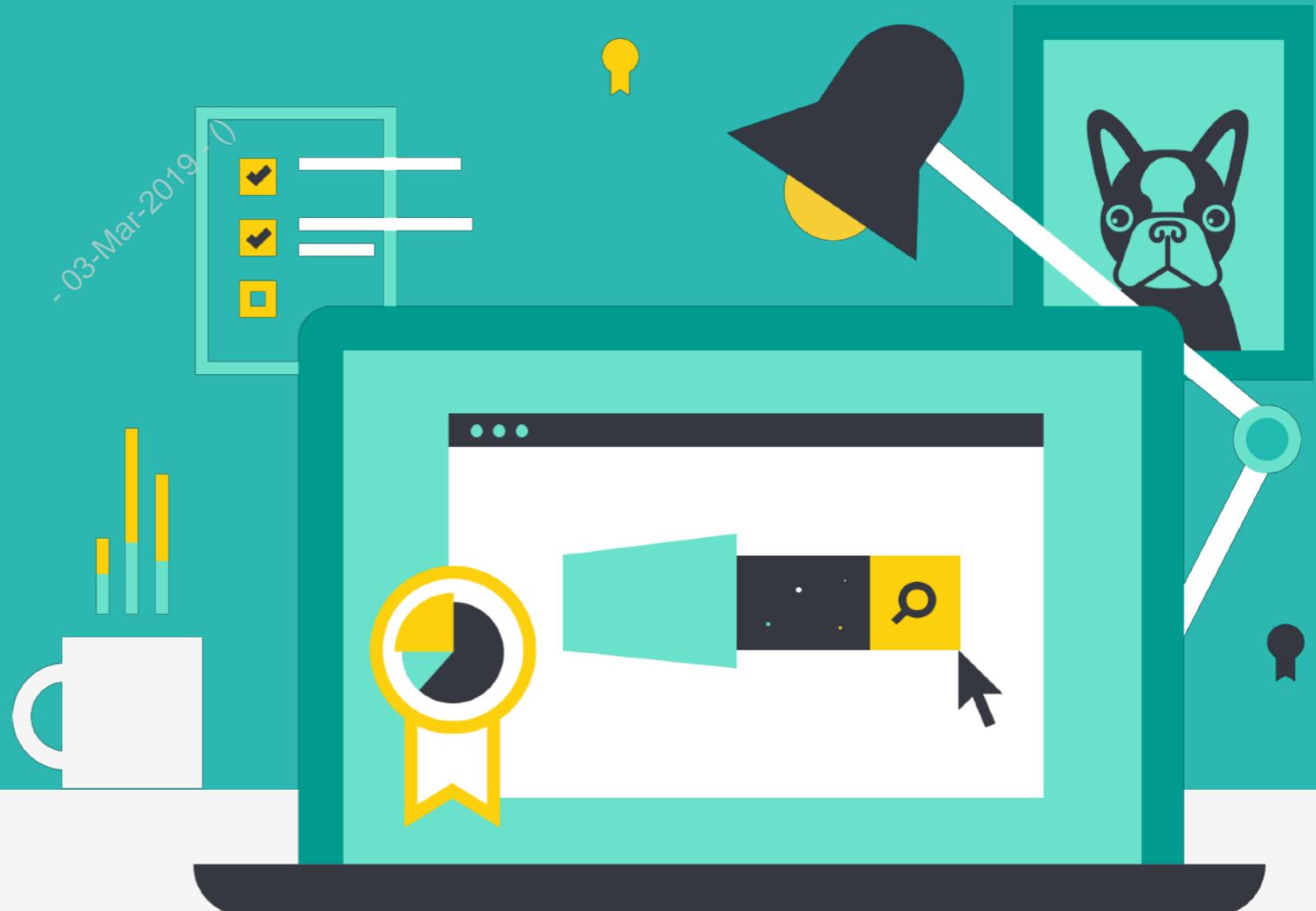




Elasticsearch Engineer I

An Elastic Training Course



6.5.1

elastic.co/training

6.x.x

Elasticsearch Engineer I

Course: Elasticsearch Engineer I

Version 6.5.1

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

~03-Mar-2019~0

Welcome to This Virtual Training

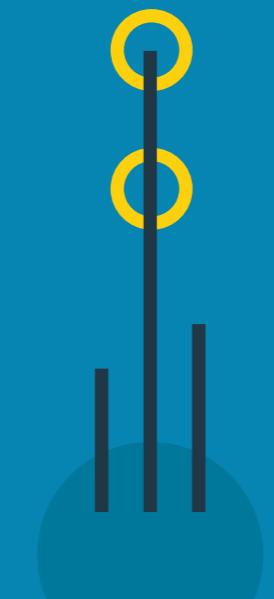
- We will start momentarily
- The training will start with an audio/video test, to make sure that everyone can hear and see the instructors
- To prevent any audio/video issues, please:
 - disable any ad blockers or script blockers
 - use a supported web browser: Chrome or Firefox
- In case of problems, 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 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

Agenda and Introductions

,03-Mar-2019,0



About This Training

- Environment
- Introductions
- Agenda...

~03-Mar-2019~0

Course Agenda

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices

03-Mar-2019 - 0

Chapter 1

Elastic Stack Overview

03-Mar-2019 - 0

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- The Story of the Elastic Stack
- Why Elastic Stack?
- Elastic Stack Components
- Our Dataset

~03-Mar-2019~0

The Story of the Elastic Stack

.03-Mar-2019 - 0



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 (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



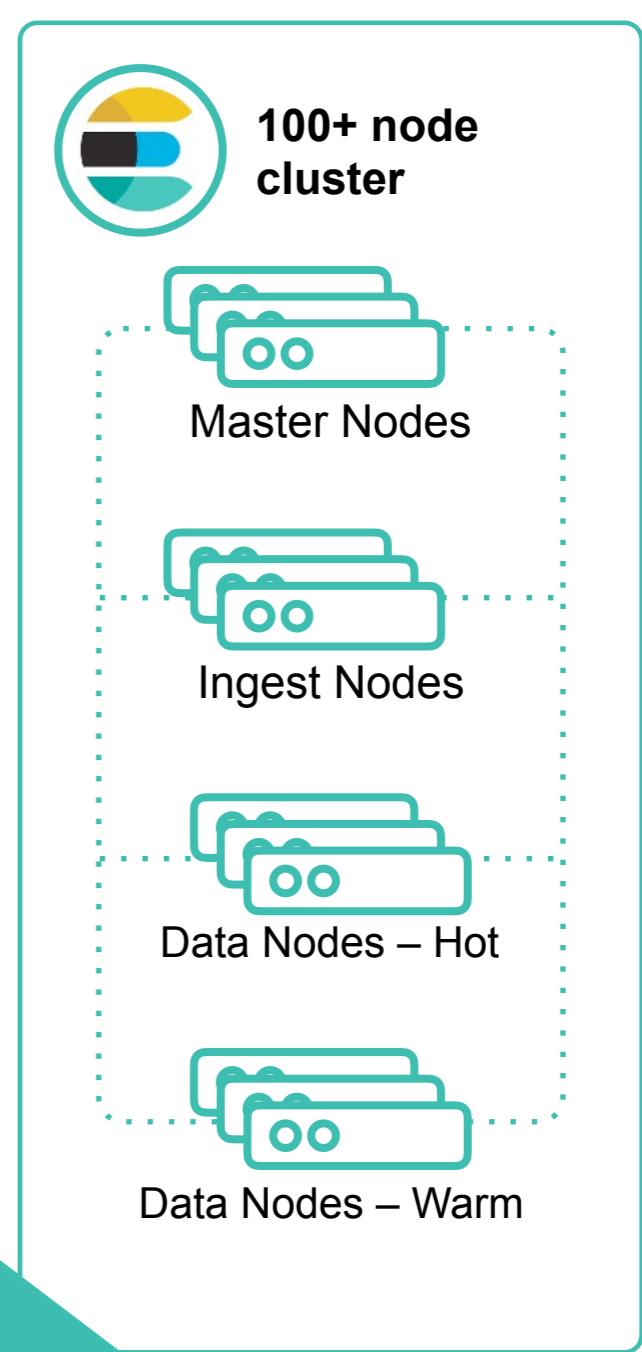
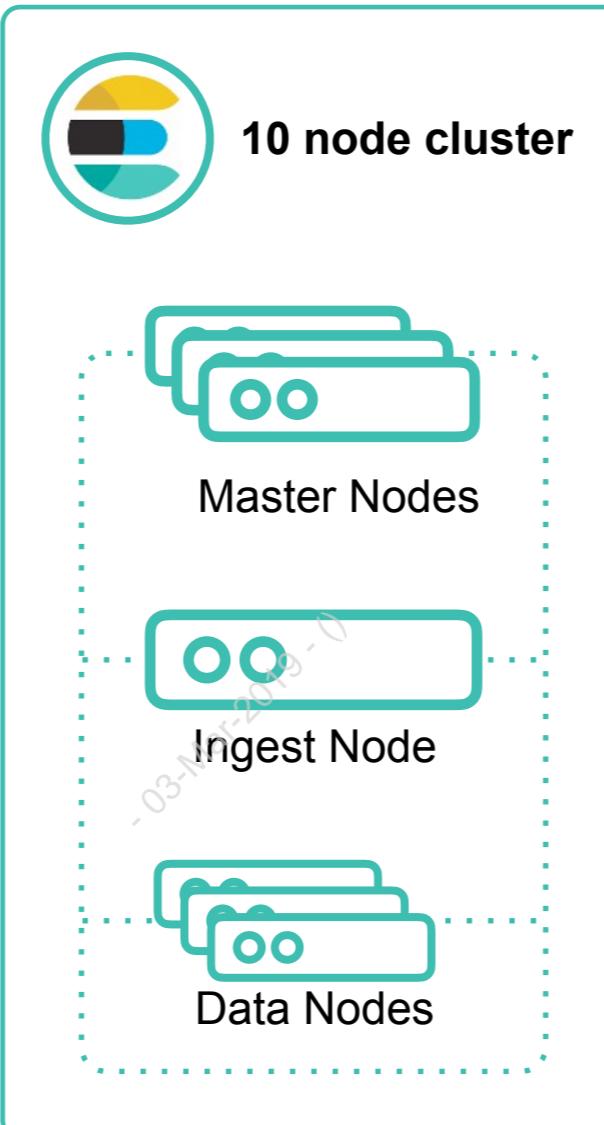
1. Distributed search:

- Elasticsearch is distributed and scales horizontally:



A **node** is an instance of Elasticsearch

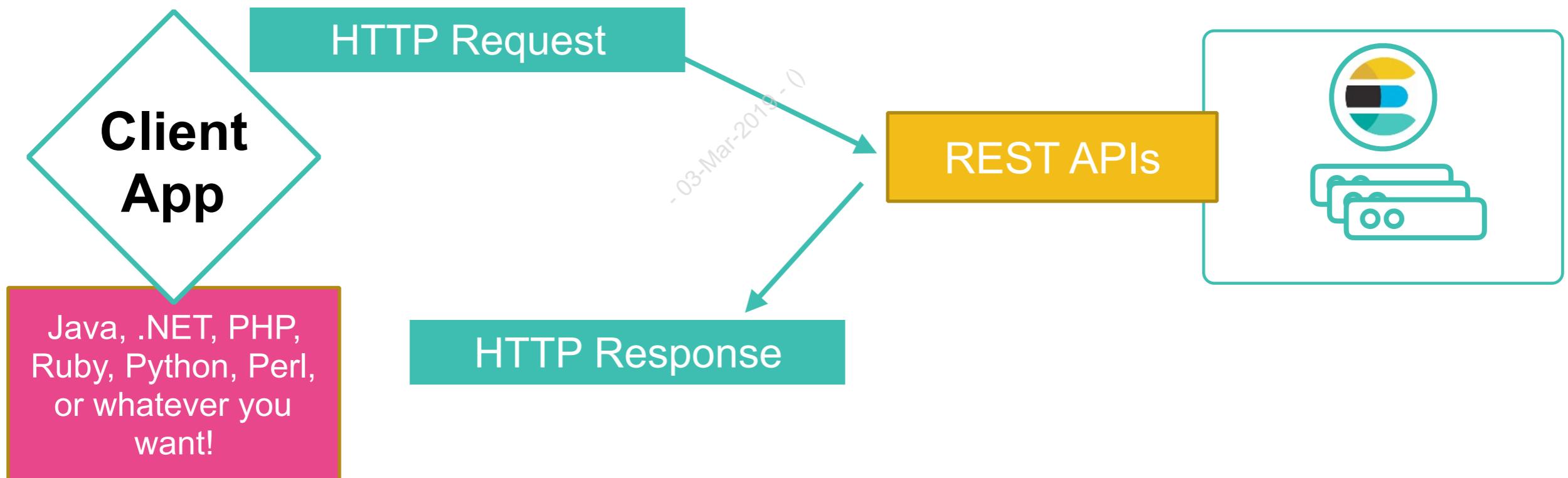
A **cluster** is a collection of Elasticsearch nodes



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



First version of **Elasticsearch** (0.4) released

Elasticsearch becomes a company

Kibana and **Logstash** OS projects join

Elasticsearch 1.0 GA

Elastic Cloud acquired

Beats OS joins

Elasticsearch 2.0 GA

Elastic Stack

Machine Learning acquired

5.0 GA

APM acquired

6.0 GA

Swiftype acquired

2010

2012

2013

2014

2015

2016

2017

2018



Why Elastic Stack?

„03-Mar-2019“



You Know, for Search!

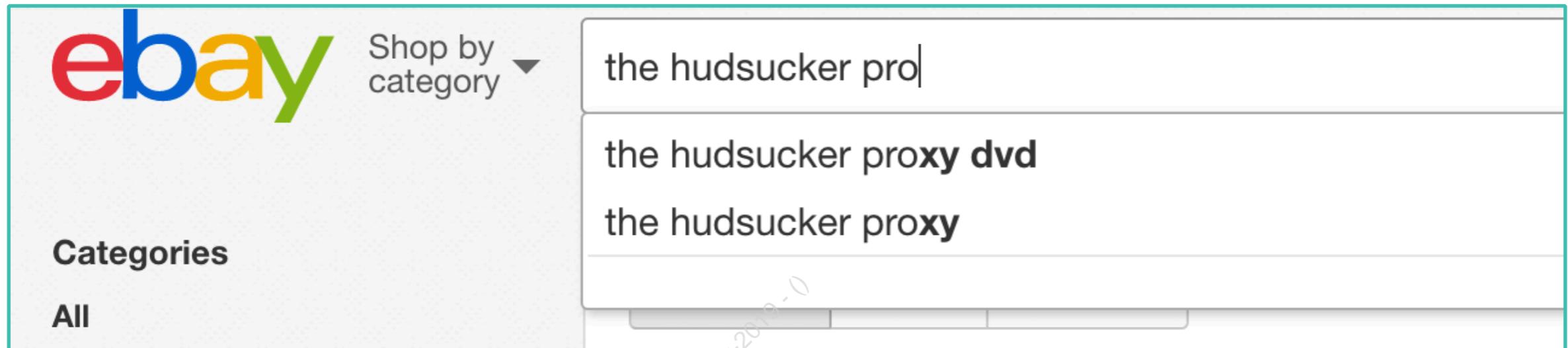
- ...and *logging*,
- and *metrics*,
- and ***security analytics***,
- and lots more!

~03-Mar-2019~0



Search

- **Full-text search** is where it all began:



eBay uses Elasticsearch
to search 800 million
listings in sub-seconds

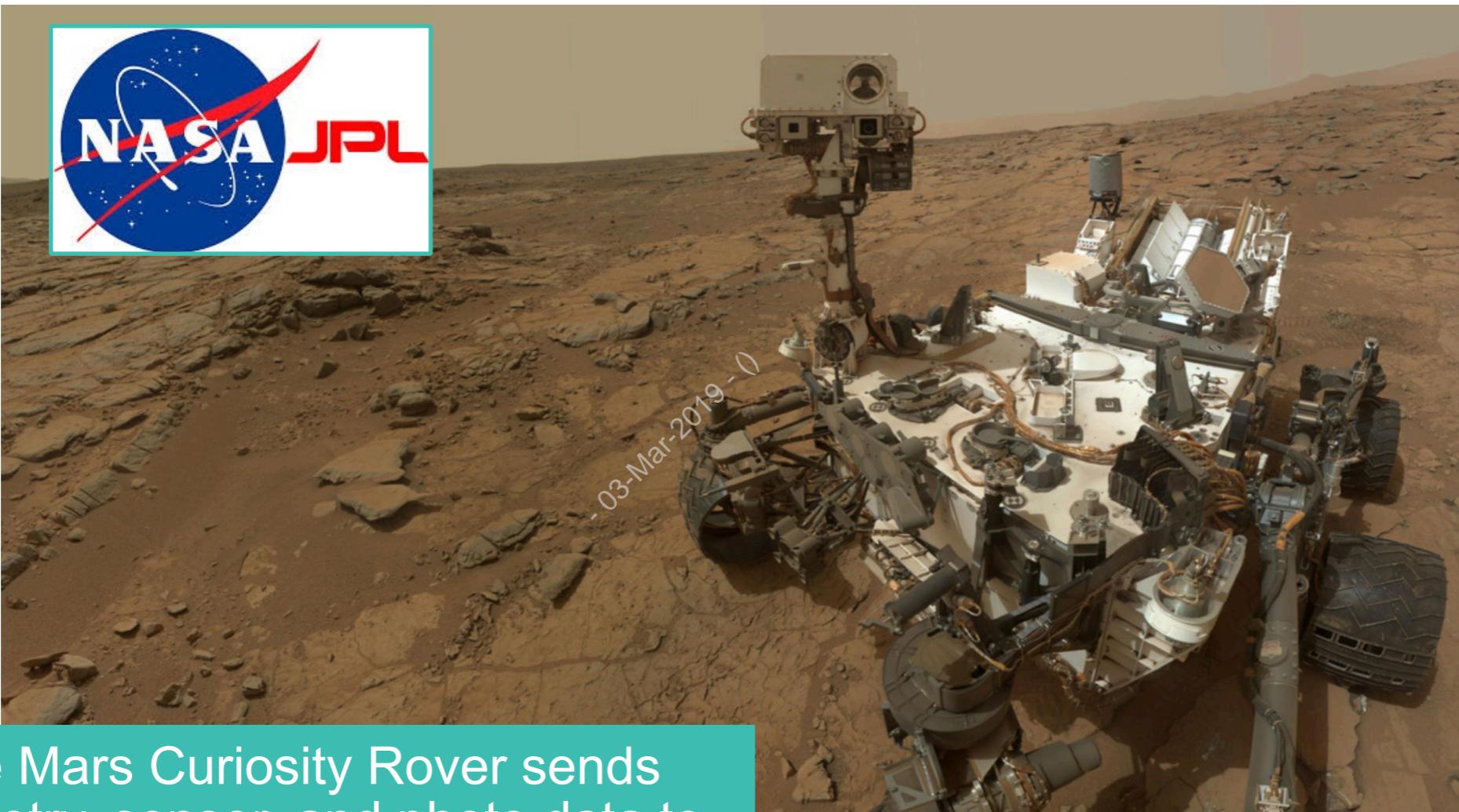
Logging

- A popular use case for the Elastic Stack is storing and analyzing *log data*:



Metrics

- Using the Elastic Stack to analyze all types of *metric data*, from CPU usage to the weather on Mars:



The Mars Curiosity Rover sends telemetry, sensor, and photo data to Elasticsearch for real-time analysis



Business Analytics

- Using the Elastic Stack to *analyze business needs* and build custom applications
 - from studying purchasing patterns, to improving medical care, to matching people together

It's a Match!



03-Mar-2019 - 0

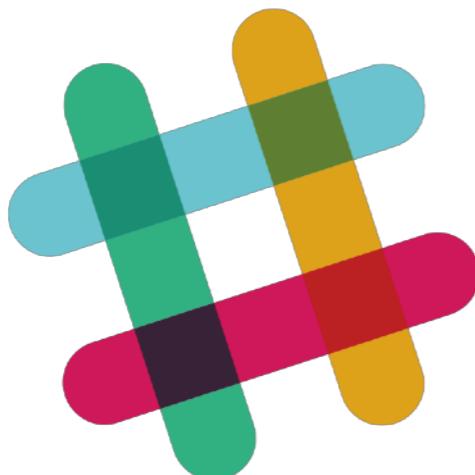


Elasticsearch is at the core
of Tinder's framework



Security Analytics

- Using the Elastic Stack to *analyze security issues and threats*:



Slack uses Elasticsearch to monitor malicious activity



Elastic Stack Components

-03-Mar-2019-0



Elastic Stack Components



Beats



Elasticsearch



Kibana



Logstash

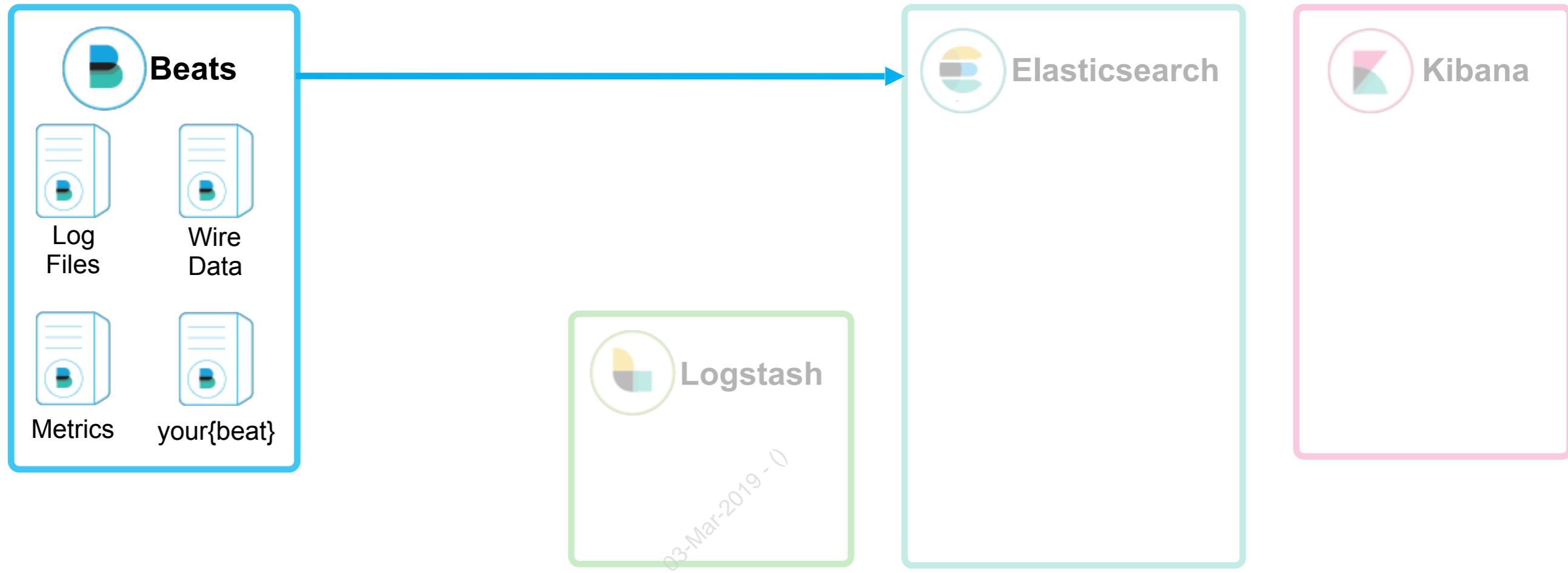
03-Mar-2019 - 0



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



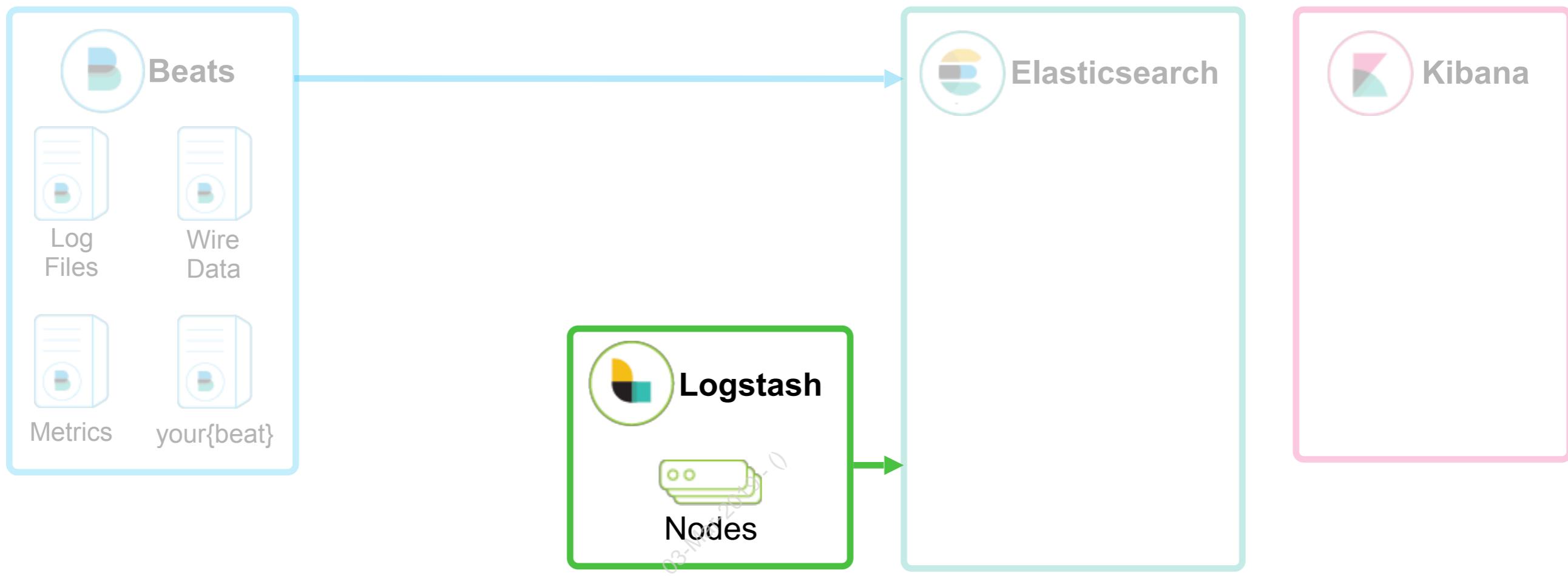
Beats



- **Beats** are single-purpose data shippers
 - e.g. **Filebeat**, **Metricbeat**, **Packetbeat**, **Winlogbeat**, etc.
 - lightweight agents that send data from your machines to Logstash or Elasticsearch



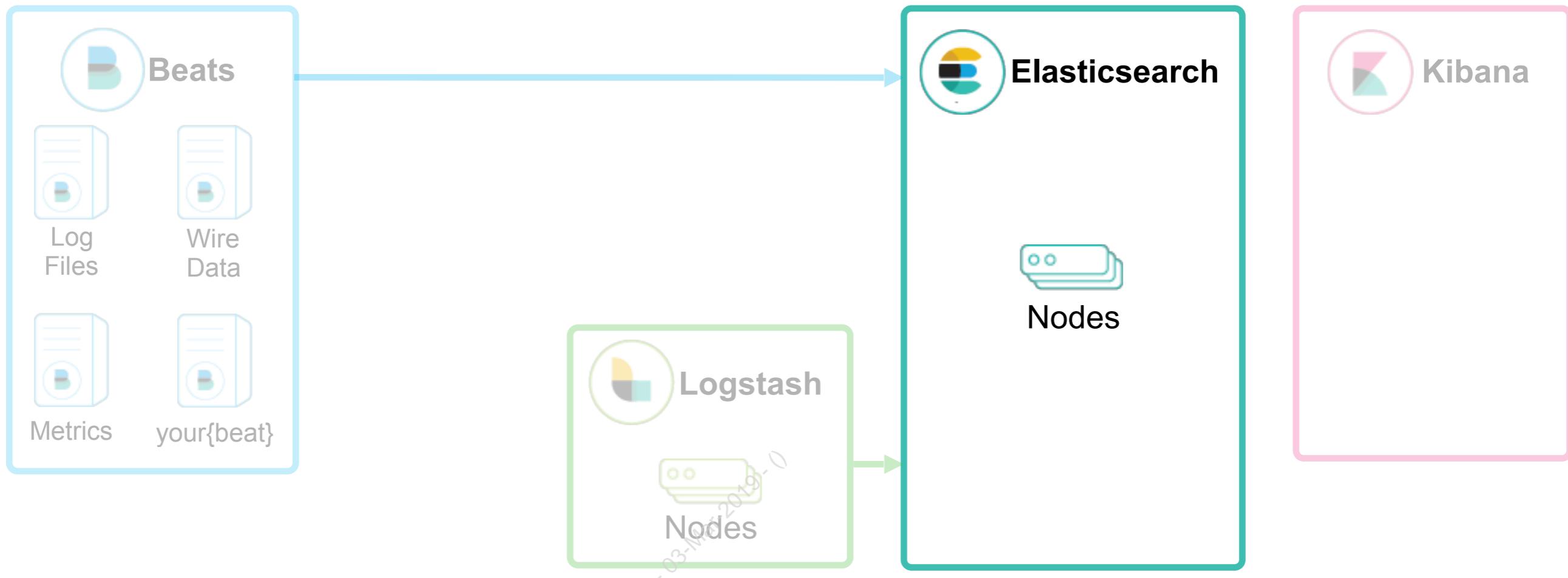
Logstash



- **Logstash** is a server-side data processing pipeline
 - ingest data from a multitude of sources simultaneously
 - parse, transform, and prepare your data for ingestion



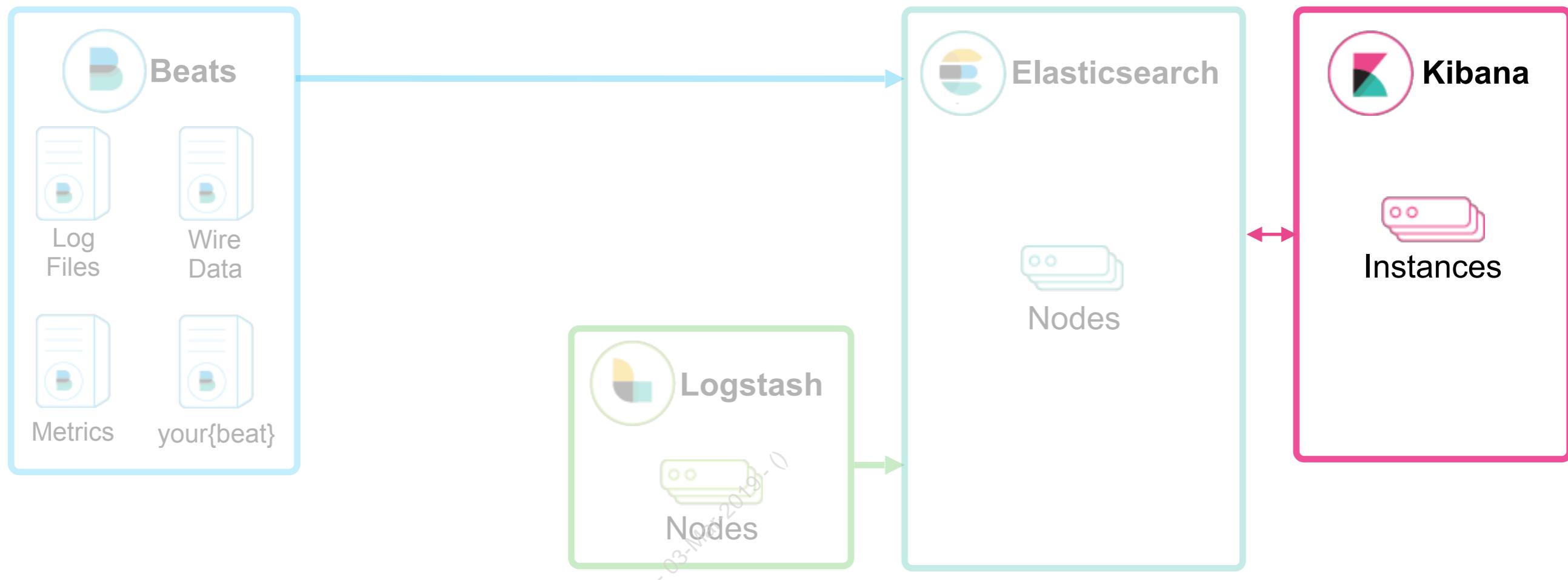
Elasticsearch



- ***Elasticsearch*** is a distributed, RESTful search and analytics engine that centrally stores your data
 - a ***node*** is a single instance of Elasticsearch
 - a ***cluster*** is a collection of one or more nodes



