes
  - if the master goes down your cluster is unavailable



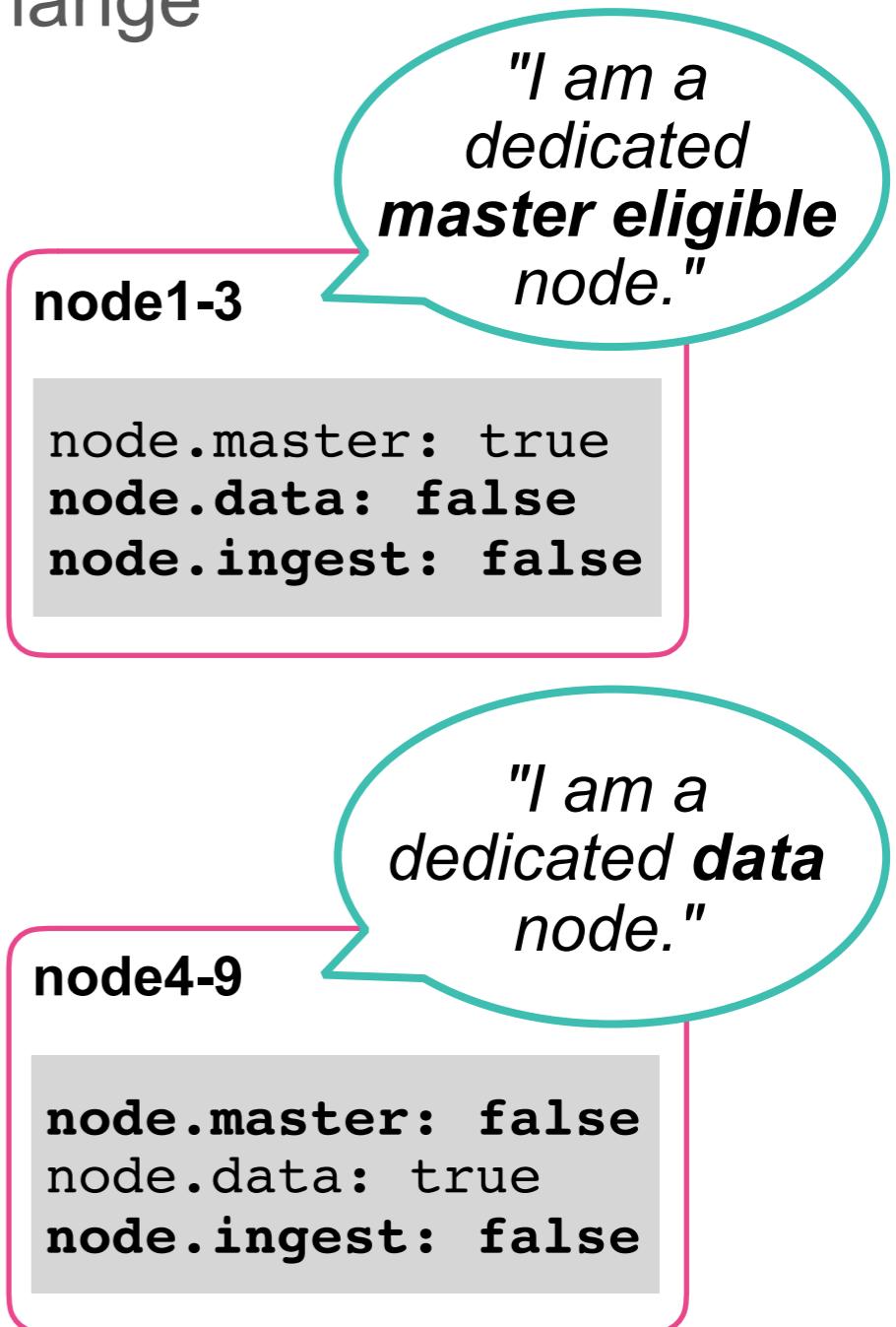
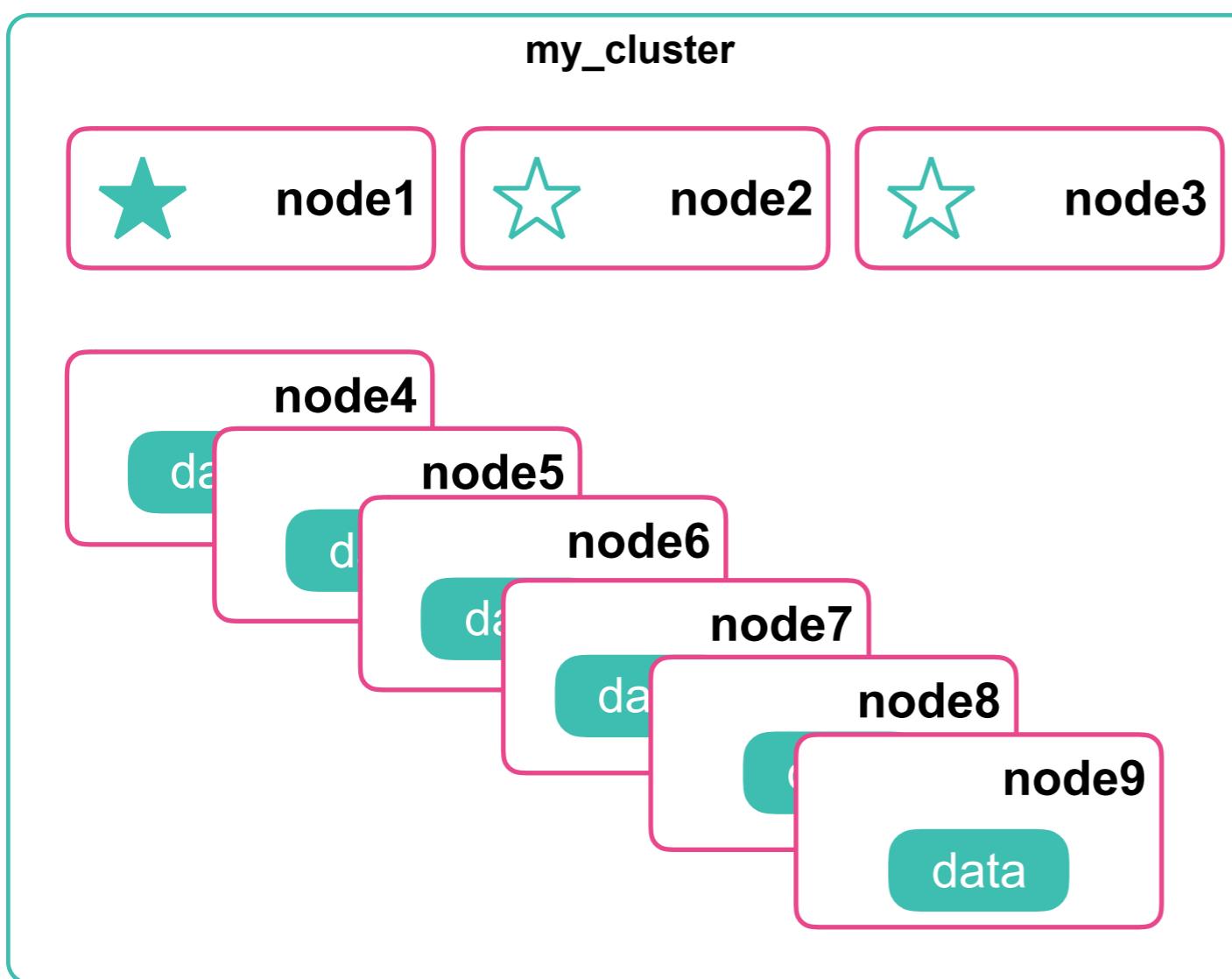
# 3-5 Node Clusters

- Suppose you have a small production cluster
  - where it is difficult to have dedicated types
- It is always preferred to have 3 master-eligible nodes



# Larger Clusters

- Larger, high-volume clusters should have dedicated nodes
  - add data nodes as your requirements change
  - covered in details in *Engineer II*





Elasticsearch Nodes and Shards

Lesson 2

# Review - Node Roles



# Summary

- There are different node roles and nodes can have one or multiple roles at the same time
- ***Data nodes*** hold shards and execute data-related operations like CRUD, search, and aggregations
- ***Ingest nodes*** can pre-process documents before indexing to Elasticsearch
- ***Machine Learning nodes*** can run machine learning jobs
- Larger, high-volume clusters should have dedicated nodes to improve performance and resource usage

# Quiz

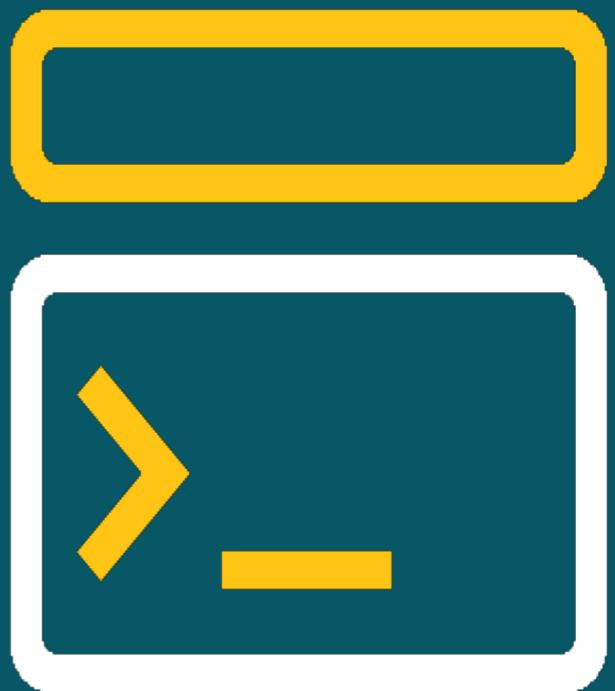
1. How would you configure a node to be a ***dedicated data*** node?
2. **True or False:** A 2-node cluster is a good option for a small production cluster.
3. **True or False:** Larger, high-volume clusters should have dedicated nodes?



Elasticsearch Nodes and Shards

Lesson 2

# Lab - Nodes Roles





Elasticsearch Nodes and Shards

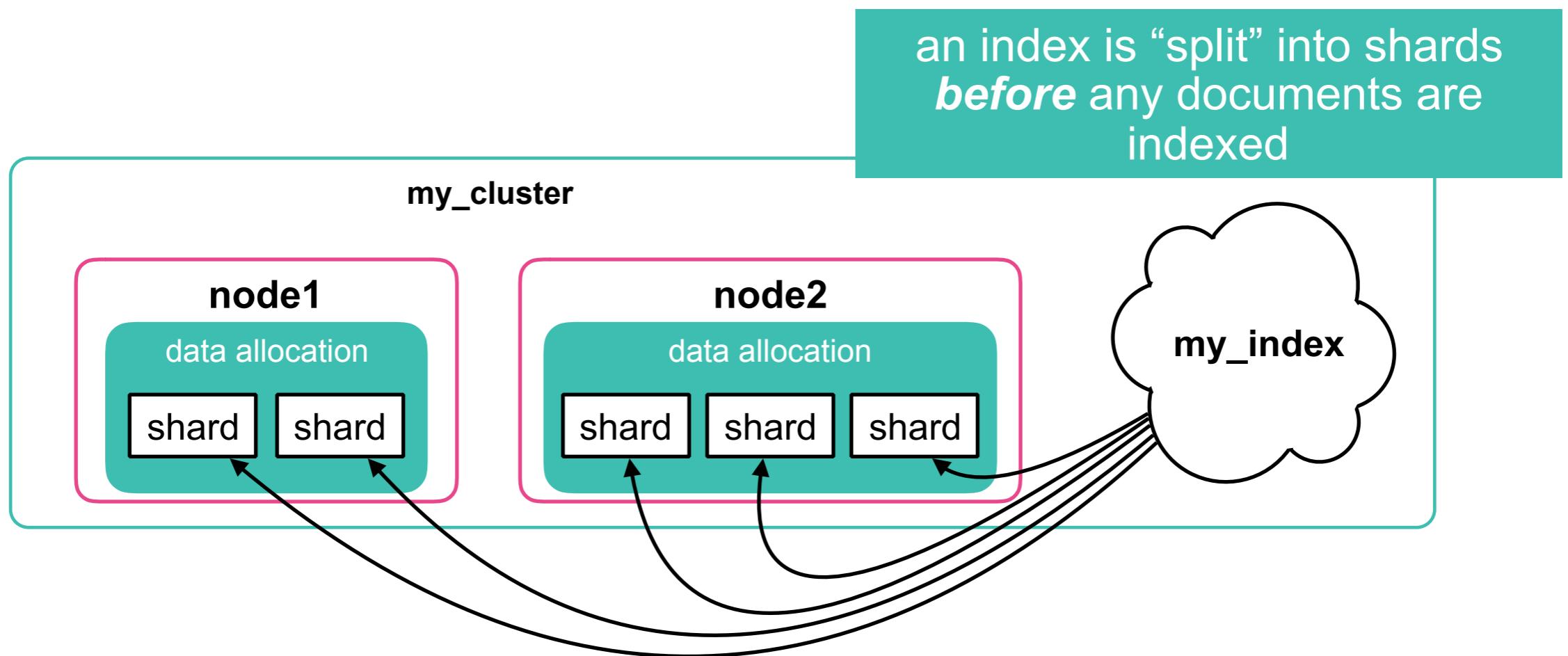
Lesson 3

# Understanding Shards



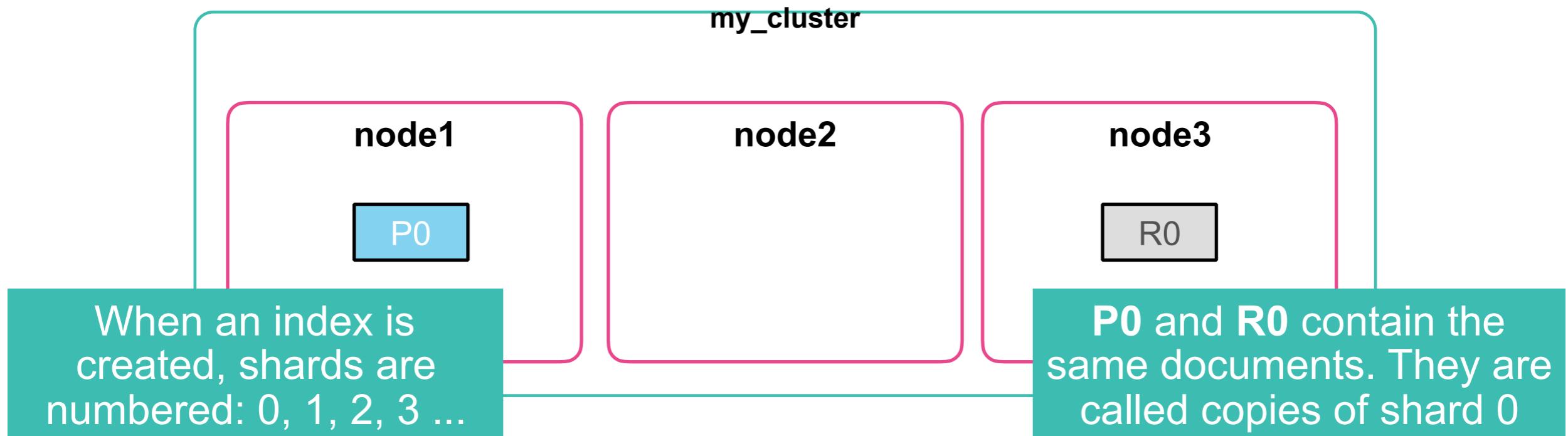
# Remember Shards?

- 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



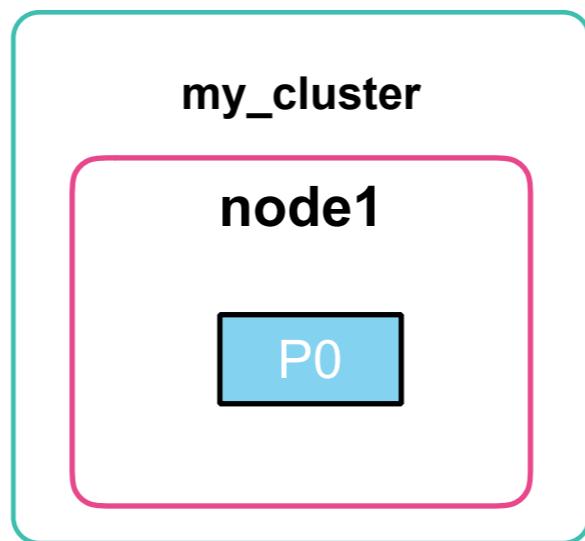
# Primary vs. Replica

- 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



# The Shards of the Blogs Dataset

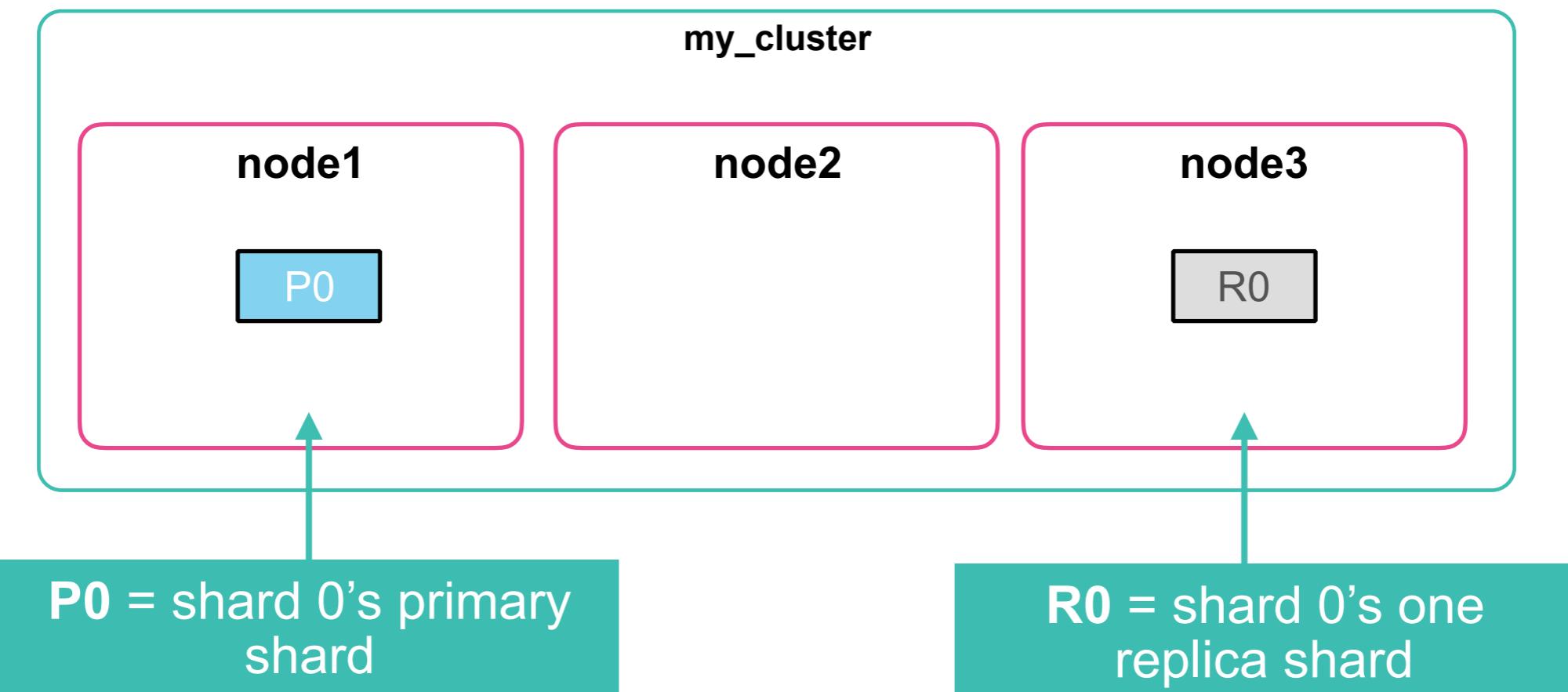
- Our blog index had the **default number of one** primary shards with one replica
  - but our cluster only had one node
- Where were the replicas?