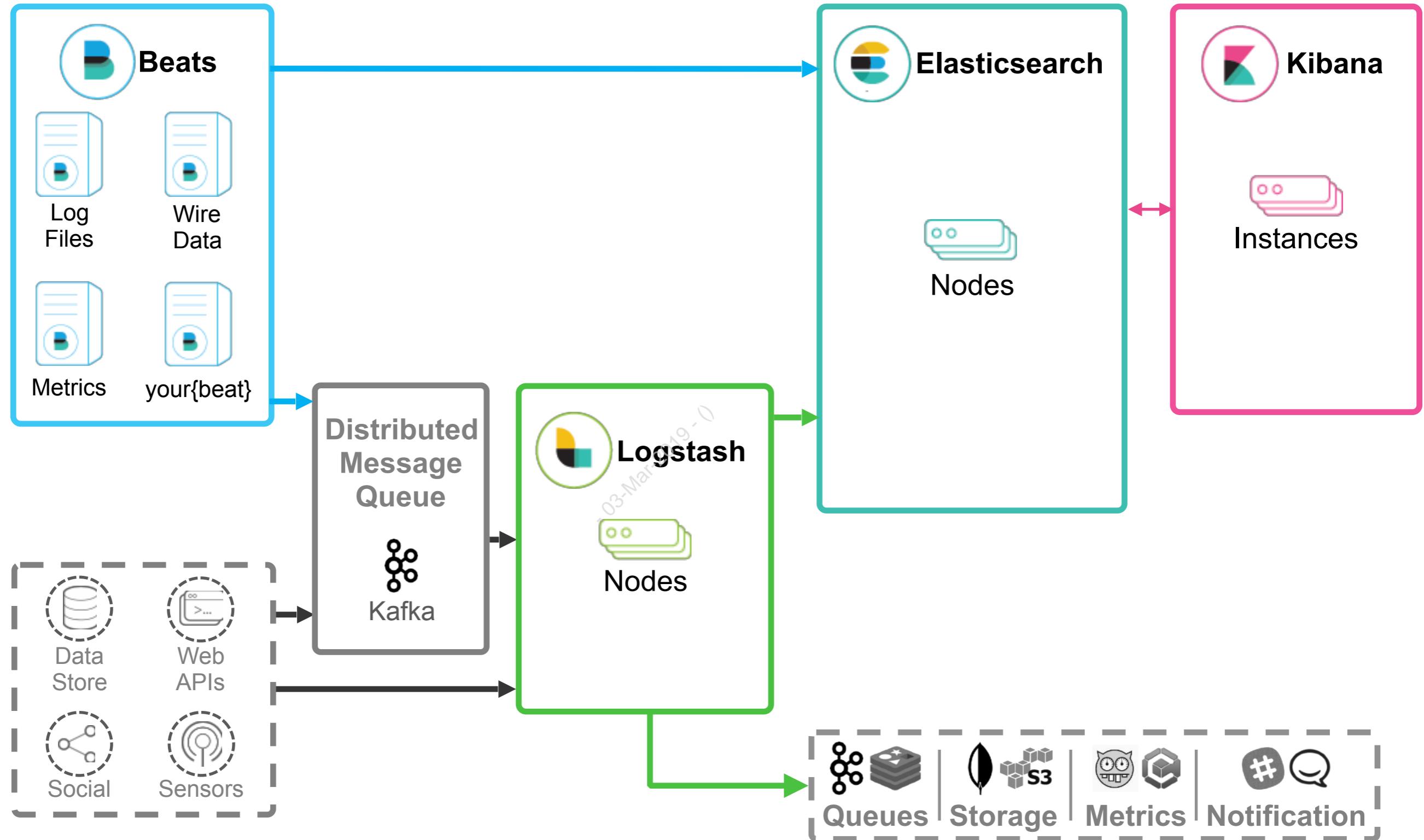
Kibana



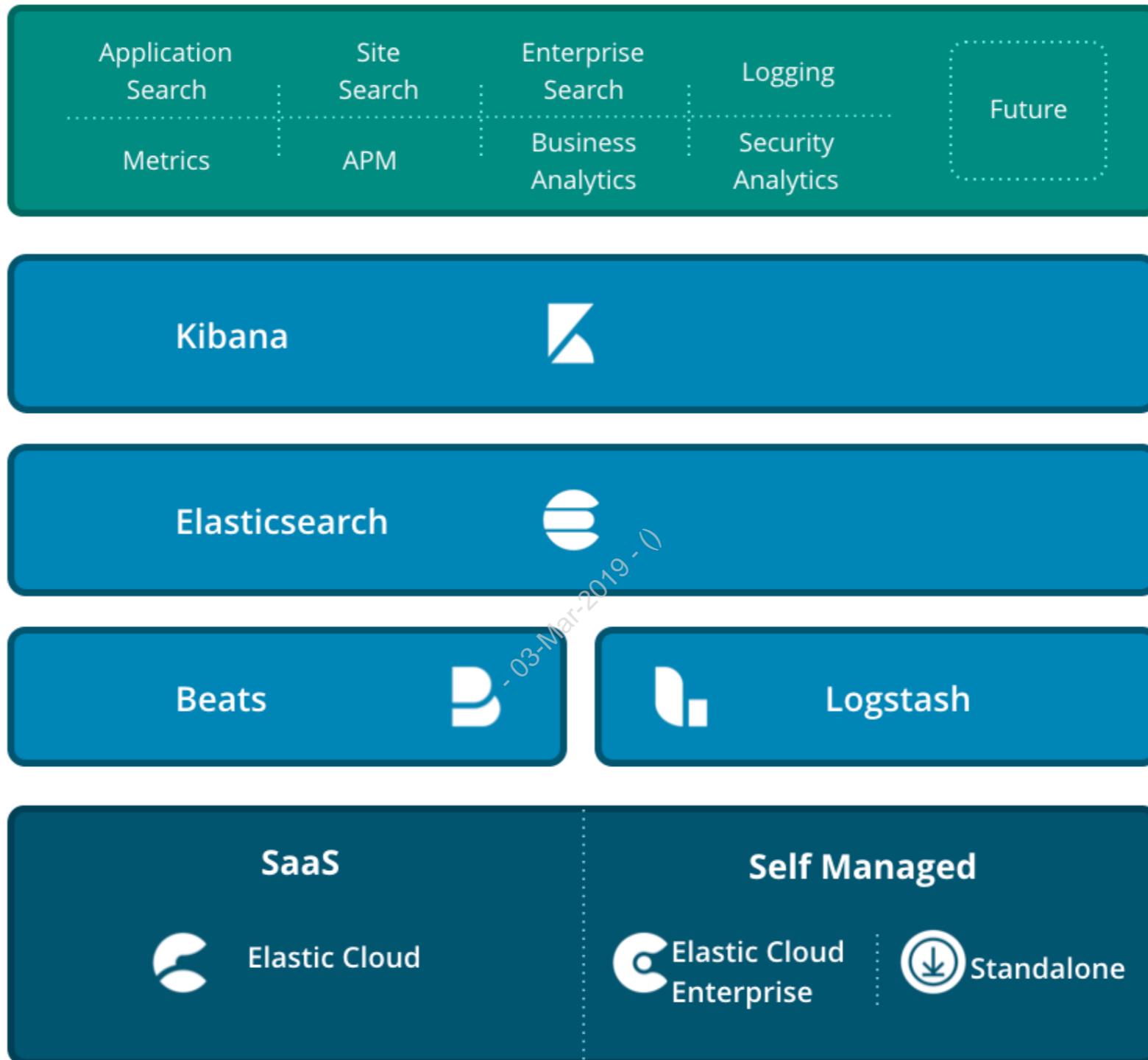
- ***Kibana*** is an analytics and visualization platform
 - manage the stack
 - search, view and interact with your Elasticsearch data
 - visualize your data in a variety of charts, tables, and maps



The Elastic Journey of an Event



Elastic



Our Dataset

~03-Mar-2019~

Static Data vs. Time Series Data

- In general, we can categorize most data in our customers' 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?

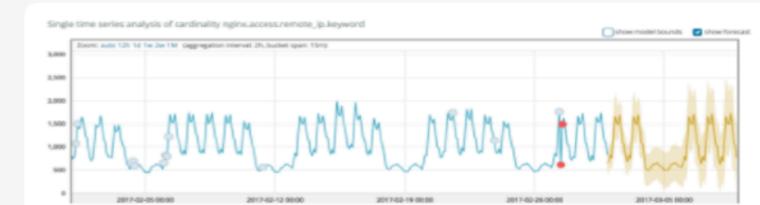


12 JANUARY 2018 **ENGINEERING**

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

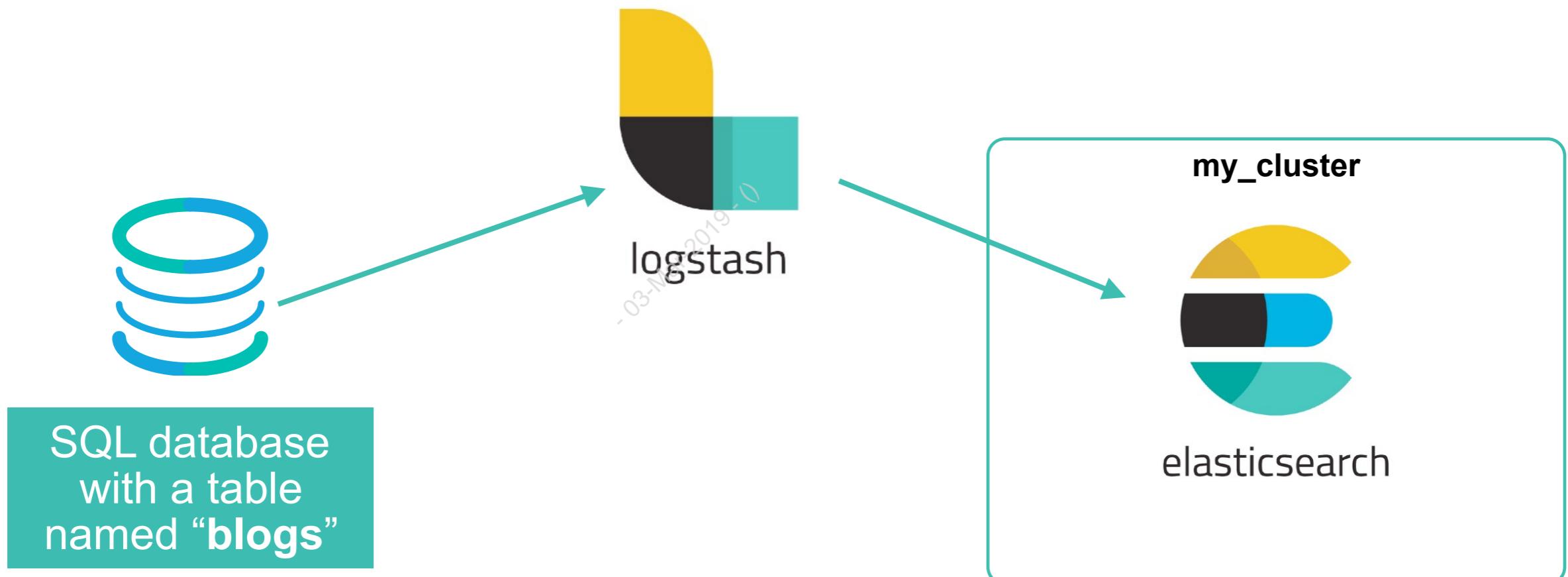
By Hendrik Muhs • Thomas Grabowski

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

logstash

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

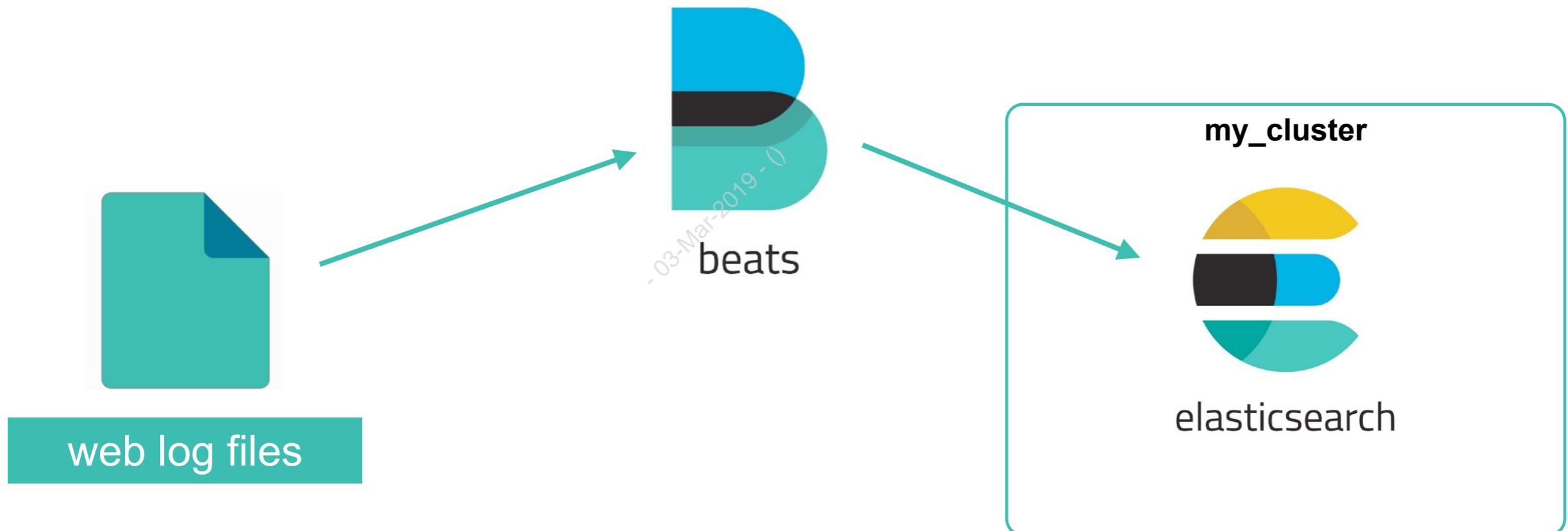
- For time series, we are going to use the web access log files from elastic.co/blog:

```
{  
    "@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"  
    }  
}
```



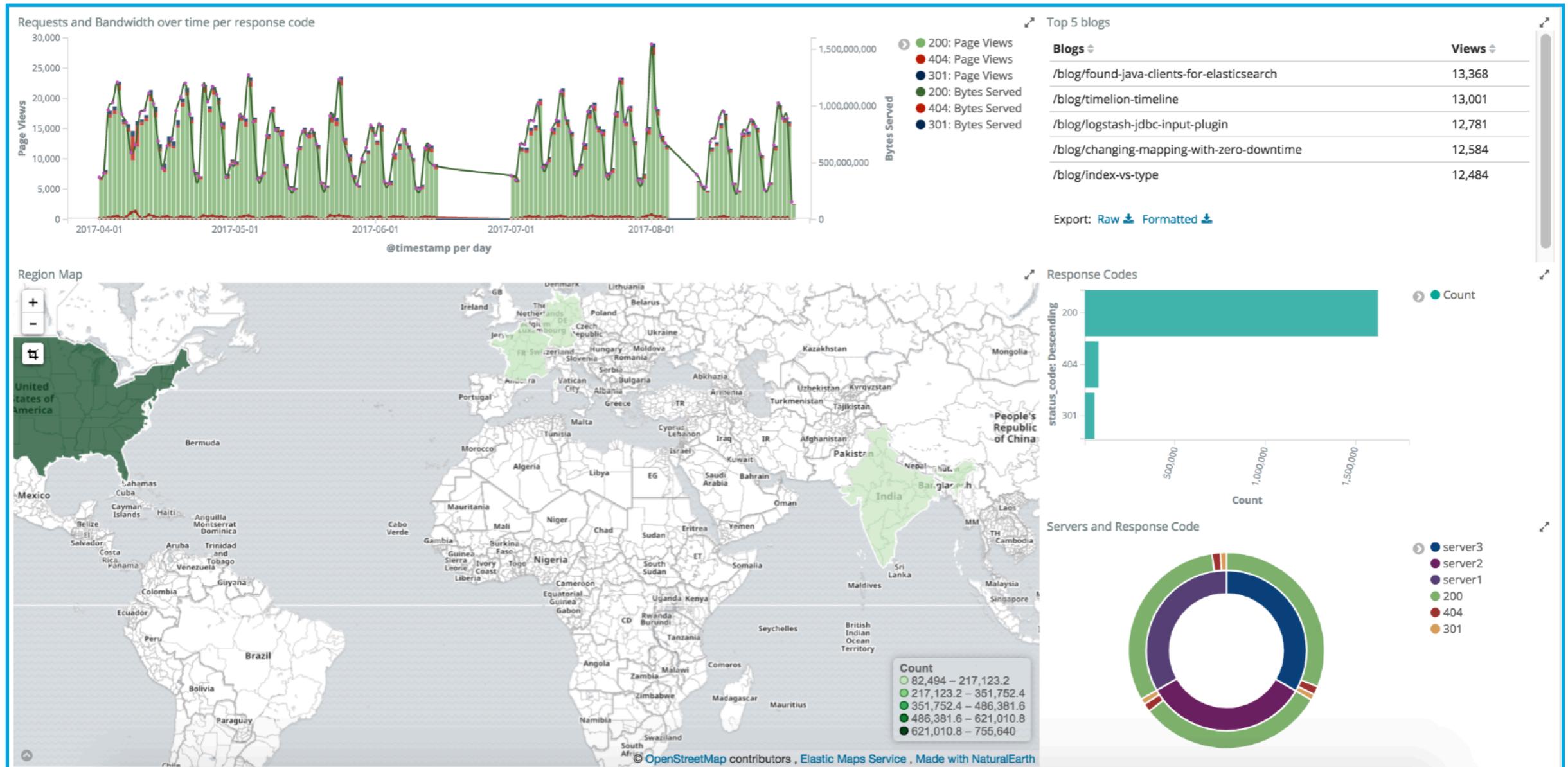
Ingesting the Log Data

- 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:



Chapter Review

·03-Mar-2019·0

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. **True or False:** Elasticsearch uses Apache Lucene behind the scenes to index and search data.
2. What were two of Shay's main objectives when designing Elasticsearch?
3. In general, most data in our customers' use cases can be categorized into what two types of data?
4. You have ***data in a database***. What component of the Elastic Stack might you use to ingest it into Elasticsearch?
5. You want to ***monitor the performance of your servers***. How might you collect that data and ingest it into Elasticsearch?
6. You want to ***monitor all of your networks' traffic***. How might you collect that data and ingest it into Elasticsearch?

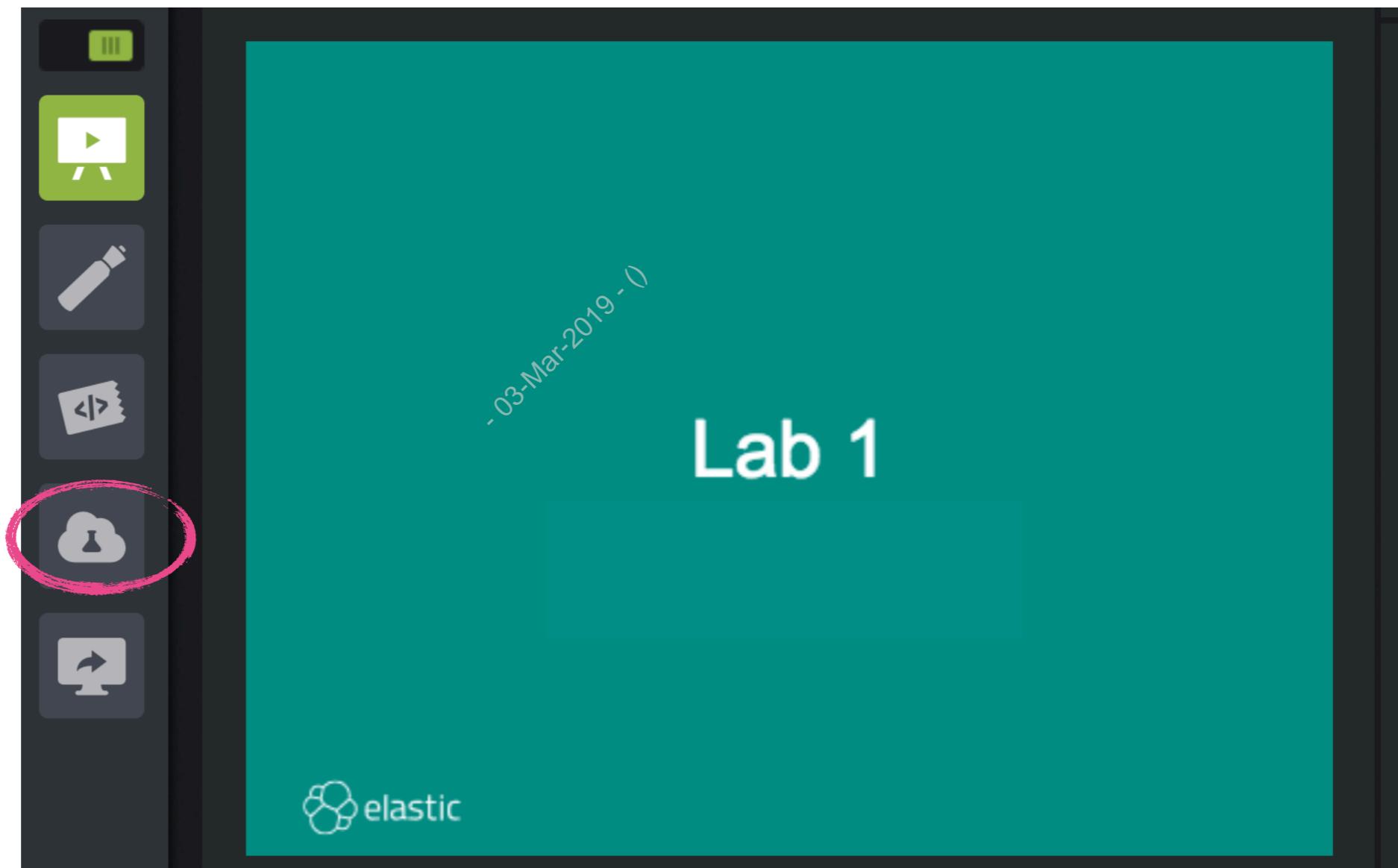


Lab Environment

~03-Mar-2019~

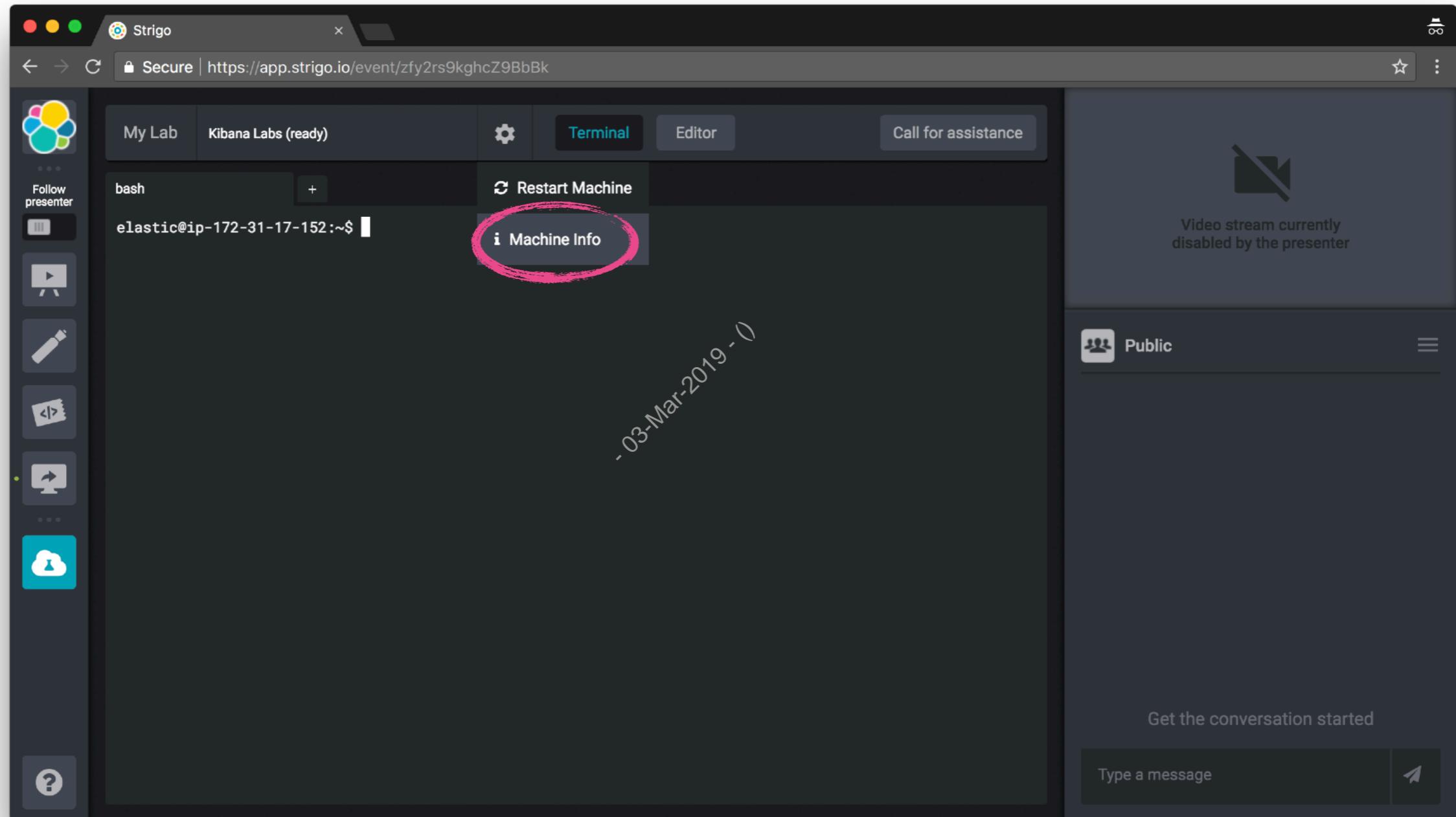
Lab Environment

- 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



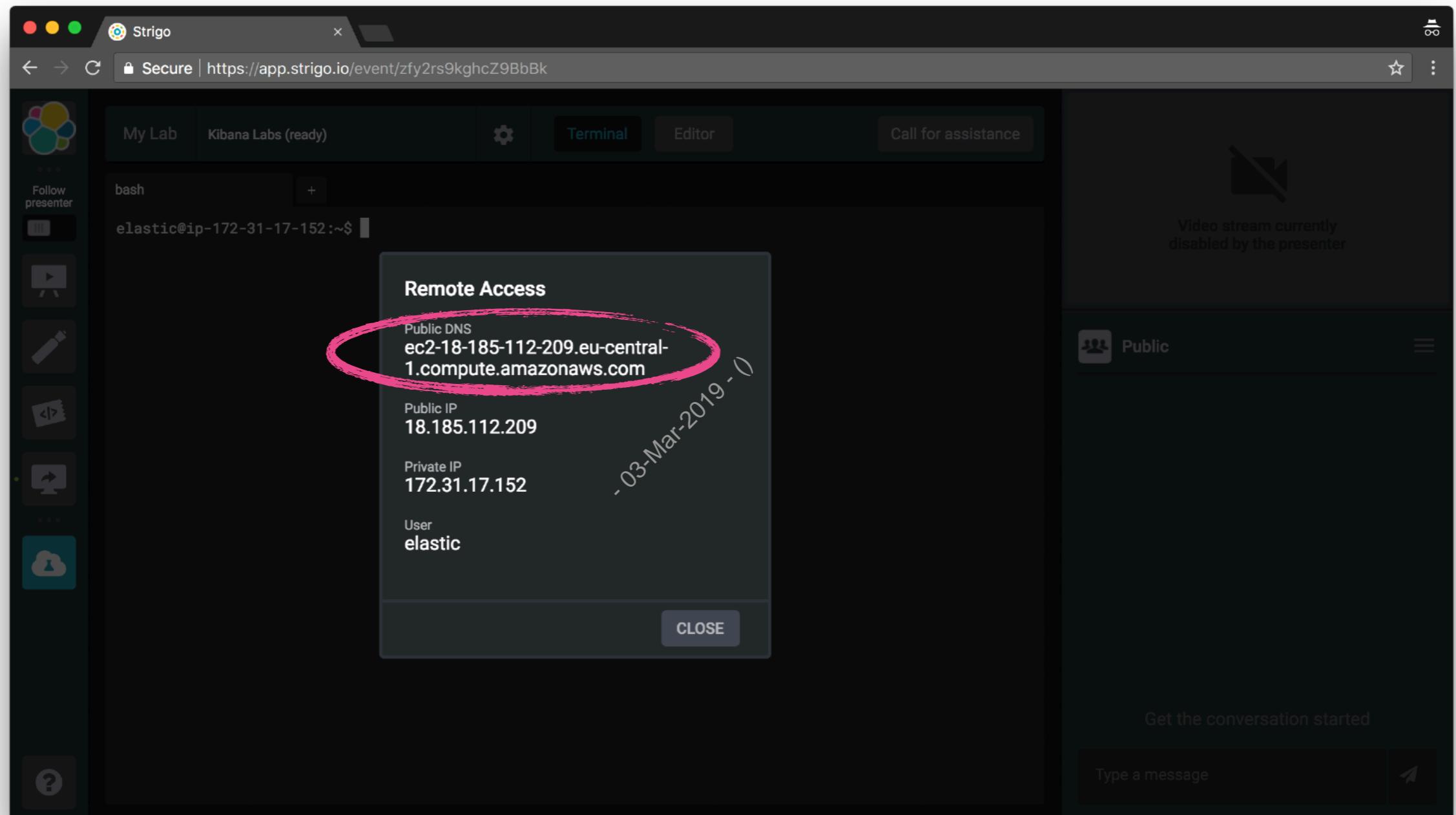
Lab Environment

- 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:



elastic

Welcome to Elasticsearch Engineer I

09-Mar-2019 - 0

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

Lab 1

Elastic Stack Overview



Chapter 2

Getting Started with Elasticsearch

203-Mar-2019-0



- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- Installing Elasticsearch
- Starting Elasticsearch
- Elasticsearch Communication
- Securing Elasticsearch

~03-Mar-2019~0



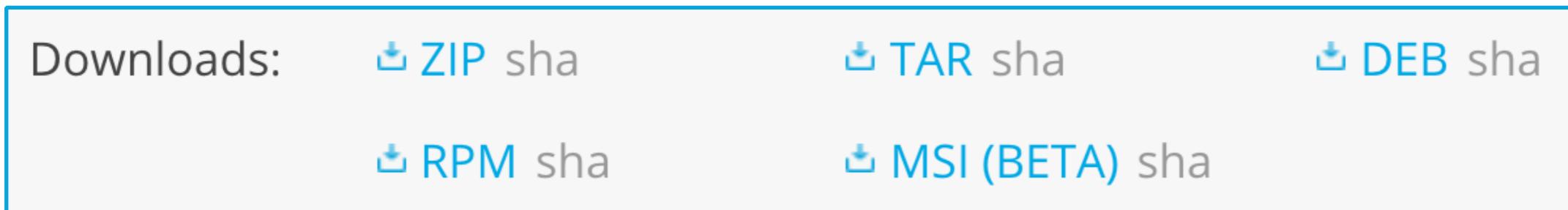
Installing Elasticsearch

~03-Mar-2019 ~00:00



Acquiring Elasticsearch

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



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

A screenshot of the Elasticsearch GitHub repository page. The page shows the following statistics:

- Code: 1,227 issues, 141 pull requests, 1 project, 1 insight.
- Open Source, Distributed, RESTful Search Engine (<https://www.elastic.co/products/elast...>)
- Tags:.elasticsearch, java, search-engine
- Commits: 29,883, Branches: 124, Releases: 204, Contributors: 938, License: Apache-2.0



Preparing for Installation

- Elasticsearch is a Java application
 - requires **JRE** (JavaSE runtime environment) or **JDK** (Java Development Kit)
 - JDK is preferred over JRE, due to tools such as **jstack** and **jps**, as well as better control on updating
- Elasticsearch 6.x requires at least Java 8
 - https://www.elastic.co/support/matrix#show_jvm
- Be sure to verify your environment is using the desired version
 - set **\$JAVA_HOME** appropriately



Installation Options

- Install options:
 - unzip the binaries (it really is that easy!)
 - MSI for Windows
 - use Linux package manager ([.deb](#), [.rpm](#))
 - CM options like Ansible, Chef, Puppet
 - [Docker](#)
 - Elastic Cloud Enterprise
- And if you do not want to install or manage yourself, use Elastic Cloud:
 - <https://www.elastic.co/cloud>



Elasticsearch Directories

Folder	Description	Setting
bin	Binary scripts including elasticsearch to start a node and elasticsearch-plugin to install plugins	
config	Configuration files including elasticsearch.yml	ES_PATH_CONF
data	The location of the data files of each index and shard allocated on the node	path.data
lib	The Java JAR files of Elasticsearch	
logs	Elasticsearch log files location	path.logs
modules	Contains various Elasticsearch modules	
plugins	Plugin files location. Each plugin will be contained in a subdirectory	



The Configuration Files

- Elasticsearch has three 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: `-Xms512mb`
 - command line:

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

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

The **path.data** Directory

- The **path.data** directory is where nodes store data
- Default is **\$ES_HOME/data**
 - except for RPM/DEB which defaults to **/var/lib/elasticsearch**
- You typically configure **path.data** to a different directory

```
path.data: /path/to/data
```

- **path.data** can be set to multiple paths, in which case all paths will be used to store data

```
path.data: [ /home/elasticsearch/data1,/home/elasticsearch/data2 ]
```



Specifying the Config Location

- Default config directory location:
 - `$ES_HOME/config`
- Default location for RPM/DEB:
 - `/etc/elasticsearch`
- The default location can be changed using the `ES_PATH_CONF` environment variable

```
$ ES_PATH_CONF=/home/elasticsearchuser/myconfig ./bin/elasticsearch
```



Starting Elasticsearch

~03-Mar-2019~



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
```

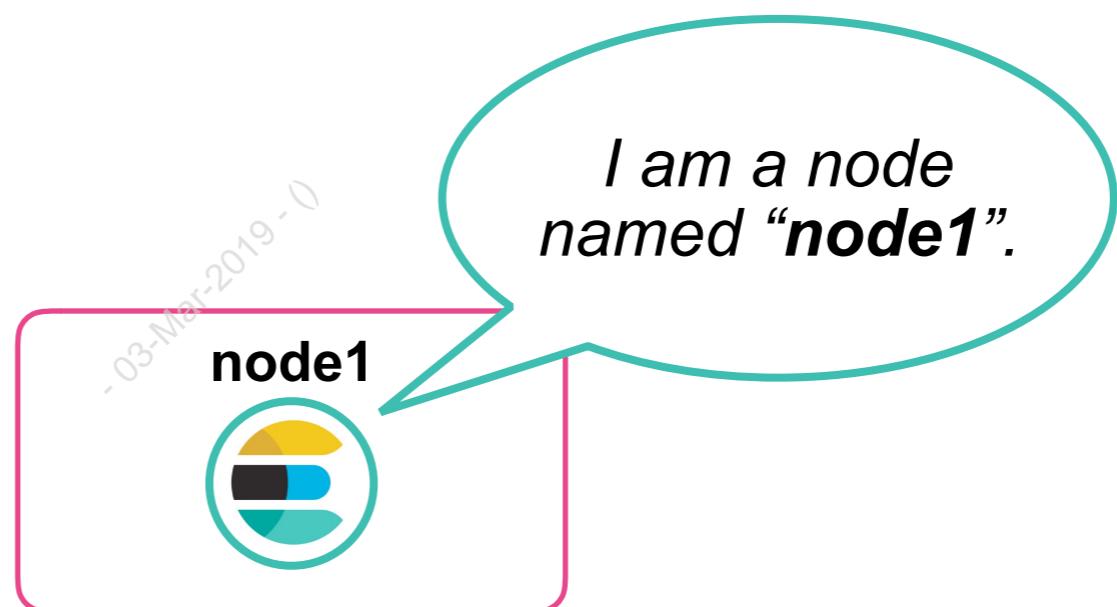
- Use **-d** to run as a daemon on Mac/Linux:

```
$ ./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

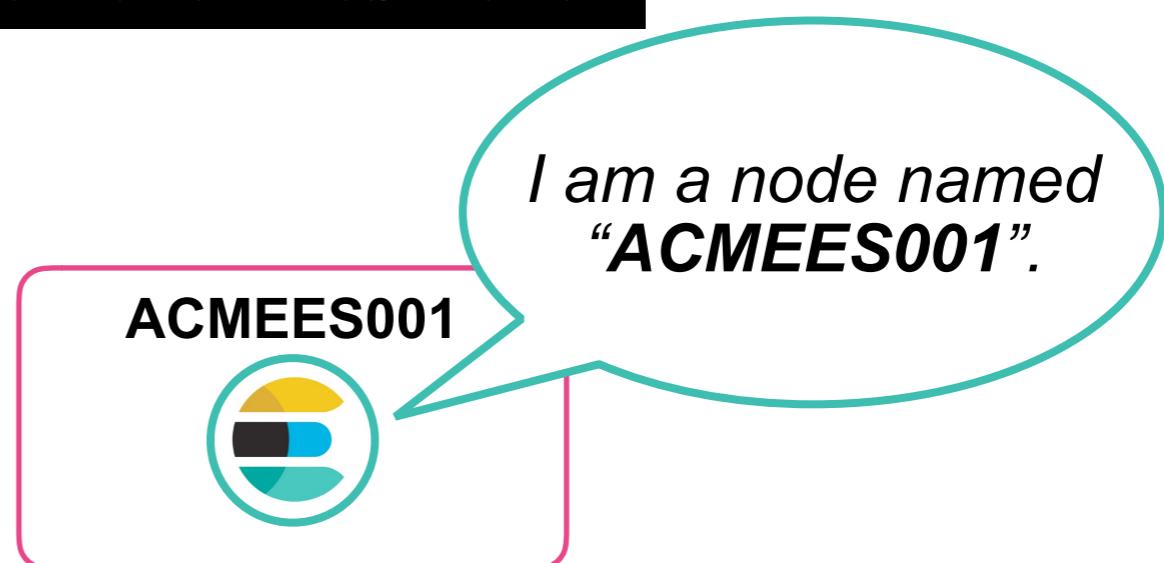


Node Name

- Every node has an **unique id**
 - randomly generated UUID
- Every node has a name
 - default is the first seven characters of the UUID
 - change to something meaningful
- Set using **node.name** (on the command line or .yml file)

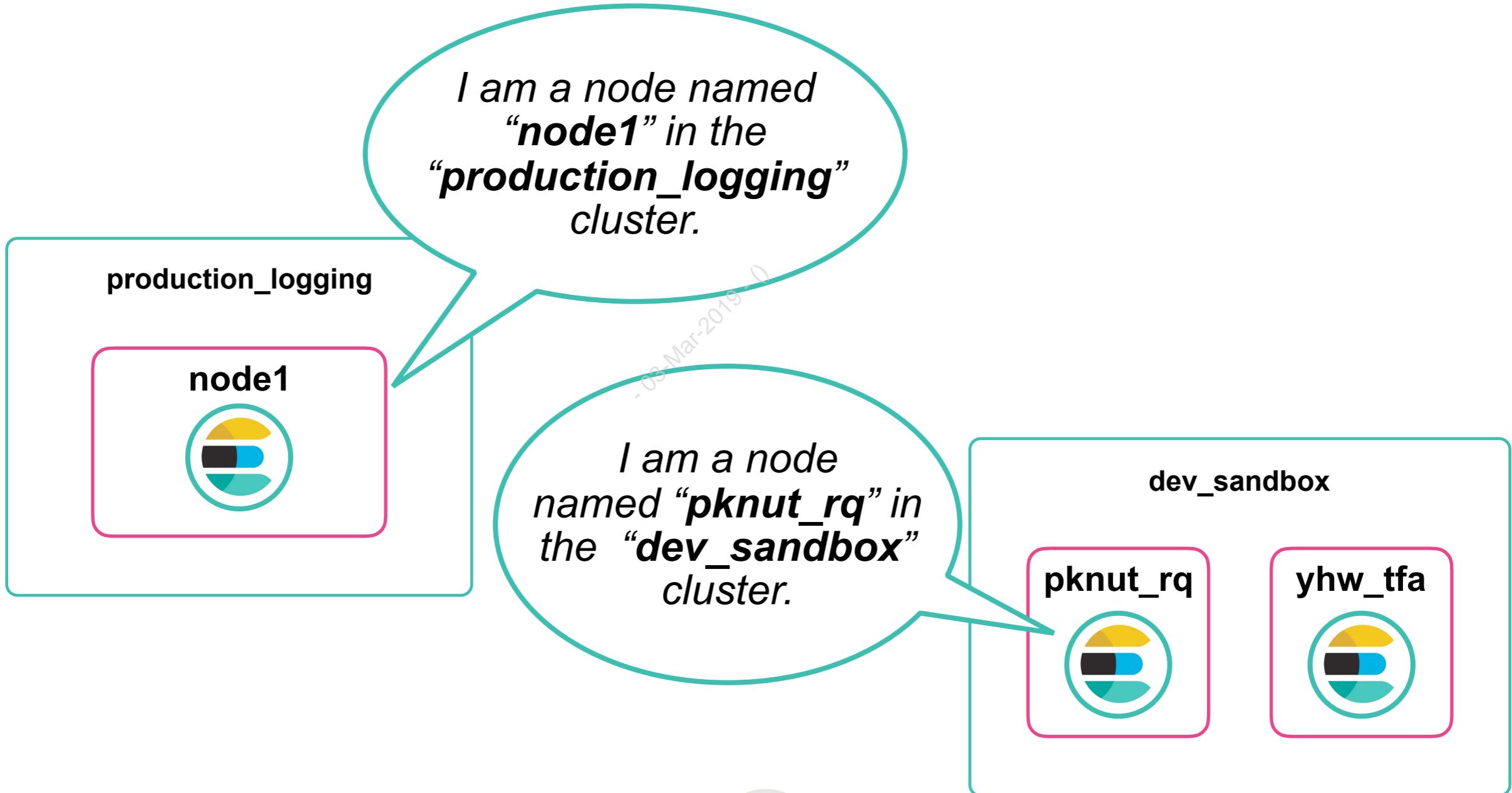
```
$ ./bin/elasticsearch -E node.name=`hostname`
```

node.name: \${HOSTNAME}



Cluster

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

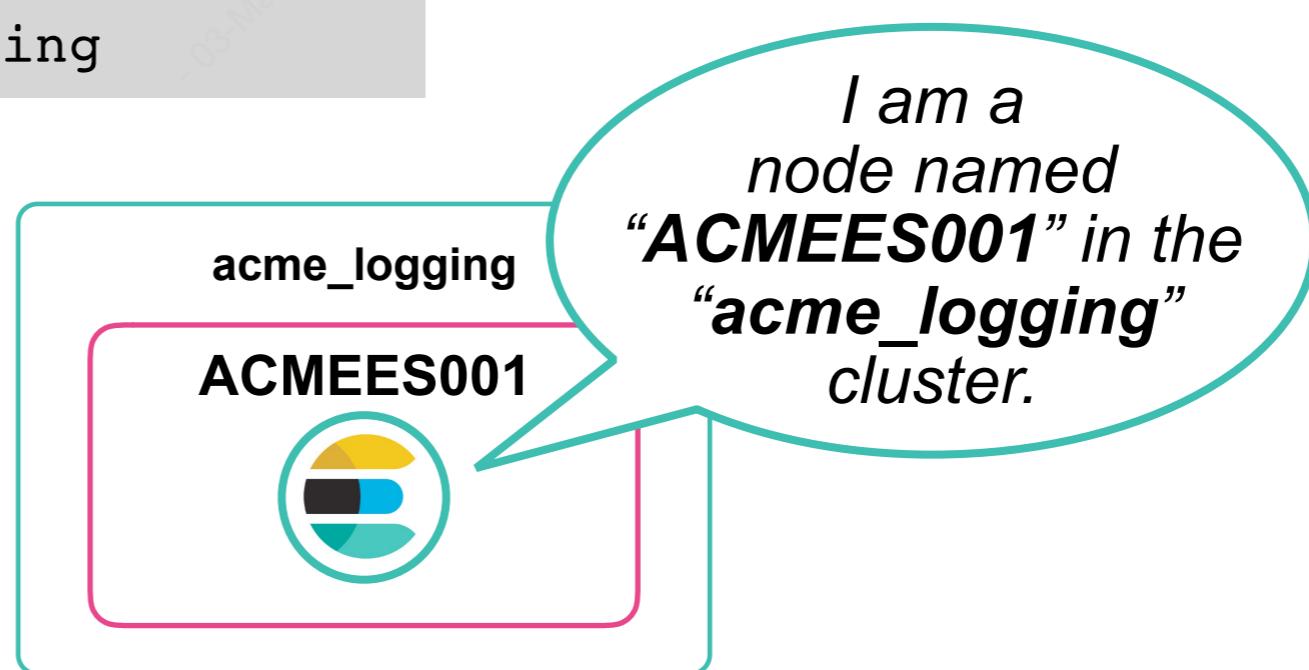


Cluster Name

- 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



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`
```

~03-Mar-2019~0

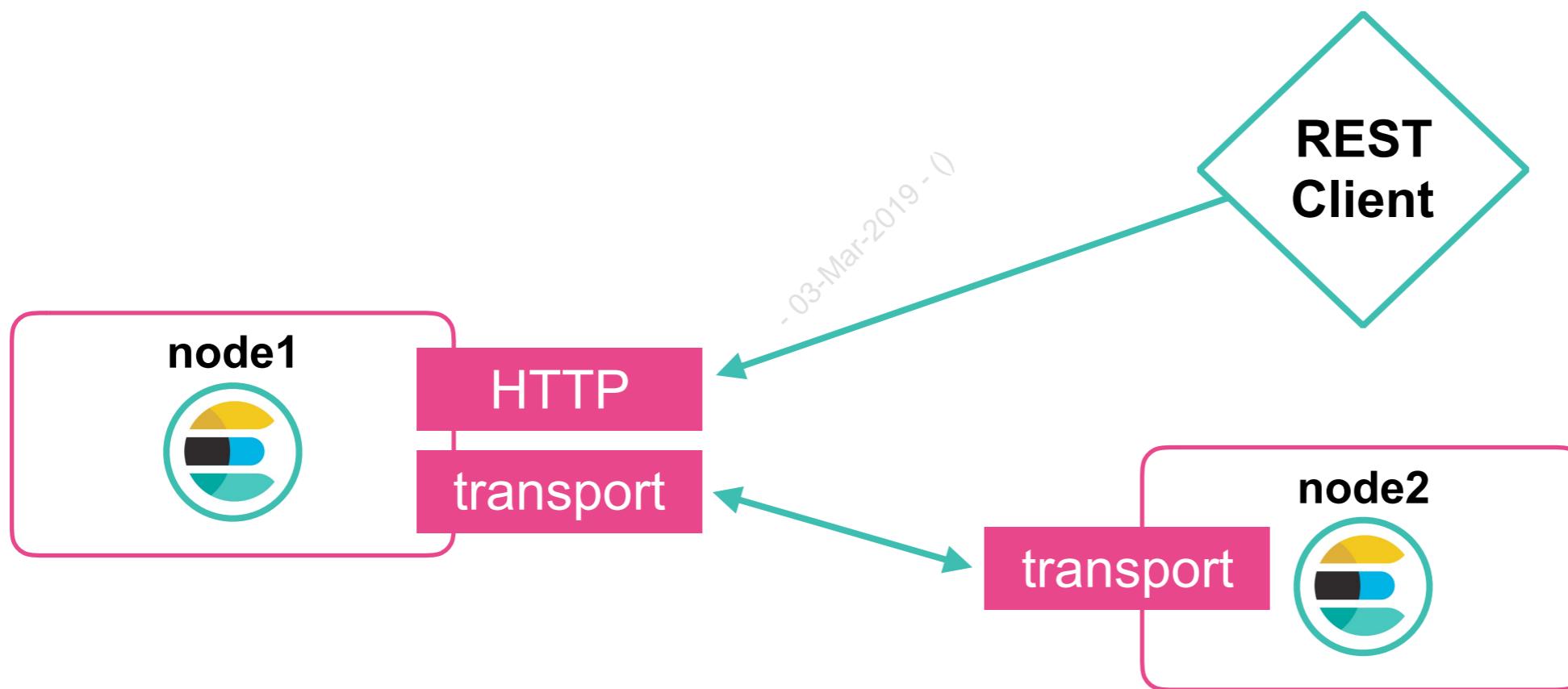
Elasticsearch Communication

-03-Mar-2019-0



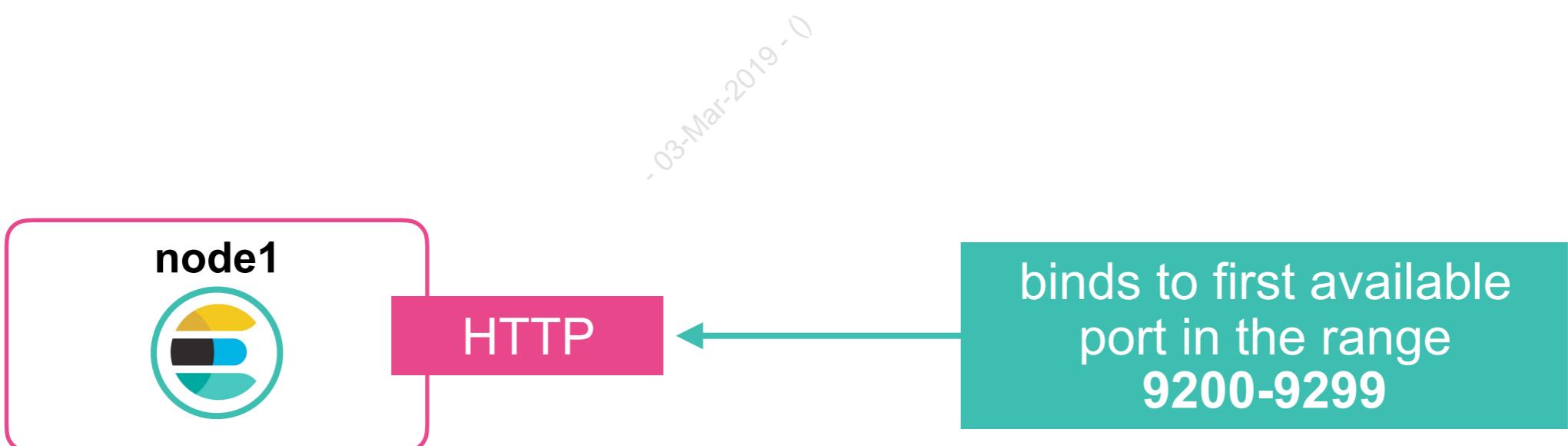
HTTP vs. Transport

- 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**



API Examples

GET command

```
$ curl -XGET 'http://localhost:9200/'
```

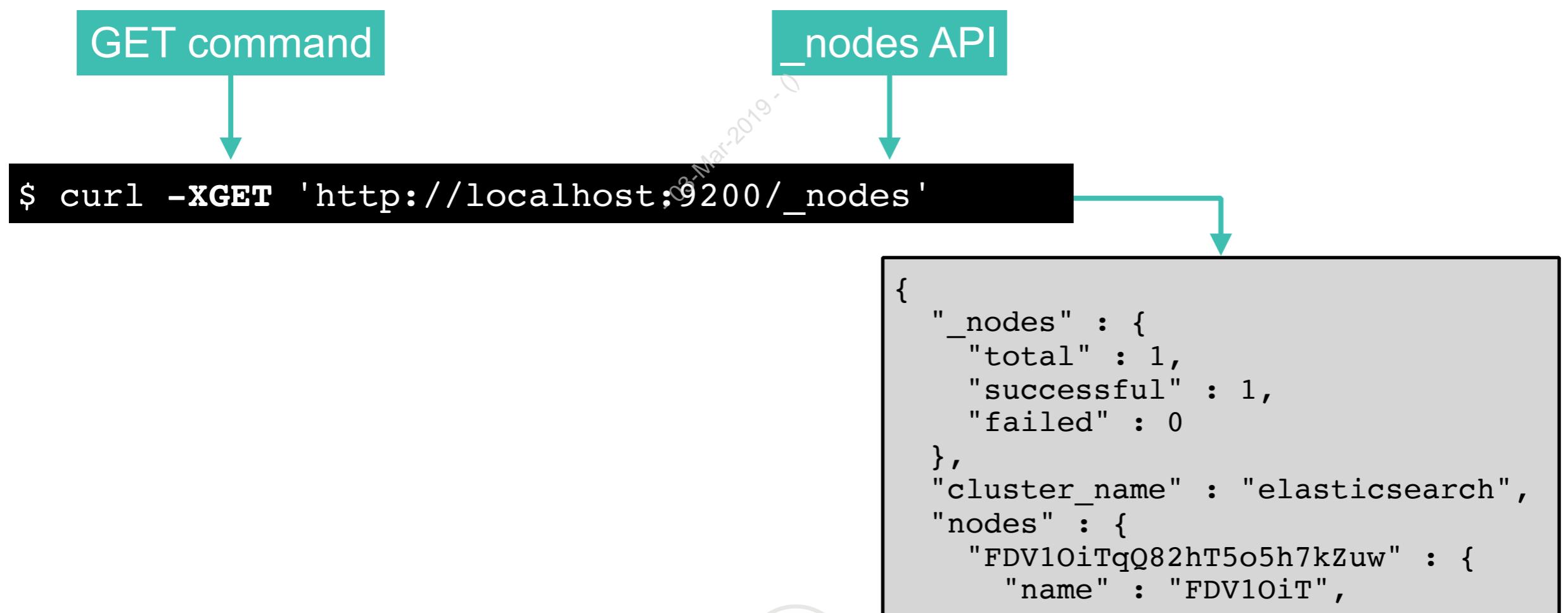
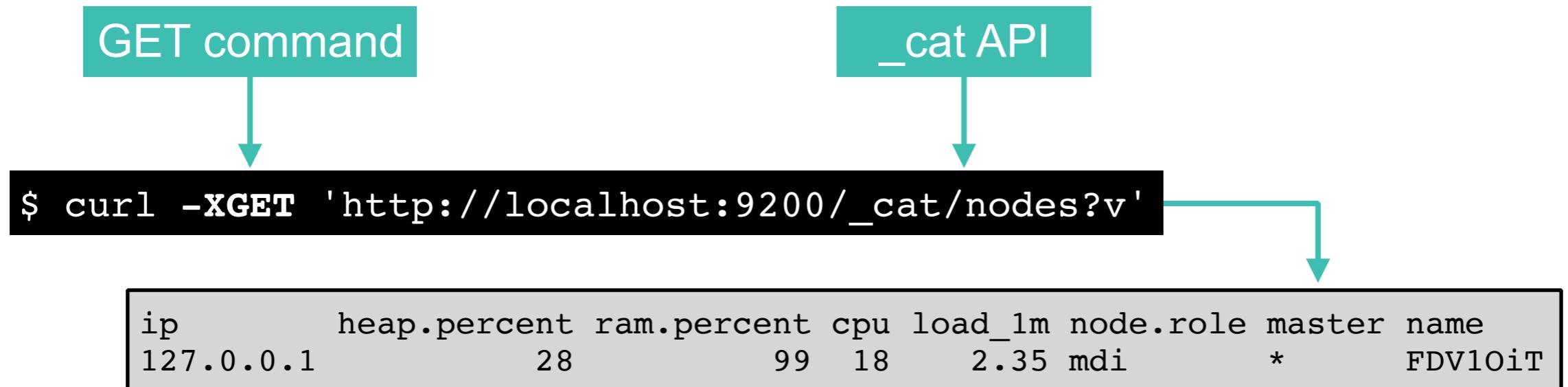
“200 OK” response if successful

```
HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8
content-length: 432

{
  "name" : "FDVlOiT",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "dHJUi1jrRLS9pBkvHZ172Q",
  "version" : {
    "number" : "6.3.1",
    "build_hash" : "bd92e7f",
    "build_date" : "2018-06-29T21:59:26.107521Z",
    "build_snapshot" : false,
    "lucene_version" : "7.3.1",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

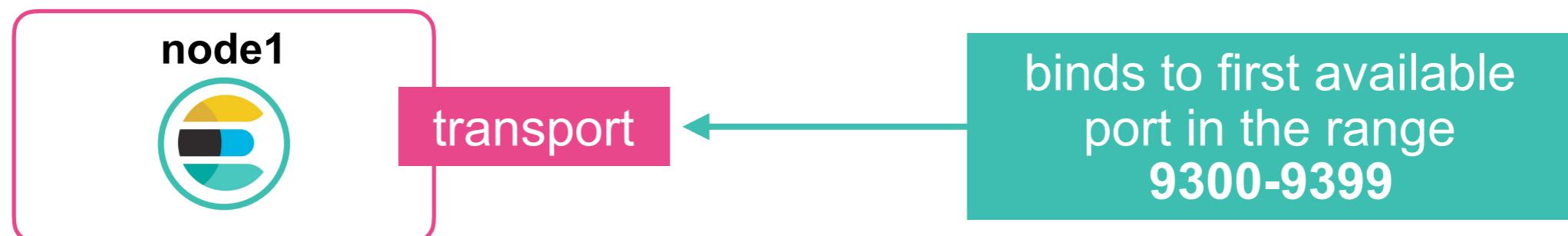


API Examples



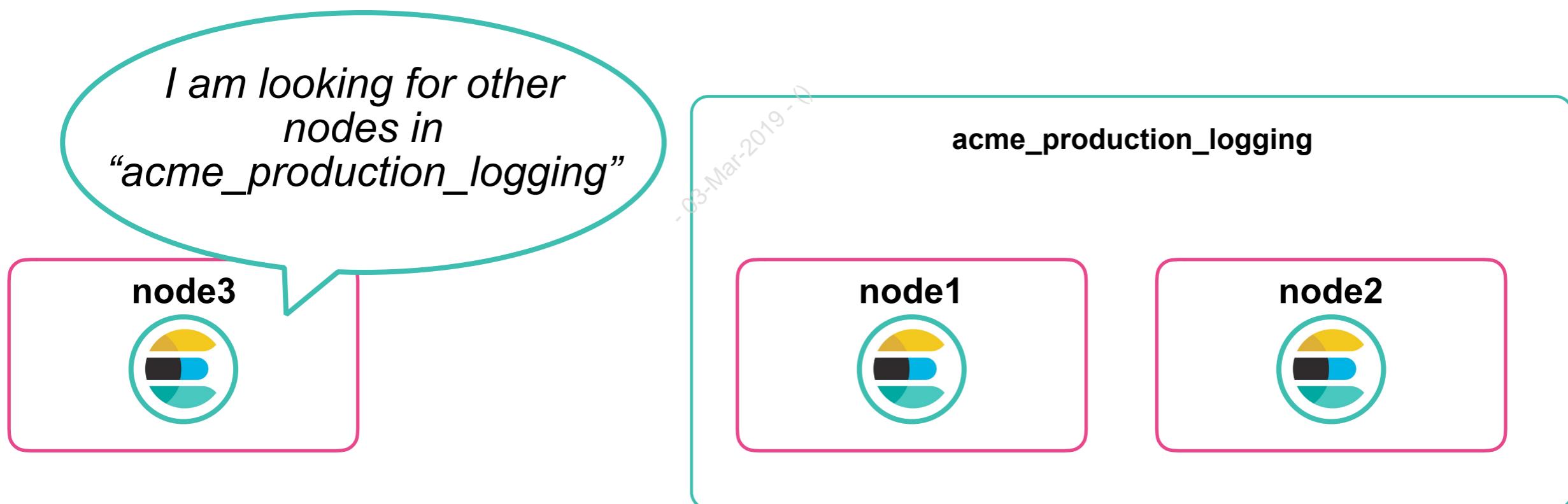
Transport Communication

- 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**



The Discovery Module

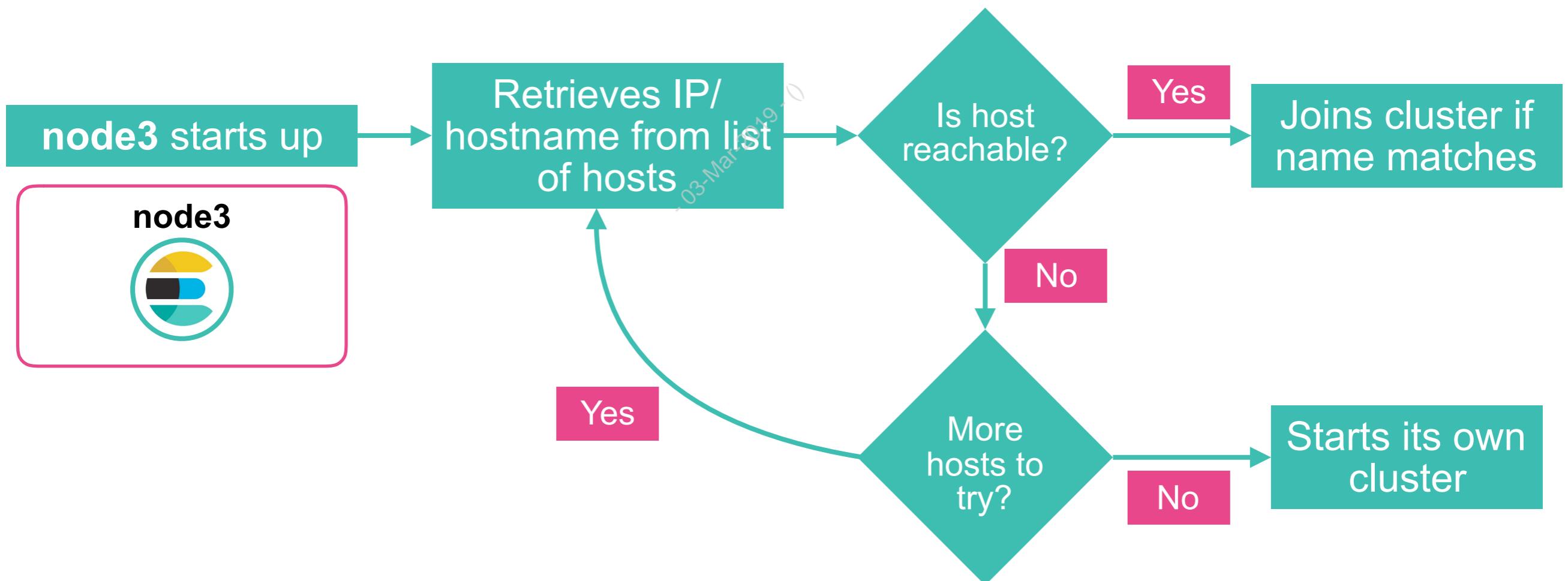
- The discovery module is responsible for discovering nodes within a cluster
 - the default is zen discovery
 - but there are also discovery modules for EC2, GCE and Azure



Adding Nodes to a Cluster

- To join a cluster, a node issues ping requests to find other nodes on the same network
 - using a list of hosts that act as *gossip routers*, configured with **discovery.zen.ping.unicast.hosts**

```
discovery.zen.ping.unicast.hosts : [ "node1:9300", "node2" ]
```



Configuring Network Settings

- Three ways to configure network settings:

network.*	specify settings for both protocols in one setting
transport.*	specify settings for the transport protocol
http.*	specify settings for the http protocol

- When configuring hosts, you can specify binding and publishing together, or separately:

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



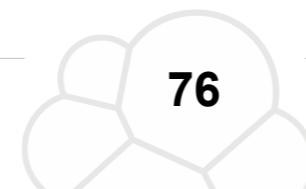
Special Values for network.host

- The following special values may be used for `network.host`:
 - 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_`

A common setting



Securing Elasticsearch

~03-Mar-2019 ~00:00



Security Considerations

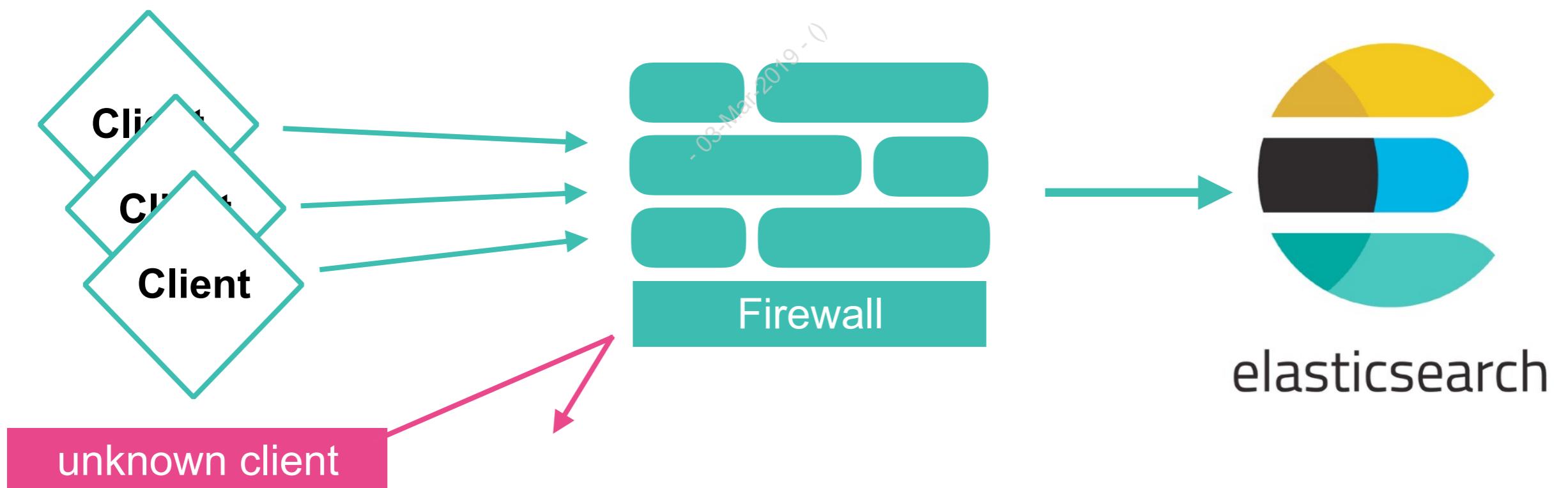
- Elasticsearch **does not offer** authentication or authorization by default
- **Make sure to secure your cluster before going live:**
 - firewalls
 - reverse proxy
 - *Elastic Security*

~03-Mar-2019~0



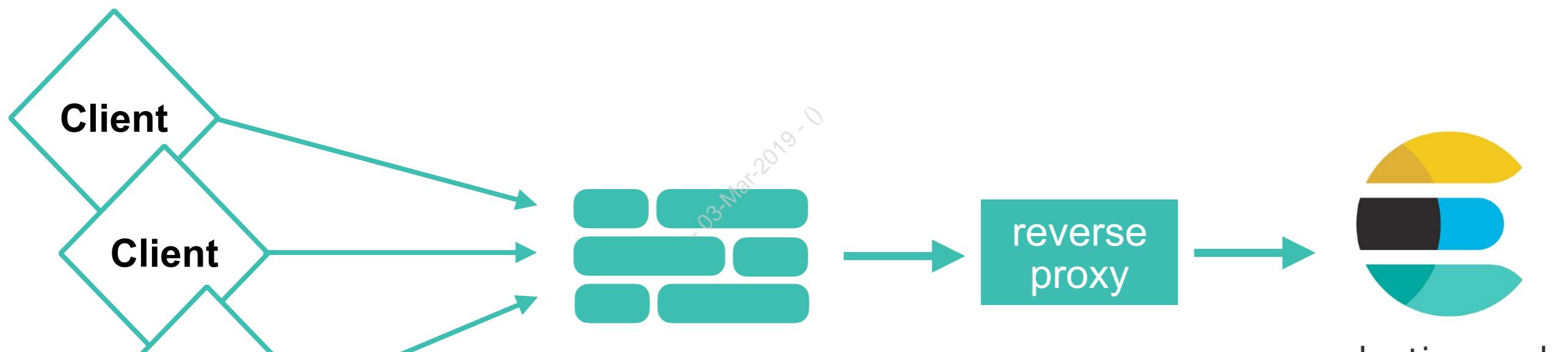
Firewall

- Add ***firewall rules*** with IP filtering to only allow access from the machines running your application
 - never expose port 9200 http publicly on the internet
 - restrict access of ports to only systems necessary
 - be careful, don't block port 9300 between the nodes



Reverse Proxy

- You can put Elasticsearch behind a *reverse proxy*
 - gain basic authentication and authorization
 - helpful blog at <https://www.elastic.co/blog/playing-http-tricks-nginx>



Read-only Configurations

- Using an HTTP web proxy (HAProxy, NGINX, etc.), you can configure:

1. read-only cluster vs. write cluster

- add rules that parse the URL path restricting requests to certain URL paths

2. read-only indices vs. write indices:

- only allow GET requests for certain logins/IPs

3. read-only Kibana dashboards and visualizations

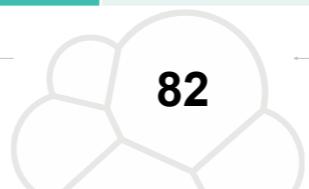
- only allow GET, HEAD, and OPTIONS requests for the particular Kibana instances that are desired to be read-only



Elastic Security

- Enables Security (Gold/Platinum):

	<i>Gold</i>	<i>Platinum</i>
<i>Encrypted communications</i>	✓	✓
<i>Role-based access control</i>	✓	✓
<i>Audit Logging</i>	✓	✓
<i>IP filtering</i>	✓	✓
<i>Native, LDAP, PKI, Active Directory authentication</i>	✓	✓
<i>SAML authentication</i>		✓
<i>Attribute-based access control</i>		✓
<i>Field and document level security</i>		✓
<i>Third-party authentication</i>		✓
<i>Encryption at rest support</i>		✓



Enabling Elastic Security

The screenshot shows the Kibana Management interface. On the left is a sidebar with icons for Discover, Visualize, Dashboard, Timelion, APM, Dev Tools, Monitoring, and Management. The Management icon is selected. The main area is titled 'Management' and shows 'Version: 6.3.1'. Below this are sections for 'Elasticsearch' (with 'Index Management' and 'License Management' buttons, where 'License Management' is circled in red), 'Kibana' (with 'Index Patterns', 'Saved Objects', 'Reporting', and 'Advanced Settings' buttons), and a footer with 'Index Patterns', 'Saved Objects', 'Reporting', and 'Advanced Settings'.

The screenshot shows the 'Management / Elasticsearch / License Management' page. The left sidebar is identical to the one in the previous screenshot. The main content area displays a message: 'Your Basic license is active' with a checkmark icon, followed by the text 'Your license will never expire.' Below this are two cards: 'Update your license' (with the sub-instruction 'If you already have a new license, upload it now.' and a 'Update license' button) and 'Start a 30-day trial' (with the sub-instruction 'Experience what security, machine learning, and all our other [Platinum features](#) have to offer.' and a 'Start trial' button). A watermark '08-Mar-2019' is visible across the page.