A primary and its replicas can not be on the same node, so we are missing some replicas!

# Scaling Elasticsearch

- Adding nodes to our cluster will cause a *redistribution of shards*
  - and the creation of replicas (if enough nodes exist)
  - the *master node* determines the distribution of shards

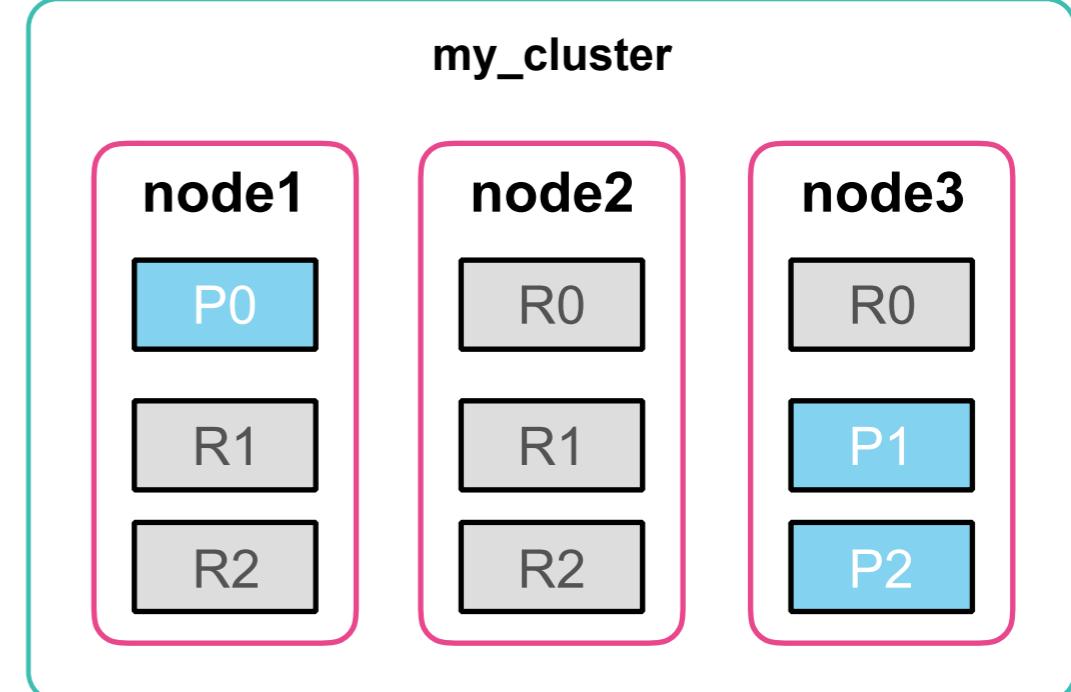


# Configuring the Number of Shards

- The default number of primary shards for an index is 1
  - you specify the number of primary shards when you create the index and you cannot change it afterwards
  - the number of replicas can be changed at any time

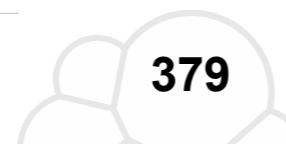
```
PUT my_new_index
{
  "settings": {
    "number_of_shards": 3,
    "number_of_replicas": 2
  }
}
```

3 primaries + 2 replicas = 9 total shards on the cluster



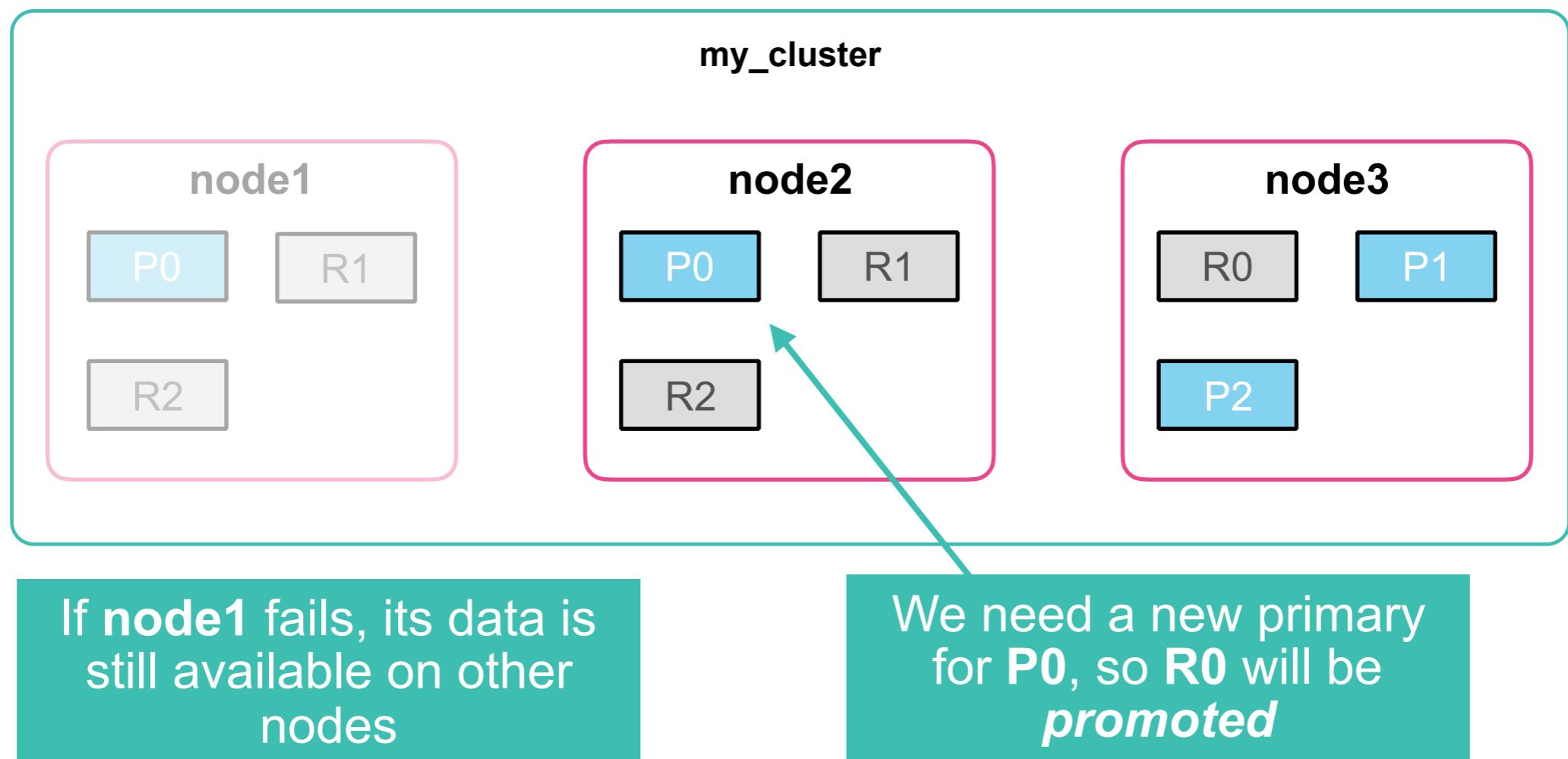
# Why 1 Instead of 5 Primaries?

- Previous versions of Elasticsearch (<7.0) had 5 primaries
- ***Oversharding*** is one of the main problems users have!
  - too many small shards consume resources for no reason
- A shard typically can hold at least 10s of gigabytes
- If more shards are needed
  - multiple indices make it easy to scale
  - else, the split API allows you to increase the number of shards
- More details in the ***Engineer II*** course



# Why Create Replicas?

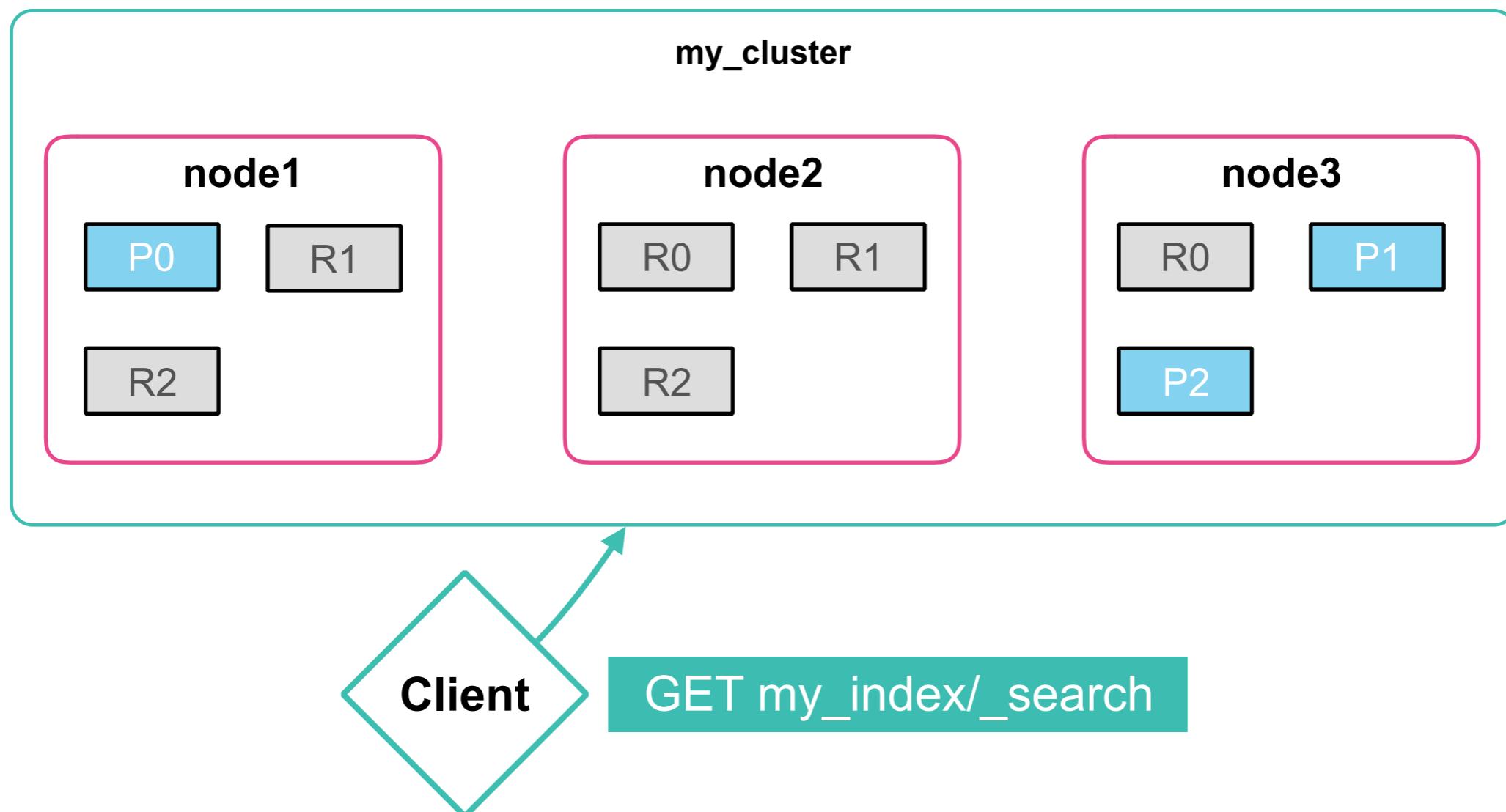
- ***High availability***
  - we can lose a node and still have all the data available
  - replicas are promoted to primaries as needed



# Why Create Replicas?

- ***Read throughput***

- a query can be performed on a primary *or* replica shard
- allows you to scale your data and better utilize cluster resources



# Cluster Health and Shard Allocation

# Cluster Health

- A cluster has a *health* that contains various details and metrics of the cluster

GET \_cluster/health



```
{  
  "cluster_name": "elasticsearch",  
  "status": "yellow",  
  "timed_out": false,  
  "number_of_nodes": 1,  
  "number_of_data_nodes": 1,  
  "active_primary_shards": 15,  
  "active_shards": 15,  
  "relocating_shards": 0,  
  "initializing_shards": 0,  
  "unassigned_shards": 15,  
  "delayed_unassigned_shards": 0,  
  "number_of_pending_tasks": 0,  
  "number_of_in_flight_fetch": 0,  
  "task_max_waiting_in_queue_millis": 0,  
  "active_shards_percent_as_number": 50  
}
```



# Health Status

- The ***health status*** is either green, yellow or red and exists at three levels: shard, index, and cluster
- ***cluster health***
  - status of the ***worst index*** in the cluster
- ***index health***
  - status of the ***worst shard*** in that index
- ***shard health***
  - ***red***: at least one primary shard is not allocated in the cluster
  - ***yellow***: all primaries are allocated but at least one replica is not
  - ***green***: all shards are allocated

# Shard Allocation

- Shards go through several states:
  - UNASSIGNED
  - INITIALIZING
  - STARTED
  - RELOCATING
- Elasticsearch manages those states **automatically**
  - without the need for manual intervention
- Let's follow the allocation of shards...

# UNASSIGNED Shards

- **UNASSIGNED** describes shards that exist in the cluster state, but cannot be found in the cluster itself

```
PUT my_index
{
  "settings": {
    "number_of_shards": 2,
    "number_of_replicas": 1
  }
}
```

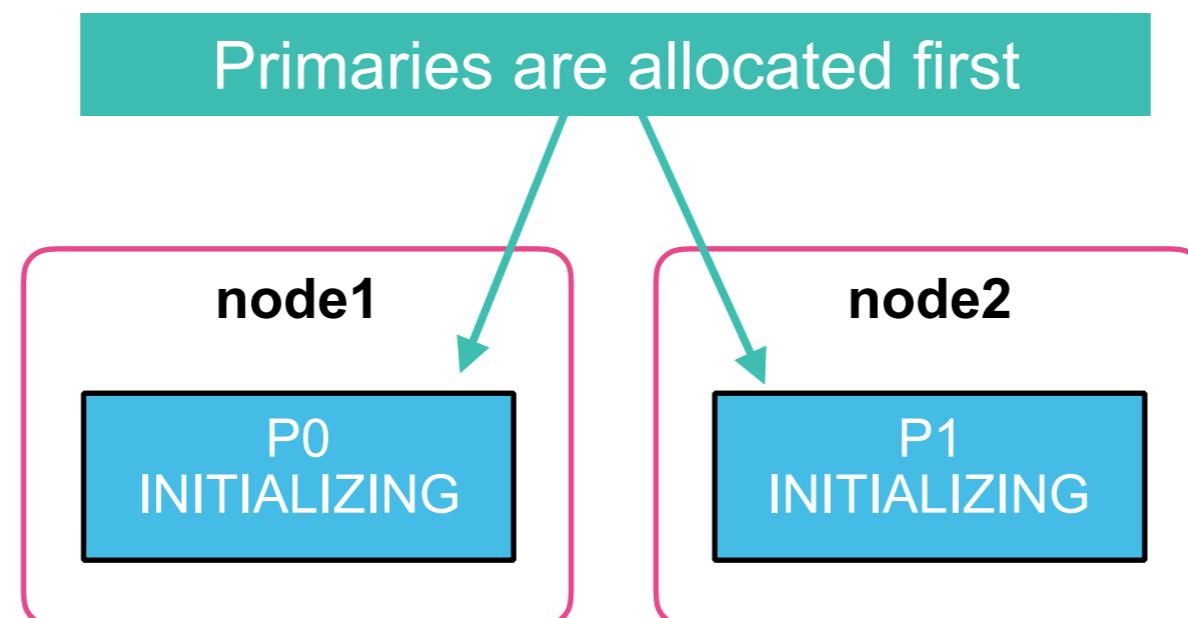
Four shards total, all in  
the UNASSIGNED state

node1

node2

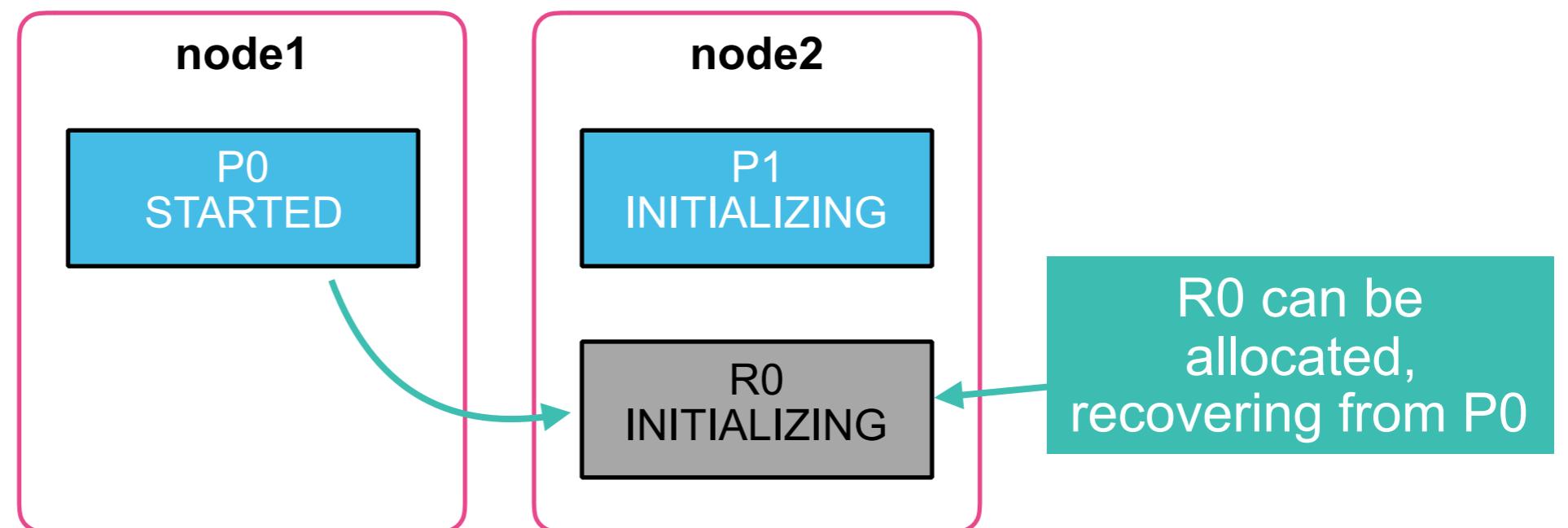
# INITIALIZING

- Shards are briefly in the **INITIALIZING** state as they are being created:



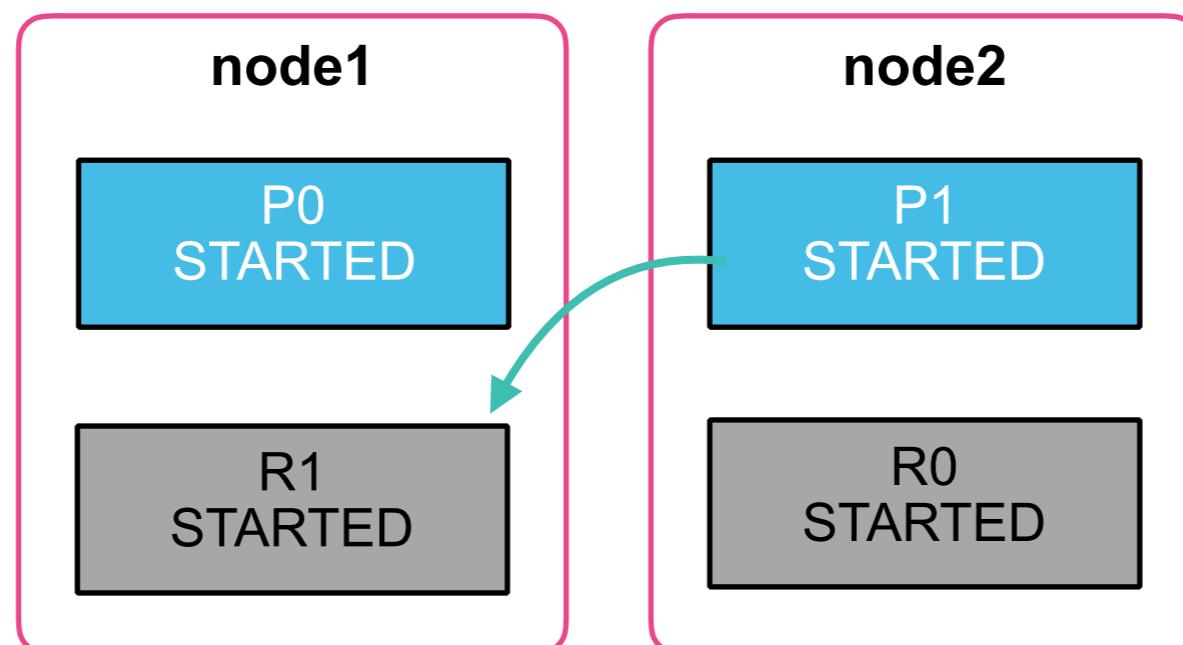
# STARTED

- When a shard is done initializing, it moves to the **STARTED** state
  - and its replicas can now be allocated



# STARTED

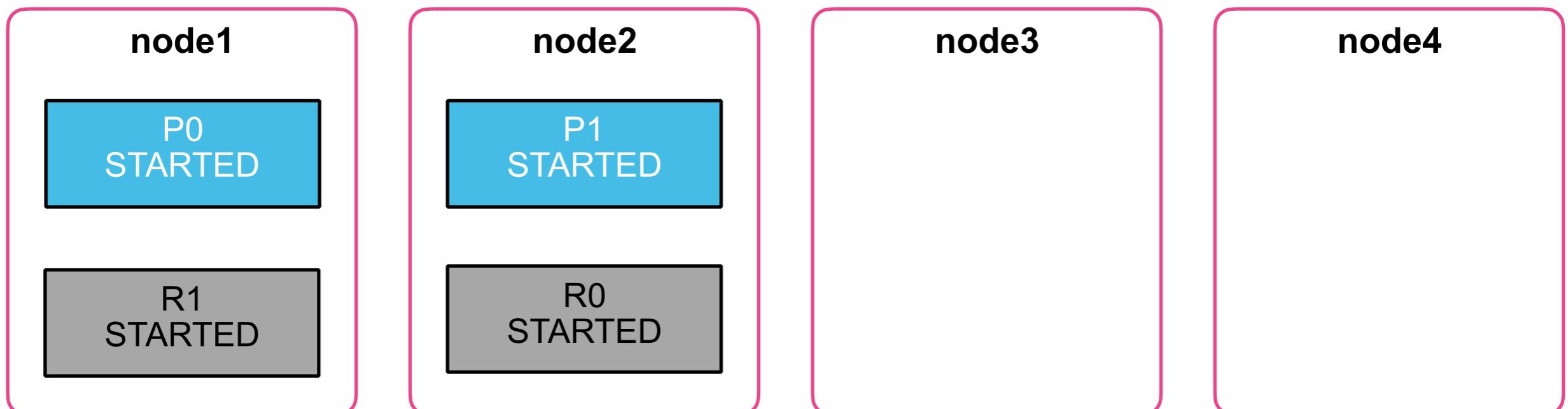
- Eventually, each primary and each replica will move from **INITIALIZING** to **STARTED**
  - the new index is fully allocated and distributed on the cluster



What is the  
cluster health?

# Adding Nodes

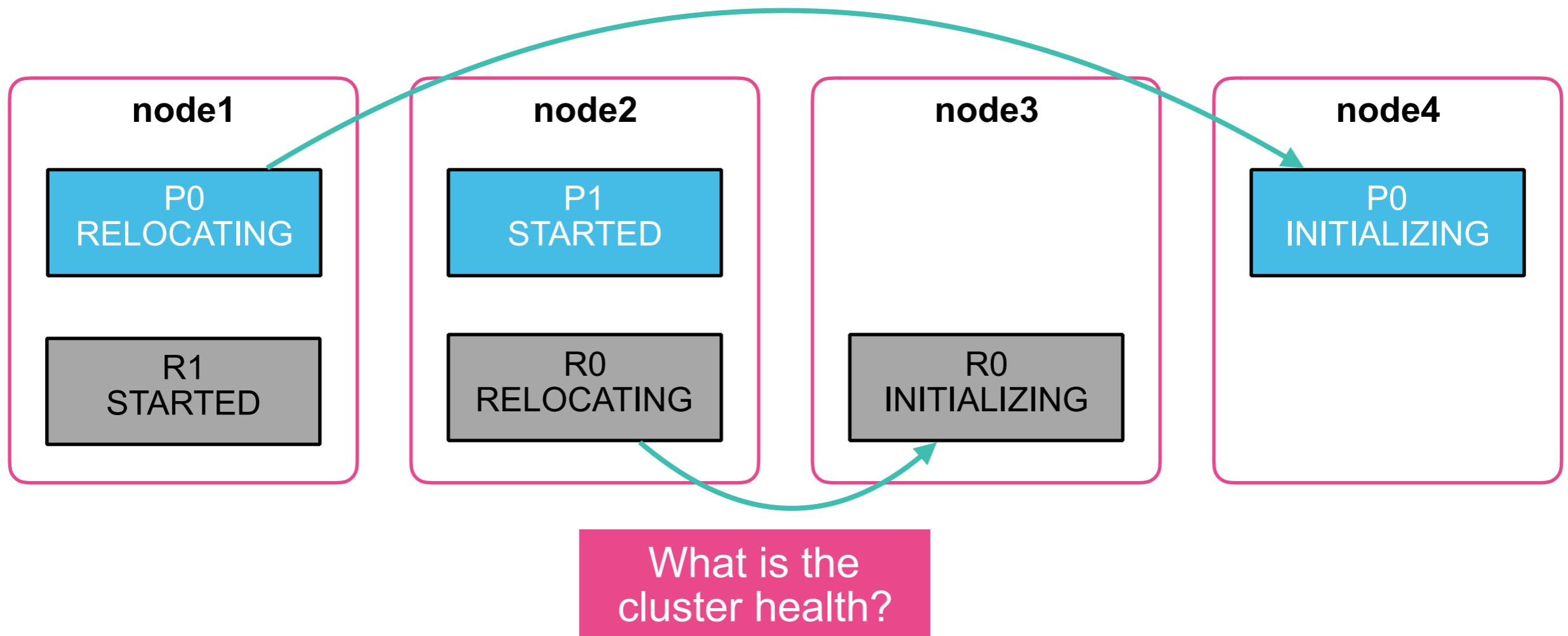
- Suppose we add a couple of nodes to our cluster:
  - the master node will attempt to balance the cluster by distributing shards to the new nodes



What is the  
cluster health?

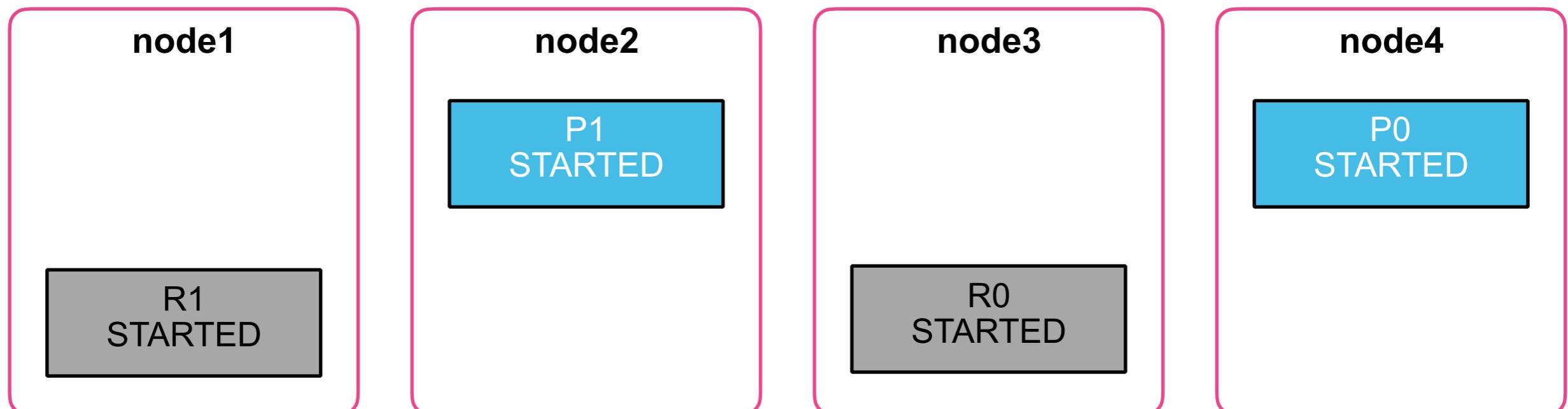
# RELOCATING

- A shard that is currently moving from one node to another is in the **RELOCATING** state
  - operations (index, delete, search, etc.) can still be done on the shard during relocation



# RELOCATING

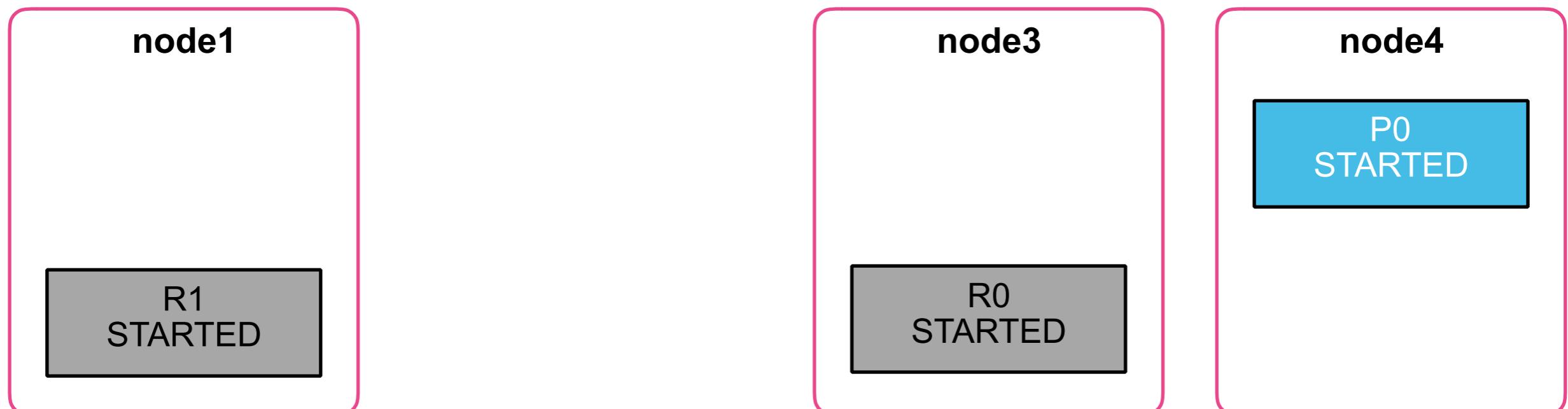
- After the relocated shard is started, the original shard gets removed:



What is the  
cluster health?

# Removing Nodes

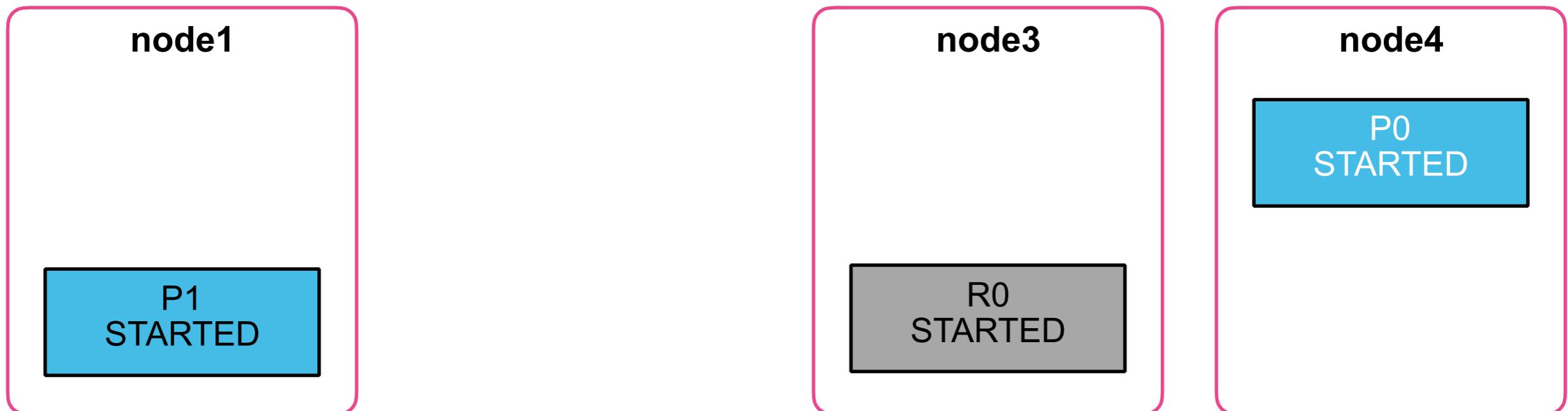
- Suppose we remove **node2** from the cluster (or it fails somehow):
  - no data is lost, because **R1** is a copy of **P1**



What is the  
cluster health?

# Shard Promotion

- R1 gets promoted to a primary shard, becomes P1:
  - replica promotion is really fast

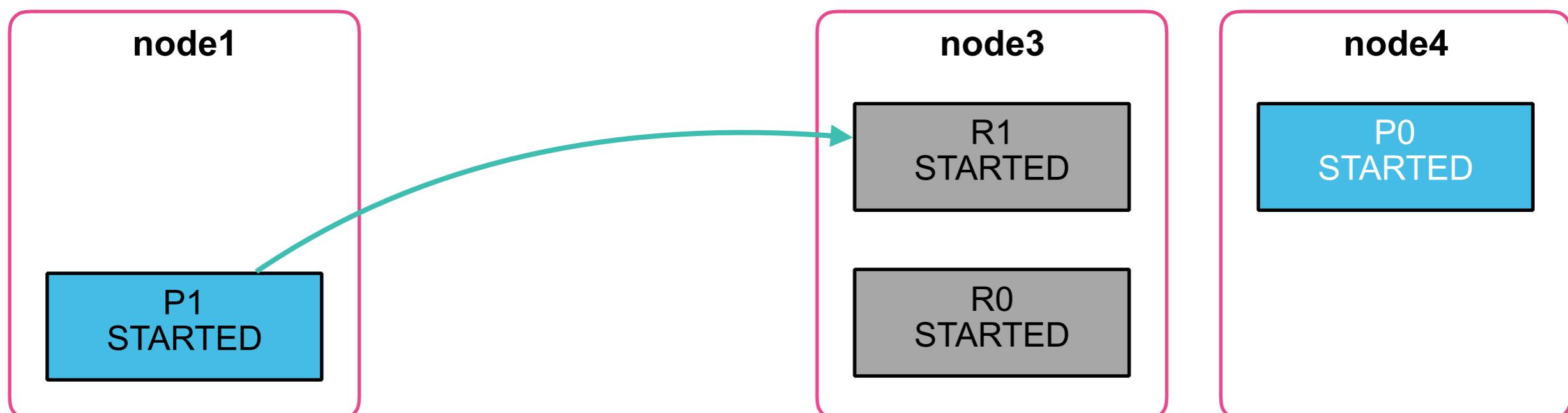


What is the  
cluster health?

# More Replicas

- P1 needs a replica, so the master node allocates a replica shard on either **node3** or **node4**:
  - recreation of missing replicas is delayed

Remember that Elasticsearch does all of this automatically without the need for manual intervention



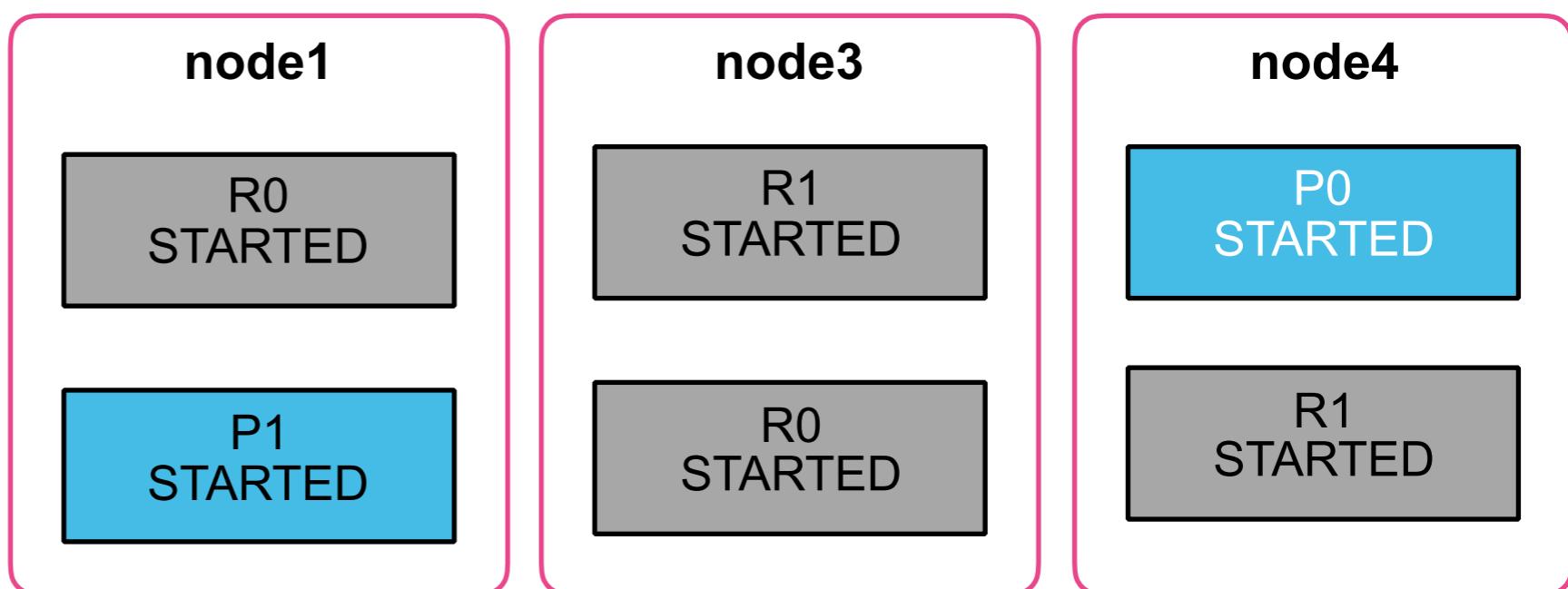
What is the cluster health?

# Adding Replicas

- If there are more shards than nodes, Elasticsearch will not allocate a replica on a node already containing a copy of this shard

```
PUT my_index/_settings
{
  "number_of_replicas": 3
}
```

What is the  
cluster health?





Elasticsearch Nodes and Shards

Lesson 3

# Review - Understanding Shards



# Summary

- Elasticsearch subdivides the data of your index into multiple pieces called ***shards***
- Each ***shard copy*** has one (and only one) ***primary*** and zero or more ***replicas***
- ***Shard allocation*** is the process of assigning a shard to a node in the cluster
- A cluster's ***health status*** is either green, yellow, or red, depending on the current shard allocation



# Quiz

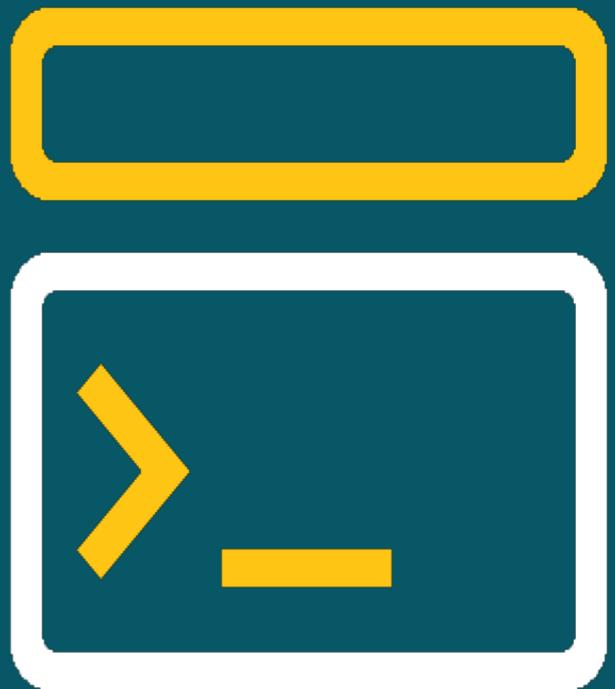
1. If **number\_of\_shards** for an index is 4, and **number\_of\_replicas** is 2, how many total shards will exist for this index?
2. **True or False:** The nodes where shards are allocated are decided by the coordinating node.
3. **True or False:** A cluster with a yellow status is missing indexed documents.