Configuring Elastic Security

- Elasticsearch:
 - enable Elastic security feature on every node (elasticsearch.yml)

```
xpack.security.enabled: true
```

- set the passwords for all built-in users

```
./elasticsearch/bin/elasticsearch-setup-passwords interactive
```

- Kibana:
 - configure Kibana to use the appropriate built-in user (kibana.yml)

```
elasticsearch.username: "kibana"  
elasticsearch.password: "kibanapassword"
```



Kibana Keystore

- In order to avoid having passwords in clear text in the configuration file, a keystore can be used:

```
bin/kibana-keystore create
```

- Once the keystore is created, sensitive information can be added to it:

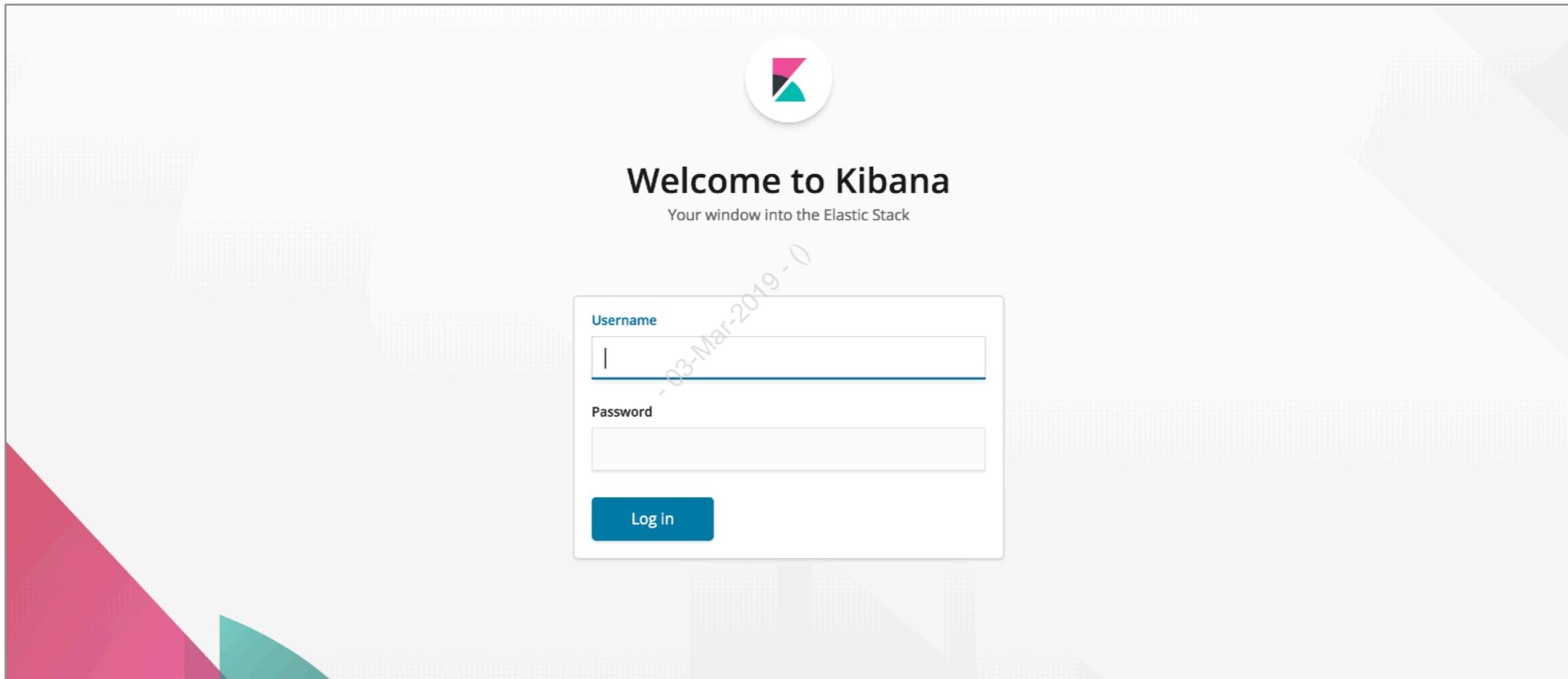
```
bin/kibana-keystore add elasticsearch.username  
kibana  
bin/kibana-keystore add elasticsearch.password  
password
```

- Those parameters don't need to be defined in the kibana.yml file anymore



After Configuring Elastic Security

- Login with user and password that you defined before



Defining Users and Roles

- Users, roles, privileges and permissions can be defined in Kibana:

Management / Security

Users Roles

<input type="checkbox"/>	Full Name ↑	Username	Roles	Reserved ?
<input type="checkbox"/>	Blog User	blogs_user	blogs_readonly , kibana_user	
<input type="checkbox"/>		elastic	superuser	✓
<input type="checkbox"/>		kibana	kibana_system	✓
<input type="checkbox"/>		logstash_system	logstash_system	✓

Search... + Create user

03-Mar-2019



Defining Roles and Privileges

- Roles and privileges can be defined at a very fine-grained level:

New Role

Name
my_role

Cluster Privileges

- all
- monitor
- manage
- manage_security
- manage_index_templates
- manage_pipeline
- manage_ingest_pipelines
- transport_client
- manage_ml
- monitor_ml
- manage_watcher
- monitor_watcher

Run As Privileges
Add a user...

Index Privileges

Indices
my_index*

Privileges
manage ✖ monitor ✖ read ✖ read_cross_cluster ✖

Granted Documents Query Optional

Granted Fields Optional
* ✖ +



X-Pack <6.3

- Before following the steps discussed in the previous slides, install the x-pack plugin in

- Elasticsearch:

```
./elasticsearch/bin/elasticsearch-plugin install x-pack
```

- Kibana:

```
./kibana/bin/kibana-plugin install x-pack
```

the kibana install will
take a few minutes



Chapter Review

·03-Mar-2019·0

Summary

- Installing Elasticsearch can be as easy as unzipping a file!
- Elasticsearch uses two network communication mechanisms: ***HTTP*** for REST clients; and ***transport*** for inter-node communication
- A ***node*** is an instance of Elasticsearch
- A ***cluster*** is one or multiple nodes working together in a distributed manner
- ***Elastic Security*** provides a complete solution for securing Elasticsearch Make sure to secure your cluster
- **Secure your cluster!**

Quiz

1. What are the three files you will find in the Elasticsearch **config** folder?
2. How do nodes in a cluster communicate with each other?
3. How does a new node joining a cluster find the cluster?
4. What is the result of setting **network.host** to **_site_**?
5. **True or False:** An Elasticsearch cluster comes with security available by default.
6. Explain three options to secure your cluster.



Lab 2

03-Mar-2019 10

Getting Started with Elasticsearch



Chapter 3

CRUD Operations

· 03-Mar-2019 · 0

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- Getting Data In
- Cheaper in Bulk
- Getting Data Out

~03-Mar-2019~0



Getting Data In

·03-Mar-2019·0

Documents must be JSON Objects

- For example, our blogs are currently in a database table:

title	category	date	author_first_name	author_last_name	author_company
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** (e.g. “**title**” and “**author.first_name**”)

...

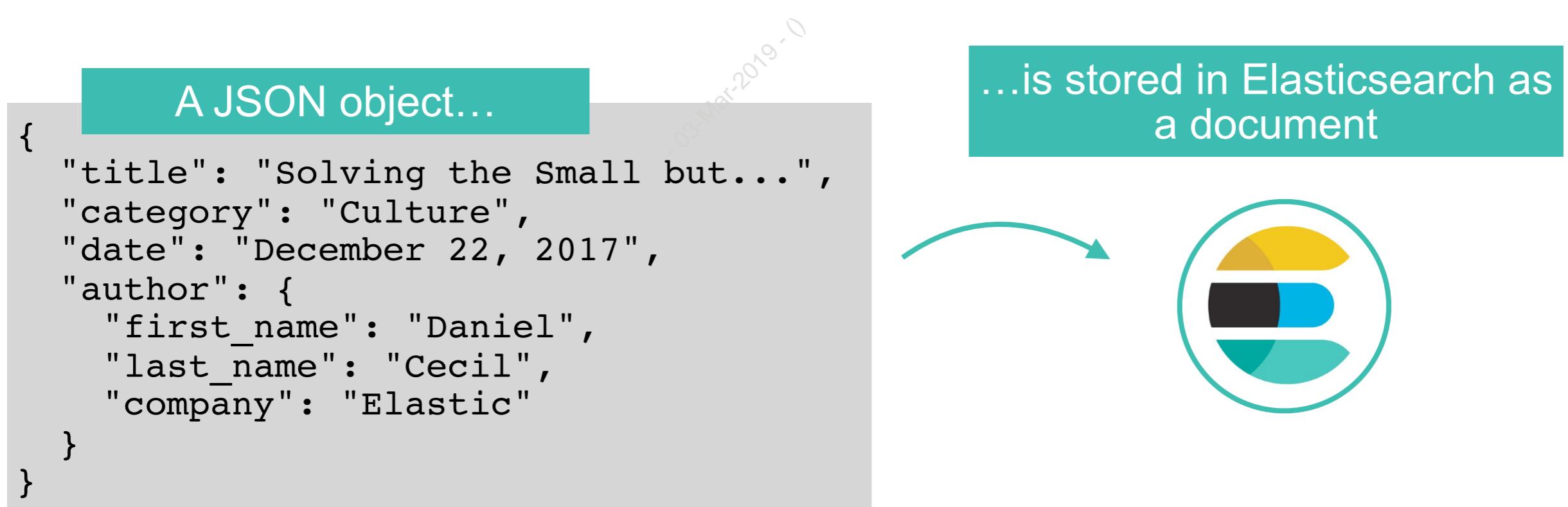
```
{  
  "title": "Fighting Ebola with Elastic",  
  "category": "User Stories",  
  "author": {  
    "first_name": "Emily",  
    "last_name": "Mosher"  
  }  
}
```

...and **values**



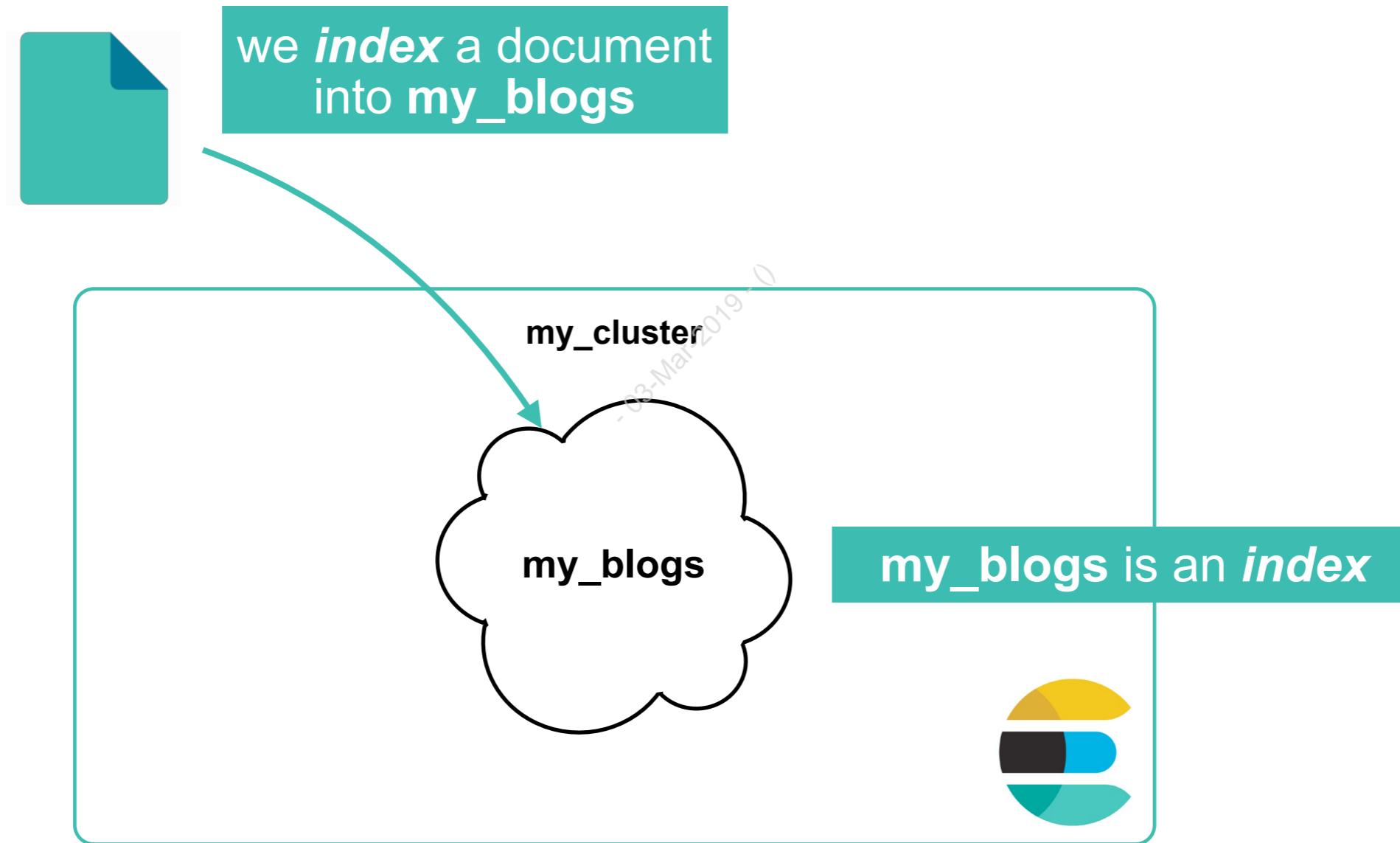
Document Store

- 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



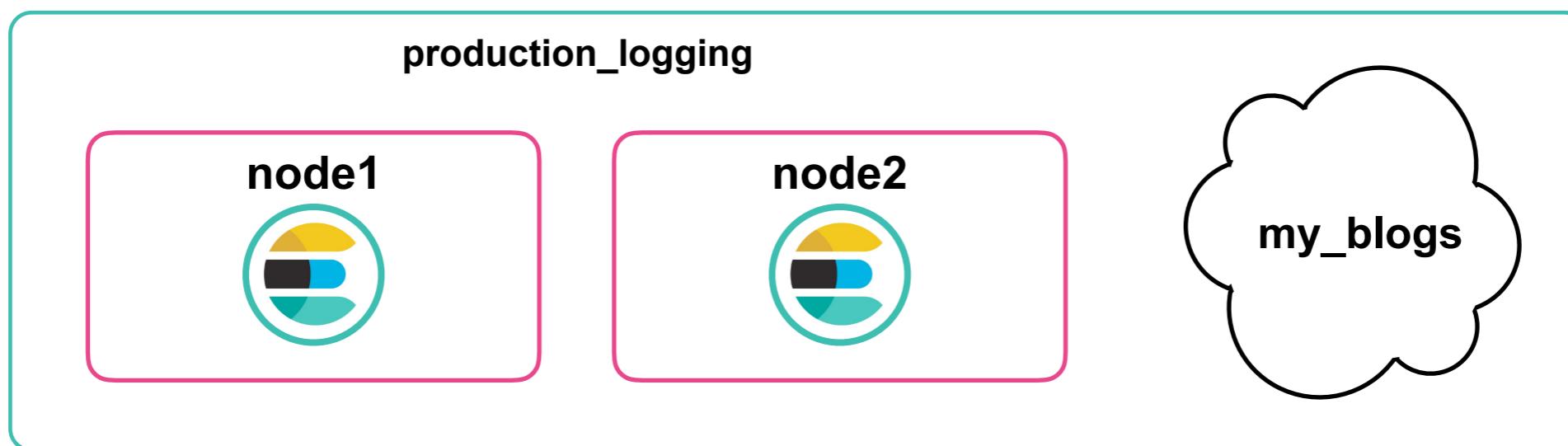
Documents are Indexed into an Index

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



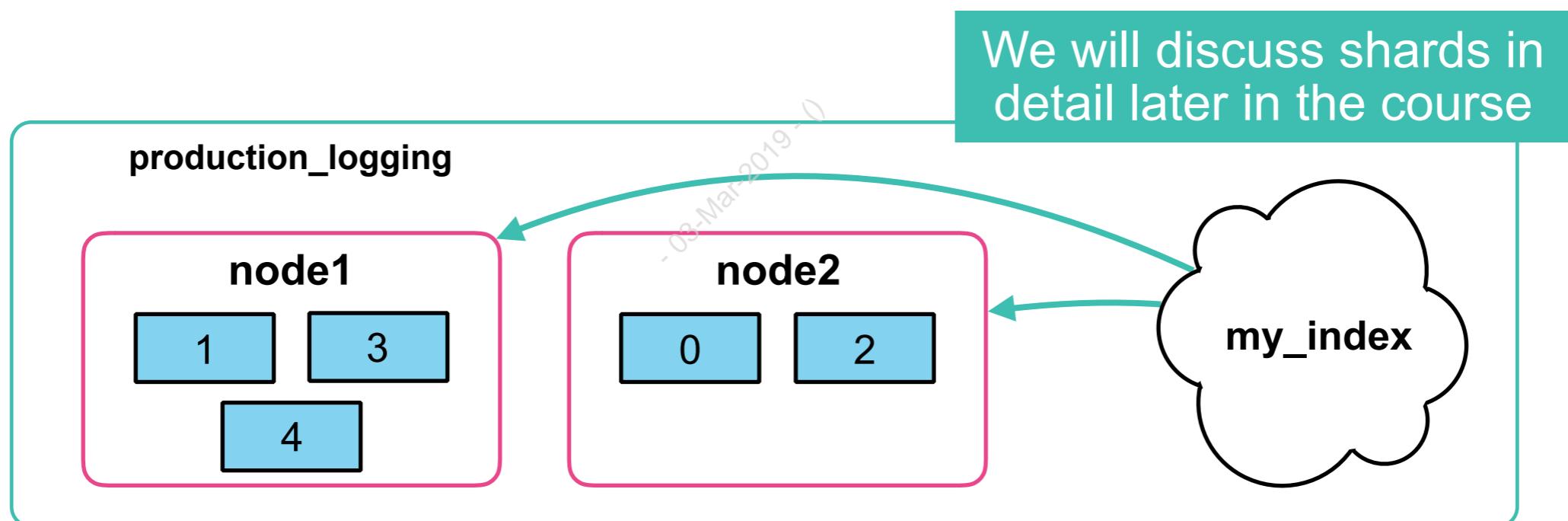
Index

- An **index** in Elasticsearch is a ***logical*** way of grouping data
 - an index has a ***mapping*** that defines the fields in the index
 - an index is a ***logical namespace*** that maps to where its contents are stored in the cluster
- There are two different concepts in this definition:
 - an index has some type of ***data schema*** mechanism
 - an index has some type of mechanism to ***distribute data*** across a cluster



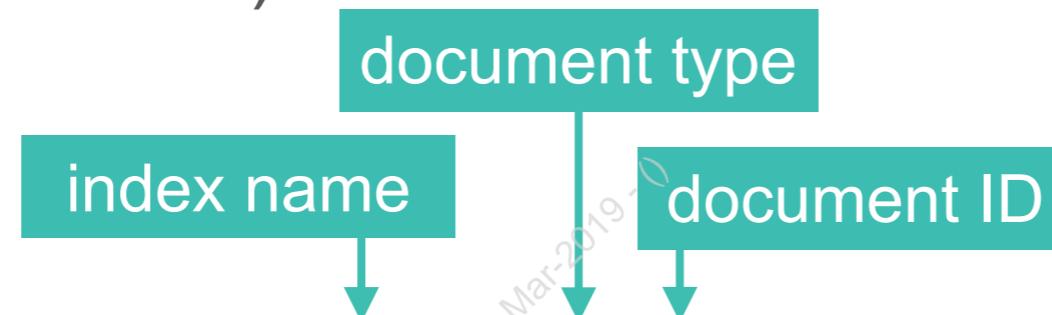
Shards

- Elasticsearch is a ***distributed*** document store
- A ***shard*** is a single piece of an Elasticsearch index
 - Indexes are partitioned into shards so they can be distributed across multiple nodes



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**, the **type** and an **ID**
 - if no ID is provided, elasticsearch will generate one (as we 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

The screenshot shows the Kibana interface with the Dev Tools tab selected. In the Dev Tools section, the Console tab is active. A code editor window displays a PUT request to the index 'my_blogs' with document ID '1'. The request body contains a JSON object with fields: title, category, author (first_name and last_name). The response pane shows the successful creation of the document, including fields like _index, _type, _id, _version, result, _shards, total, successful, failed, _seq_no, and _primary_term.

```
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
11 {
12   "_index": "my_blogs",
13   "_type": "_doc",
14   "_id": "1",
15   "_version": 1,
16   "result": "created",
17   "_shards": {
18     "total": 2,
19     "successful": 1,
20     "failed": 0
21   },
22   "_seq_no": 0,
23   "_primary_term": 1
24 }
```

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



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": {
    ...
  }
}
```

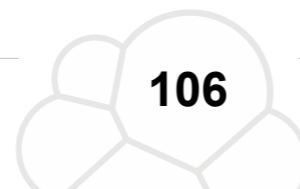
We typically create the index first, then index our documents

What if the document ID already exists?

- What do you think happens if you index a document using and 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/_doc/1/_create
{
  "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 update fields in a document, use the `_update` endpoint:
 - make sure to add the “doc” context

```
POST my_blogs/_doc/1/_update
{
  "doc": {
    "date": "September 26, 2016"
  }
}
```

update the “date” field

The diagram illustrates the flow of the update operation. A curved green arrow points from the 'date' field in the request body (inside the 'doc' object) down to the 'result' field in the response body. The request body is shown in a light gray box, and the response body is shown in a white box with a black border.

```
{
  "_index": "my_blogs",
  "_type": "_doc",
  "_id": "1",
  "_version": 3,
  "result": "updated",
  ...
}
```



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

„03-Mar-2019“

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, which can be queued up and indexed in batches of hundreds or thousands
- 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/_doc/_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

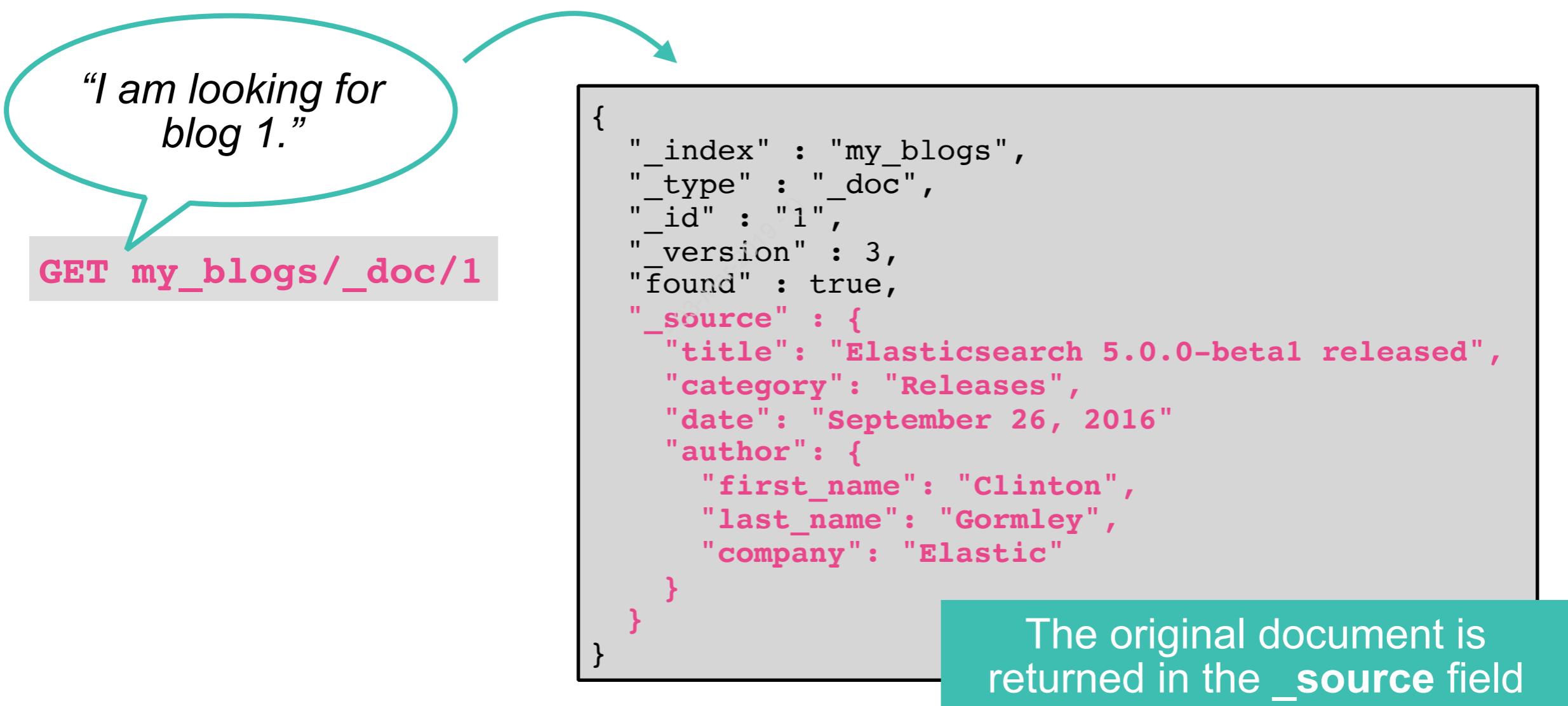


Getting Data Out

· 03-Mar-2019 · 0

Retrieving a Document

- Use **GET** to retrieve an indexed document
 - notice we specify the **index**, the **type** 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",
      "_type" : "_doc",
      "_id" : 3
    },
    {
      "_index" : "blogs",
      "_type" : "_doc",
      "_id" : "F1oSq2EBOOytT5ZTHpaE"
    }
  ]
}
```

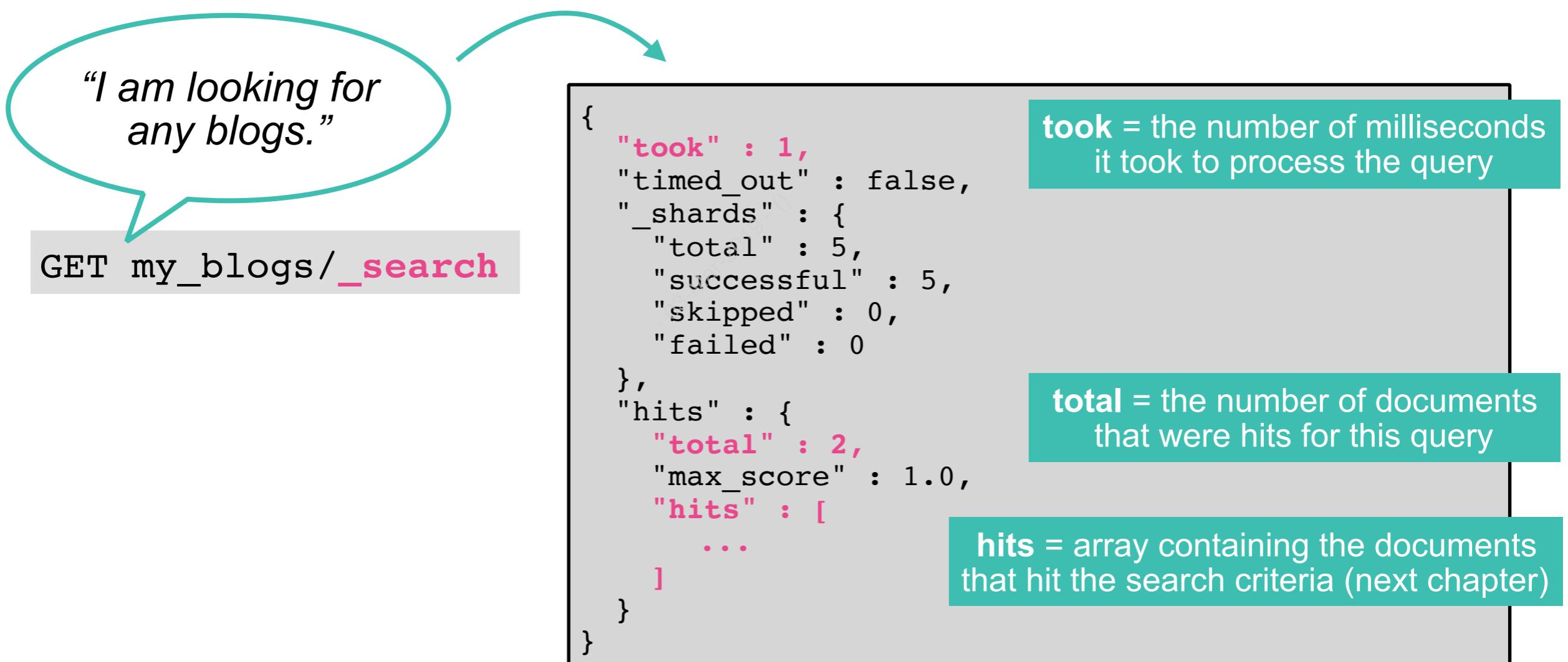
Use the `_mget` endpoint

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



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



Chapter Review

·03-Mar-2019·0

CRUD Operations

Index	<pre>PUT my_blogs/_doc/4 { "title" : "Elasticsearch released", "category": "Releases" }</pre>
Create	<pre>PUT my_blogs/_doc/4/_create { "title" : "Elasticsearch released", "category": "Releases" }</pre>
Read	<pre>GET my_blogs/_doc/4</pre>
Update	<pre>POST my_blogs/_doc/4/_update { "doc" : { "title" : "Elasticsearch 6.2 released" } }</pre>
Delete	<pre>DELETE my_blogs/_doc/4</pre>
Search	<pre>GET my_blogs/_search</pre>



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

~03-Mar-2019~0

Quiz

1. How is an index distributed across a cluster?
2. **True or False:** If an index does not exist when indexing a document, Elasticsearch will automatically create the index for you.
3. What is the ***default number of hits*** that a query returns?
4. What happens if you index a document and the `_id` already appears in the index?
5. **True or False:** Using the Bulk API is more efficient than sending multiple, separate requests.



Lab 3

CRUD Operations

03-Mar-2019



Chapter 4

Querying Data

,03-Mar-2019,0

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- Relevance
- Searching for Terms
- Scoring
- Searching for Phrases
- Searching within Date Ranges
- Combining Searches
- Filtering Searches
- Improving Relevancy

~03-Mar-2019~0

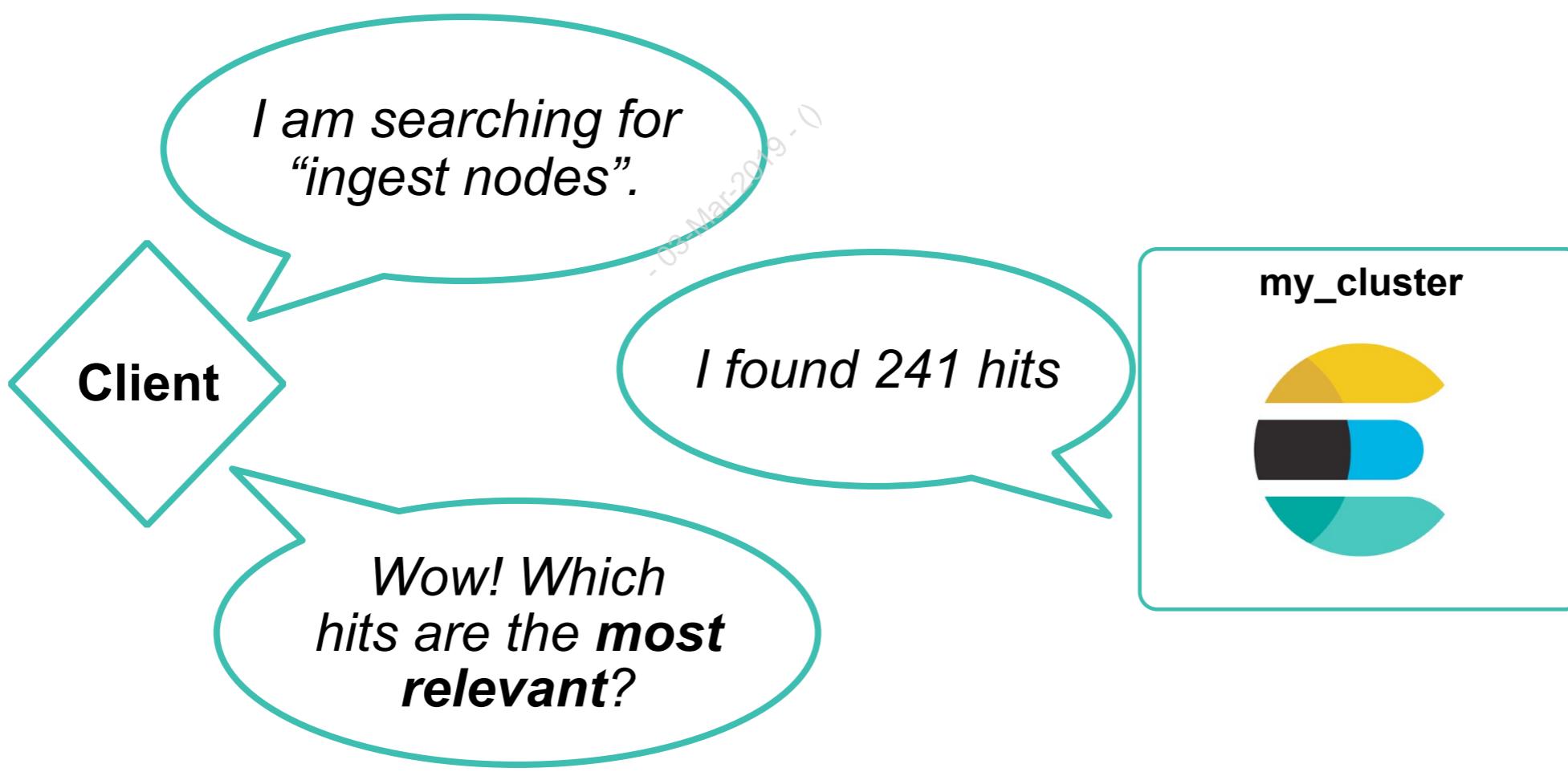


Relevance

~03-Mar-2019

Relevance

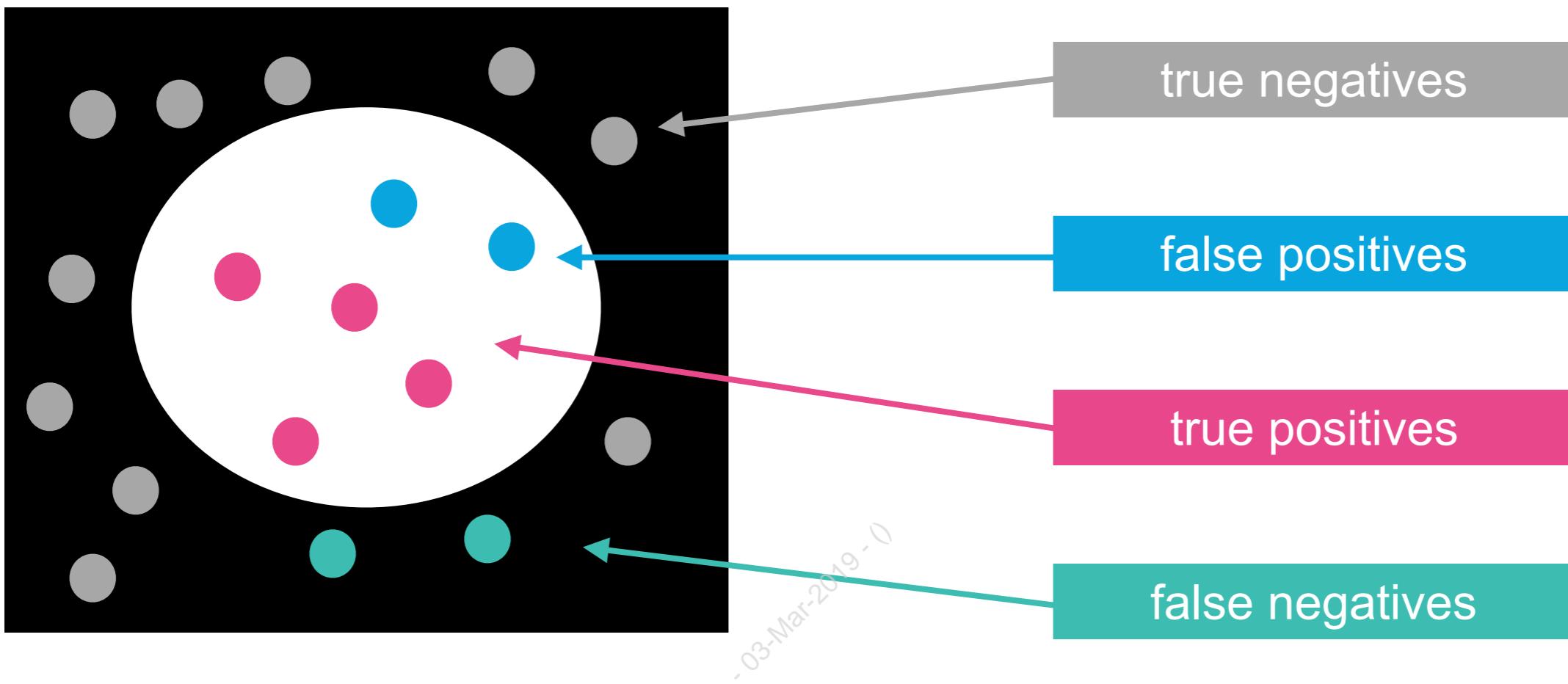
- When querying data, we are interested in the most *relevant* documents
 - Are you able to find all the documents you are looking for?
 - How many irrelevant documents are returned?
 - How well are the documents ranked?



Measuring Relevance

- **Precision**
 - did we get irrelevant results in addition to the relevant results?
- **Recall**
 - did we miss any relevant results that were not returned?
- **Ranking**
 - are the results ordered from most relevant at the top, to least relevant at the bottom?
- Elasticsearch uses a **score** to determine the ranking of the matching documents

Precision and Recall



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

Improving Precision and Recall

- **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



Searching for Terms

~03-Mar-2011~

Searching Documents

- Suppose we are interested in learning about “*ingest nodes*”:

<i>Blog Title</i>	<i>Relevant?</i>
"Ingest Node: A Client's Perspective"	yes
"A New Way To Ingest - Part 1"	yes
"A New Way To Ingest - Part 2"	yes
"Ingest Node to Logstash Config Converter"	kind of
"Monitoring Kafka with Elastic Stack: Filebeat"	kind of
"Elasticsearch for Apache Hadoop 5.0.0-beta1"	no
"Logstash 6.0.0 GA Released"	no

Even though the fourth line looks like a close match, it is not very relevant. Using the DSL we can improve our ranking and relevancy.



Query String vs. Query DSL

- ***Query string*** is a simplified approach to search documents
 - can be used in the URL
 - simple and easy-to-use
 - but hard to write complex queries
 - unforgiving for small typos (quotes, parentheses, ...)
- ***Query DSL*** (Domain Specific Language) allows you to write queries in a JSON format
 - must send a body
 - exposes the entire collection of Elasticsearch APIs
 - very powerful



The match Query

- The *match* query is a simple but powerful search for fields that are text, numerical values, or dates
- This is your “*go to*” query
 - and often the starting point of your searches
- The syntax looks like:

```
GET my_blogs/_search
{
  "query": {
    "match": {
      "FIELD": "TEXT"
    }
  }
}
```

The field you want to
search

The text you want to
search for

Let's Search for Terms

- Let's search for “*ingest nodes*” 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"
    }
  }
}
```

Query DSL

```
GET blogs/_search?q=content:(ingest nodes)
```

query string

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": 241,           ← 241 documents were a match  
  "max_score": 9.3005905,  
  "hits": [  
    {  
      "_index": "blogs",  
      "_type": "_doc",  
      "_id": "7ItcB2cBZWSWiUi4z_dx",  
      "_score": 9.3005905,           ← Each hit has a _score  
      "_source": {  
        "title": "A New Way To Ingest - Part 2"  
      }  
    },  
    {  
      "_index": "blogs",  
      "_type": "_doc",  
      "_id": "2otcB2cBZWSWiUi41Pkb",  
      "_score": 8.616863,           ← Results are sorted by  
      "_source": {  
        "title": "Ingest Node: A Client's Perspective"  
      }  
    },  
    ...  
  ]  
}
```



Analyze the Hits

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

"A New Way To Ingest - Part 2"

Relevancy is good here, but the part 1 is missing.

"Ingest Node: A Client's Perspective"

Precision was good here, but it does not answer the question.

"A New Way To Ingest - Part 1"

Relevancy is very good here, but it was hit #3



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”

```
GET blogs/_search?q=content:(ingest AND nodes)
```



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

```
GET blogs/_search?q=content:(ingest nodes logstash)
```



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 nodes logstash",
        "minimum_should_match": 2
      }
    }
  }
}
```

Only 82 hits this time

2 of the 3 terms
need to match



Scoring

~03-Mar-2018



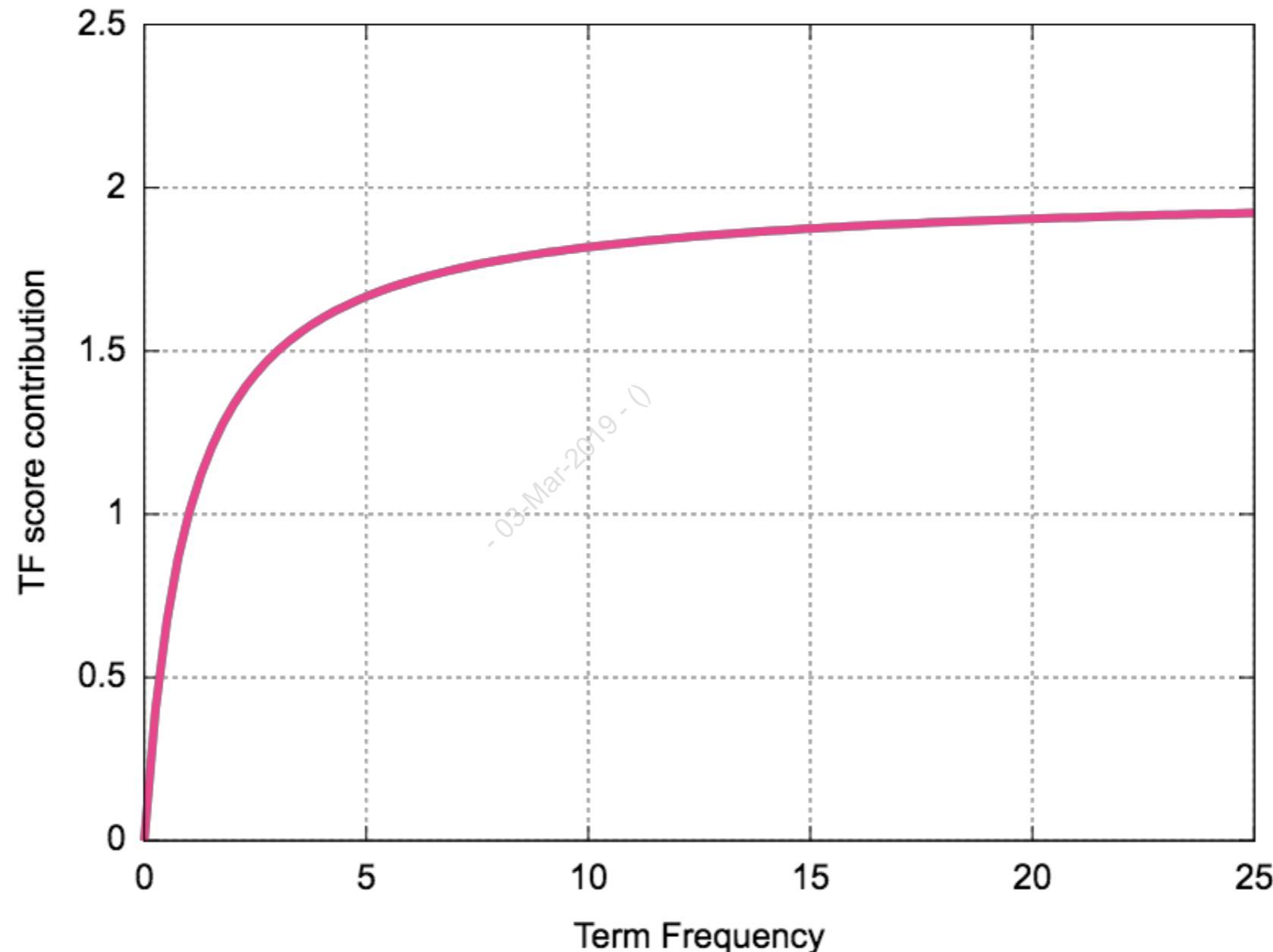
Score!

- 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



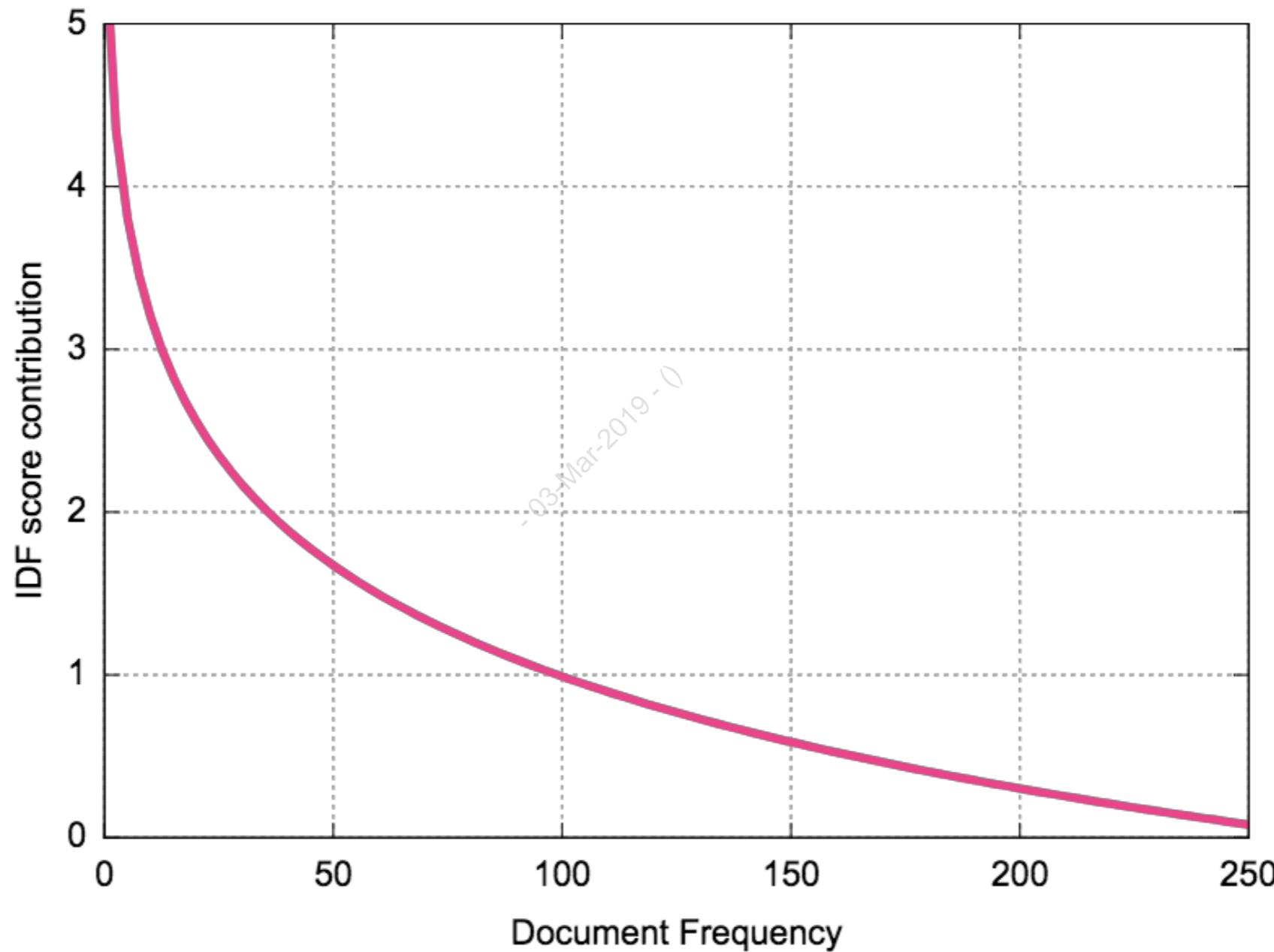
Term Frequency

- 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



Searching for Phrases

„03-Mar-2019“

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:

"Ingest Node: A Client's Perspective"

"Elasticsearch for Apache Hadoop 5.0.0-beta1"

"A New Way To Ingest - Part 1"

"Monitoring Kafka with Elastic Stack: Filebeat"

"A New Way To Ingest - Part 2"

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
- A `match_phrase` query can be quite expensive! Use them wisely

```
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**:

```
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

Only 8 hits this time

“A New Way To Ingest - Part 1”

“A New Way To Ingest - Part 2”

“Logstash 6.0.0 GA Released”

“Elastic Ingest Node: A Client's Perspective”

```
GET blogs/_search?q=content:( "ingest nodes" )
```

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"
    }
  }
}
```

```
GET blogs/_search?q=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”

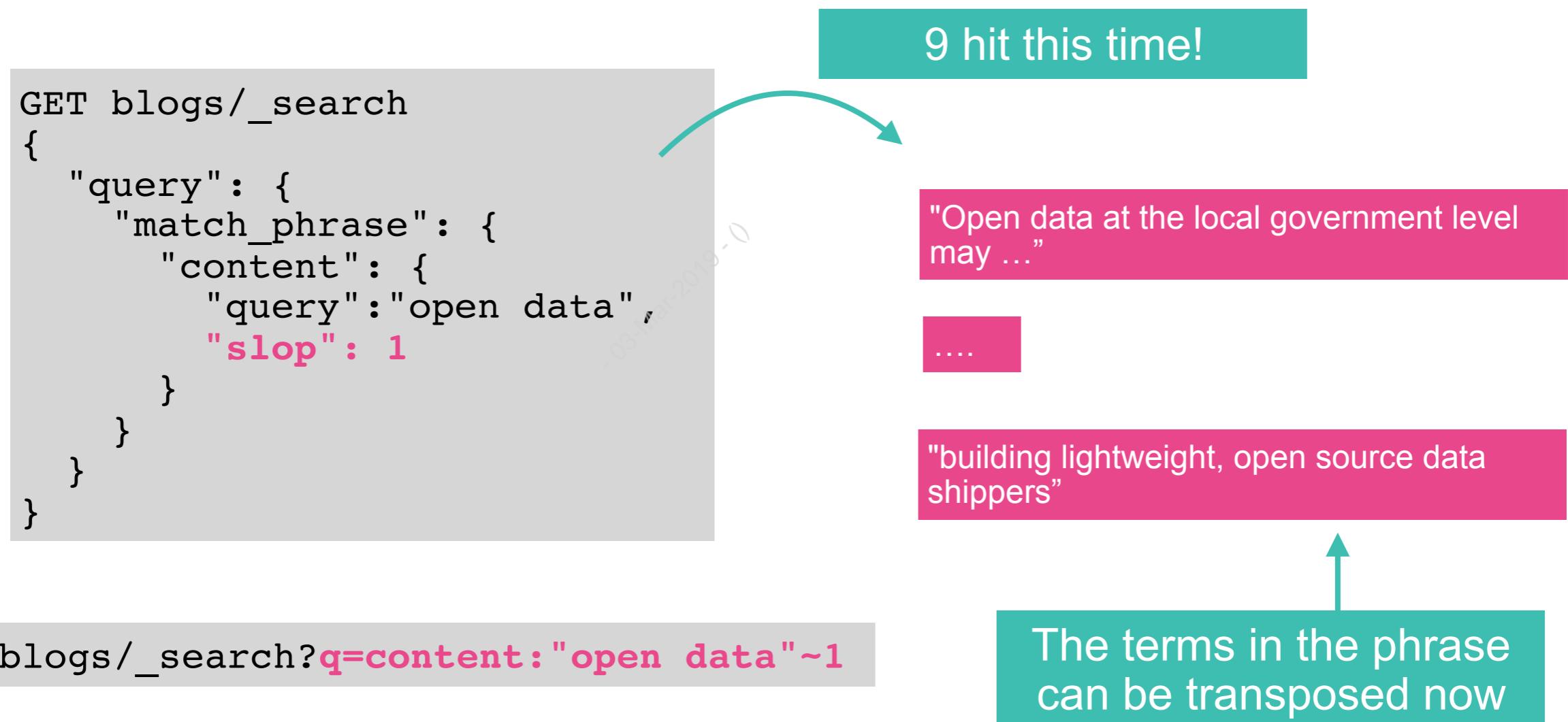
“Open data at the local government level
may seem “smaller” ...”

“...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



Searching within Date Ranges

,03-Mar-2019, 0

Specifying Date Ranges

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

A screenshot of a search interface. At the top, there is a search bar containing the text "logstash". Below the search bar, there is a section titled "Categories" which contains four items: "Engineering (3)", "Releases (3)", "The Logstash Lines (2)", and "Brewing in Beats (1)". Below the categories, there is a section titled "Dates (yyyy-MM-dd)" with two input fields: "from: 2017-12-01" and "to: 2017-12-31". A teal callout box with the text "Filter by a date range" is positioned to the left of the "from" field, with a thin teal line pointing from the center of the box to the "from" field. A faint watermark "03-Mar-2019 - 0" is visible diagonally across the interface.



The range Query on Dates

- Range queries work on **date** fields
 - You can specify the dates as a string
 - Or use a special syntax called **date math**

```
GET blogs/_search
{
  "query": {
    "range": {
      "publish_date": {
        "gte": "2017-12-01",
        "lt": "2018-01-01"
      }
    }
  }
}
```

*"I am looking for
blogs from December,
2017"*



Date Math

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

```
GET blogs/_search
{
  "query": {
    "range": {
      "publish_date": {
        "gte": "now-3M"
      }
    }
  }
}
```

"I am looking blogs from the last three months."

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

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

If now = 2017-10-19T11:56:22	
now-1h	2017-10-19T10:56:22
now+1d	2017-10-20T11:56:22
now+1h+30m	2017-10-19T13:26:22
2018-01-15 +1M	Jan 15, 2018, plus 1 month



Combining Searches

~03-Mar-2018

Combining Searches

- Suppose we want to write the following query:
 - *Find blogs about the “Elastic Stack” published before 2017*
- This search is actually a combination of two queries:
 - We need “Elastic Stack” in the **content** or **title** field,
 - and the **publish_date** field needs to be before 2017
- 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

- Let's answer the question we posed earlier:
 - Find blogs about the “Elastic Stack” published before 2017*

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": {
        "match_phrase": {
          "content": "elastic stack"
        }
      },
      "filter": {
        "range": {
          "publish_date": {
            "lt": "2017-01-01"
          }
        }
      }
    }
  }
}
```

98 hits

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>Exclude docs</i>	<i>Scoring</i>
must	Y	Y
must_not	Y	N
should	N*	Y
filter	Y	N



Improving Relevancy

·03-Mar-2019·
e



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

~03-Mar-2019~ Ø



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:

The screenshot shows an Elasticsearch search request and its results. On the left, a grey box contains the search query in JSON format. On the right, a teal box at the top displays the total number of hits. Below it, several pink boxes list the titles of the matching documents.

Search Query (Grey Box):

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"title": "elastic"}}
      ],
      "should": [
        {"match": {"title": "stack"}},
        {"match": {"title": "query"}},
        {"match": {"title": "speed"}}
      ]
    }
  }
}
```

Results (Teal Box): 259 hits

Document Titles (Pink Boxes):

- “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”

Annotation (Teal Box): This will cast a wide net because no **should** clause needs to match



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:

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"title": "elastic"}}
      ],
      "should": [
        {"match": {"title": "stack"}},
        {"match": {"title": "query"}},
        {"match": {"title": "speed"}}
      ],
      "minimum_should_match": 1
    }
  }
}
```

Only 71 hits this time

`_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:

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

At least one clause must match for a hit



A “should not” Query

- The following example implements “**should not**” logic (notice how you can nest **bool** queries):

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": {"match": {"title": "elastic"}},
      "should": {
        "bool": {
          "must_not": {
            "match": {
              "title": "stack"
            }
          }
        }
      }
    }
  }
}
```

*I am looking for blogs that contain “**elastic**” but should not contain “**stack**”.*

259 hits



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



Chapter Review

·03-Mar-2019·0

Summary

- **Precision** refers to how many irrelevant results are returned in addition to the relevant results
- **Recall** refers to how many relevant results are not returned
- **Query DSL** (Domain Specific Language) allows you to write queries in a JSON format
- The **match** query is a simple but powerful search for fields that are text, numerical values, or dates
- The **match_phrase** query is for searching text when you want to find terms that are near each other
- The **range** query is for finding documents with fields that fall in a given range
- The **bool** query allows you to combine queries using Boolean logic



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. How could you improve the precision of a **match** query that consists of 5 terms?
4. 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?
03-Mar-2019
5. **True or False:** If a document matches a **filter** clause, the score is increased by a factor of 2.



Lab 4

Querying Data

03-Mar-2019



Chapter 5

Text Analysis and Mappings

· 03-Mar-2019 · 0

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- What is a Mapping?
- Analysis and the Inverted Index
- Multi-Fields
- Analyzers

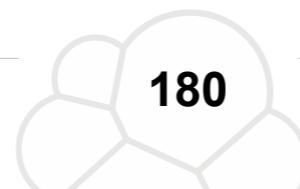
~03-Mar-2019~ Ø

What is a Mapping?

·03-Mar-2019·

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 a mapping:

GET my_index/_mapping

This particular mapping has a **type** named **_doc**

The "**properties**" section contains the fields and data types in your documents

```
{  
  "my_index": {  
    "mappings": {  
      "_doc": {  
        "properties": {  
          "comment": {  
            "type": "text",  
            "fields": {  
              "keyword": {  
                "type": "keyword",  
                "ignore_above": 256  
              }  
            }  
          },  
          "details": {  
            "properties": {  
              "created_at": {  
                "type": "date"  
              },  
              "employee": {  
                "type": "boolean"  
              },  
              ...  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

What's with the *type* section?

- Each document has a *type*
 - Prior to 6.0: an index could have multiple types
 - Since 6.0: an index can only have a single type
- Types were a design mistake that have caused issues with some users
 - so we are moving away from types
 - moving forward, you will eventually need to redesign your old indices to only have a single document type
- <https://www.elastic.co/guide/en/elasticsearch/reference/current/removal-of-types.html>



Elasticsearch Data Types for Fields

- **Simple Types, including:**
 - **text**: for full text (analyzed) strings
 - **keyword**: for exact value strings
 - **date**: 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** and **percolator**
- Plus **range** types and more

we will talk about
these next

Defining a Mapping

- In most 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, or
 - **add to** a mapping of an existing index

~03-Mar-2019-0

```
PUT comments
{
  "mappings": {
    "define mapping 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

```
PUT /my_index/_doc/1
{
  "username": "kimchy",
  "comment": "Search is something that any application should have",
  "details": {
    "created_at": "2010-04-23T15:48:50",
    "visibility": "public",
    "employee": true
  }
}
```

The diagram illustrates the dynamic field mapping process. It shows a PUT request to create a document in an index. The document contains several fields: 'username' (string), 'comment' (string), 'details' (object), and 'details.created_at' (date). The 'details' object itself contains 'visibility' (string) and 'employee' (boolean). Arrows from the field names to teal boxes indicate the inferred types: 'username', 'comment', and 'details.visibility' are mapped to 'strings'; 'details.created_at' is mapped to 'date'; and 'details.employee' is mapped to 'boolean'. A watermark '03-Mar-2019 - 0' is present.



Elasticsearch “guesses” data types:

Sometimes it conveniently
guesses what you want...

```
PUT comments/_doc/2
{
  "num_of_views" : 430
}
```

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

Elasticsearch “guesses” data types:

Dates in the default format are also recognized

```
PUT comments/_doc/3
{
  "comment_time" : "2016-11-06"
```

```
"comment_time": {
  "type": "date"
}
```

Elasticsearch “guesses” data types:

Sometimes it may choose a type you do **not** want...

```
PUT comments/_doc/4
{
  "average_length" : "24.8"
```

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

- Strings are mapped twice: once as type **text** and once as type **keyword**
- To explain what these are we will talk about **text analysis** first...

Analysis and the Inverted Index

-03 Nov 2019 - 0

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 *analysis* which occurs when your fields are indexed

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

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

These two queries return
the same hits with the
same scoring



What is an Inverted Index?

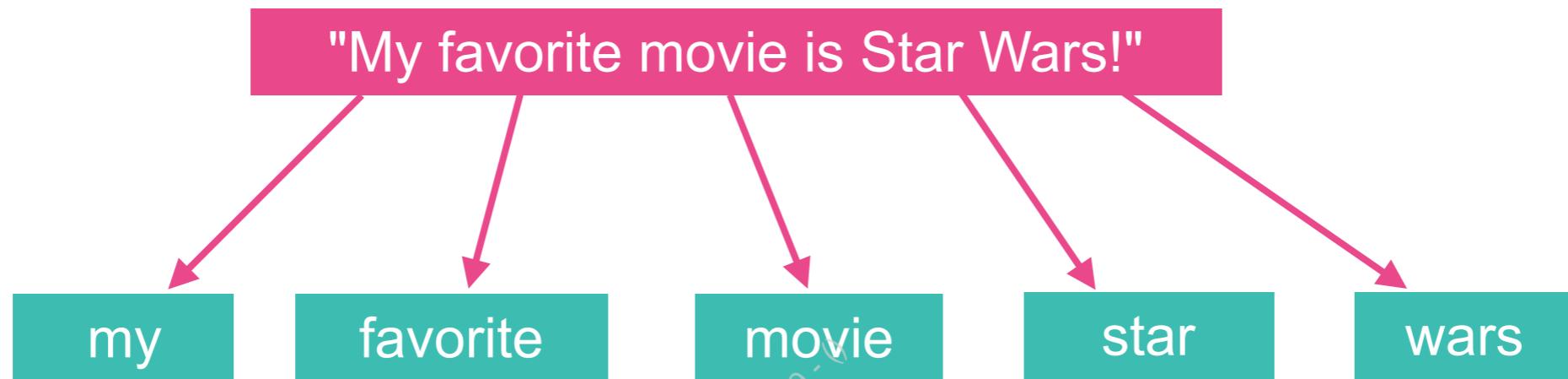
- You are familiar with the index in the back of a book
 - Useful terms from the book are sorted, and page numbers tell you where to find that term
- Lucene creates a similar *inverted index* with your text fields

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

Analysis Makes Full Text Searchable

- An inverted index gets built when you index data
- At that time, full text gets *analyzed*



- Similarly, the text in your queries gets analyzed (typically using the same analyzer)



Building an Inverted Index

- Let's look at how text is analyzed and added to an inverted index:

- text is broken into tokens
- indexed by a unique document id

1 The quick brown fox jumps over the lazy dog

2 Fast jumping spiders



term	id
Fast	2
The	1
brown	1
dog	1
fox	1
jumping	2
jumps	1
lazy	1
over	1
quick	1
spiders	2
the	1



Lowercase

- Searching for “Fast” and “fast” should yield the same results
 - common to lowercase all tokens

1 The quick brown fox jumps over the lazy dog

2 Fast jumping spiders

“the” was already
indexed for doc 1

term	id
brown	1
dog	1
fast	2
fox	1
jumping	2
jumps	1
lazy	1
over	1
quick	1
spiders	2
the	1

Stop Words

- Searching for “the” rarely makes sense
 - typically we do not bother to index “the”
 - or other **stop words**

- 1 The quick brown fox jumps over the lazy dog
- 2 Fast jumping spiders

term	id
brown	1
dog	1
fast	2
fox	1
jumping	2
jumps	1
lazy	1
over	1
quick	1
spiders	2

Stemming

- “jumps” and “jumping” share the same stem: “jump”
 - want “jump” to match both
 - so only index the stem

- 1 The quick brown fox jumps over the lazy dog
- 2 Fast jumping spiders

term	id
brown	1
dog	1
fast	2
fox	1
jump	1,2
lazy	1
over	1
quick	1
spider	2

Analyze the Search Query Too

- If we are going to analyze our text during indexing, we better also analyze our search queries in the same manner



token	id
brown	1
dog	1
fast	2
fox	1
jump	1,2
lazy	1
over	1
quick	1
spider	2

Multi-Fields

~03-Mar-2019 10:00



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 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***
- You can override this behavior by defining an explicit mapping for your string fields



Understanding Multi-Fields

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

```
PUT comments/_doc/20
{
  "comment": "I like Elasticsearch!"
}
```

“comment” is analyzed

03-Mar-2019 - 0

i
like
elasticsearch

“comment.keyword” is not analyzed

"I like Elasticsearch!"

Inverted Index per Text Field

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

```
"properties": {  
    "comment": {  
        "type": "text",  
        "fields": {  
            "keyword": {  
                "type": "keyword",  
                "ignore_above": 256  
            }  
        }  
    },  
    "movie": {  
        "type": "text",  
        "fields": {  
            "keyword": {  
                "type": "keyword",  
                "ignore_above": 256  
            }  
        }  
    },  
    "username": {  
        "type": "text",  
        "fields": {  
            "keyword": {  
                "type": "keyword",  
                "ignore_above": 256  
            }  
        }  
    }  
}
```

How many inverted indexes are built from this mapping?



Inverted Index per Text Field

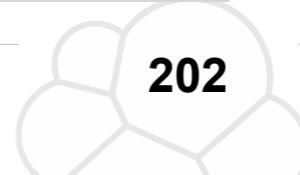
- This mapping has 6 inverted indexes that can be queried
 - Notice “comment”, “movie” and “username” each have two:

```
"properties": {  
    "comment": {  
        "type": "text",  
        "fields": {  
            "keyword": {  
                "type": "keyword",  
                "ignore_above": 256  
            }  
        }  
    },  
    "movie": {  
        "type": "text",  
        "fields": {  
            "keyword": {  
                "type": "keyword",  
                "ignore_above": 256  
            }  
        }  
    },  
    "username": {  
        "type": "text",  
        "fields": {  
            "keyword": {  
                "type": "keyword",  
                "ignore_above": 256  
            }  
        }  
    }  
}
```

The “**fields**” section is for defining ***multi-fields***

Two different inverted indices are built: “**movie**” and “**movie.keyword**”

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



Analyzers

~03-Mar-2019 10



Anatomy of an Analyzer

- An analyzer consists of three parts:
 1. Character Filters
 2. Tokenizer
 3. Token Filters



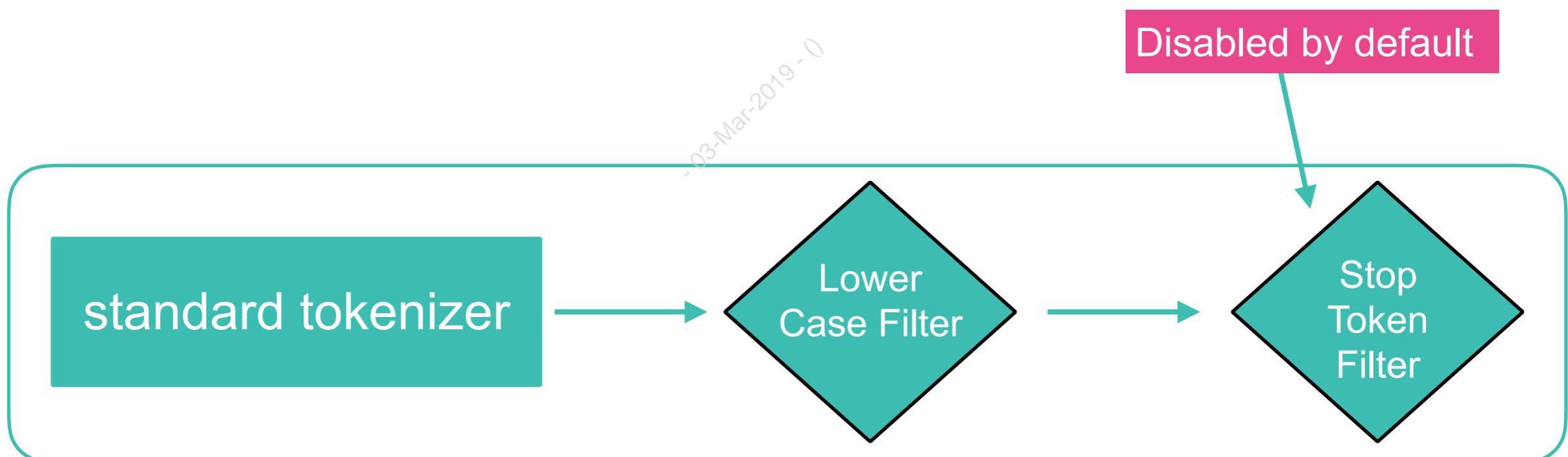
Built-in Analyzers

- **standard** - which is the default analyzer
- **simple** - breaks text into terms whenever it encounters a character which is not a letter
- **keyword** - simply indexes the text exactly as is
- Others include:
 - **whitespace, stop, pattern, language**, 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 often you may need to define your own analyzers



The standard Analyzer

- The **standard** analyzer is the default in Elasticsearch
 - it has no character filters
 - uses the **standard** tokenizer
 - it lowerscases all tokens, and optionally removes stop words



The Analyze API

- Use the `_analyze` API to test what an analyzer will do to your text:

Use the `_analyze` API