Elasticsearch Nodes and Shards

Lesson 3

# Lab - Understanding Shards





Elasticsearch Nodes and Shards

Lesson 4

# Distributed Operations

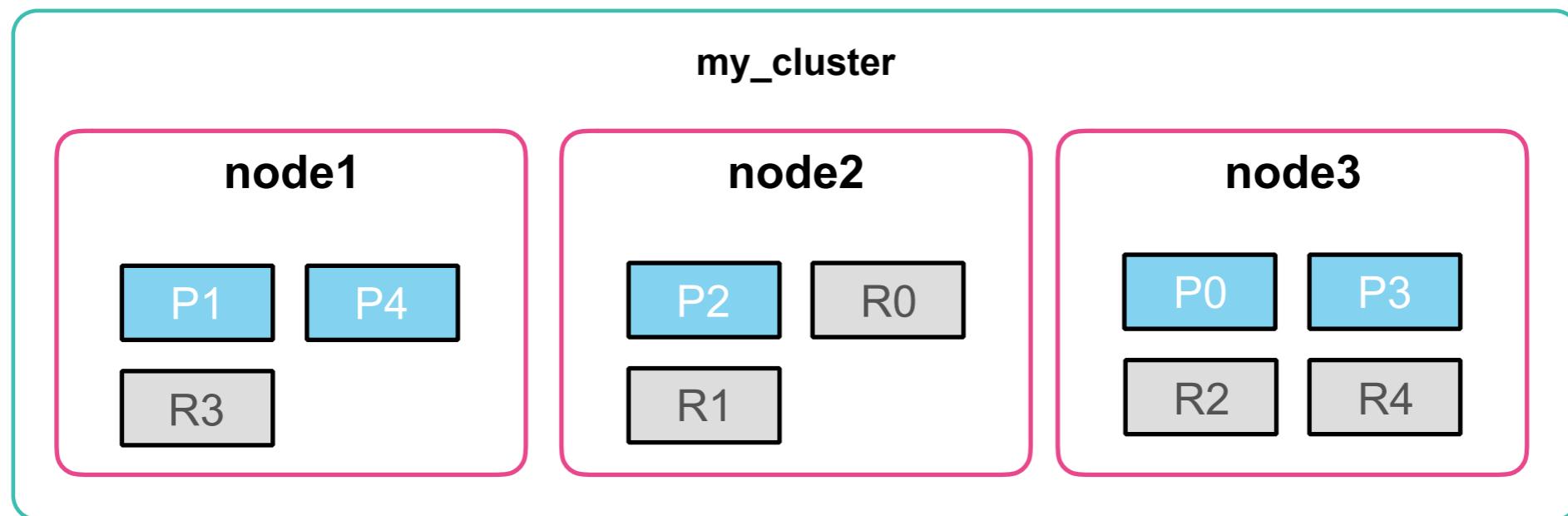


# Anatomy of a Write Operation

# How Data Gets In

- Let's take a look at the details of how a document is indexed into a cluster
  - suppose we index the following document into our **blogs** index, which currently has 5 primary shards with 1 replica

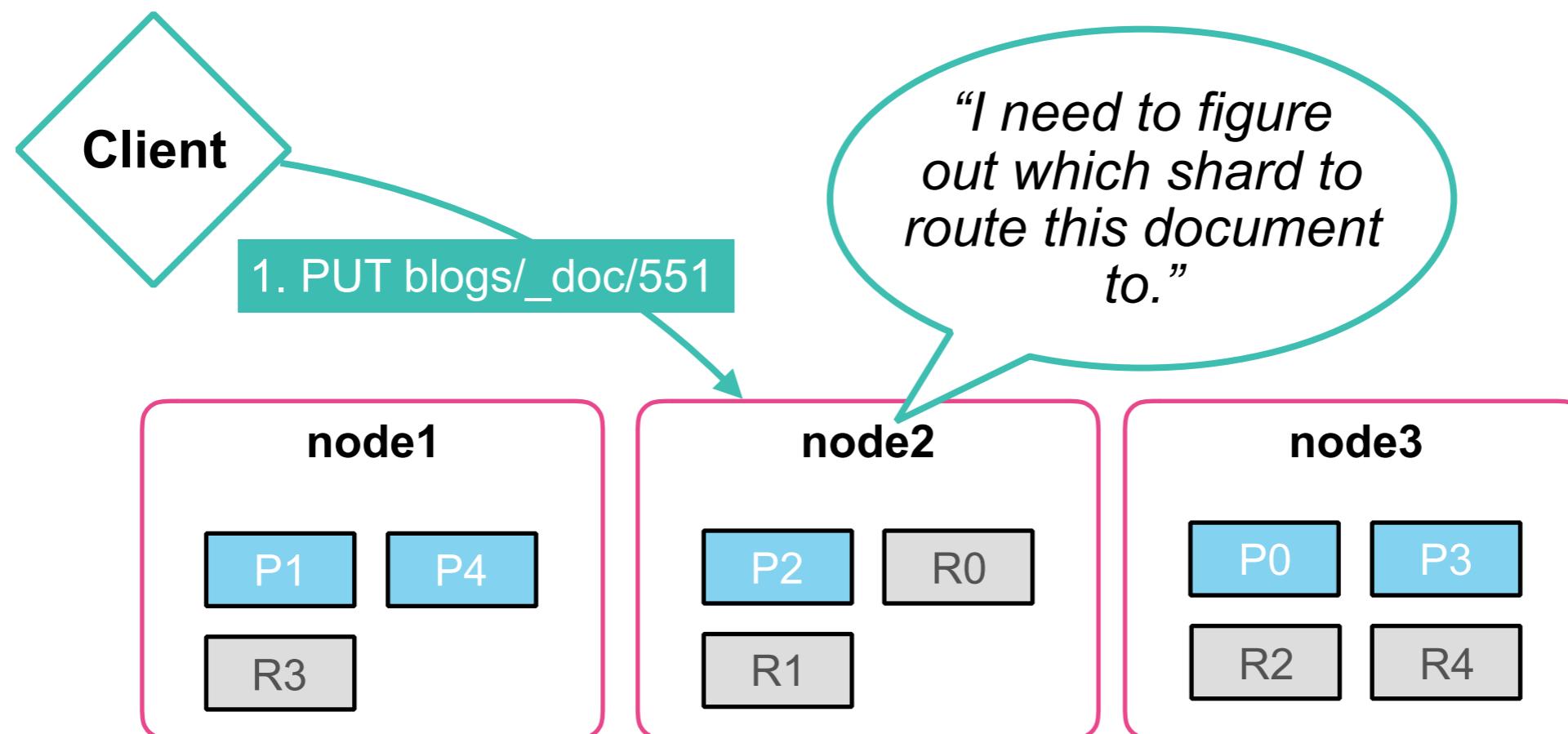
```
PUT blogs/_doc/551
{
  "title": "A History of Logstash Output Workers",
  "category": "Engineering",
  ...
}
```



# Document Routing

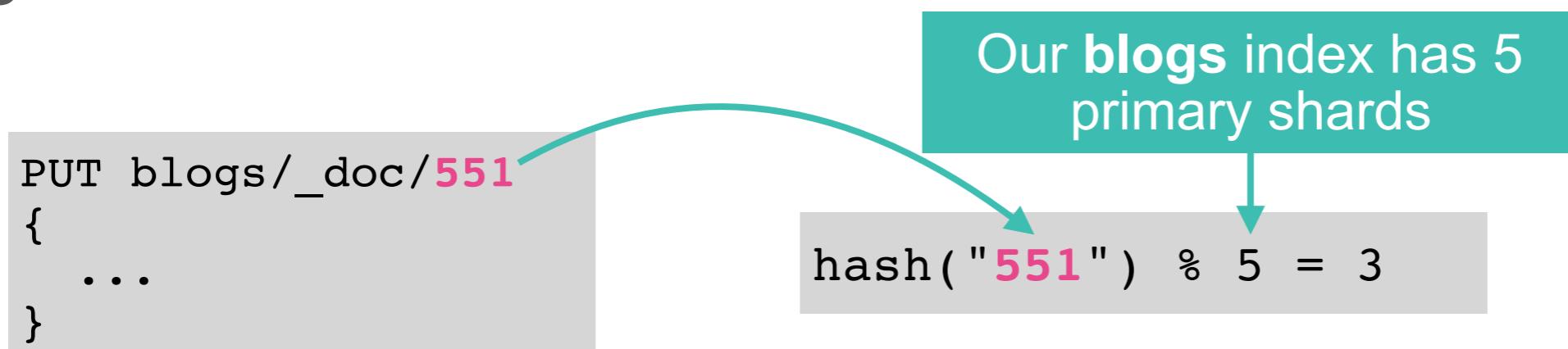
- When a document is indexed, it needs to be determined **which shard to index the document to**
  - the shard is chosen based on a simple formula

```
shard = hash(_routing) % number_of_primary_shards
```



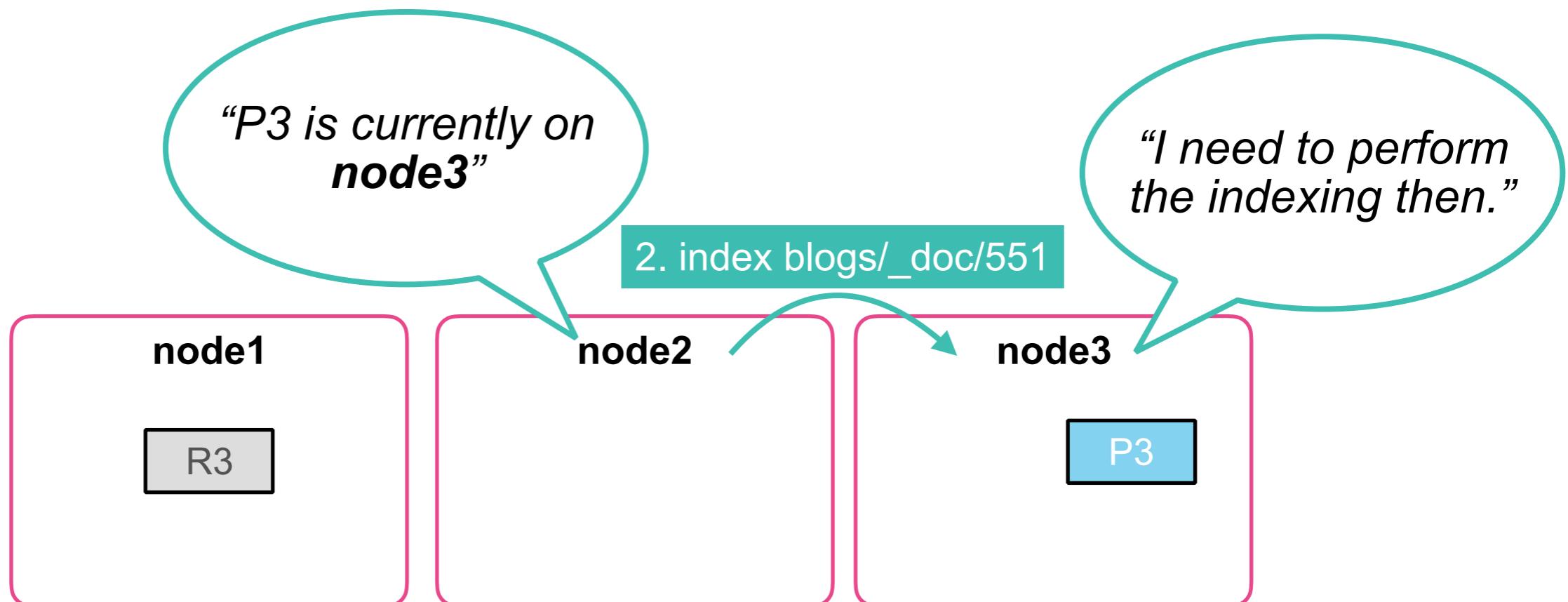
# Document Routing

- ***Why does it matter which shard a document is routed to?***
  - because Elasticsearch needs to be able to retrieve the document later!
- ***What is the value of \_routing?***
  - the `_id` is used by default, but you can specify a different value if desired
- In our **blogs** index, the document with `_id` equal to “**551**” gets routed to **shard 3**



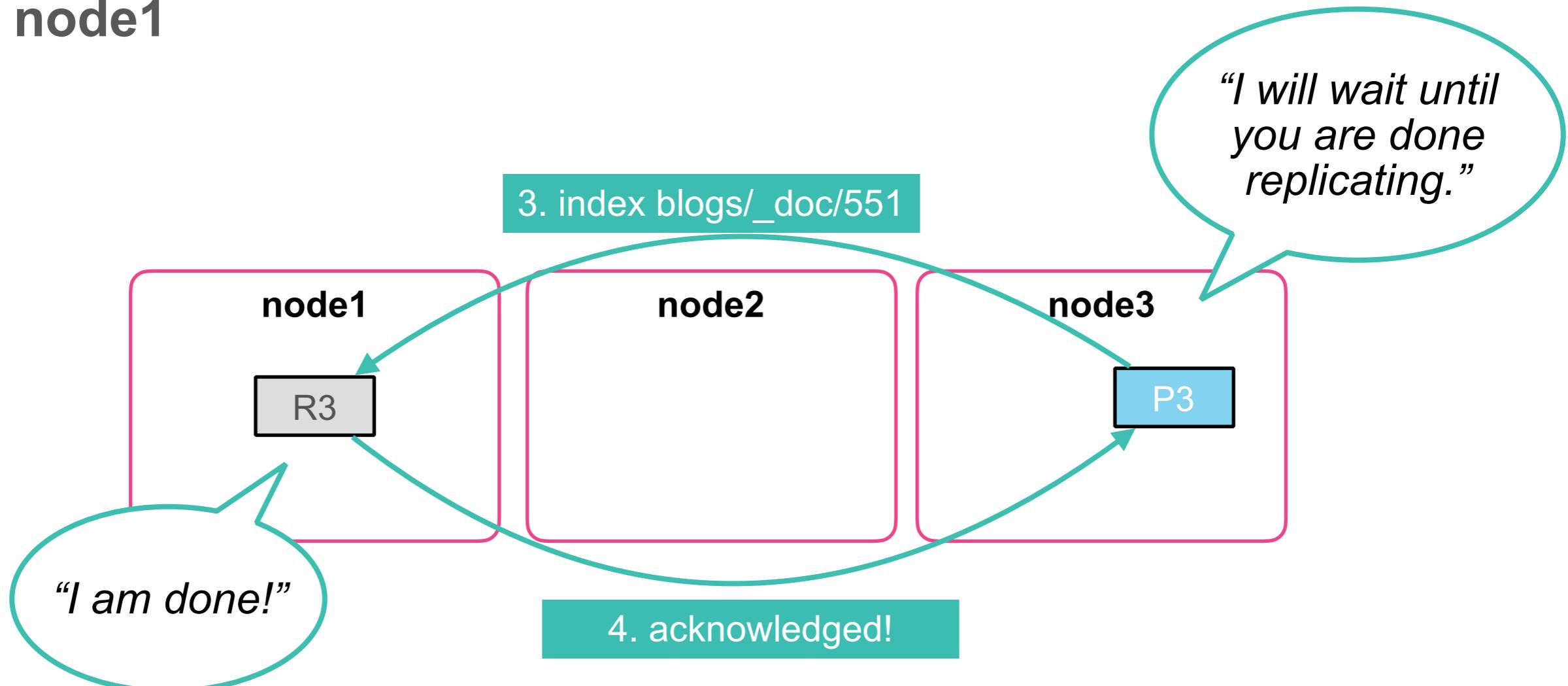
# Write Operations on the Primary Shard

- When you index, delete, or update a document, the **primary shard** has to perform the operation first
  - so node2 is going to forward the indexing request to node3



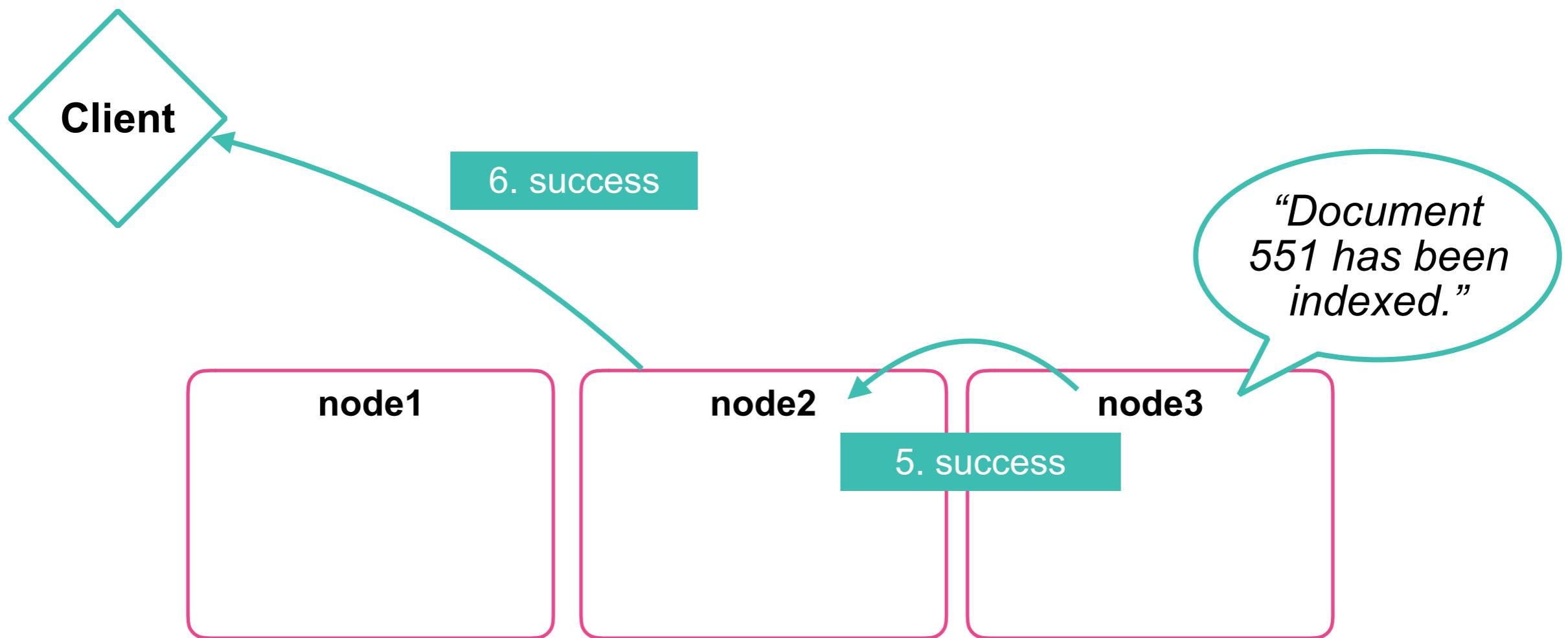
# Replicas are Synced

- node3 indexes the new document, then forwards the request (in parallel) to all replica shards
  - in our example, P3 has one replica (R3) that is currently on node1



# Client Response

- node3 lets the coordinating node (node2 in our example) know that the write operation is successful on every shard
  - and node2 sends the details back to the client application



# Updates and Deletes

- The process of an **\_update** or **DELETE** is similar to the indexing of a document
- An **\_update** to a document is actually three steps:
  - the source of the current document is retrieved
  - the current version of the document is deleted
  - then a merged new version of the entire document is indexed

# The refresh\_interval Setting

- New indexed documents are **not searchable** until a **refresh** occurs
  - by default, it takes up to 1 second
  - this is a trade-off between responsiveness and speed
  - you can still **GET** documents by *ID*
- the **refresh\_interval** setting controls that behavior
  - it is a dynamic index level setting

```
PUT my_index/_settings
{
  "refresh_interval": "30s"
}
```

Increase the **refresh\_interval** in index heavy scenarios to achieve better indexing performance

# The refresh Parameter

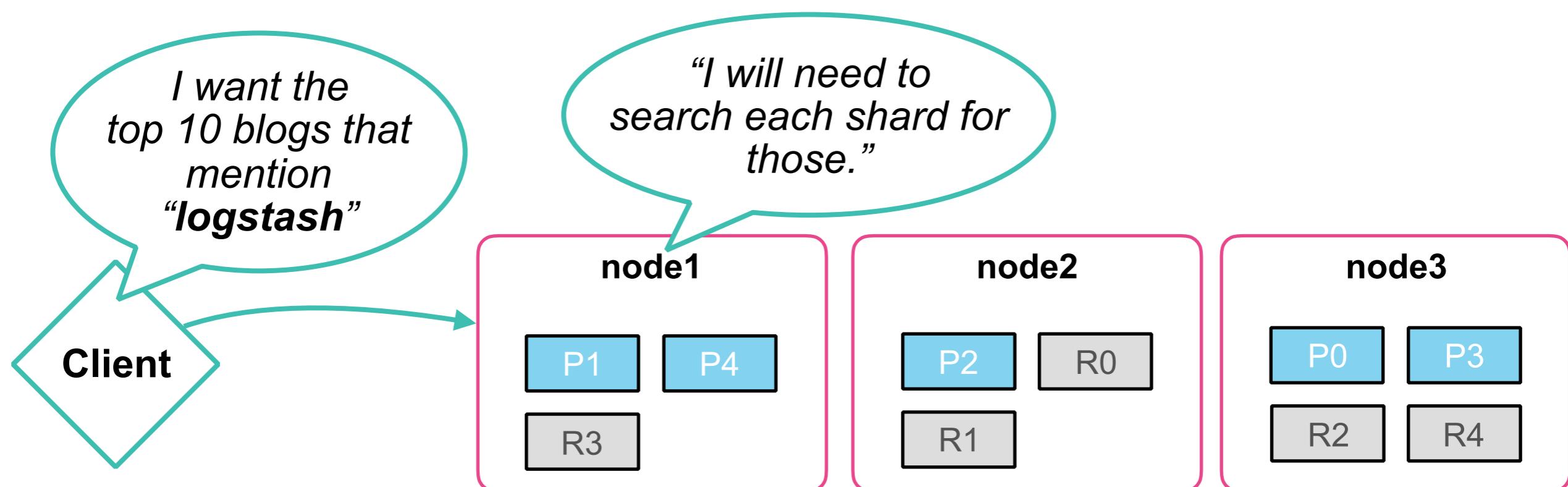
- Controls the refresh behavior on a per request basis:
  - **false**: the default value, any changes to the document are *not* visible immediately
  - **true**: forces a refresh in the affected primary and replica shards so that the changes are visible immediately
  - **wait\_for**: synchronous request that waits for a refresh to happen

```
PUT blogs/_doc/551?refresh=wait_for
{
  "title": "A History of Logstash Output Workers",
  "category": "Engineering",
  ...
}
```

# Anatomy of a Search

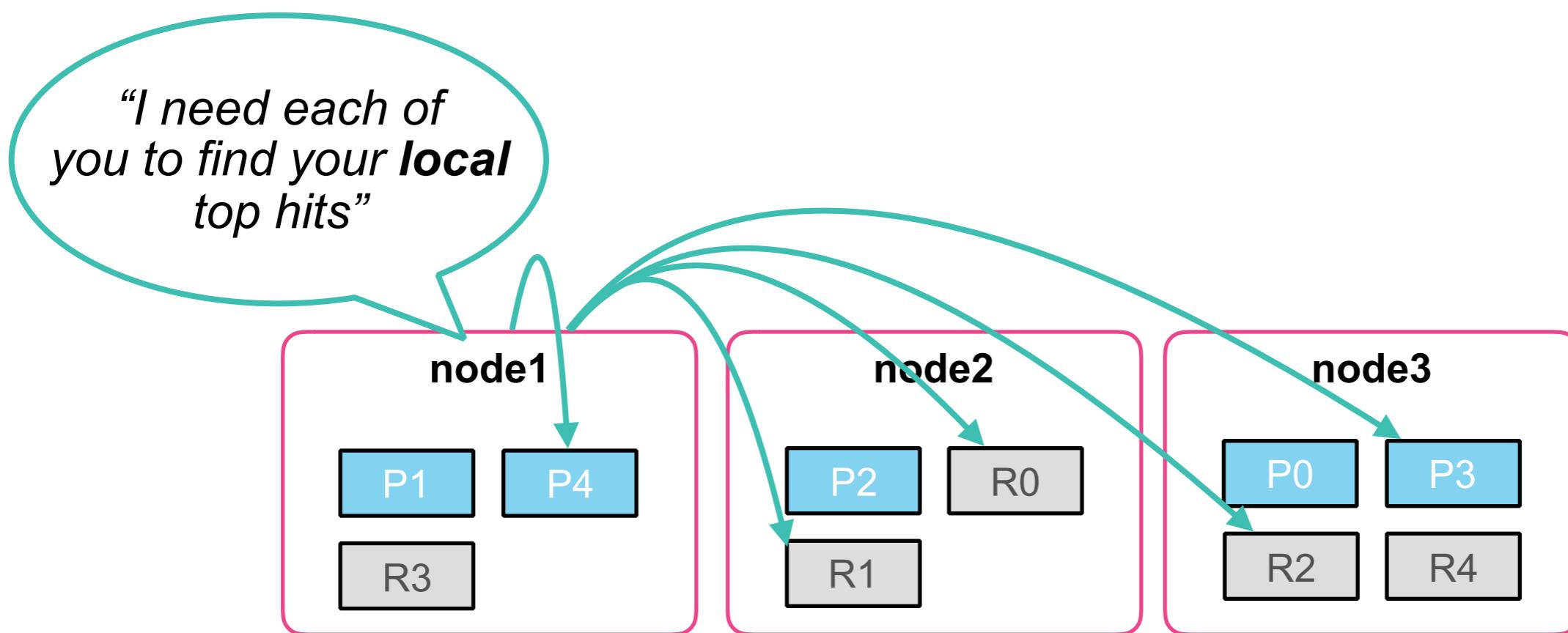
# Anatomy of a Search

- Distributed search is a challenging task
  - we have to search for hits in *a copy of every shard* in the index
- And finding the documents is only half the story!
  - the hits must be combined into a single, sorted list of documents that represents a page of results



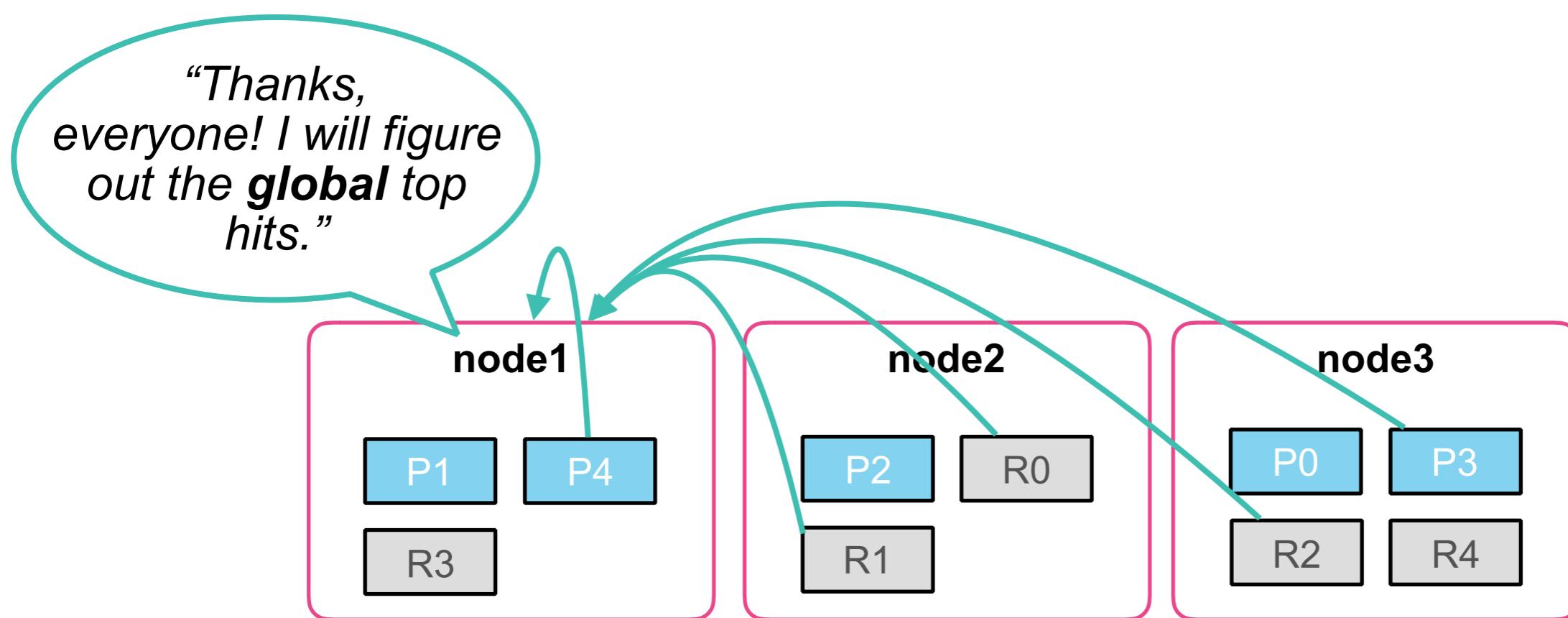
# The Query Phase

- The initial part of a search is referred to as the *query phase*
  - the query is broadcast to a **shard copy** of every shard in the index,
  - and each shard executes the query *locally*



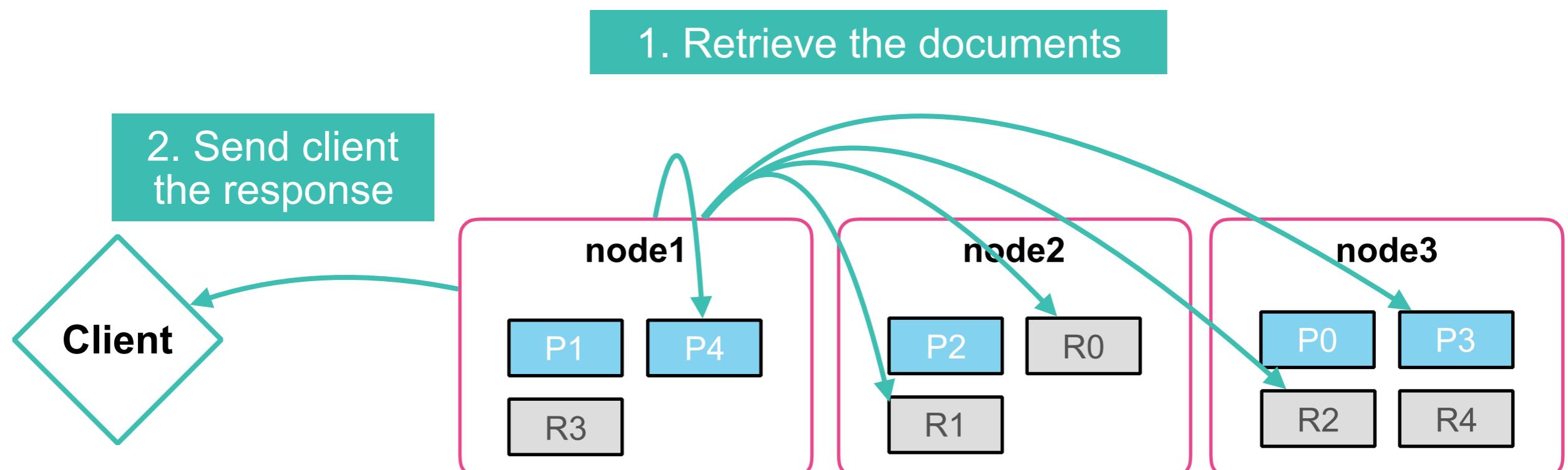
# The Query Phase

- Each shard *returns the doc IDs and sort values* of its top hits to the coordinating node
- The coordinating node *merges these values* to create a *globally sorted* list of results



# The Fetch Phase

- Now that the coordinating node knows the doc ID's of the top 10 hits, it can now ***fetch the documents***
  - and then returns the top documents to the client





Elasticsearch Nodes and Shards

Lesson 4

# Review - Distributed Operations



# Summary

- A search consists of a *query phase* and a *fetch phase*
- By default, the id of a document is considered when determining which shard of the index to route the document to
- A *shard* is a single instance of Lucene that holds data

# Quiz

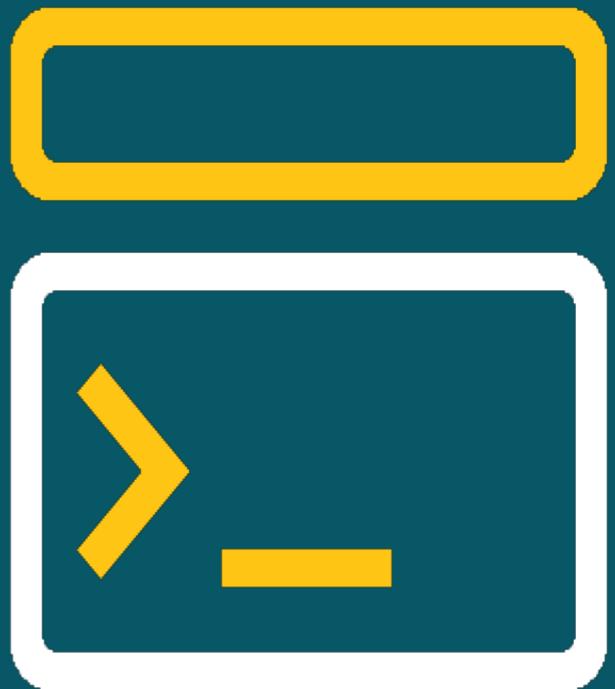
- 1. True or False:** An index operation has to be executed on the primary shard first before being synced to replicas.
- 2. True or False:** After an index operation documents might not be searchable up to 1 second.
- 3. What happens if you use ?refresh=wait\_for in an index request. Explain one use case that benefits from it?**



Elasticsearch Nodes and Shards

Lesson 4

# Lab - Distributed Operations



- Elasticsearch Fundamentals
- Elasticsearch Queries
- Elasticsearch Aggregations
- Elasticsearch Text Analysis and Mappings
- Elasticsearch Nodes and Shards
- Elasticsearch Monitoring and Troubleshooting

Module 6

# Elasticsearch Monitoring and Troubleshooting



# Topics

- HTTP Response and Shard Allocation Issues
- Monitoring
- Diagnosing Performance Issues



Elasticsearch Monitoring & Troubleshooting

Lesson 1

# HTTP Response and Shard Allocation Issues



# Elasticsearch Responses

- Elasticsearch uses the REST APIs
- There are two sections in each response:
  - *HTTP response status*: 200, 201, 404, 501
  - and the *response body*

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



201 HTTP Status

```
{
  "_index": "test",
  "_type": "_doc",
  "_id": "1",
  "_version": 1,
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 0,
  "_primary_term": 1
}
```

# Common HTTP Errors

Issue	Reason	Action
<b>Unable to connect</b>	networking issue or cluster down	check network status and cluster health
<b>connection unexpectedly closed</b>	node died or network issue	retry (with risk of creating a duplicate document)
<b>4xx error</b>	client error	fix bad request before retrying
<b>429 error</b>	Elasticsearch is too busy	retry (ideally with linear or exponential backoff)
<b>5xx error</b>	internal server error in Elasticsearch	look into Elasticsearch logs to see what is the issue

# Understanding the Index Response Body

- Write operations like **delete**, **index**, **create**, and **update** return shard information:

```
"_shards": {  
  "total": 2,  
  "successful": 2,  
  "failed": 0  
},
```

The response contains shard totals

- **total**: how many shard copies (primary and replica shards) the index operation **should be executed** on
- **successful**: the number of shard copies the index operation **successfully** executed on
- **failed**: the number of shard copies the index operation **failed** on
- **failures**: in case of failures, an array that contains related **errors**

# Understanding the Index Response Body

```
"_shards": {  
  "total": 2,  
  "successful": 1,  
  "failed": 0  
},
```

1 replica is not available

```
"_shards": {  
  "total": 2,  
  "successful": 0,  
  "failed": 2  
},
```

Both primary and replica failed

```
"_shards": {  
  "total": 2,  
  "successful": 1,  
  "failed": 1,  
  "failures": [  
    {  
      "_index": "test",  
      "_shard": 0,  
      "_node": "-mz8qioOQk-sIx9X4A3w_w",  
      "reason": {  
        "type": "node_disconnected_exception",  
        "reason": "[node3][172.18.0.4:9300]  
[indices:data/write/bulk[s][r]] disconnected"  
      },  
      "status": "INTERNAL_SERVER_ERROR",  
      "primary": false  
    }  
  ]  
}
```

In case of failures  
Elasticsearch will return  
the reasons in the body



# Understanding the Search Response Body

- Search responses are similar to the index ones, but
  - search is only executed in one copy (primary or replica)

```
_shards": {  
  "total": 47,  
  "successful": 47,  
  "skipped": 0,  
  "failed": 0  
},
```

All shards succeeded

- **total**: the number of shards the search *should be executed* on
- **successful**: the number of shards the search *succeeded* on
- **skipped**: the number of shards that *cleverly avoided* search execution because they contain data which cannot possibly match the query
- **failed**: the number of shards the search *failed* on
- **failures**: in case of failures, an array that contains related *errors*

# Causes of Partial Results

```
"_shards": {  
  "total": 5,  
  "successful": 3,  
  "skipped": 0,  
  "failed": 0  
},
```

2 shards were not available in the cluster when the request was received

```
"_shards": {  
  "total": 5,  
  "successful": 3,  
  "skipped": 0,  
  "failed": 2,  
  "failures": [  
    {  
      "shard": 1,  
      "index": "test",  
      "node": "-mz8qio0Qk-sIx9X4A3w_w",  
      "reason": {  
        "type": "node_disconnected_exception",  
        "reason": "[node3][172.18.0.4:9300][indices:data/read/search[phase/query]] disconnected"  
      } } , ... }
```

# Finding Health Issues

- Suppose your cluster is having issues
  - getting the health at the *cluster level* can reveal if nodes are down

GET `_cluster/health`

Suppose you have a 4 node cluster, but only 2 of them appear in the health

```
{  
  "cluster_name": "my_cluster",  
  "status": "red",  
  "timed_out": false,  
  "number_of_nodes": 2,  
  "number_of_data_nodes": 2,  
  "active_primary_shards": 60,  
  "active_shards": 120,  
  ...  
}
```

GET `_cluster/health?wait_for_status=yellow`

Blocks until the **status** of the cluster becomes **yellow** or **green**

# Drilling Down to Indices

- A red cluster means you have at least one red index
  - getting the health at the *index level* reveals the problematic index (or indices)