```
GET _analyze
{
  "analyzer": "simple",
  "text": "My favorite movie is Star Wars!"
}
```

Test the “simple” analyzer on the given “text”

“My favorite movie is Star Wars!”

“simple”
analyzer

my

favorite

movie

is

star

wars



Chapter Review

·03-Mar-2019·0

Summary

- ***Analysis*** is the process of converting full text into terms for the inverted index
- Every type has a ***mapping*** that defines a document's fields and their data types
- Mappings are created ***dynamically***, or you can define your own
- ***Multi-fields*** allow a field to be indexed in multiple ways

03-Mar-2019



Quiz

1. The process of converting full text into terms for the inverted index is called _____
2. What data type would “**value**” : 300 get mapped to dynamically?
3. What data type would “**value**”: “January” get mapped to dynamically?
4. **True or False:** You can change a field’s data type from “**integer**” to “**long**” because those two types are compatible.
5. What are the three components that make up an analyzer?

Lab 5

03-Mar-2019

Text Analysis and Mappings



Chapter 6

Custom Mappings

· 03-Mar-2019 · 0

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- Defining a Custom Analyzer
- Improving the Blogs Mapping
- Defining Explicit Mappings

~03-Mar-2019~0

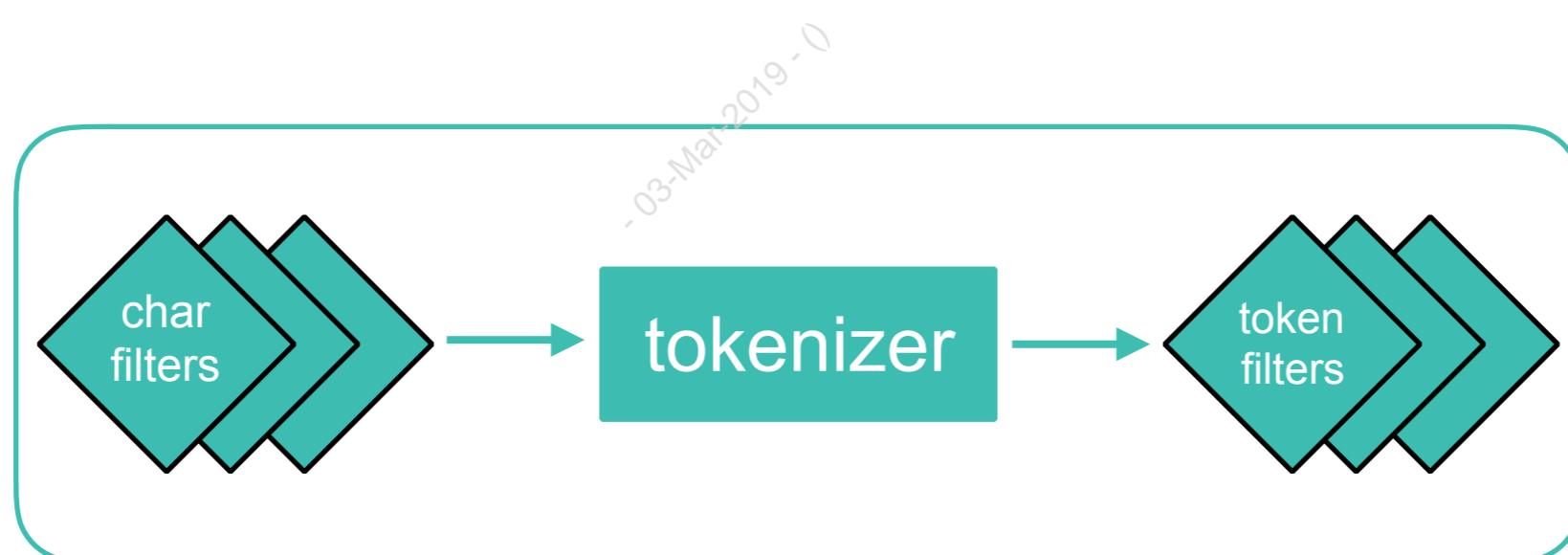


Defining a Custom Analyzer

-03-Mar-2019-

Custom Analyzers

- You can build your own custom analyzer, out of the three building blocks:
 1. Character Filters
 2. Tokenizer
 3. Token Filters



Custom Analyzers

- Let's say we want to have an analyzer that is similar to the standard analyzer, but which also drops stop words:

```
PUT blogs_test2
{
  "settings": {
    "analysis": {
      "filter": {
        "my_stopwords": {
          "type": "stop",
          "stopwords": ["to", "and", "or", "is", "the"]
        }
      },
      "analyzer": {
        "my_content_analyzer": {
          "type": "custom",
          "char_filter": [],
          "tokenizer": "standard",
          "filter": ["lowercase", "my_stopwords"]
        }
      }
    }
  }
}
```

List the stop words that need to be removed in the analysis chain

Define a custom analyzer that uses the stop filter



Test the Custom Analyzer

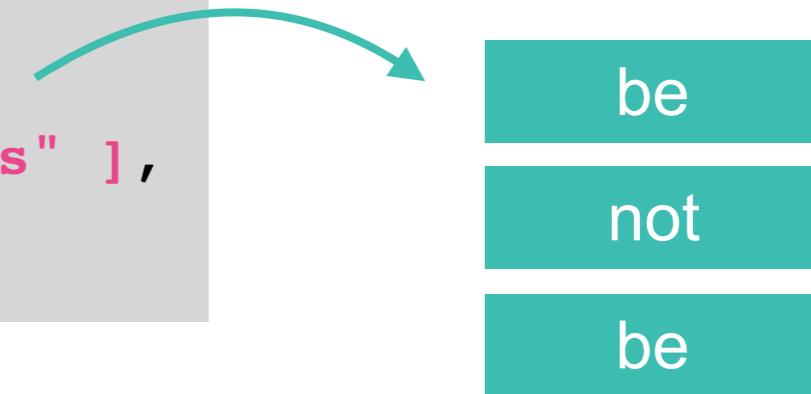
```
POST blogs_test2/_analyze
{
  "text": ["To be or not to be"],
  "analyzer": "my_content_analyzer"
}
```



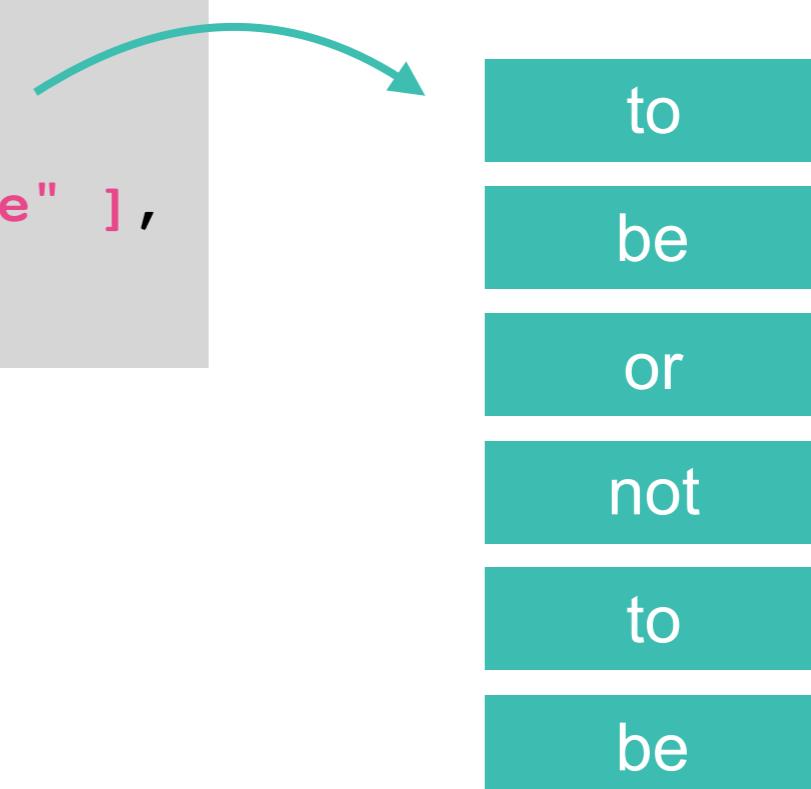
be
not
be

Order Matters in Filters

```
GET blogs_test2/_analyze
{
  "tokenizer": "standard",
  "filter": ["lowercase", "my_stopwords"],
  "text": "To Be Or Not To Be"
}
```



```
GET blogs_test2/_analyze
{
  "tokenizer": "standard",
  "filter": [ "my_stopwords", "lowercase" ],
  "text": "To Be Or Not To Be"
}
```



Improving the Blogs Mapping

-03-Mar-2019-Q

Current Blogs Mapping

- Currently, our blog dataset is mapped with the default data types and analyzers:

Our **text** fields do not specify any analyzers, so **standard** is used by default

keyword fields are not analyzed at all

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



How Could We Improve the Blogs Mapping?

- To answer this question, you need to understand how each field is going to be queried in your application
- For our blog dataset, we want users to be able to:
 - search for a term or phrase within the “**content**” or “**title**” field
 - search by date using the “**publish_date**” field
 - search by “**author**”
 - filter by “**locales**” and/or “**categories**”



The “category” Field

- The “category” field currently is set to one of 10 different values:

Engineering

Releases

This week in Elasticsearch

User Stories

News

Brewing in Beats

The Logstash Lines

Culture

Kurrently in Kibana

<blank>

What is a good way to map a fixed set of text values?

~03-Mar-2019~0



The “category” Field

- For our use case, it would probably make sense to map this field as a **keyword** type
 - searches will have to be exact matches
 - but we will create a user interface where users do not have to type in a category (use a checkbox or drop-down)

```
"properties": {  
    ...  
    "category": {  
        "type": "keyword"  
    },  
    ...  
}
```



The “content” Field

- What is probably the best choice for mapping the “content” field?
 - How would you map a large field of raw text?

"Today, we are incredibly excited to announce the Elastic Stack Monitoring Service. This service extends our commitment to improving product usability and quality of support by providing you with a dedicated monitoring cluster to host your Elastic Stack monitoring data. This service is available to our Gold and Platinum self-hosted customers at no additional charge."



The “content” Field

- We could start with the **standard** analyzer and see how it works
 - Over time, we will likely need to **configure a custom analyzer**
 - And maybe **add multi-fields** to analyze the text in different ways

```
"properties": {  
    ...  
    "content": {  
        "type": "text",  
        "analyzer": "standard",  
        "fields": {  
            "english": {  
                "type": "text",  
                "analyzer": "english"  
            }  
        }  
    },  
    ...  
}
```

Use “analyzer” to specify a field’s analyzer

You can use you custom analyzer defined in settings



The “author” Field

- We want users to:
 - **search by author**, but it is unlikely they will enter a person's first and last name exactly
 - be able to **filter by author**, so an exact match is needed
- How could we satisfy both of these criteria?

"author": "Bohyun Kim"

The “author” Field

- Seems like a perfect candidate for mapping as **text** and **keyword**:

```
"properties": {  
    ...  
    "author": {  
        "type": "text",  
        "fields": {  
            "keyword": {  
                "type": "keyword"  
            }  
        }  
    },  
    ...  
}
```

03-Mar-2019

```
GET blogs/_search  
{  
    "query": {  
        "match": {  
            "author": "kim"  
        }  
    }  
}  
  
GET blogs/_search  
{  
    "query": {  
        "bool": {  
            "filter": {  
                "match": {  
                    "author.keyword": "Bohyun Kim"  
                }  
            }  
        }  
    }  
}
```



The “publish_date” Field

- OK, this one is easy, especially since Elasticsearch guessed the correct date type for us!

```
"publish_date": "2017-06-29T00:00:00.000Z"
```

```
"properties": {  
    ...  
    "publish_date": {  
        "type": "date"  
    },  
    ...  
}
```



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",  
        "format": "dd/MM/yyyy"  
    },  
    "comment_time" : {  
        "type": "date",  
        "format" : "basic_date||epoch_millis"  
    }  
}
```

Use || for multiple formats

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



The “locales” Field

- “locales” is a comma-separated list of strings
 - Searches will most likely be for a single locale
 - And the specific value of each locale probably does not need analysis

```
"locales": "de-de,fr-fr,ja-jp"
```

The “locales” Field

- We should use an **array** of values for this field
 - and map each entry as **keyword**
- In Elasticsearch, there is no dedicated array type
 - But, **any field can contain multiple values** (as long as each value in the array is the same data type)

"locales": "fr-fr"

This field is a single keyword

"locales": ["de-de", "fr-fr", "ja-jp"]

This field is an array of keywords (both formats work)



The “locales” Field

- The mapping for “locales” is simply as a **keyword** type
 - because arrays have no special mapping required

```
"properties": {  
    ...  
    "locales": {  
        "type": "keyword"  
    },  
    ...  
}
```

- But we have a different problem right now!
 - The “locales” data did not get ingested as an array
`"locales": "de-de,fr-fr,ja-jp"` ← We did not index it as an array!
- We will need some special processing to split the comma-separated values into an array (discussed in *Engineer II*)



Can you change a mapping?

- **No** - not without reindexing your documents
 - adding fields is possible
 - ... all other schema 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



Defining Explicit Mappings

Q3-Mar-2019 - 0

Defining Explicit 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
 - but use a type name that you want to keep

```
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": {  
      "_doc": {  
        "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**”

The diagram illustrates the modification of Elasticsearch mapping fields. On the left, a large gray box contains the original mapping code. On the right, a smaller gray box shows the updated mapping code. Two teal arrows point from the original code to the updated one, indicating the changes made.

Original Mapping (Left):

```
"category": {  
    "type": "text",  
    "fields": {  
        "keyword": {  
            "type": "keyword",  
            "ignore_above": 256  
        }  
    }  
},  
"content": {  
    "type": "text",  
    "fields": {  
        "keyword": {  
            "type": "keyword",  
            "ignore_above": 256  
        }  
    }  
},  
...  
}
```

Updated Mapping (Right):

```
"category": {  
    "type": "keyword"  
},  
"content": {  
    "type": "text"  
},  
"date": {  
    "type": "date",  
    "format": ["M dd, yyyy"]  
},  
"locales": {  
    "type": "keyword"  
},  
...
```



4. Define a New Index with the Mapping

```
PUT blogs
{
  "mappings": {
    "_doc": {
      "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!



Chapter Review

·03-Mar-2019·0

Summary

- Use the `_analyze` API to test an analyzer (or the components individually)
- In most use cases, you will need to define your own mappings
- A useful trick to create a custom mapping is to index a document in a dummy index and to use the default mapping as a base for the new mapping

Quiz

- 1. True or False:** When defining an analyzer, token filters are applied in the order they are listed.
- 2. True or False:** A custom analyzer needs to be defined in the settings of an index.
- 3. True or False:** It is possible to define multiple tokenizers in an analyzer.

~03-Mar-2019~ Ø



Lab 6

Custom Mappings

03-Mar-2019 10



Chapter 7

Node Types

,03-Mar-2019, 0

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- Node Roles
- Cluster State and Master Nodes
- Data Nodes
- Sample Architectures

~03-Mar-2019~ Ø



Node Roles

-03-Mar-2019-



Node Roles

- There are several roles a node can have:
 - Master eligible
 - Data
 - Ingest
 - Coordinating
 - Machine Learning
- Nodes can take on multiple roles at the same time
 - or they can be ***dedicated*** nodes that only take on a single role
 - ingest and coordinating only nodes are covered in detail in the ***Engineer II*** course



Configuring Node Roles

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

Node type	Configuration parameter	Default value
master eligible	node.master	true
data	node.data	true
ingest	node.ingest	true



Cluster State and Master Nodes

13-Mar-2019-0

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:

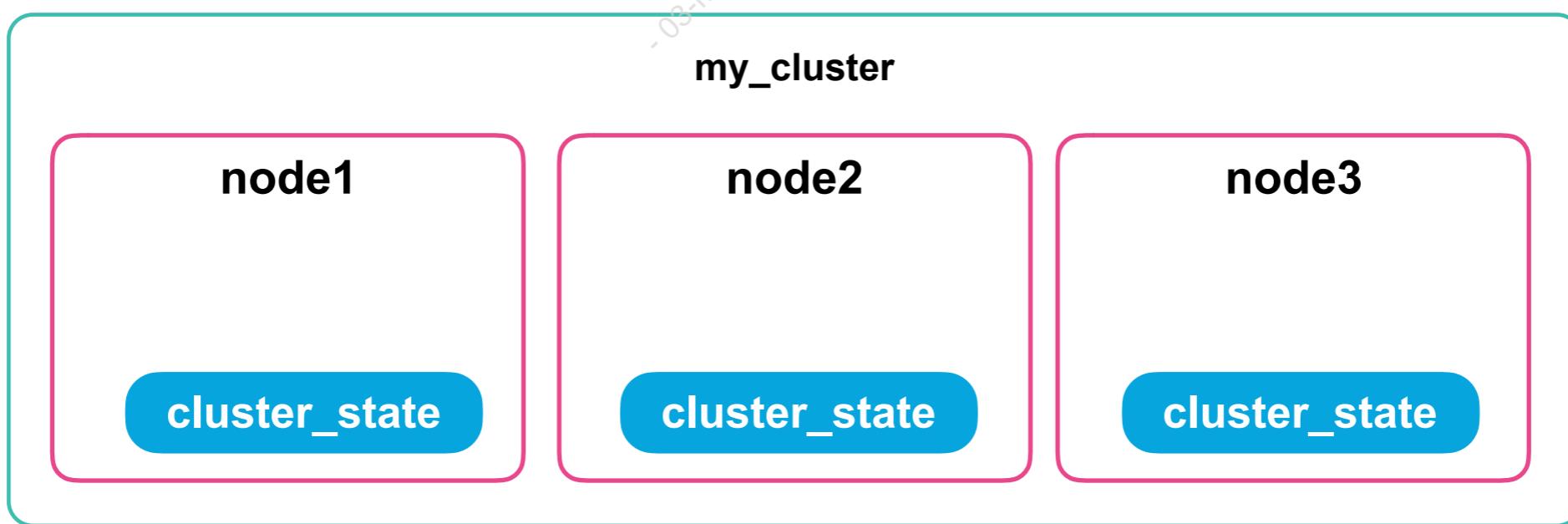
GET `_cluster/state`

```
{  
  "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": {...}  
}
```



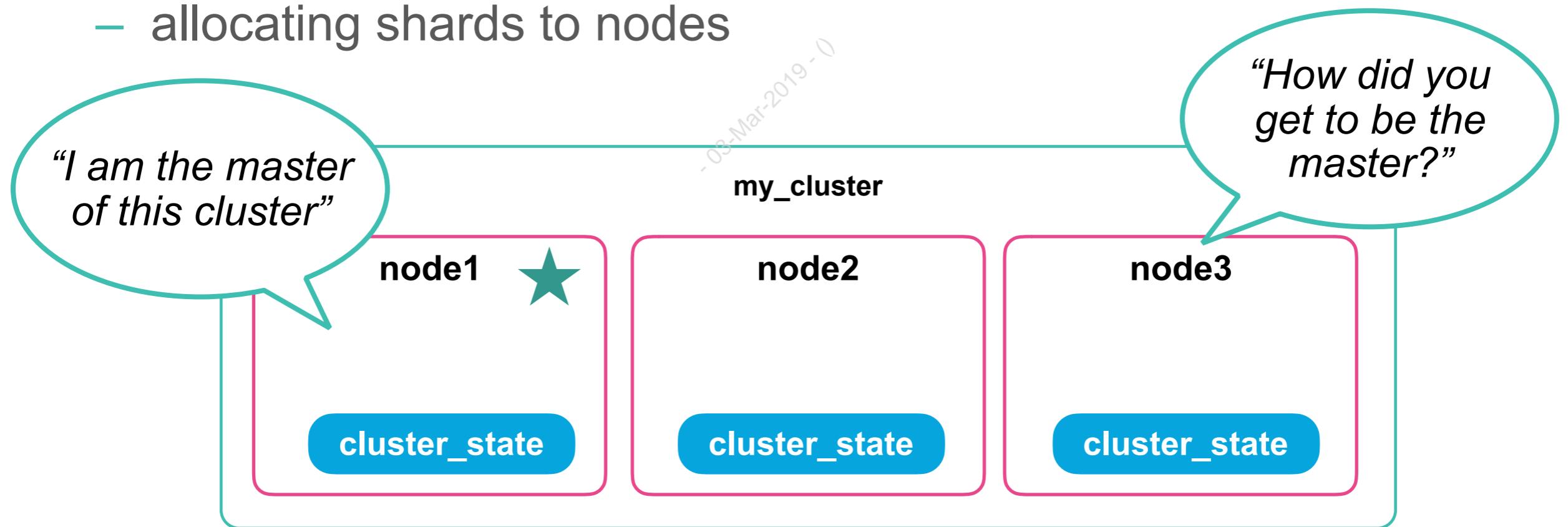
Cluster State

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



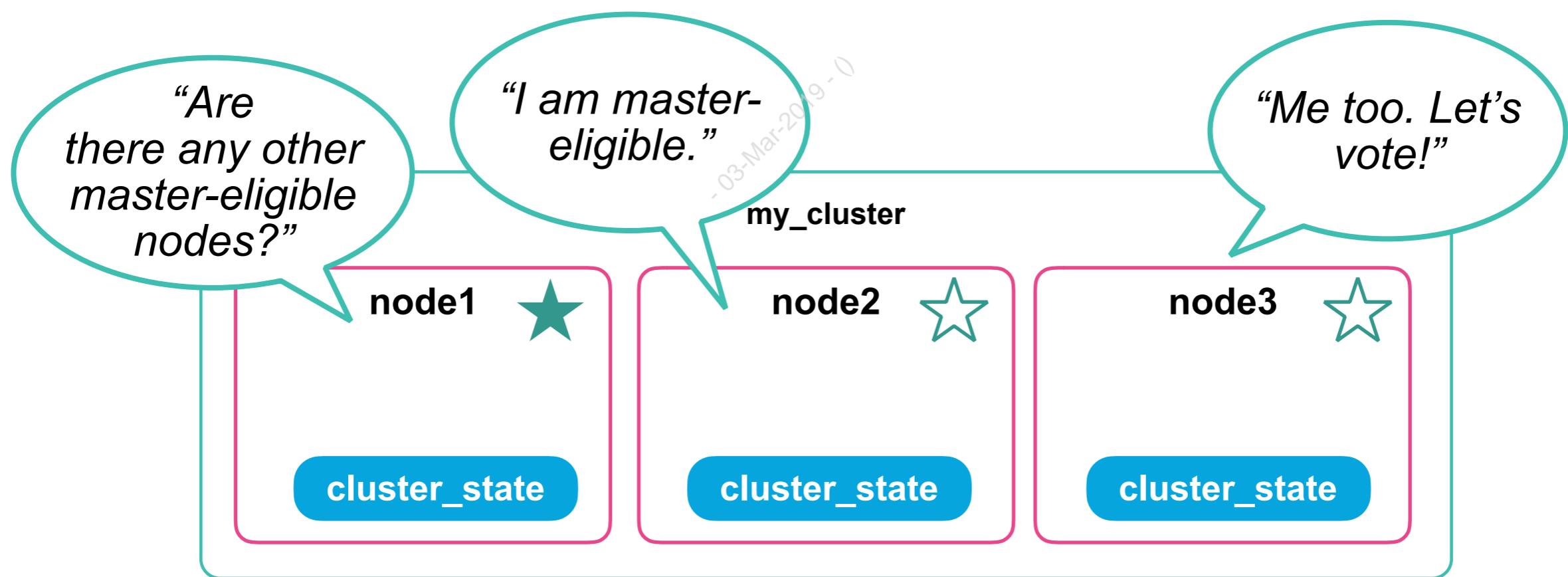
The Master Node

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



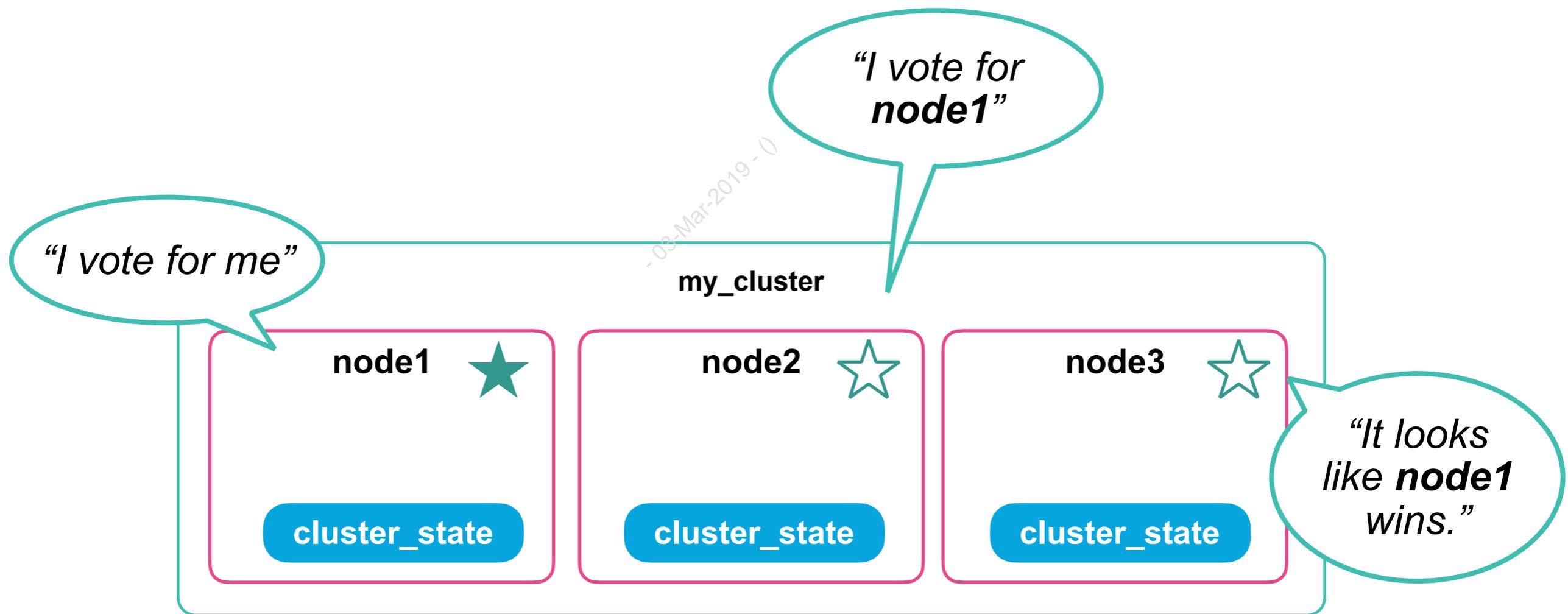
Master-eligible Nodes

- 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)
 - each master-eligible node votes



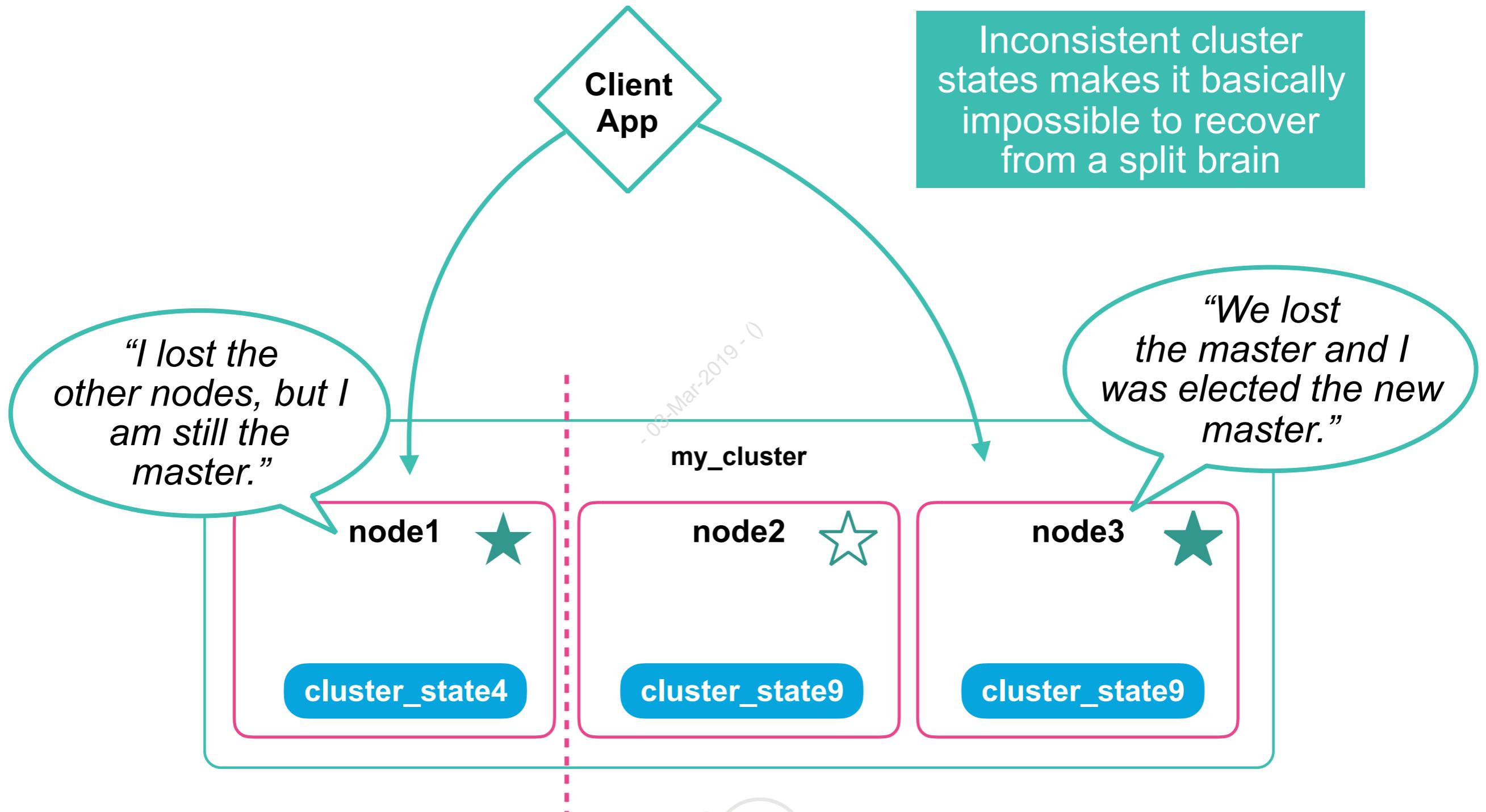
Master-eligible Nodes Election

- The number of votes to win the election is configured with the `discovery.zen.minimum_master_nodes` setting
 - set it to $N/2+1$, where N is the number of master-eligible nodes
- Why is it important to have a **quorum**?