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

Set “**level**” equal to “**indices**” to view the health of all indices

“**my\_index**” should have 5 active primaries, but it only has 2

```
{  
  "cluster_name": "my_cluster",  
  "status": "red",  
  ...  
  "indices": {  
    "my_index": {  
      "status": "red",  
      "number_of_shards": 5,  
      "number_of_replicas": 1,  
      "active_primary_shards": 2,  
      "active_shards": 3,  
      "relocating_shards": 0,  
      "initializing_shards": 0,  
      "unassigned_shards": 5  
    },  
    ...  
  }  
}
```

# Health of a Specific Index

- You can request the health of a specific index using the following syntax:

```
GET _cluster/health/my_index
```

Shards stats are for  
“my\_index” only

```
{  
  "cluster_name": "my_cluster",  
  "status": "red",  
  "timed_out": false,  
  "number_of_nodes": 2,  
  "number_of_data_nodes": 2,  
  "active_primary_shards": 2,  
  "active_shards": 3,  
  "relocating_shards": 0,  
  "initializing_shards": 0,  
  "unassigned_shards": 5,  
  "delayed_unassigned_shards": 0,  
  "number_of_pending_tasks": 0,  
  "number_of_in_flight_fetch": 0,  
  "task_max_waiting_in_queue_millis": 0,  
  "active_shards_percent_as_number": 47.014  
}
```

# Drilling Down to Shards

- You can drill down even further and determine which shards within the index are red:

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

Helpful, but often  
too detailed

```
{  
  "cluster_name": "my_cluster",  
  "status": "red",  
  ...  
  "indices": {  
    "my_index": {  
      "status": "red",  
      "number_of_shards": 5,  
      "number_of_replicas": 1,  
      "active_primary_shards": 2,  
      "active_shards": 3,  
      "relocating_shards": 0,  
      "initializing_shards": 0,  
      "unassigned_shards": 5,  
      "shards": {  
        "0": {  
          "status": "red",  
          "primary_active": false,  
          "active_shards": 0,  
          "relocating_shards": 0,  
          "initializing_shards": 0,  
          "unassigned_shards": 0  
        },  
        ...  
      },  
      ...  
    },  
    ...  
  },  
  ...  
}
```

# Cluster Allocation Explain API

- Having an **UNASSIGNED** primary shard means your cluster is not in a good state!
  - lots of failures when attempting to index documents, retrieve documents, run queries, etc.
- Elasticsearch provides the ***Cluster Allocation API*** to help you locate any **UNASSIGNED** shards:
  - also known as the ***explain API*** for short

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

Returns detailed information of  
the first unassigned shard it finds

# Locating Unassigned Shards

- The response from the ***explain API*** lists unassigned shards
  - and an explanation of why they are unassigned

GET \_cluster/allocation/explain

Caused by either a node  
that went down or improper  
permission settings

Shard 1 is an unassigned  
primary shard

```
{  
  "index": "my_index",  
  "shard": 1,  
  "primary": true,  
  "current_state": "unassigned",  
  "unassigned_info": {  
    "reason": "INDEX_CREATED",  
    "at": "2017-02-02T22:59:36.686Z",  
    "last_allocation_status": "no_attempt"  
  },  
  "can_allocate": "no",  
  "allocate_explanation": "cannot allocate because  
allocation is not permitted to any of the nodes",  
  ...  
}
```

# Specific Shard Allocation Details

- You can also request the allocation details of a specific shard in a specific index:

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

Returns detailed information for just this particular shard



Lesson 1

# Review - HTTP Response and Shard Allocation Issues



# Summary

- Every write or read operation returns shard information, such as, the number of shards it should be executed on
- A cluster's **health** reports various statistics and the status of the cluster
- The ***Cluster Allocation API*** helps locating **UNASSIGNED** shards

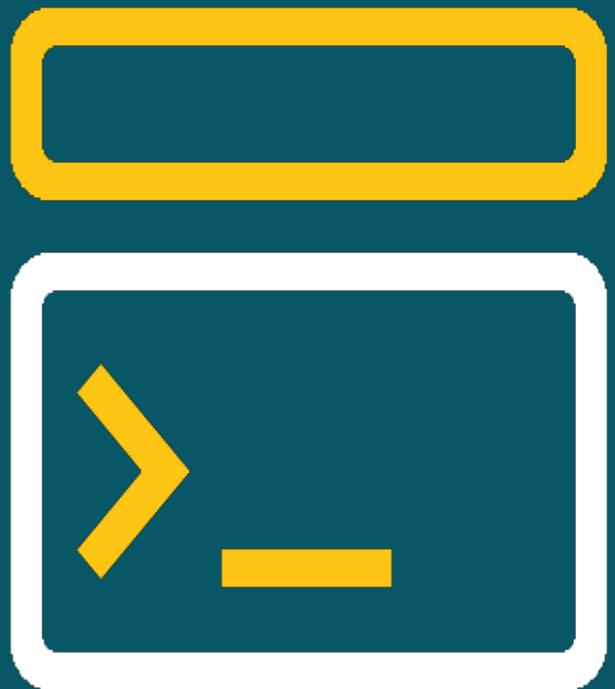
# Quiz

1. Why should you bother checking the “`_shards`” section of a response?
2. **True or False:** Having an **UNASSIGNED** primary shard means your cluster is not in a good state.
3. **True or False:** The “**skipped**” section of a response indicates the number of shards that failed to execute a search request.



Lesson 1

# Lab - HTTP Response and Shard Allocation Issues





Elasticsearch Monitoring & Troubleshooting

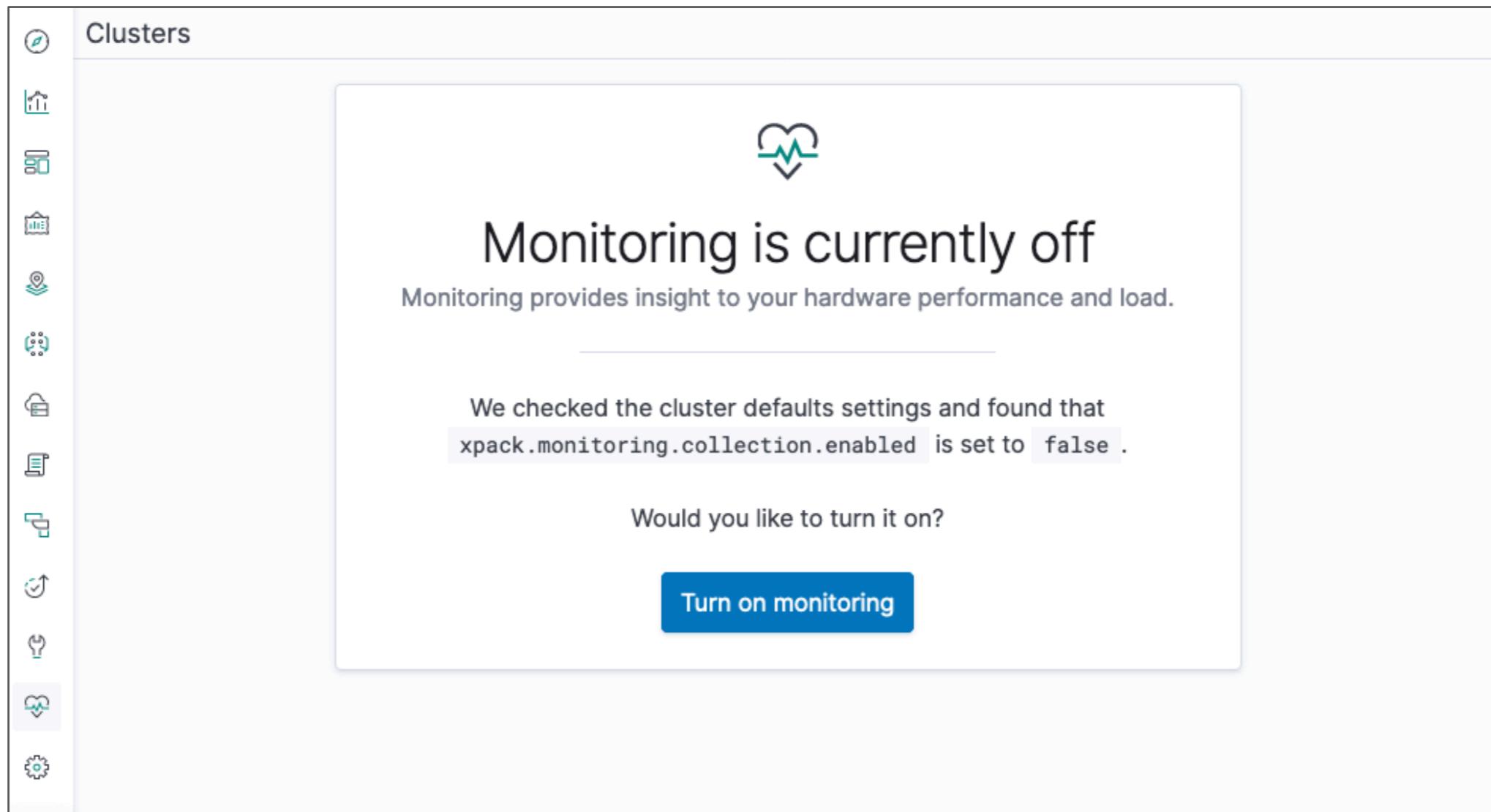
# Lesson 2

# Monitoring



# Elastic Monitoring

- **Monitoring** uses Elasticsearch to monitor Elasticsearch
  - `xpack.monitoring.collection.enabled` defaults to `false`
  - the stats of all the nodes are indexed into Elasticsearch



# Configuring Monitoring

- By default, each node contains an agent to collect data



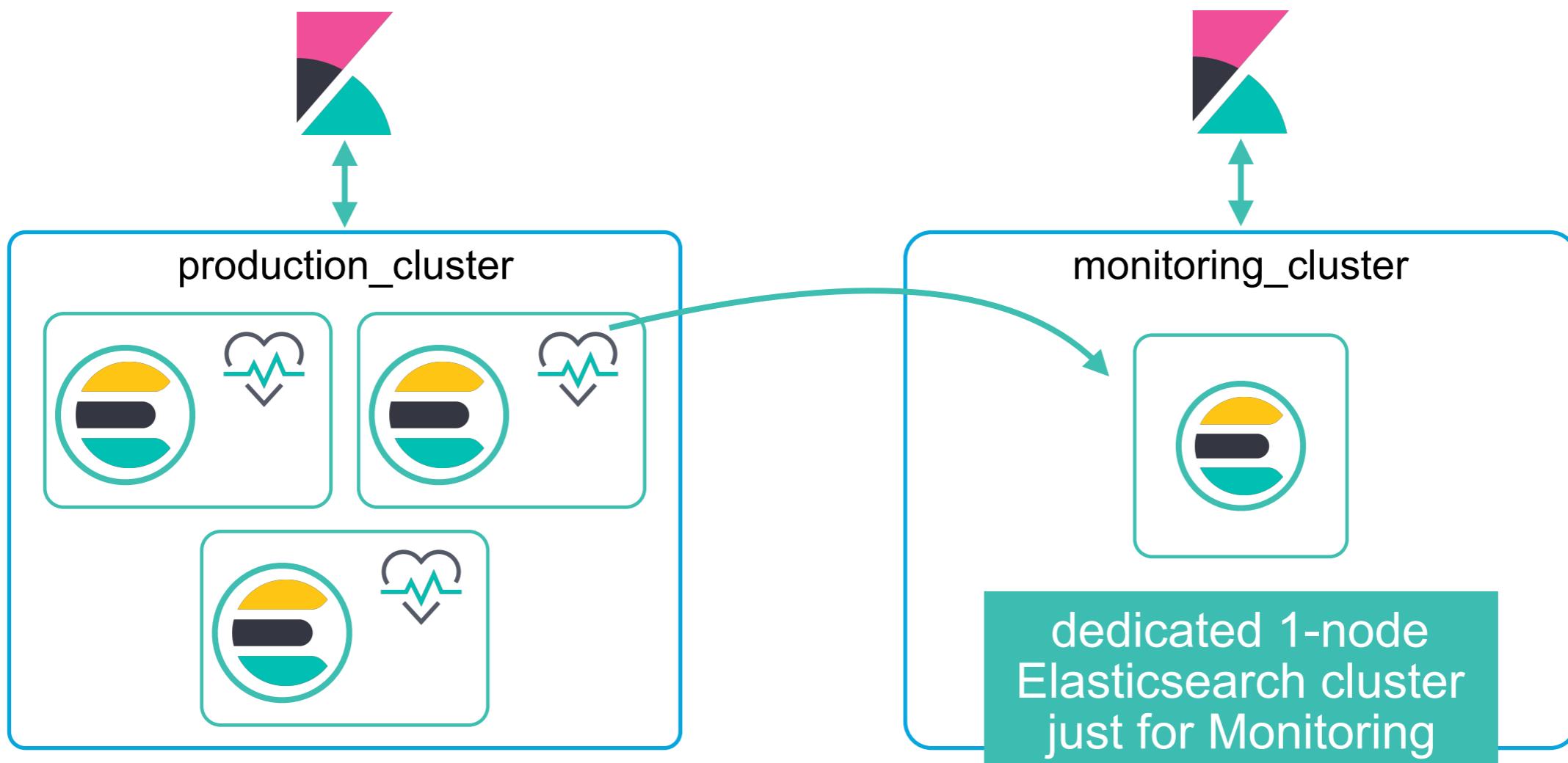
- Here are a few common Monitoring settings
  - which can be configured in `elasticsearch.yml`:

<code>xpack.monitoring.collection.indices</code>	The indices to collect data from. Defaults to all indices, but can be a comma-separated list.
<code>xpack.monitoring.collection.interval</code>	How often data samples are collected. Defaults to 10s
<code>xpack.monitoring.exporters</code>	Configures where the agent stores monitoring data. By default, the agent uses a <b>local exporter</b> that indexes monitoring data on the cluster where it is installed.



# Dedicated Monitoring Cluster

- Use a *dedicated cluster* for Monitoring (**recommended**)
  - to reduce the load and storage on the monitored clusters
  - to keep access to Monitoring even for unhealthy clusters
  - to support segregation duties (separate security policies)



# Configuring a Dedicated Monitoring Cluster

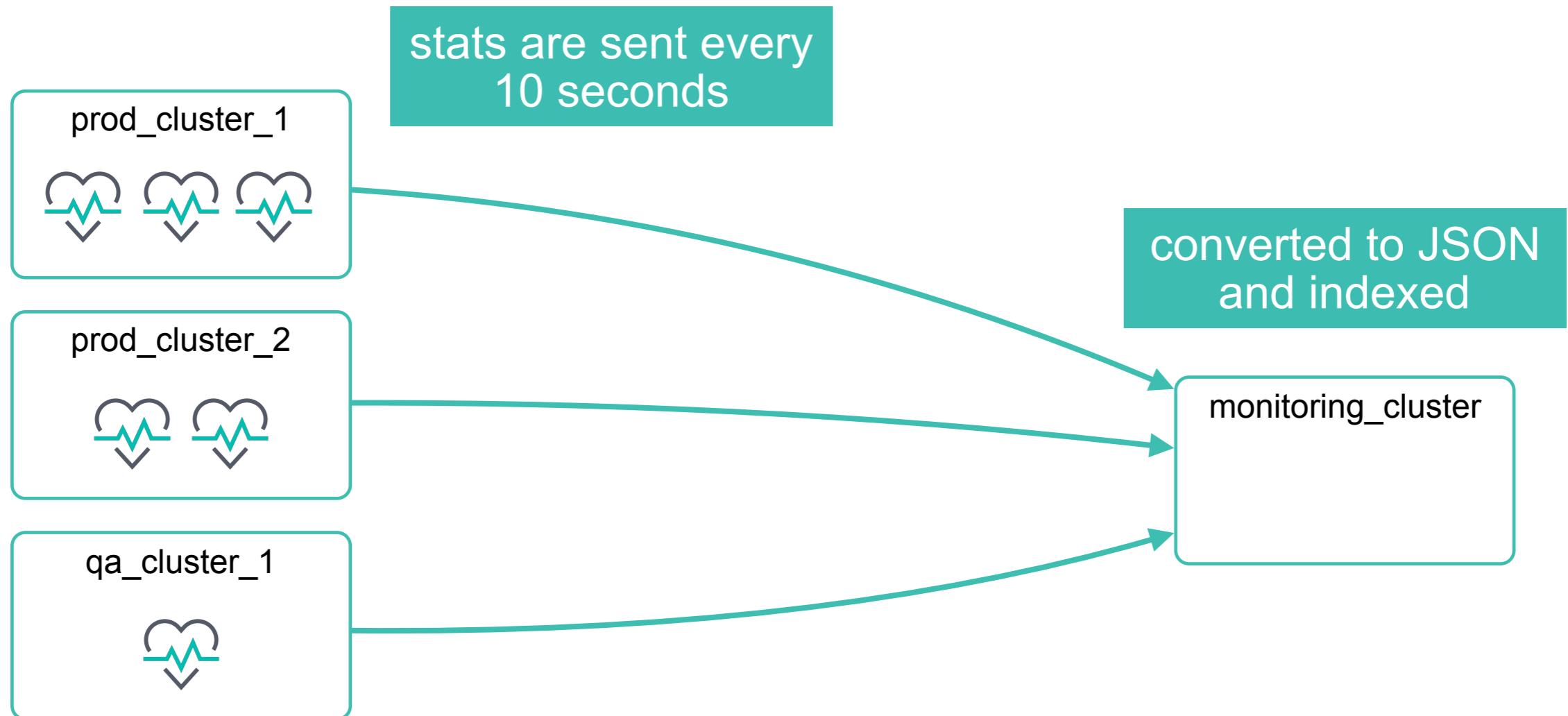
- If Monitoring is on a different cluster than the one being monitored
  - you need to tell the monitored cluster where to send its stats to
- If Elastic Security is enabled on the Monitoring cluster, then provide credentials
  - you can also use SSL/TLS (see [docs](#) for details)

```
xpack.monitoring.exporters:  
  id1:  
    type: http  
    host: ["http://monitoring_cluster:9200"]  
    auth.username: username  
    auth.password: changeme
```

configure in **elasticsearch.yml**  
of each node

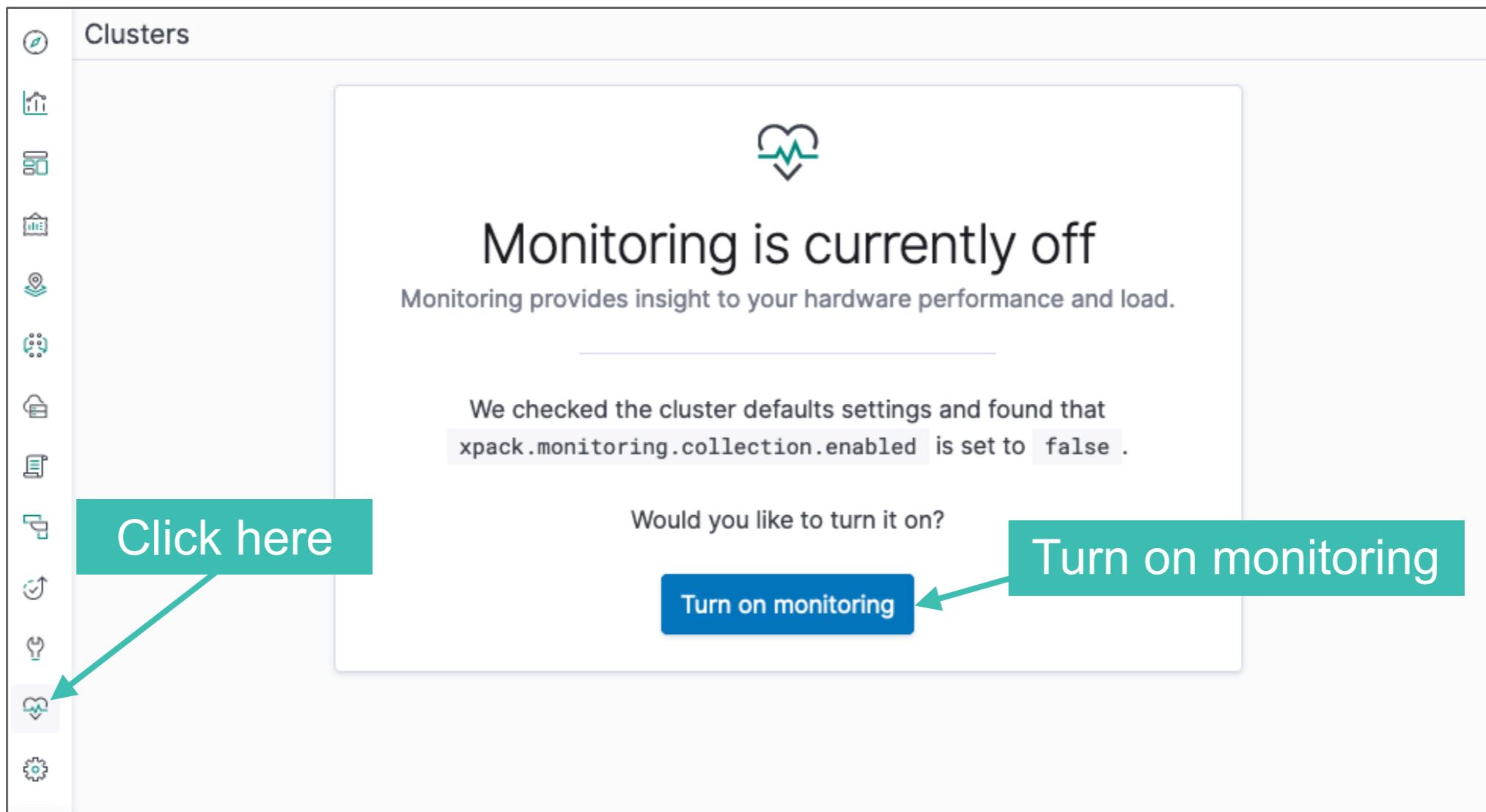
# Monitoring Multiple Clusters

- Multiple clusters can be monitored by a single dedicated monitoring cluster (Gold/Platinum)



# The Monitoring UI

- Open up Kibana
- Click on the **Monitoring** shortcut in the left toolbar:



# Clusters Dashboard

- The initial Monitoring page is the “Clusters” dashboard
  - it shows all the monitored clusters
  - and also other monitored Elastic Stack components (e.g. Kibana)

The screenshot shows the Elasticsearch Monitoring interface for a cluster named "my\_cluster". The top navigation bar indicates "Health is green" and "Basic license". The left sidebar lists various monitoring components: Elasticsearch, Kibana, Logstash, Filebeat, Metricsbeat, Auditbeat, and CloudWatch Metrics.