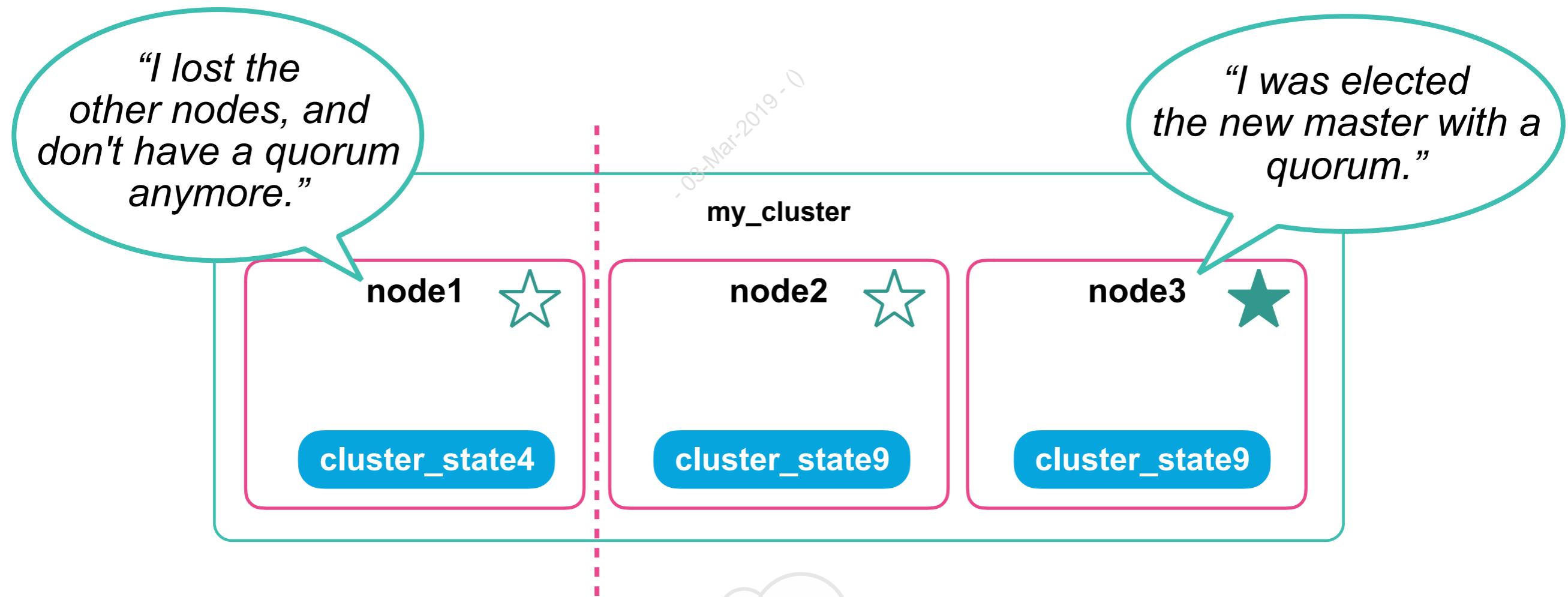
Split Brain

- Two masters can lead to inconsistency:



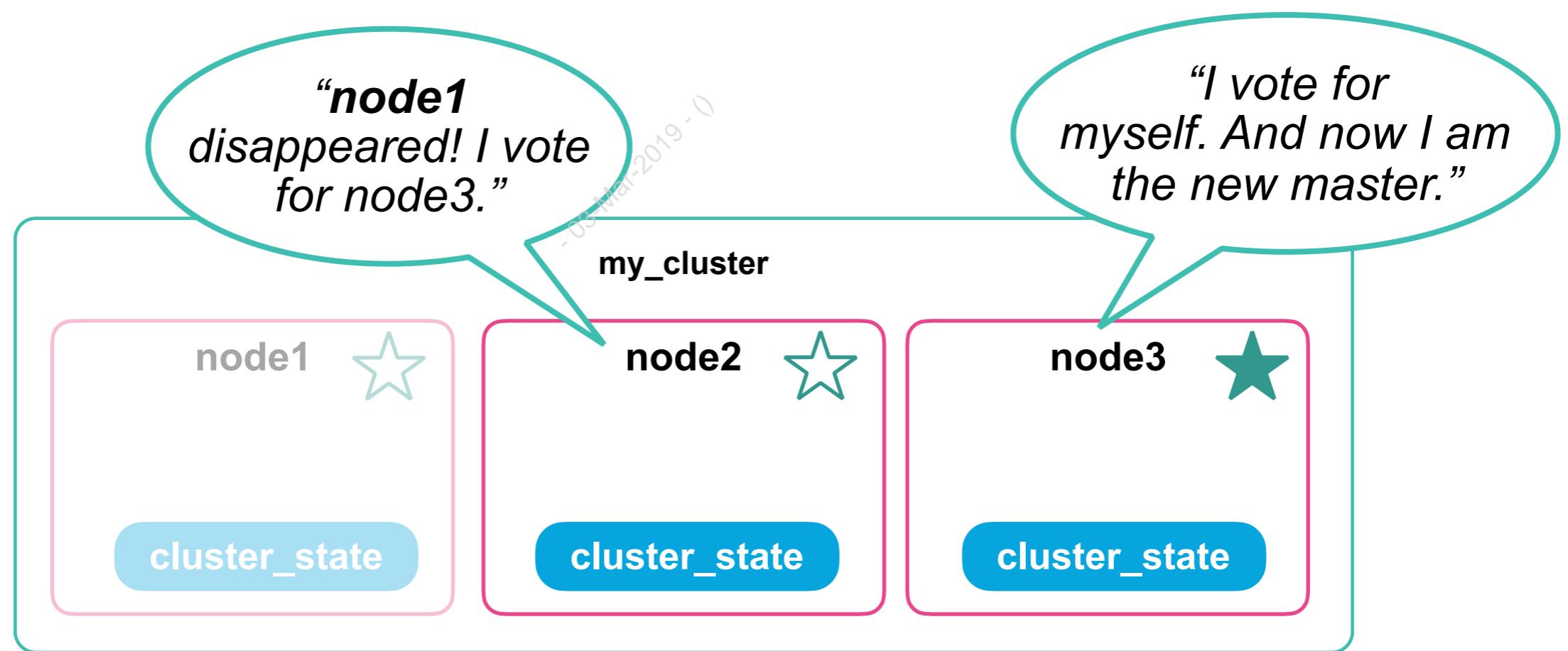
Configuring minimum_master_nodes

- We recommend your production clusters have **3 master-eligible** nodes with **minimum_master_nodes** set to 2
- Should be defined in the config file (elasticsearch.yml):
 - an **extremely important setting** for the stability of your cluster
 - it must be configured properly to avoid a "**split brain**"



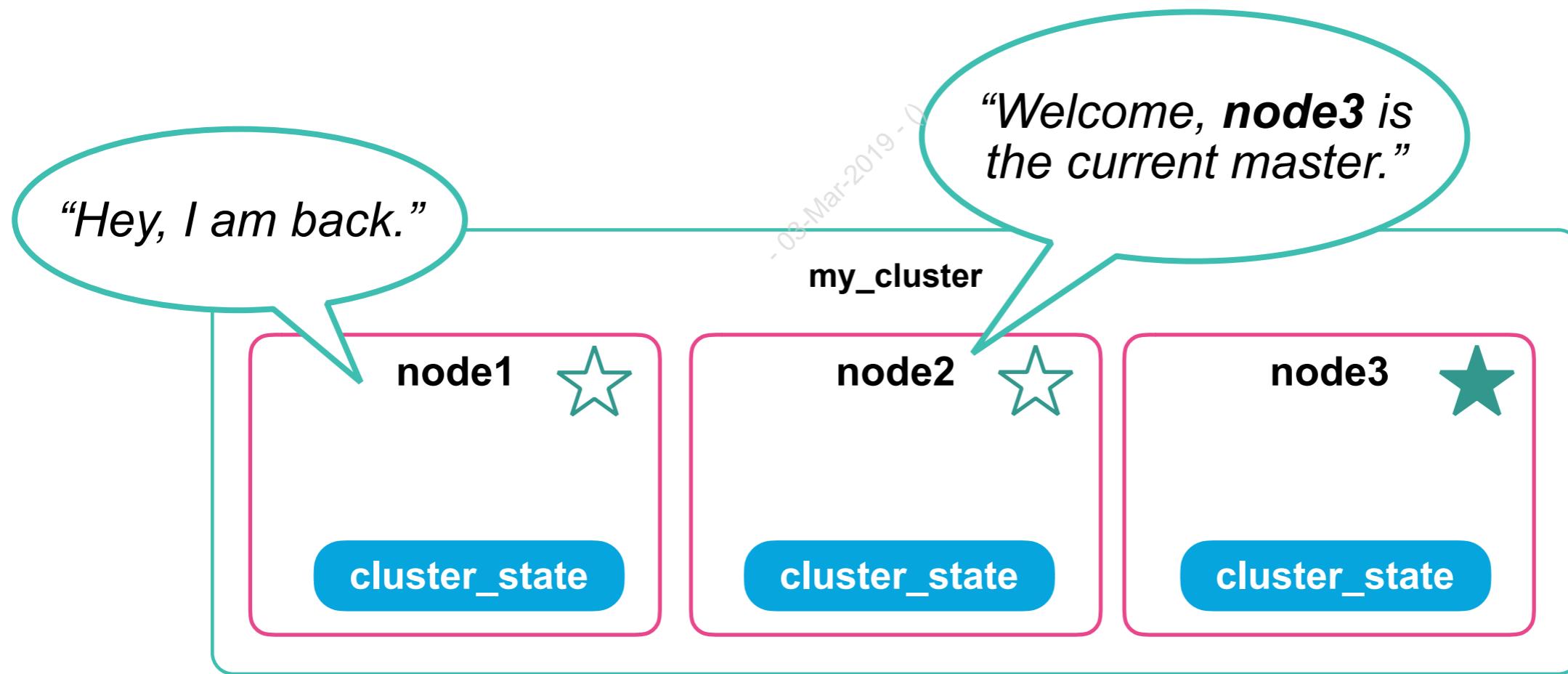
High Availability of the Master Node

- 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



Data Nodes

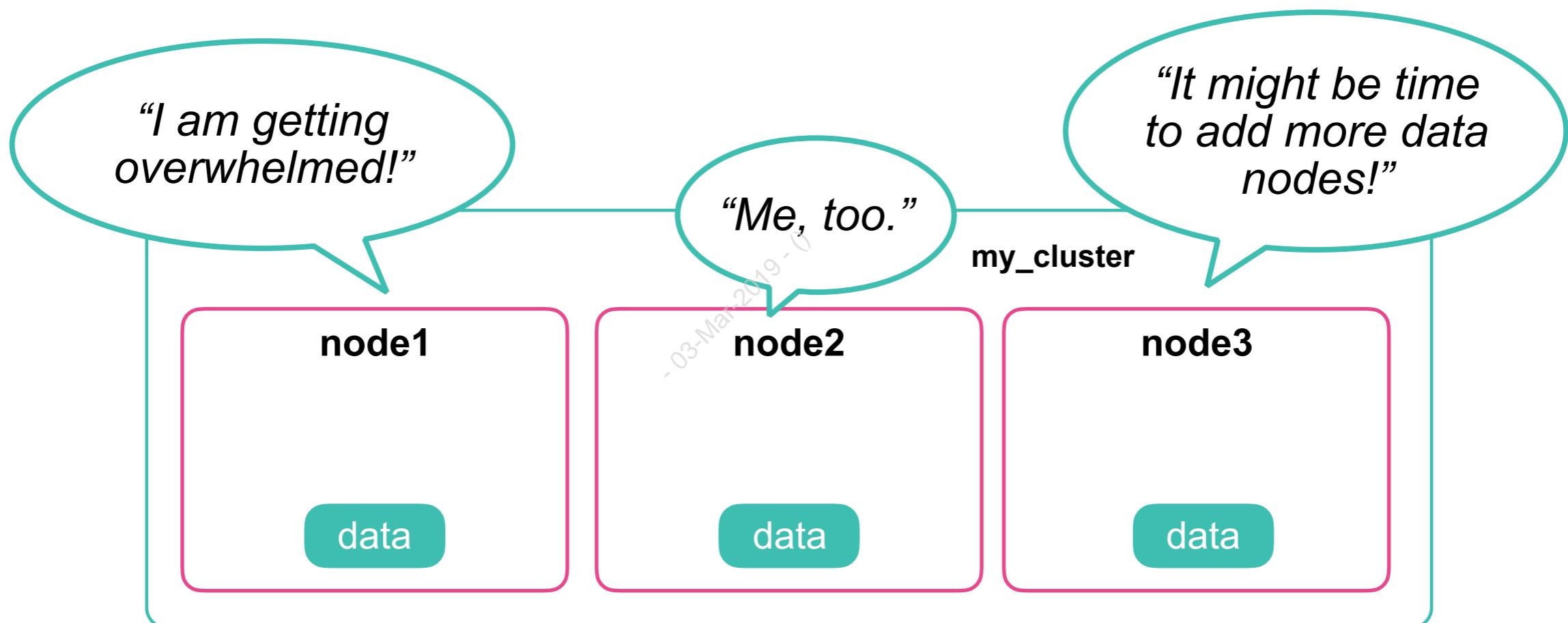
„03-Mar-2019“

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

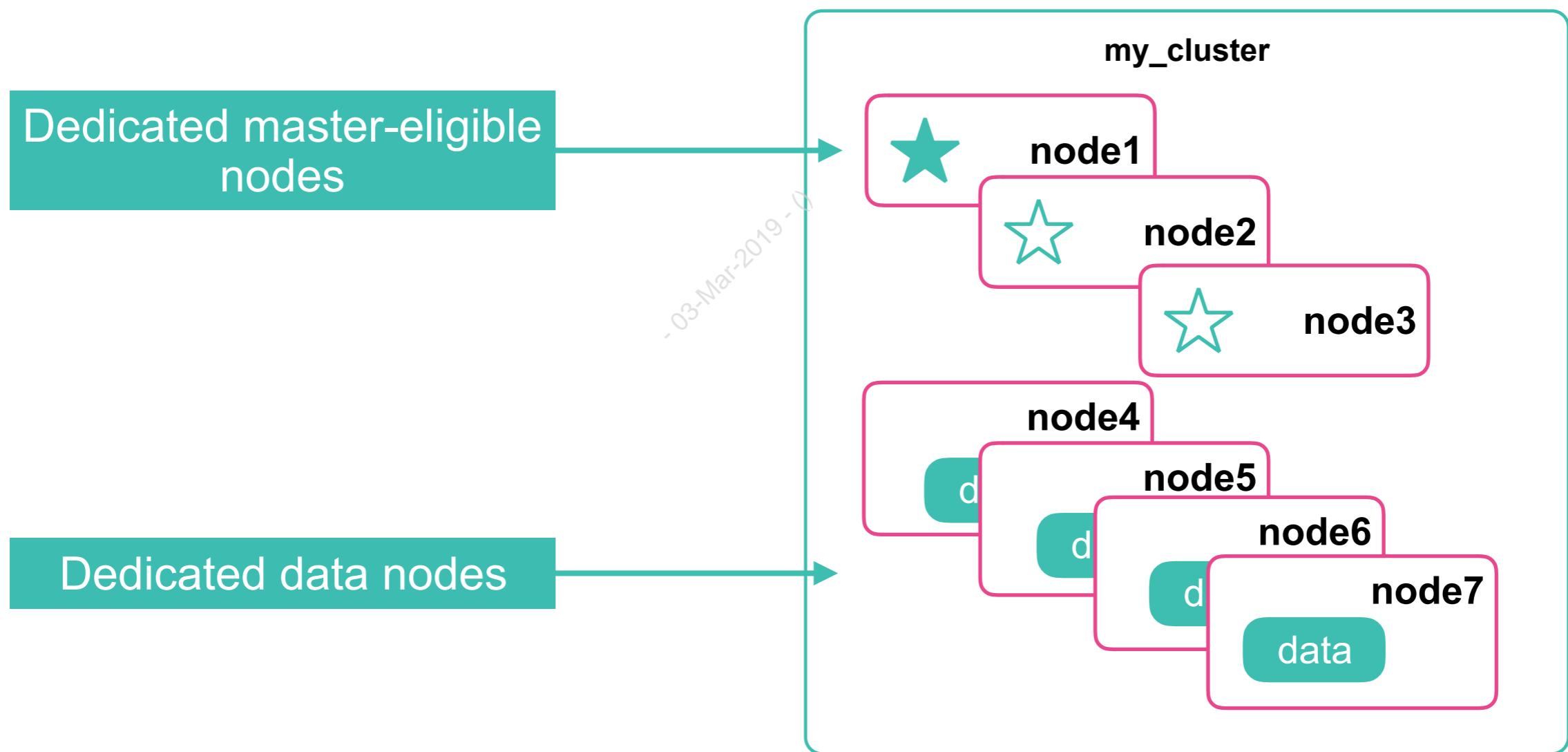
Data Nodes Scale

- 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



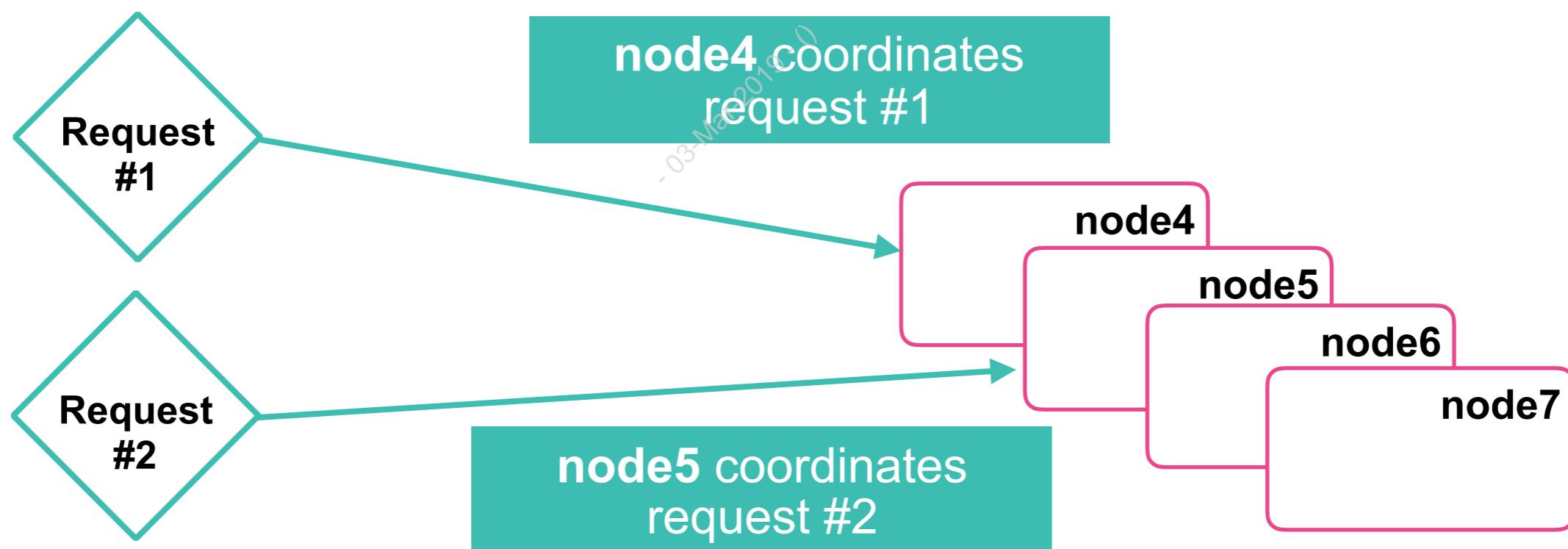
Dedicated Master and Data Nodes

- You can configure a node to be *just* a data or master-eligible node
 - by setting `node.master` or `node.data` to `false`
 - provides a nice separation of the master and data roles



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

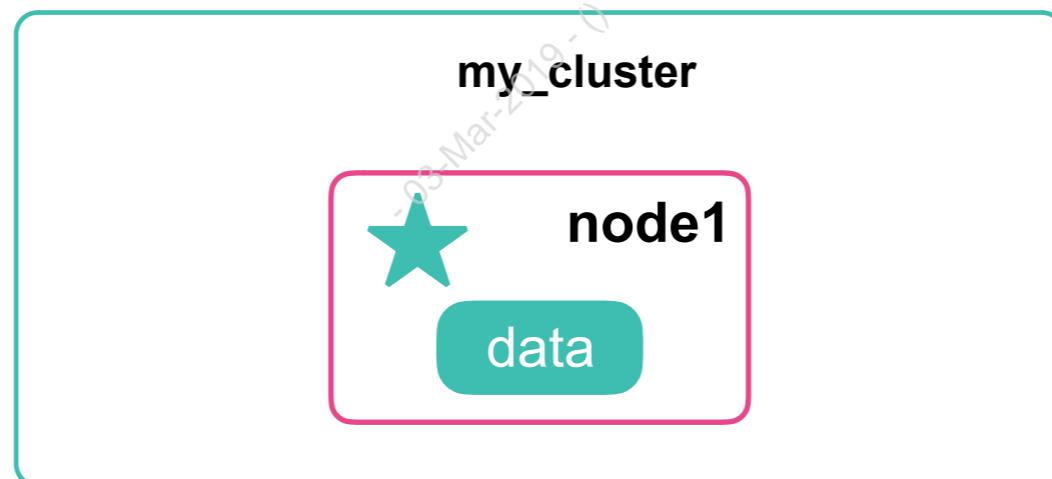


Sample Architectures

~03-Mar-2019 ~

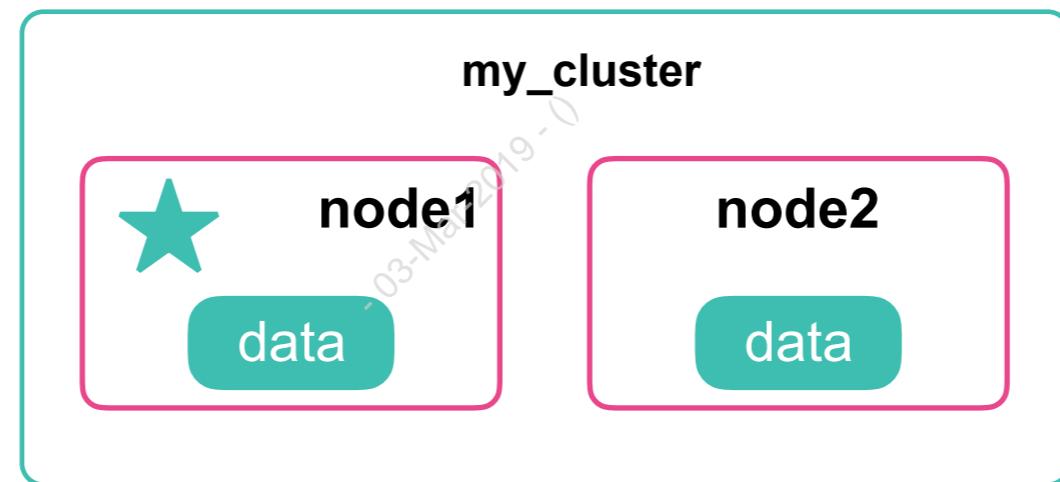
1-Node Cluster

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



2-Node Clusters

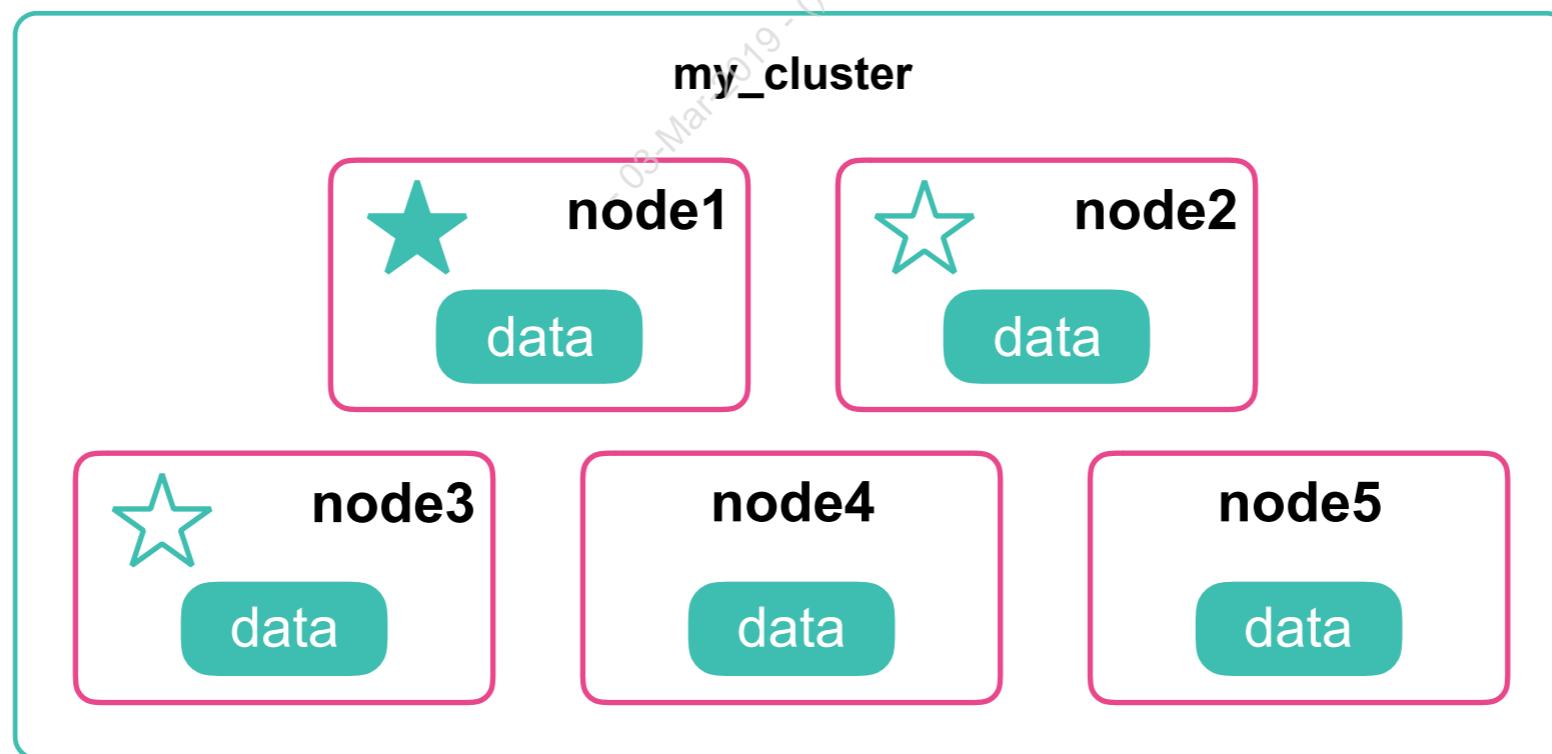
- 2-node clusters have the potential for split brain
 - either configure one dedicated master-eligible node,
 - or set **minimum_master_nodes = 2**



Be careful with this cluster - it has the potential for split brain

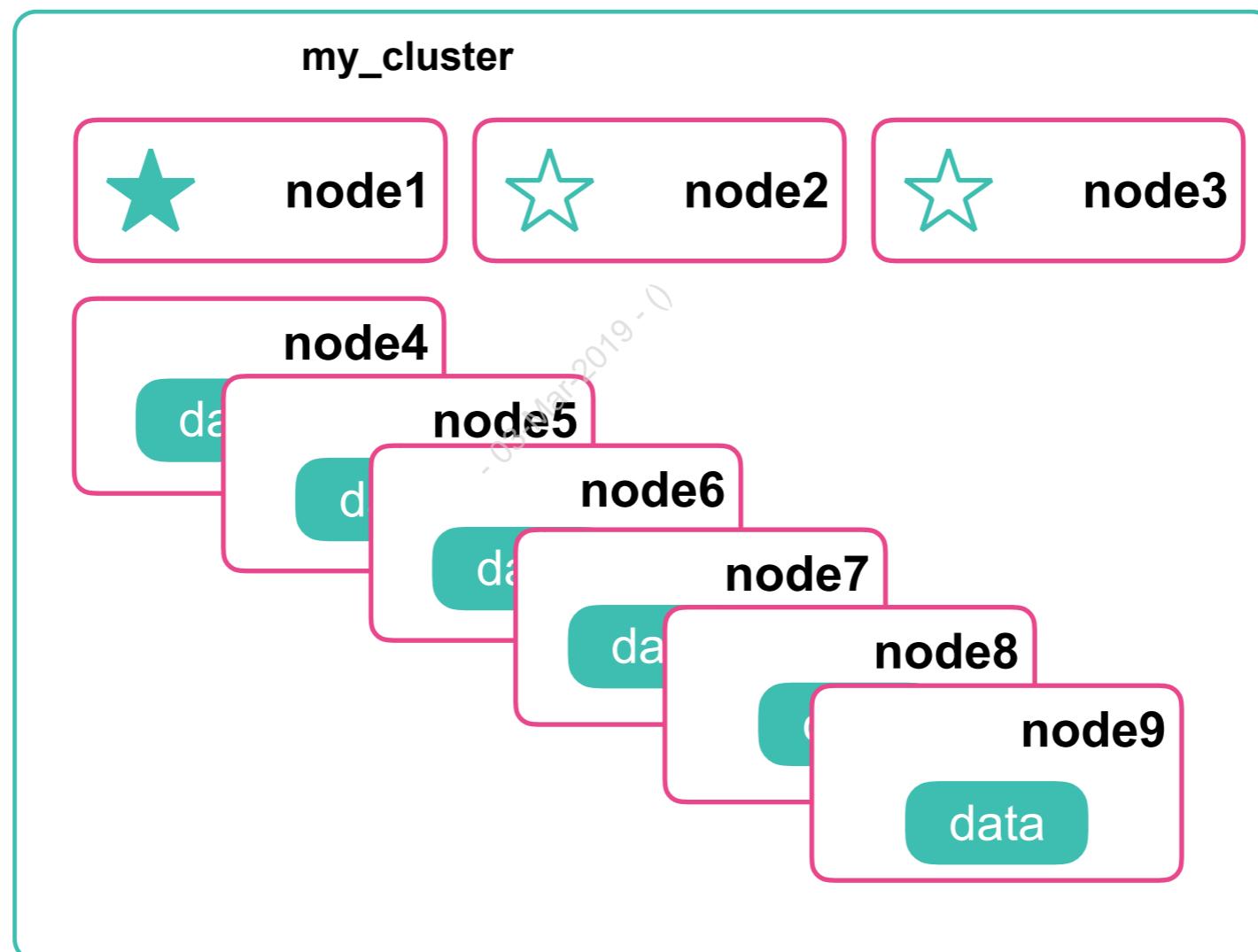
3-5 Node Cluster

- 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,
 - and set **minimum_master_nodes = 2**



Larger Clusters

- Larger, high-volume clusters should have dedicated nodes
 - add data nodes as your requirements change



Chapter Review

·03-Mar-2019·0

Summary

- The details of a cluster are maintained in the ***cluster state***
- Every cluster has ***one node*** designated as the ***master***
- A master-eligible node needs at least **`minimum_master_nodes`** votes to win an election
- ***Data nodes*** hold shards and execute data-related operations like CRUD, search, and aggregations

~03-Mar-2019~0



Quiz

1. If you have ***three*** master-eligible nodes in your cluster, what should you set **`minimum_master_nodes`** to?
2. If you have ***two*** master-eligible nodes in your cluster, what should you set **`minimum_master_nodes`** to?
3. How would you configure a node to be a ***dedicated master-eligible*** node?

~03-Mar-2019~0



Lab 7

Nodes Types

~03-Mar-2019~0



Chapter 8

Understanding Shards

03-Mar-2019 - 0

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- Understanding Shards
- Anatomy of a Write Operation
- Anatomy of a Search
- Number of Shards

~03-Mar-2019~0



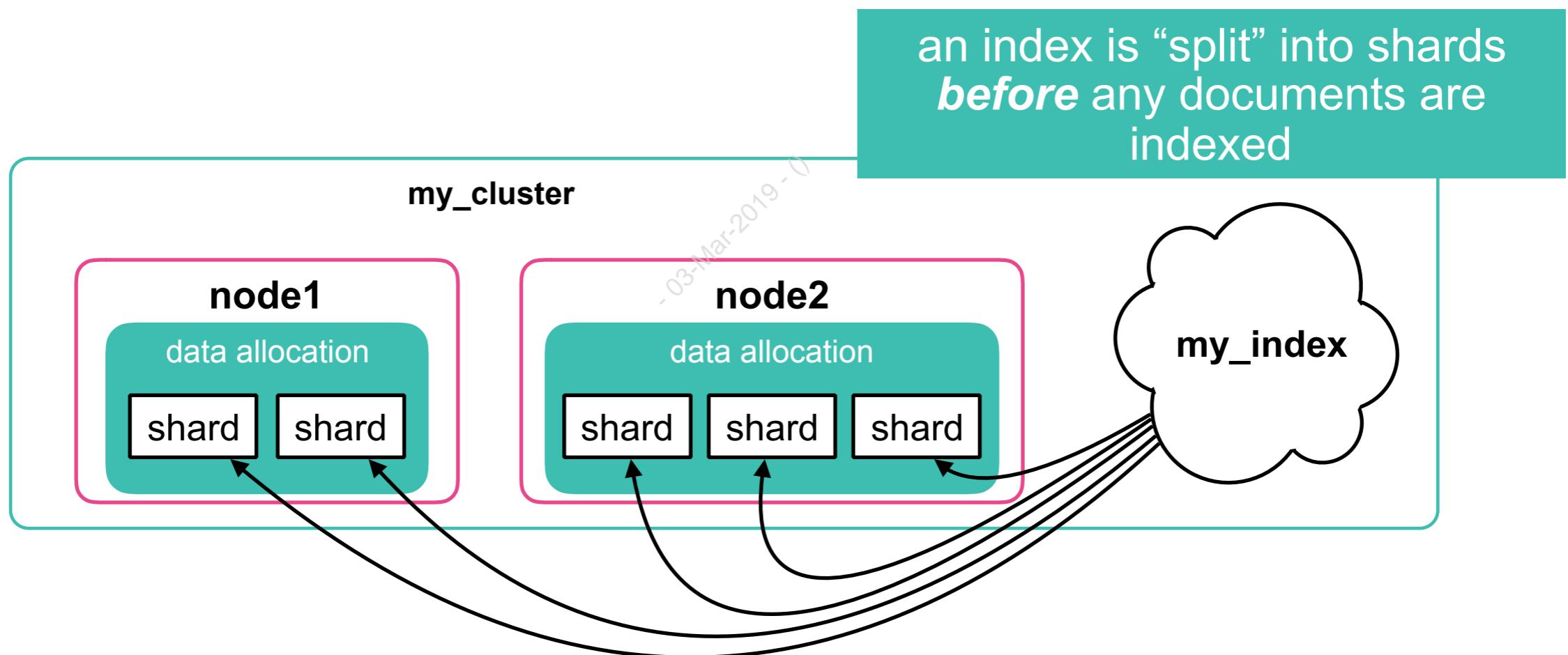
Understanding Shards

-03-Mar-2016-



Remember Shards?

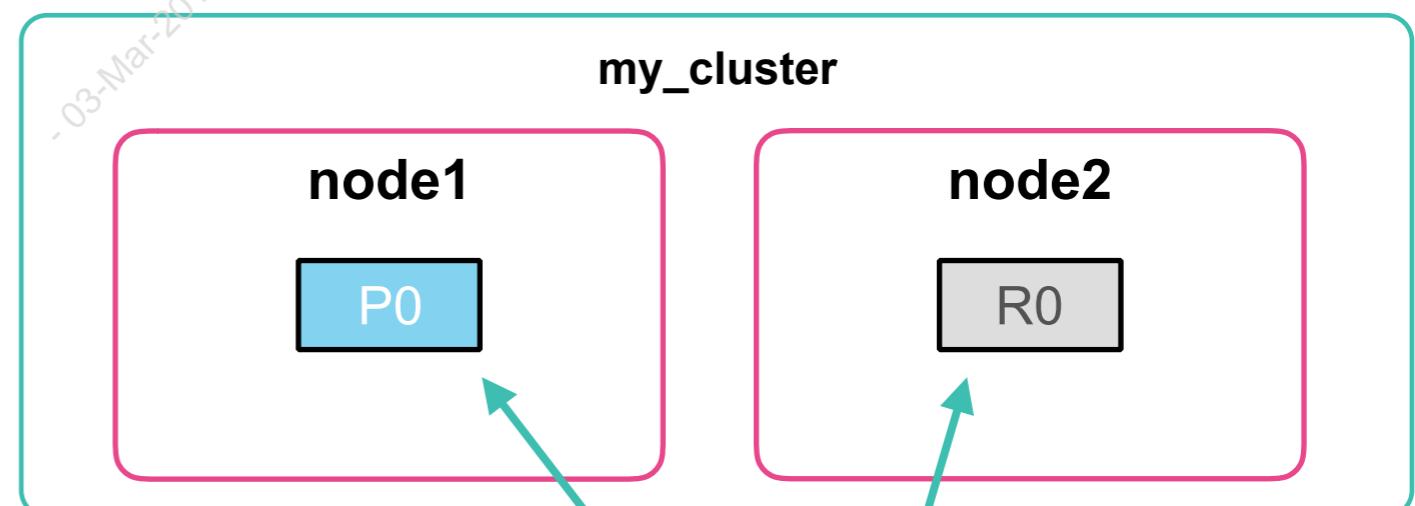
- A **shard** is a worker unit that holds data and can be assigned to nodes
 - An index is a virtual namespace which points to a number of shards



Primary vs. Replica

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

When an index is created, shards are numbered: 0, 1, 2, 3 ...