**Elasticsearch Overview:**

- Version: 7.1.1
- Uptime: 3 minutes

**Elasticsearch Nodes:**

- Disk Available: 74.35% (129.6 GB / 174.3 GB)
- JVM Heap: 36.15% (715.9 MB / 1.9 GB)

**Elasticsearch Indices:**

- Documents: 1,753,330
- Disk Usage: 1.2 GB
- Primary Shards: 8
- Replica Shards: 8

**Kibana Overview:**

- Requests: 1
- Max. Response Time: 26 ms

**Kibana Instances:**

- Connections: 0
- Memory Usage: 15.81% (230.2 MB / 1.4 GB)

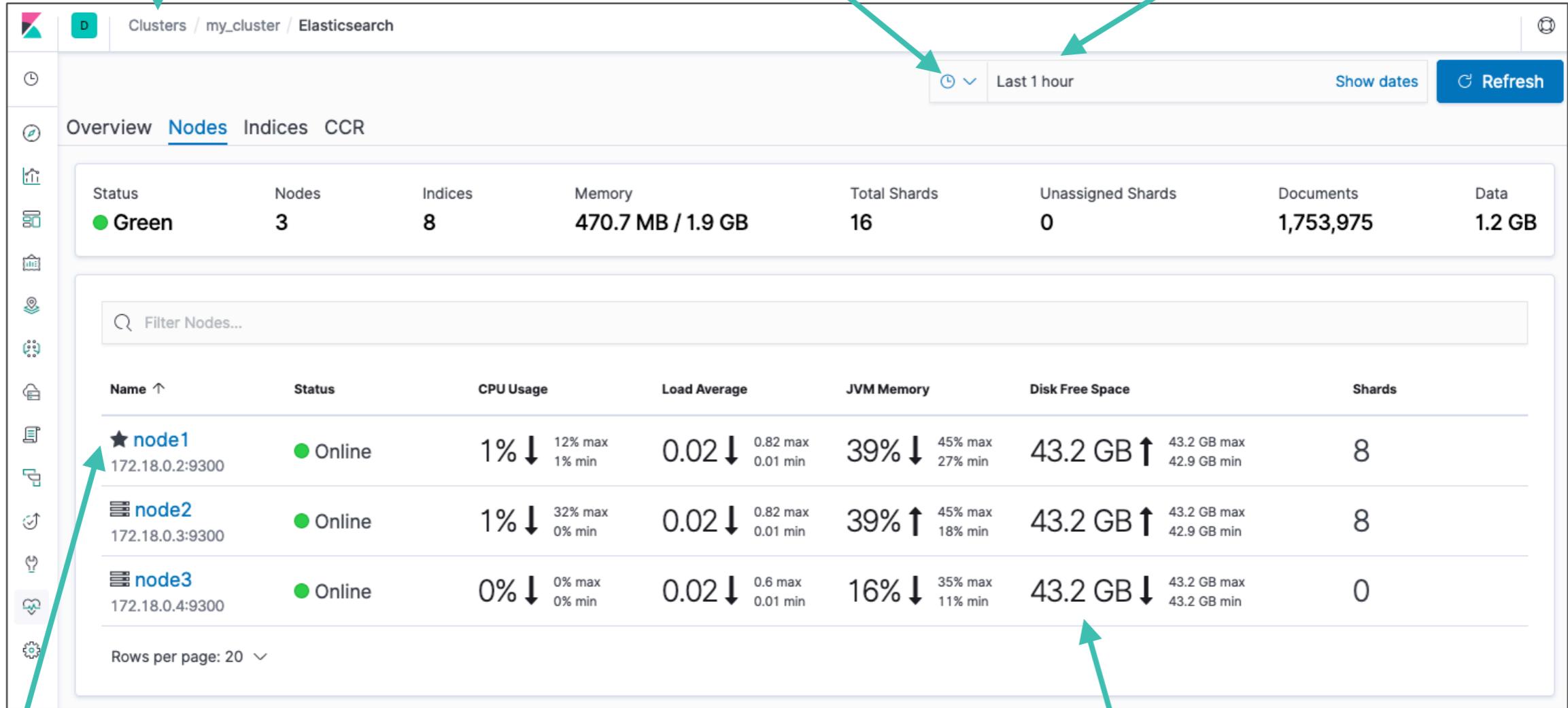
**Note:** Note the free version of Monitoring can only monitor a single cluster

# Nodes Dashboard

Breadcrumbs for easy navigation

Configuration of time period and refresh interval

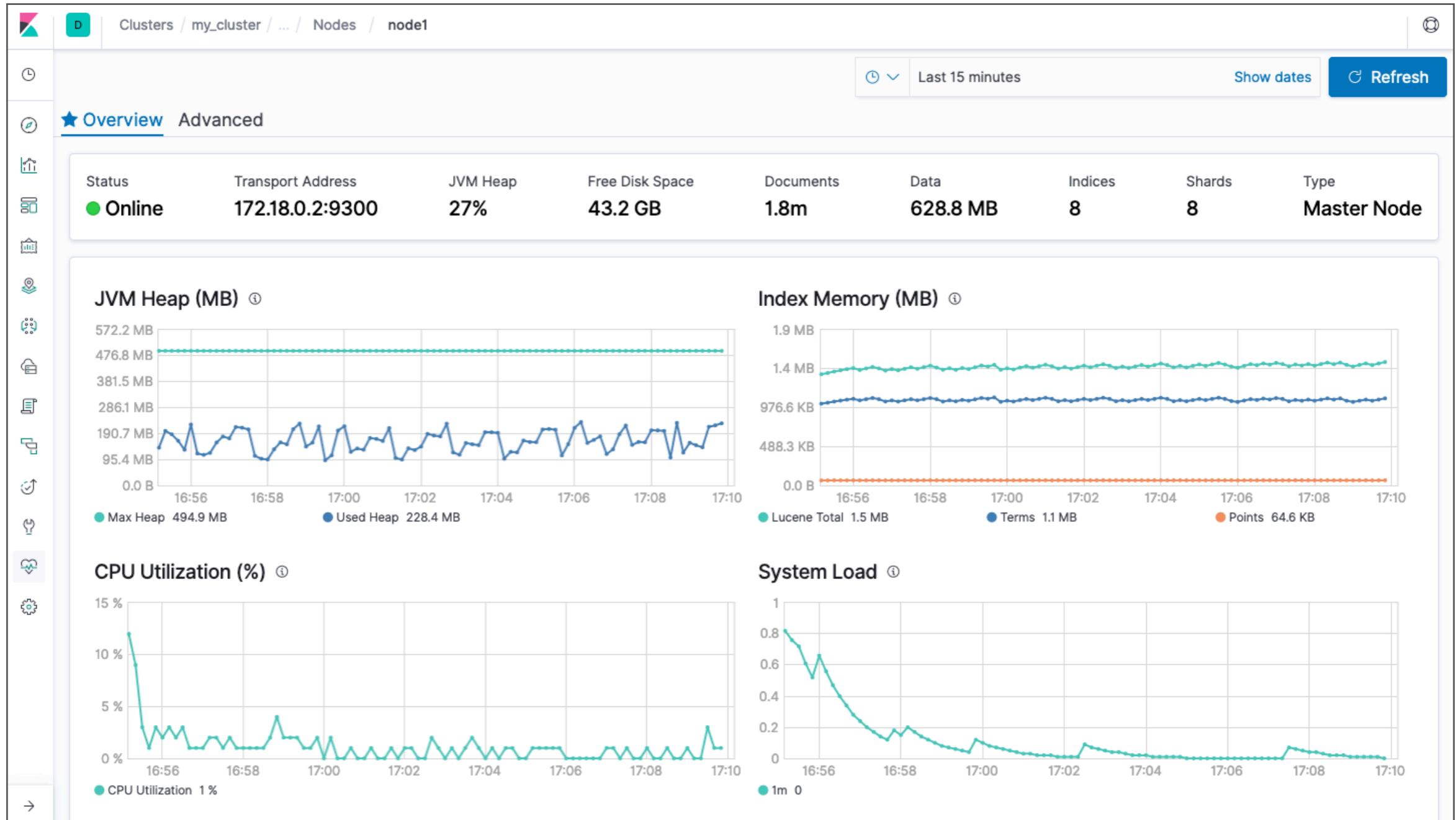
Time period of the currently displayed data



the master node is indicated by a “star”

stats updates in real-time

# Individual Node Dashboard



the actual page has more details

# Indices Dashboard

quick way to find unhealthy indices

The screenshot shows the Elasticsearch Indices Dashboard for a cluster named 'my\_cluster'. The 'Indices' tab is selected. At the top, there's an overview section with metrics: Status (Green), Nodes (3), Indices (8), Memory (487.7 MB / 1.9 GB), Total Shards (16), Unassigned Shards (0), Documents (1,754,914), and Data (1.2 GB). Below this is a table of indices with columns: Name, Status, Document Count, Data, Index Rate, Search Rate, and Unassigned Shards. All indices listed are green and have 0 unassigned shards. A teal arrow points from the 'Status' column header to the 'Status' column in the table. Another teal arrow points from the 'Unassigned Shards' column header to the 'Unassigned Shards' column in the table.

Name	Status	Document Count	Data	Index Rate	Search Rate	Unassigned Shards
blogs	Green	1.6k	12.0 MB	0 /s	0 /s	0
logs_server1	Green	582.1k	411.9 MB	0 /s	0 /s	0
logs_server2	Green	584.6k	415.6 MB	0 /s	0 /s	0
logs_server3	Green	584.8k	415.3 MB	0 /s	0 /s	0



Elasticsearch Monitoring & Troubleshooting

Lesson 2

# Review - Monitoring



# Summary

- The ***Elastic Monitoring*** component uses Elasticsearch to monitor Elasticsearch
- Best practice is to use a ***dedicated cluster*** for Monitoring
- It is easier to spot issues on your cluster with the dedicated Monitoring UI

# Quiz

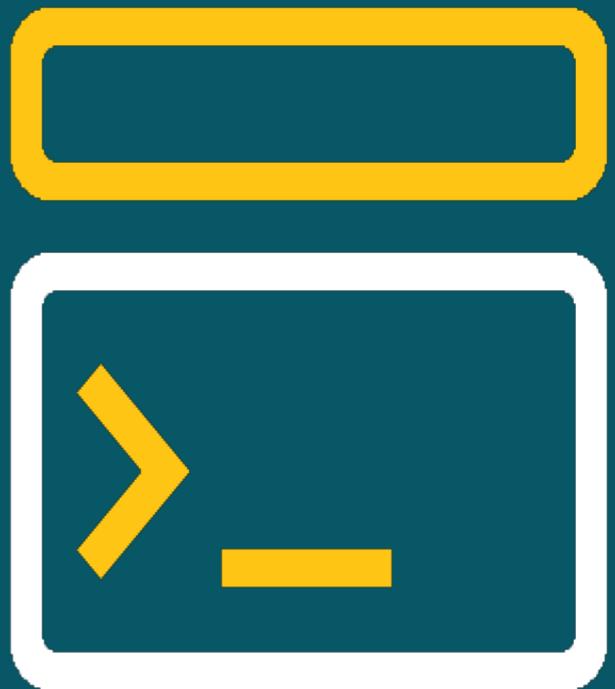
1. **True or False: Elastic Monitoring** can monitor multiple clusters.
2. What are the benefits of using a dedicated cluster for the **Monitoring** component?
3. The default **Monitoring** collection interval is \_\_\_\_\_ seconds.



Elasticsearch Monitoring & Troubleshooting

Lesson 2

# Lab - Monitoring





Elasticsearch Monitoring & Troubleshooting

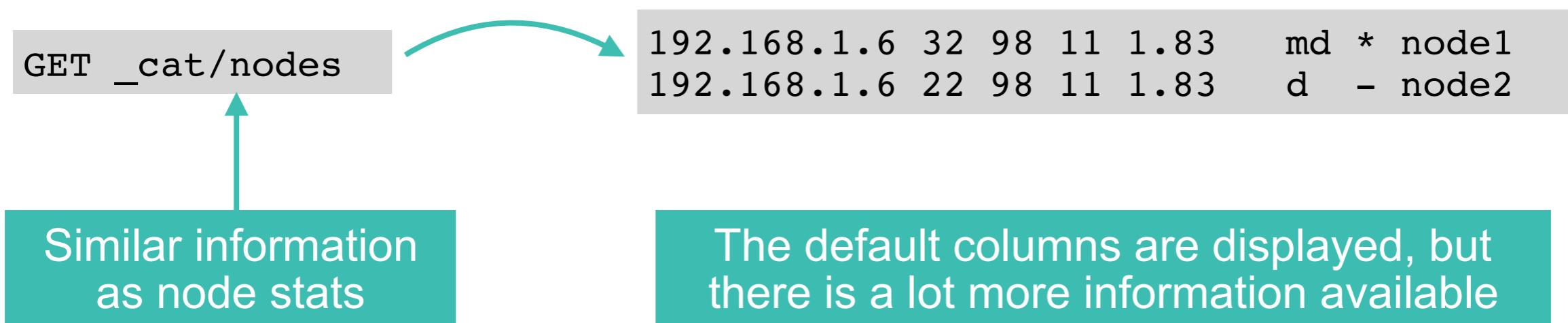
Lesson 3

# Diagnosing Performance Issues



# The cat API

- You have seen the `_cat` API throughout the labs
  - it is a wrapper around many of the Elasticsearch JSON APIs, including the stats APIs discussed so far in this chapter
  - command-line tool friendly
  - helpful for simple monitoring (e.g. Nagios)



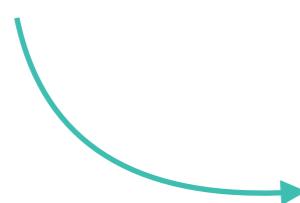
# Specifying Columns

- Add the “h” parameter to specify which columns to return
  - Use “help” to see all the available columns for a command

```
GET _cat/nodes?help
```

- use “h=” to retrieve all columns
- add the “v” parameter to display the column names in the response

```
GET _cat/nodes?v&h=name,disk.avail,search.query_total,heap.percent
```



name	disk.avail	search.query_total	heap.percent
node1	144.4gb	3800	43
node2	144.4gb	0	26

number of query operations on the node



# Pending Tasks

- The *Pending Tasks API* shows cluster-level changes that have not been executed yet:

```
GET _cluster/pending_tasks
```

```
{  
  "tasks": [  
    {  
      "insert_order": 101,  
      "priority": "URGENT",  
      "source": "create-index [my_index], cause [api]",  
      "time_in_queue_millis": 86,  
      "time_in_queue": "86ms"  
    }  
  ]  
}
```

The response is often empty because cluster-level changes are fast

# Task Management API

- The *Task Management API* shows tasks currently executing on the nodes
  - provides a nice view of how busy the cluster is

GET \_tasks

You can also use this API to cancel a task

```
{  
  "nodes": {  
    "OmWJPhToQ0iyNfz-Qd9i8g": {  
      "name": "node1",  
      "transport_address": "192.168.1.6:9300",  
      "host": "192.168.1.6",  
      "ip": "192.168.1.6:9300",  
      "roles": [  
        "master",  
        "data"  
      ],  
      "tasks": {  
        "OmWJPhToQ0iyNfz-Qd9i8g:37432": {  
          "node": "OmWJPhToQ0iyNfz-Qd9i8g",  
          "id": 37432,  
          "type": "direct",  
          "action": "cluster:monitor/tasks/lists[n]",  
          "start_time_in_millis": 1486702376488,  
          "running_time_in_nanos": 2157349,  
          "cancelable": false,  
          "parent_task_id": "OmWJPhToQ0iyNfz-Qd9i8g:37431"  
        },  
      }  
    }  
  }  
}
```

# Identifying Running Tasks

- Use the **X-Opaque-Id** header if you want to
  - track certain calls
  - or associate certain tasks with a client that started them

```
curl -i -H "X-Opaque-Id: 123456" http://server1:9200/_tasks
```



The X-Opaque-Id is also available in the **audit logs** and the **search slow logs**

```
{
  "nodes": {
    "OvD79L1lQme1hi06Ouiu7Q": {
      ...
      "tasks": {
        "OvD79L1lQme1hi06Ouiu7Q:105434": {
          "node": "OvD79L1lQme1hi06Ouiu7Q",
          "id": 105434,
          "type": "netty",
          "action": "cluster:monitor/tasks/lists[n]",
          ...
          "parent_task_id": "N7SB1AGdS9ioU5r67LJNRg:83863",
          "headers": {
            "X-Opaque-Id": "123456"
          }
        }
      }
    }
  }
}
```

# Thread Pool Queues

- Many cluster tasks (bulk, index, get, search, etc.) use thread pools to improve performance
  - these thread pools are fronted by queues
  - when a queue is full, 429 status code is returned

```
GET _nodes/thread_pool
```

```
...  
"write": {  
  "type": "fixed",  
  "size": 4,  
  "queue_size": 200  
}  
...
```

```
GET _nodes/stats/thread_pool
```

```
...  
"write": {  
  "threads": 8,  
  "queue": 0,  
  "active": 0,  
  "rejected": 0,  
  "largest": 8,  
  "completed": 177  
}  
...
```

# Thread Pool Queues

- The `_cat` API provides a nice view of the thread pools:

```
GET _cat/thread_pool?v
```

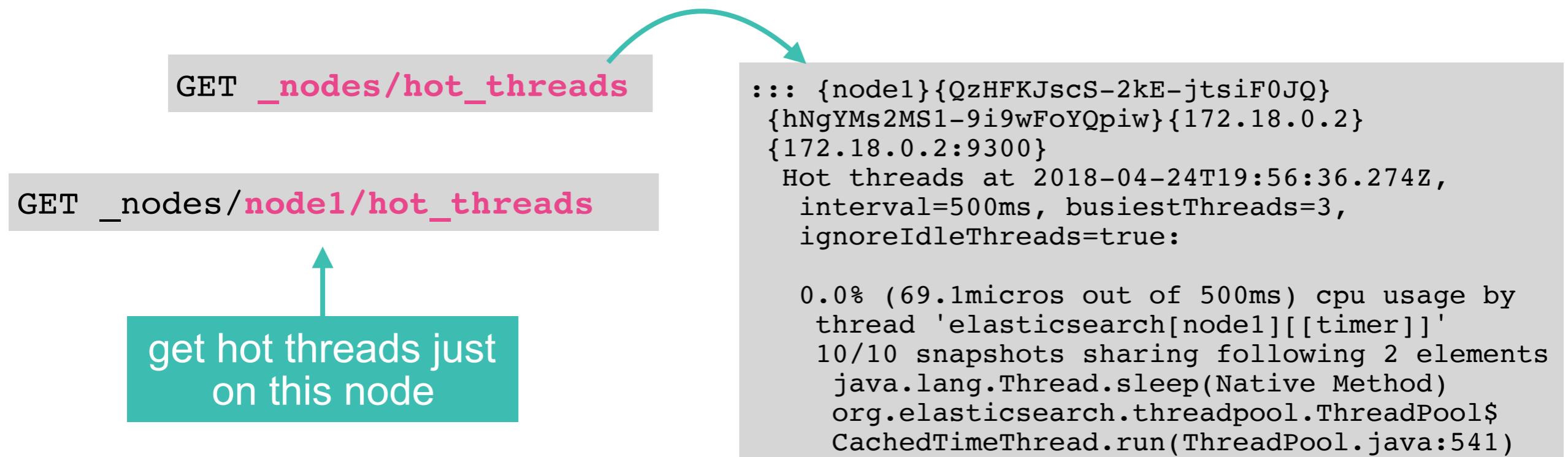


node_name	name	active	queue	rejected
node1	analyze	0	0	0
node1	ccr	0	0	0
node1	fetch_shard_started	0	0	0
node1	fetch_shard_store	0	0	0
node1	flush	0	0	0
node1	force_merge	0	0	0
node1	generic	0	0	0
node1	get	0	0	0
node1	listener	0	0	0
node1	management	1	0	0
node1	ml_autodetect	0	0	0
node1	ml_datafeed	0	0	0
node1	ml_utility	0	0	0
node1	refresh	0	0	0
node1	rollup_indexing	0	0	0
node1	search	0	0	0
node1	search_throttled	0	0	0
node1	security-token-key	0	0	0
node1	snapshot	0	0	0
node1	warmer	0	0	0
node1	watcher	0	0	0
node1	write	0	0	0

- A full queue may be good or bad (“It depends!”)
  - OK if bulk indexing is faster than ES can handle
  - bad if search queue is full

# The hot\_threads API

- The *Nodes hot\_threads API* allows you get to view the current hot threads on each node
  - invoke it on all nodes or a specific node



# The Indexing Slow Log

- The ***Indexing Slow Log*** captures information about long-running index operations into a log file
- Logs ***indexing events*** that take longer than configured thresholds
  - already configured in **log4j2.properties**

```
PUT my_index/_settings
{
  "index.indexing.slowlog" : {
    "threshold.index" : {
      "warn" : "10s",
      "info" : "5s",
      "debug" : "2s",
      "trace" : "0s"
    },
    "level" : "trace",
    "source" : 1000
  }
}
```

Request index slow log level

Current index slow log level

The first 1,000 characters of the document's source will be logged

# The Search Slow Log

- The **Search Slow Log** captures information about *long-running searches* (query and fetch phases) into a log file
  - the log file is already configured in **log4j2.properties**, but disabled by default
  - useful, but can be limited since it logs per shard
  - packetbeat may be a better solution

```
PUT my_index/_settings
{
  "index.search.slowlog": {
    "threshold": {
      "query": {
        "info": "5s"
      },
      "fetch": {
        "info": "800ms"
      }
    },
    "level": "info"
  }
}
```

This example sets level to info and defines thresholds

# The Profile API

- Elasticsearch has a powerful **Profile API** which can be used to inspect and analyze your search queries
  - just set “profile” to true in your query:

```
GET logs_server*/_search
{
  "size": 0,
  "profile": "true", ← Enable profiling for this search
  "aggs": {
    "top_cities": {
      "terms": {
        "field": "geoip.city_name.keyword",
        "size": 20
      },
      "aggs": {
        "top_urls": {
          "significant_text": {
            "field": "originalUrl.keyword",
            "size": 3
          }
        }
      }
    }
  }
}
```

# The Profiler Response...

- ...is hard to read (it is a lot of JSON):

```
"profile" : {
  "shards" : [
    {
      "id" : "[N7SB1AGdS9ioU5r67LJNRg][logs_server1][0]",
      "searches" : [
        {
          "query" : [
            {
              "type" : "MatchAllDocsQuery",
              "description" : "*:*",
              "time_in_nanos" : 25005766,
              "breakdown" : {
                "set_min_competitive_score_count" : 0,
                "match_count" : 0,
                "shallow_advance_count" : 0,
                "set_min_competitive_score" : 0,
                "next_doc" : 24170683,
                "match" : 0,
                "next_doc_count" : 582078,
                "score_count" : 0,
                "compute_max_score_count" : 0,
                "compute_max_score" : 0,
                "advance" : 0,
                "advance_count" : 0,
                "score" : 0,
                "build_scorer_count" : 30,
                "create_weight" : 13177,
                "shallow_advance" : 0,
                "create_weight_count" : 1,
                "build_scorer" : 239797
              }
            }
          ],
          ...
        }
      ]
    }
  ]
}
```

# The Search Profiler

- **Search Profiler** is a tool that transforms the JSON output into a visualization that is easy to navigate:

You can also copy-and-paste the output of a profiled query into this field

# The Query Profile Tab

- The query times are shown per shard, along with the generated Lucene query

Query Profile    Aggregation Profile

**Index:** logs\_server2  
  > [N7SBIAGdS9ioU5r67LJNRg][0]

**Index:** logs\_server1  
  < [OvD79L1IQme1hi06Ouiu7Q][0]  
    Type and description  
      ● MatchAllDocsQuery  
        \*:\*

**Index:** logs\_server3  
  > [OvD79L1IQme1hi06Ouiu7Q][0]

logs_server1	[OvD79L1IQme1hi06Ouiu7Q][0]
Type	MatchAllDocsQuery
Description	* :*
Total Time	22.976ms
Self Time	22.976ms
Timing Breakdown	
next_doc	21.8ms    97.5%
build_scorer	538.3µs    2.4%
create_weight	19.1µs    0.1%
advance	0.0ns    0.0%
compute_max_score	0.0ns    0.0%
match	0.0ns    0.0%
score	0.0ns    0.0%
set_min_competitive_score	0.0ns    0.0%
shallow_advance	0.0ns    0.0%

# The Aggregation Profile Tab

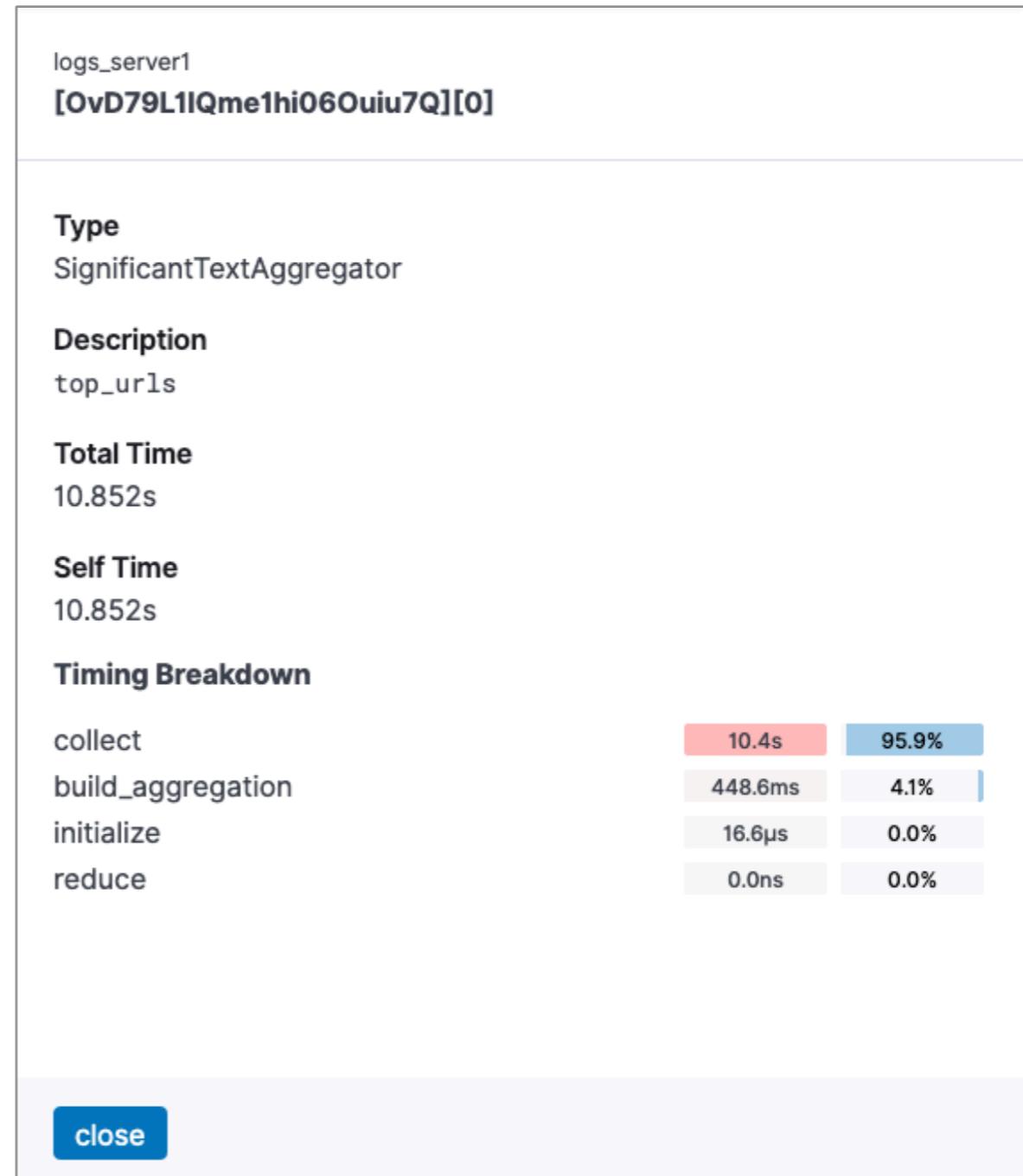
- Shows query times per shard for each aggregation that was executed

Query Profile    Aggregation Profile

**Index:** logs\_server2  
  > [N7SBIAGdS9ioU5r67LJNRg][0]

**Index:** logs\_server1  
  < [OvD79L1IQme1hi06Ouiu7Q][0]  
    Type and description  
    < GlobalOrdinalsStringTermsAggregator  
      top\_cities  
        • SignificantTextAggregator  
          top\_urls

**Index:** logs\_server3  
  > [OvD79L1IQme1hi06Ouiu7Q][0]



# Circuit Breakers

- Prevent operations from causing an **OutOfMemoryError**
  - by specifying a limit on how much memory can be used
- Elasticsearch contains multiple circuit breakers
  - parent circuit breaker
  - field data circuit breaker
  - request circuit breaker
  - in flight requests circuit breaker
  - accounting requests circuit breaker
  - script compilation circuit breaker



Elasticsearch Monitoring & Troubleshooting

Lesson 3

# Review - Diagnosing Performance Issues



# Summary

- You can use the tasks API to see cluster-level changes that have not been executed yet and the **X-Opaque-Id** header to track certain tasks
- Slow logs, thread pools, and hot threads can help you diagnose performance issues
- You can profile your search queries and aggregations to see where they are spending time
- Elasticsearch sets several circuit breakers to prevent out of memory errors

# Quiz

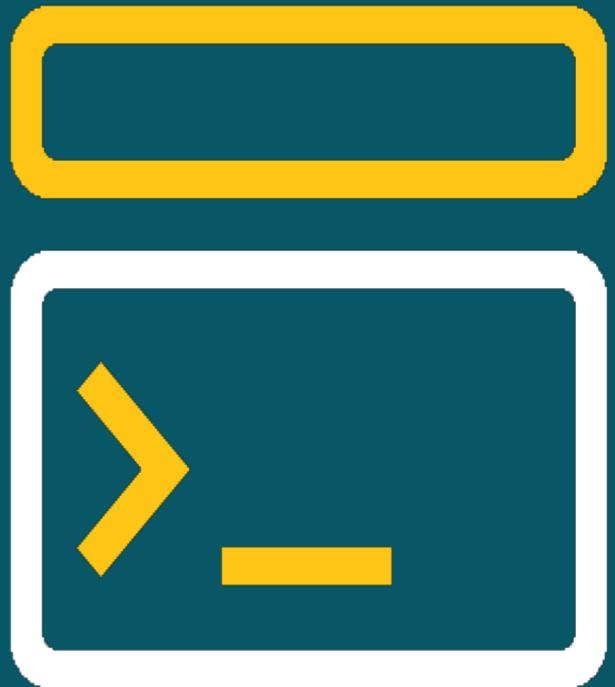
1. How would you check if index thread pool queue is full?
2. **True or False:** The only way to profile your query is setting "profile" to true and inspecting the returned JSON.
3. **True or False:** Circuit breakers prevent operations from going out of memory.



Elasticsearch Monitoring & Troubleshooting

Lesson 3

# Lab - Diagnosing Performance Issues

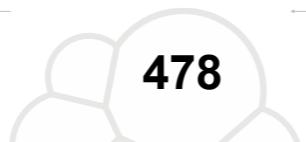


# Conclusions



# Resources

- <https://www.elastic.co/learn>
  - <https://www.elastic.co/training>
  - <https://www.elastic.co/community>
  - <https://www.elastic.co/docs>
- <https://discuss.elastic.co>
  - <https://ela.st/training-forum>



# Elastic Training

## Empowering Your People

### Immersive Learning

Lab-based exercises and knowledge checks to help master new skills

### Solution-based Curriculum

Real-world examples and common use cases

### Experienced Instructors

Expertly trained and deeply rooted in everything Elastic

### Performance-based Certification

Apply practical knowledge to real-world use cases, in real-time

FOUNDATION



SPECIALIZATIONS



# Engineer II

- 1 Elasticsearch Data Modeling
- 2 Elasticsearch Data Processing
- 3 Elasticsearch From Dev to Production
- 4 Elasticsearch Cluster Deployment
- 5 Elasticsearch Nodes and Index Management
- 6 Elasticsearch Advanced Tips and Tricks

# Elastic Consulting Services

## ACCELERATING YOUR PROJECT SUCCESS

**FLEXIBLE SCOPING**  
Shifts resource as your requirements change

**PHASE-BASED PACKAGES**  
Align to project milestones at any stage in your journey

**GLOBAL CAPABILITY**  
Provide expert, trusted services worldwide

**EXPERT ADVISORS**  
Understand your specific use cases

**PROJECT GUIDANCE**  
Ensures your goals and accelerate timelines

# Thank You!

Please complete the online survey.

# Quiz Answers



# Elasticsearch Fundamentals (Lesson 1)

1. Beats, Logstash, Elasticsearch, and Kibana
2. True
3. Scalable and easy to use

# Elasticsearch Fundamentals (Lesson 2)

1. The three configuration files are:

- elasticsearch.yml
- jvm.options
- log4j2.properties

2. False

3. Use the **node.name** property

# Elasticsearch Fundamentals (Lesson 3)

1. By default, this is true, but this behavior can be turned off (recommended for production clusters)
2. The existing document is deleted, and the new document is indexed
3. True

# Elasticsearch Fundamentals (Lesson 4)

1. Static and Time Series
2. Queries and Aggregations
3. The default is 10 hits

# Elasticsearch Queries (Lesson 1)

1. False, you are improving precision.
2. Using `minimum_should_match`. You can set it to 2, 3, or 4.
3. False. It will have very similar importance.

# Elasticsearch Queries (Lesson 2)

1. Multiple match terms use “and” or “or”, while multiple terms in match\_phrase are “and” and position matters
2. slop
3. Yes. The fuzziness value is applied per analyzed term.

# Elasticsearch Queries (Lesson 3)

1. You could use a bool query with a must match query for “scripting” in the content field and a filter for the “Engineering” category
2. False. A filter clause has no effect on a document’s score
3. False. It will return only documents that contain performance in the title field. There is a default minimum\_should\_match of 1 when there is a single should clause.

# Elasticsearch Queries (Lesson 4)

1. From and Size.
2. False. You can define your own sort criteria.
3. The one in the left is analyzed and will return any categories that contain user OR stories. The one in the right is querying the keyword field, which works as an exact match (not analyzed).

# Elasticsearch Aggregations (Lesson 1)

1. True
2. Percentile aggregation
3. Cardinality

# Elasticsearch Aggregations (Lesson 2)

1. Terms
2. How many blogs were released per month in 2017?

# Elasticsearch Aggregations (Lesson 3)

1. date\_histogram + percentiles
2. date\_histogram + terms
3. terms + significant\_text

# Chapter 1 Quiz Answers

1. true
2. long
3. false, you cannot change the mapping of a field

# Chapter 2 Quiz Answers

1. false - because text fields are analyzed, and the standard analyzer lowercases the documents as well as the queries, full text searches are case-insensitive
2. twice: once as text, once as keyword
3. true

# Chapter 3 Quiz Answers

1. false - you should use a keyword field
2. true - because you have disabled the inverted index
3. false - mapping parameters do not influence the `_source`

# Chapter 4 Quiz Answers

1. format
2. false - mapping parameters do not influence the `_source`. All this does is create a new inverted index with the values of the field that have `copy_to` defined
3. true - first you index a sample document, then you get the created mapping, update to your needs and create the new index

# Elasticsearch Nodes and Shards (Lesson 1)

1. The new node pings the servers listed in the `discovery.seed_providers` property
2. Set `cluster.initial_master_nodes` with a list of initial master-eligible nodes
3. False. Usually it is better to have an odd number of master-eligible nodes to have quorum during master election

# Elasticsearch Nodes and Shards (Lesson 2)

1. Set node.data to true, and node.master, node.ingest and node.ml to false
2. False. A 2-node cluster have high risk of being unavailable, so a 3-5 node cluster would be a better option
3. True

# Elasticsearch Nodes and Shards (Lesson 3)

1. 12. 4 primary shards, and 8 replicas
2. False
3. False. Yellow means that at least one replica shard is missing, but all primaries are allocated, so there are no missing documents.

# Elasticsearch Nodes and Shards (Lesson 4)

1. True
2. True
3. The response is only returned after the document is searchable. When the UIX returns a search response that should include the document as a result of a write operation.

# Elasticsearch Troubleshooting (Lesson 1)

1. You can have shards that failed during the query, yet the response code will show the query as a success
2. True
3. False

# Elasticsearch Troubleshooting (Lesson 2)

1. True, but only the commercial version does. The free version can only monitor one
2. If a cluster fails, you will be able to view its history and perhaps diagnose the issue. There are also performance and security benefits.
3. 10 seconds

# Elasticsearch Troubleshooting (Lesson 3)

1. Look at the thread pool queues either with `_nodes/thread_pool` or `_cat/thread_pool`
2. False. You can also use the Kibana UI
3. True

# Elasticsearch Engineer I

Course: Elasticsearch Engineer I

© 2015-2019 Elasticsearch BV. All rights reserved. Decompiling, copying, publishing and/or distribution without written consent of Elasticsearch BV is strictly prohibited.