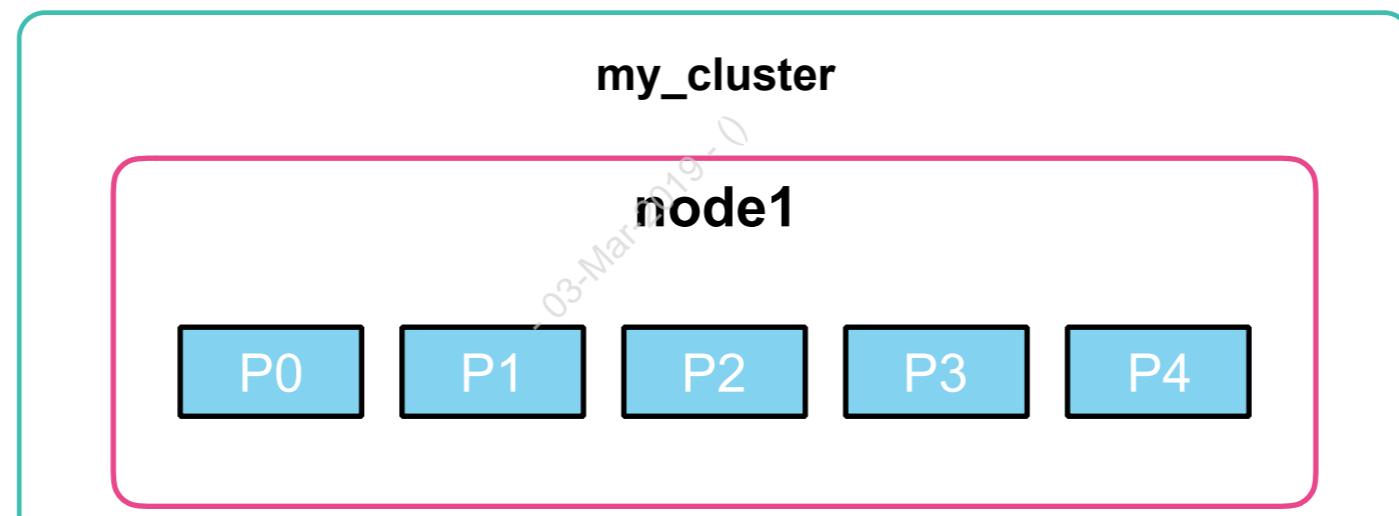


P0 and R0 contain the same documents. They are called copies of shard 0



The Shards of the Blogs Dataset

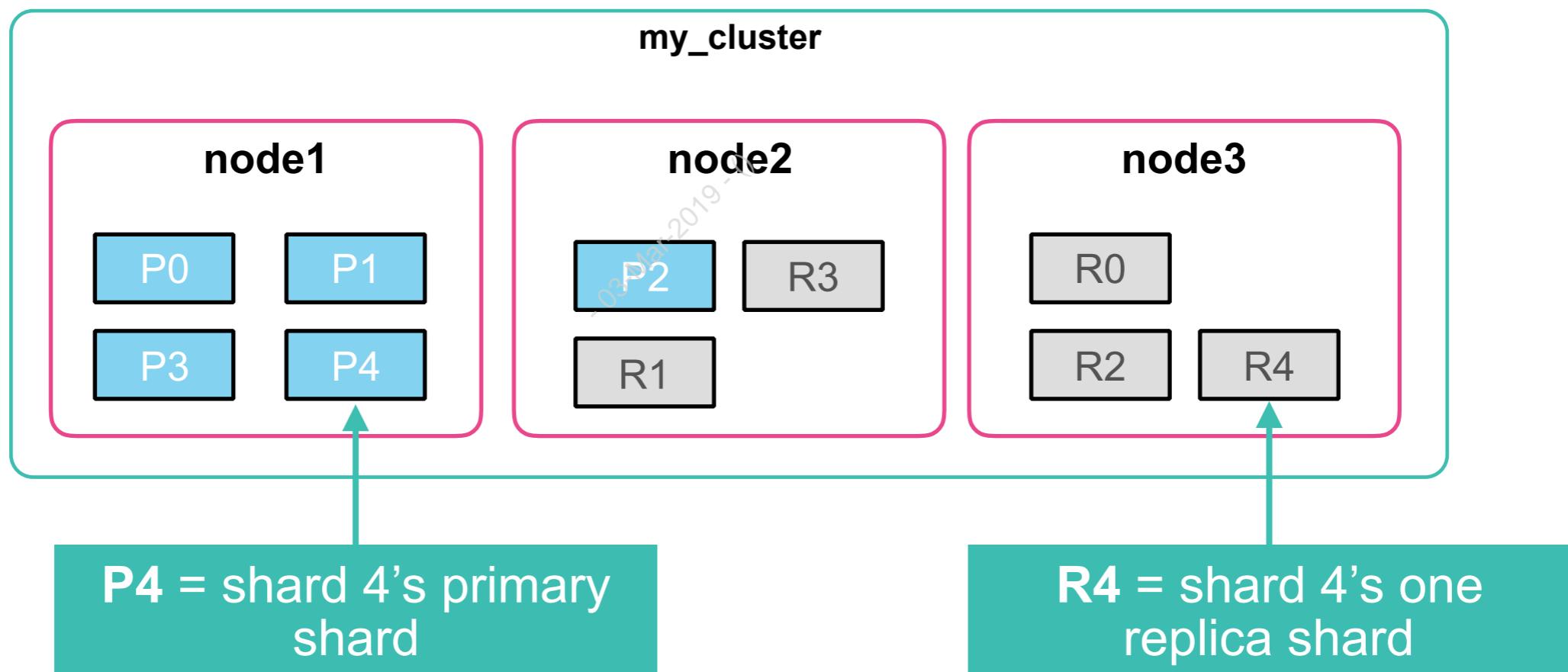
- Our blog index had the default number of five primary shards with one replica
 - but our cluster only had one node
- Where were the replicas?



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

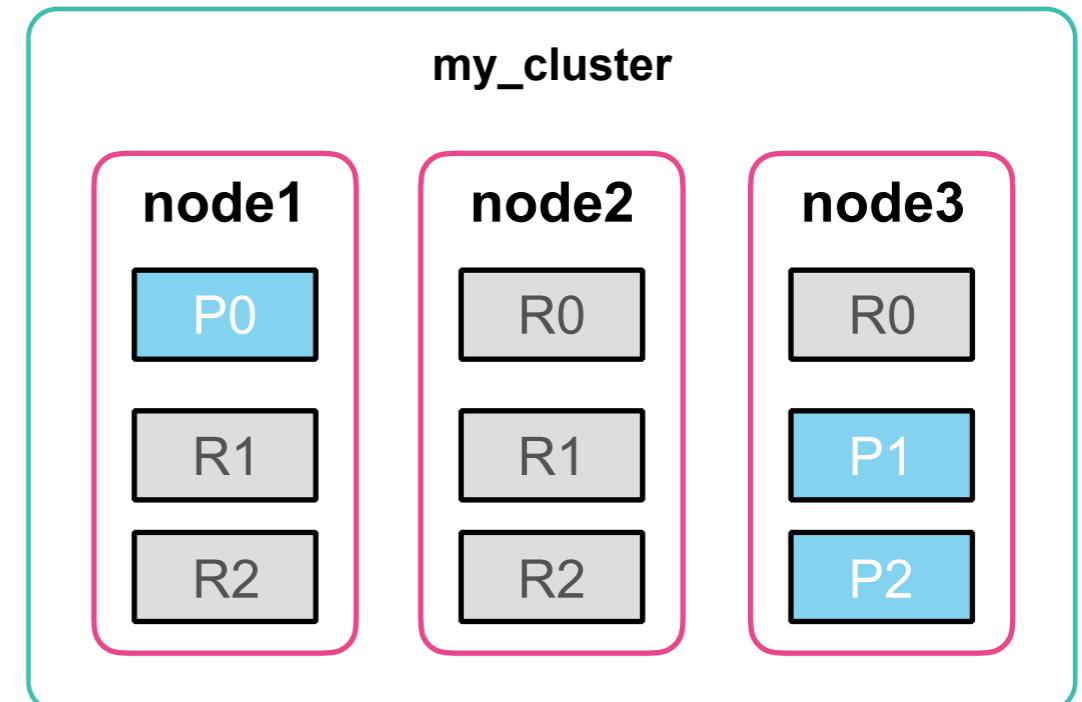


Configuring the Number of Shards

- The default number of primary shards for an index is 5
 - You specify the number of primary shards when you create the index
 - It is costly to change the number of shards, so choose wisely! (as discussed in *Engineer II*)
 - The number of replicas can be changed at any time

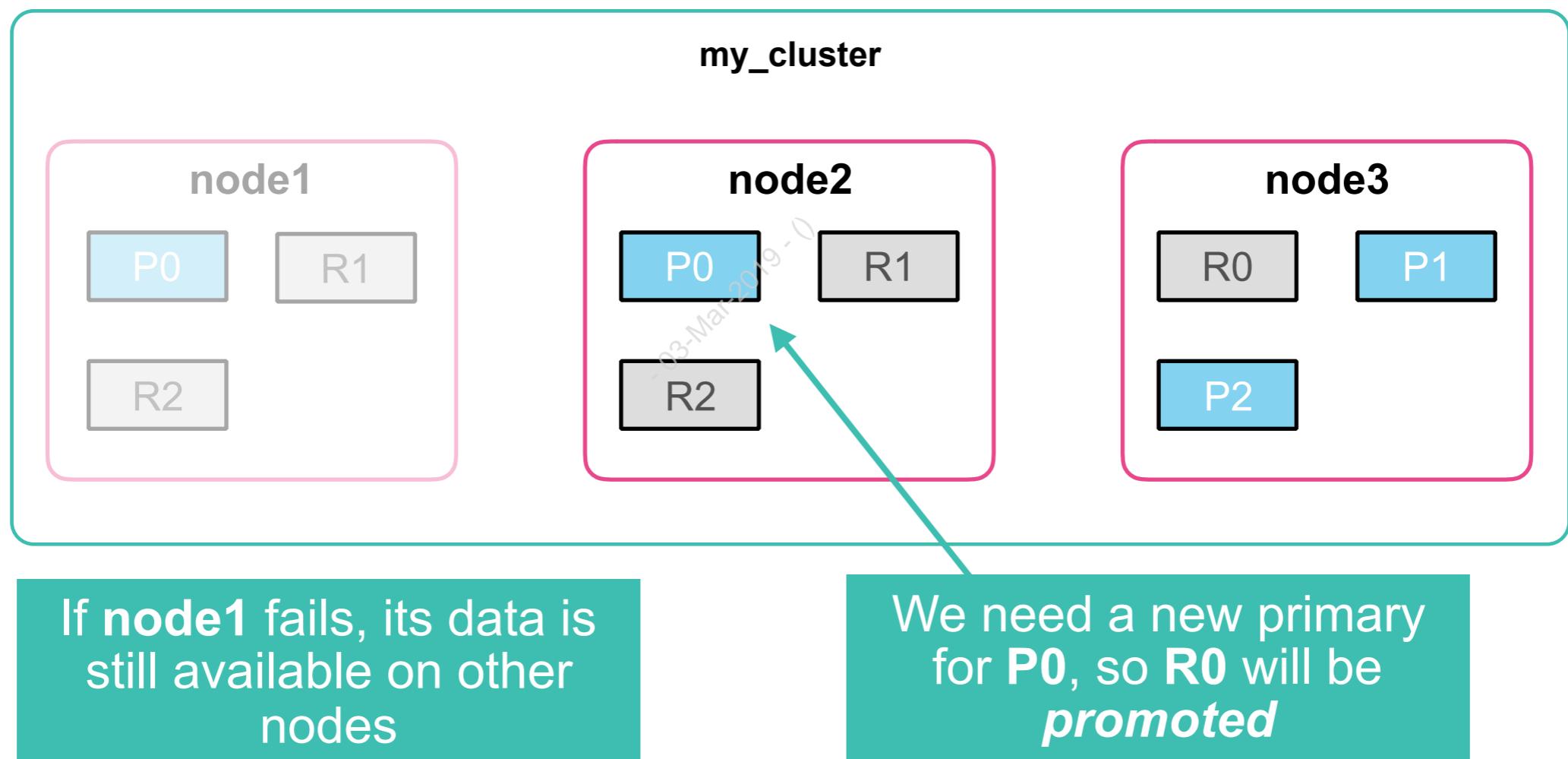
```
PUT my_new_index
{
  "settings": {
    "number_of_shards": 3,
    "number_of_replicas": 2
  }
}
```

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



Why Create Replicas?

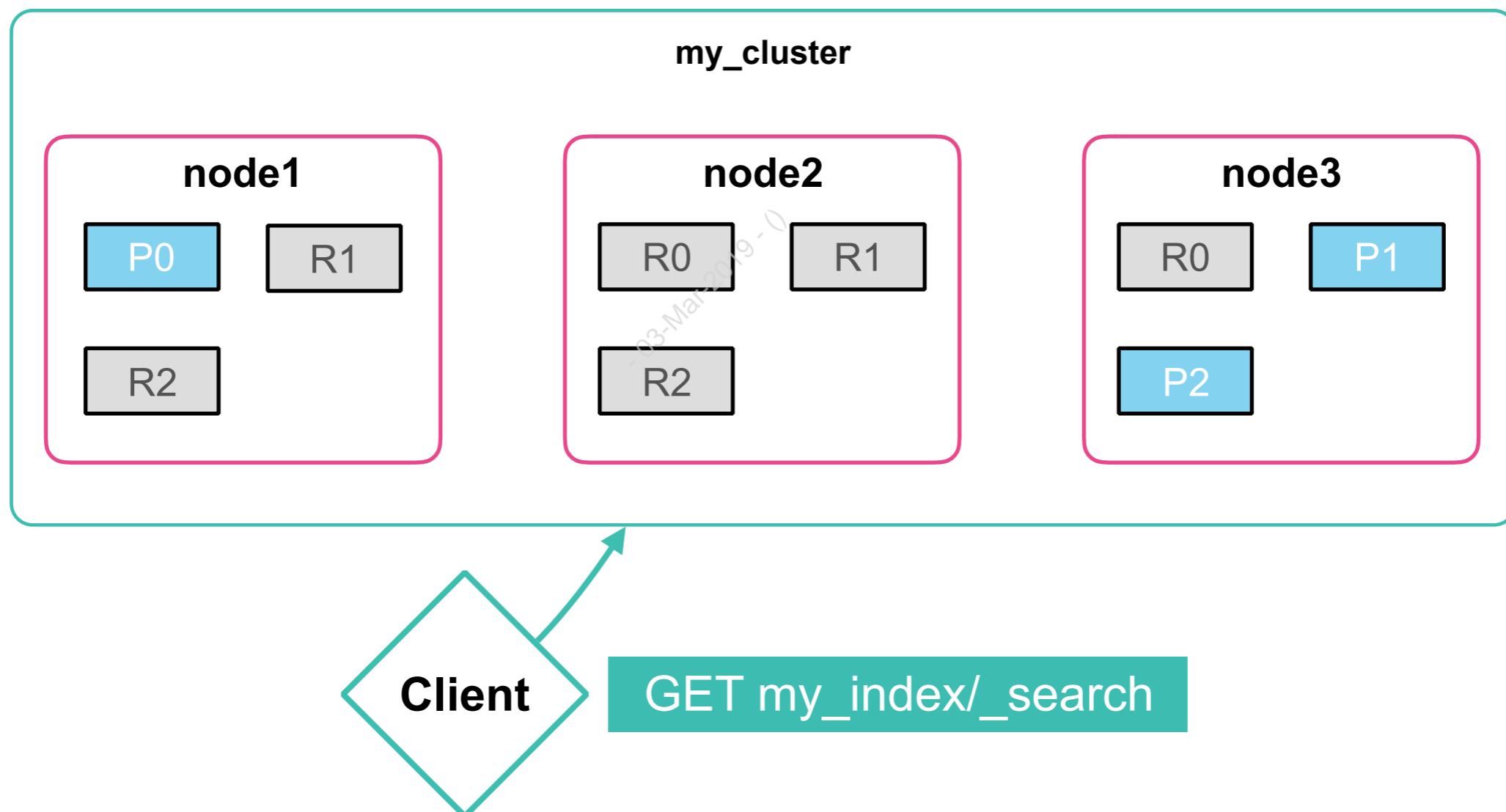
- ***High availability***
 - we can lose a node and still have all the data available
 - replicas are promoted to primaries as needed



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



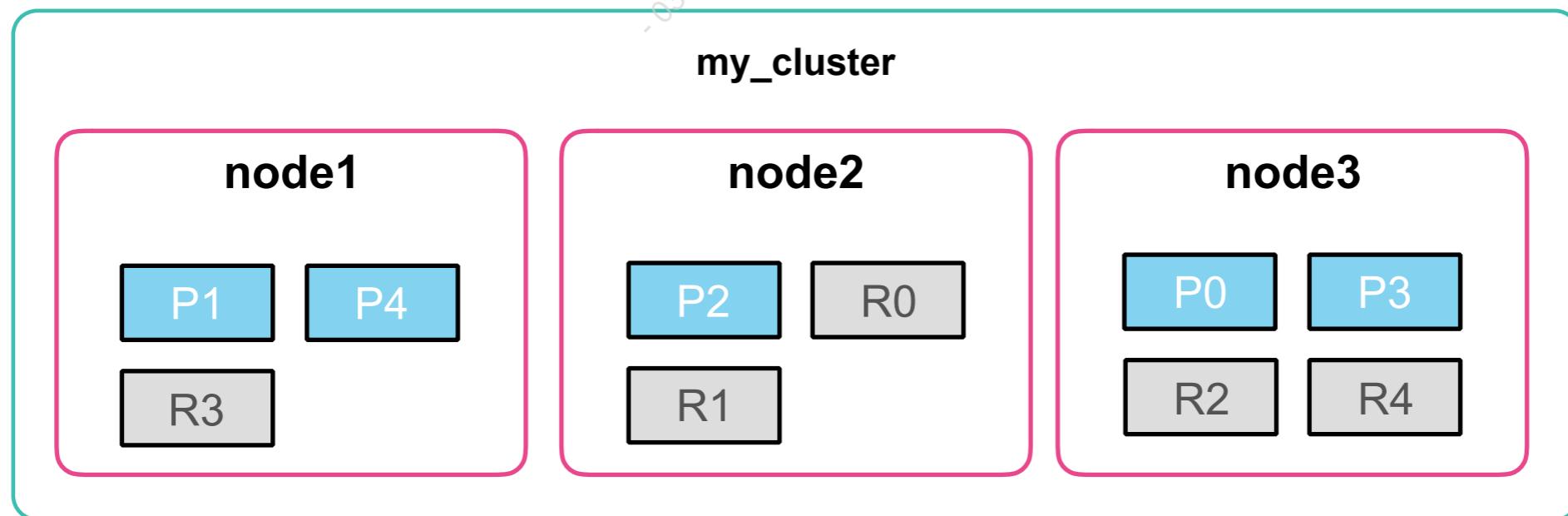
Anatomy of a Write Operation

03-Mar-2019 - 0

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

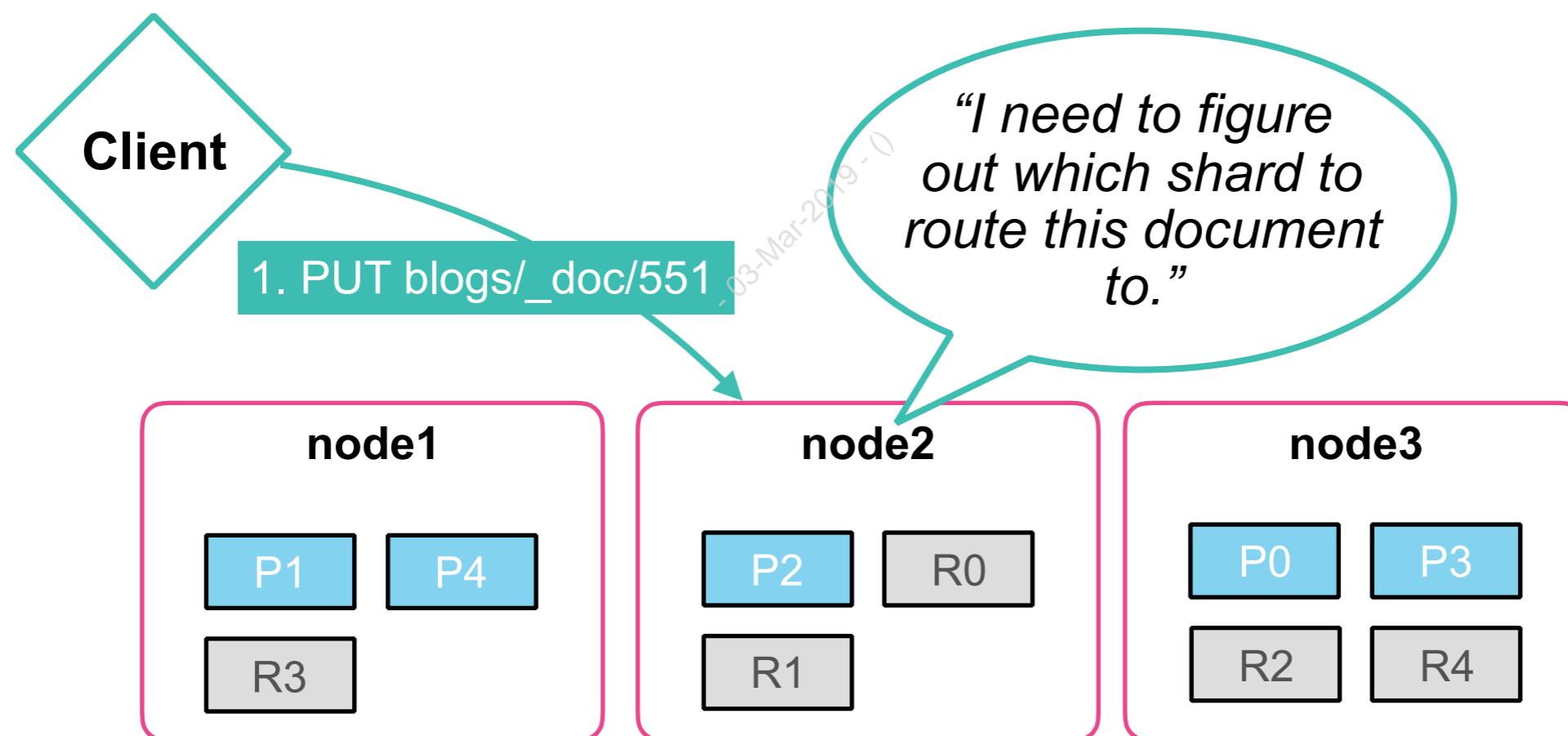
```
PUT blogs/_doc/551
{
  "title": "A History of Logstash Output Workers",
  "category": "Engineering",
  ...
}
```



Document Routing

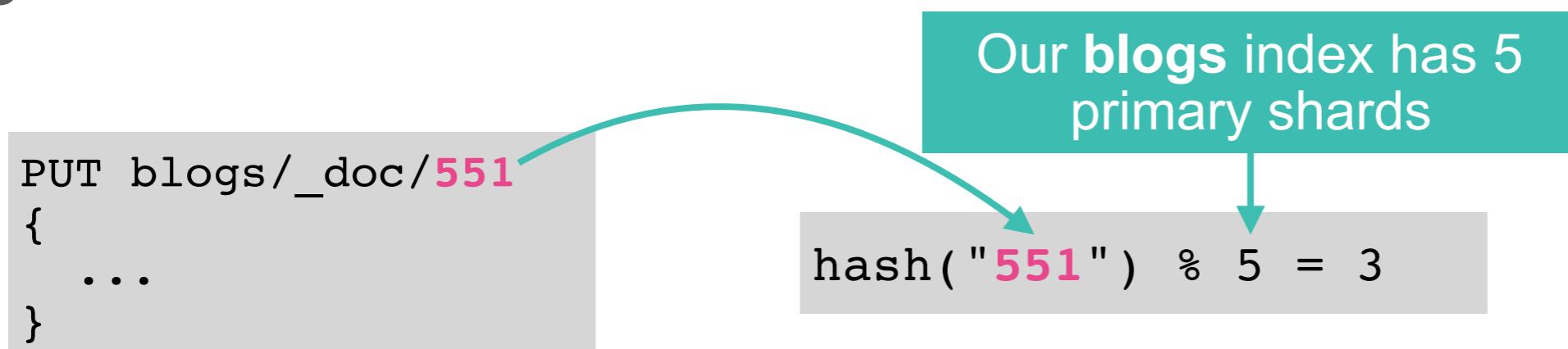
- When a document is indexed, it needs to be determined **which shard to index the document to**
 - The shard is chosen based on a simple formula

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



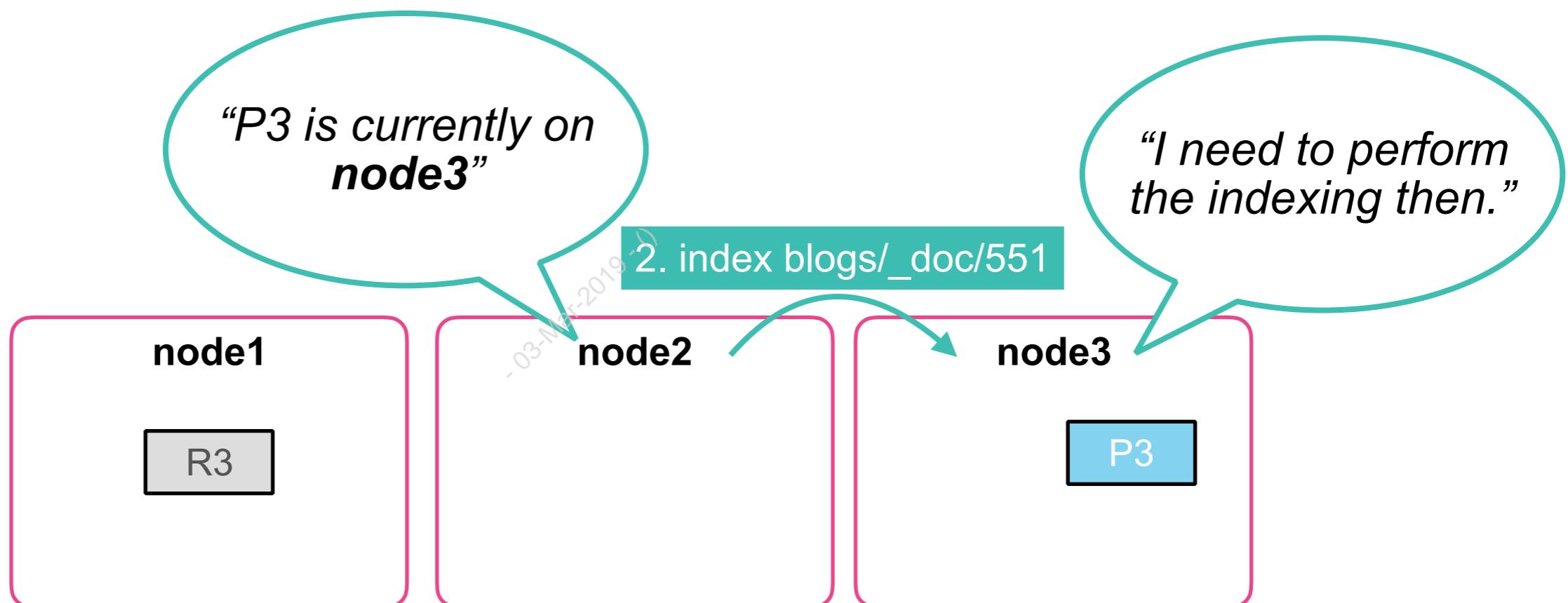
Document Routing

- ***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**



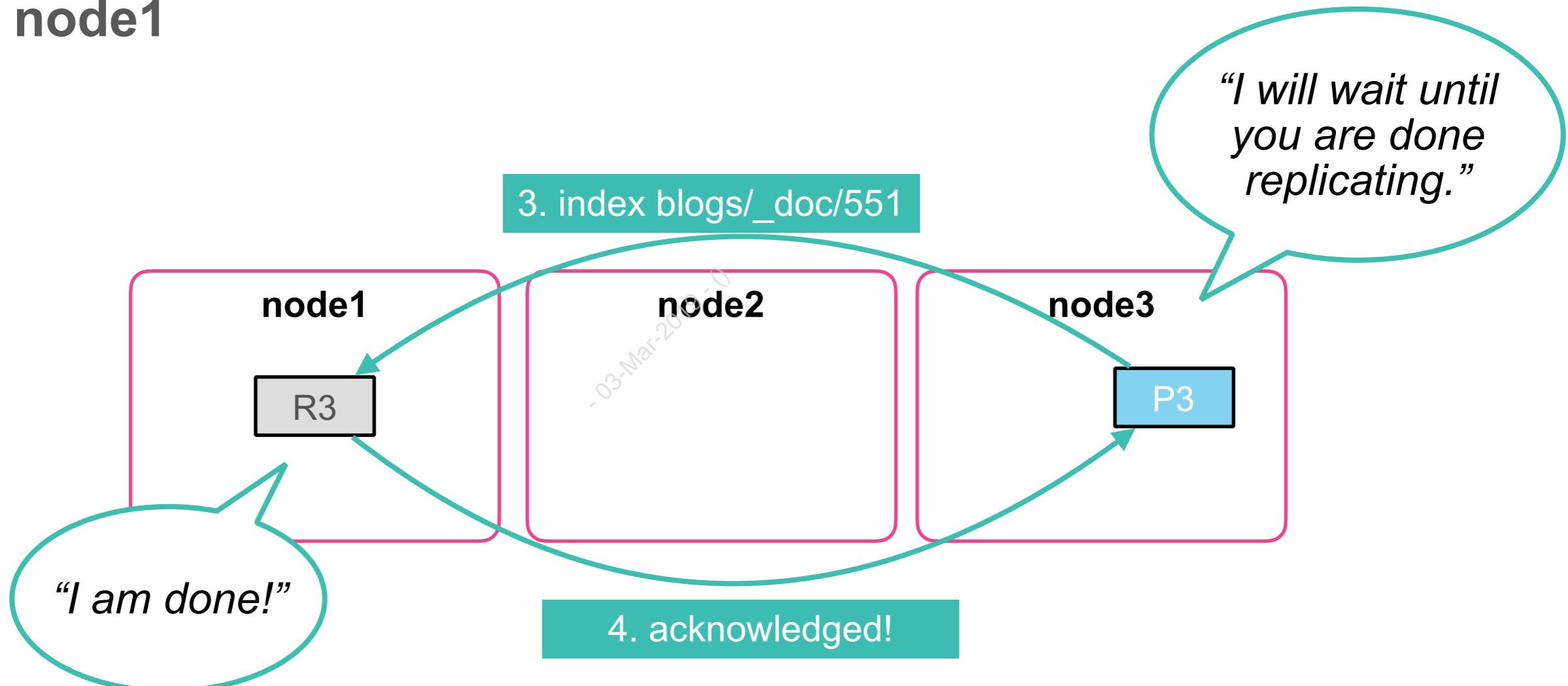
Write Operations on the Primary Shard

- 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



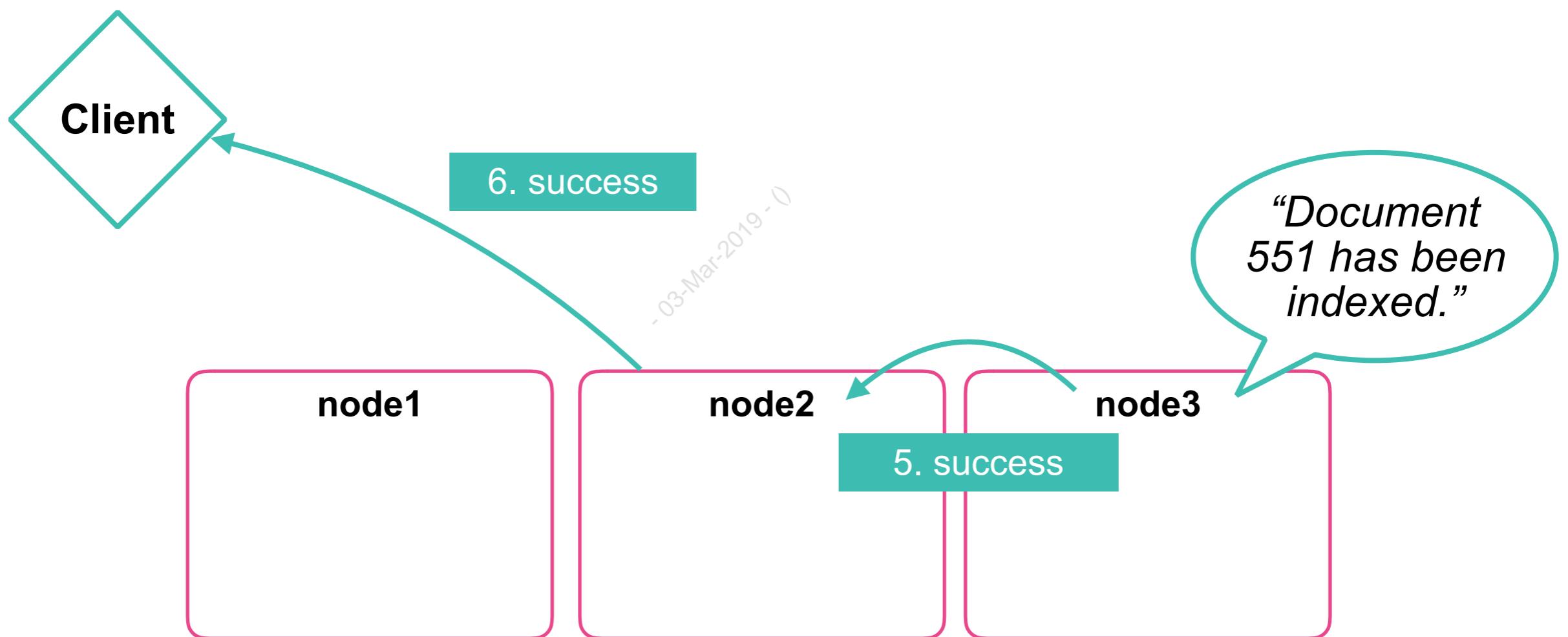
Replicas are Synced

- 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

~03-Mar-2019~0

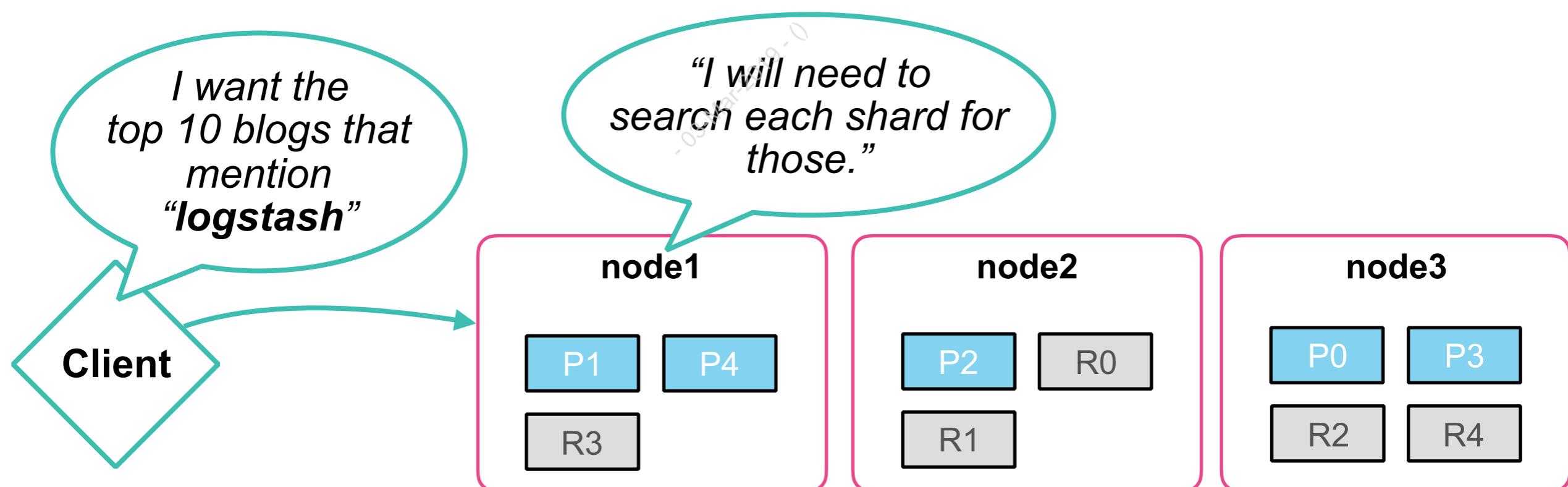


Anatomy of a Search

~03-Mar-2019

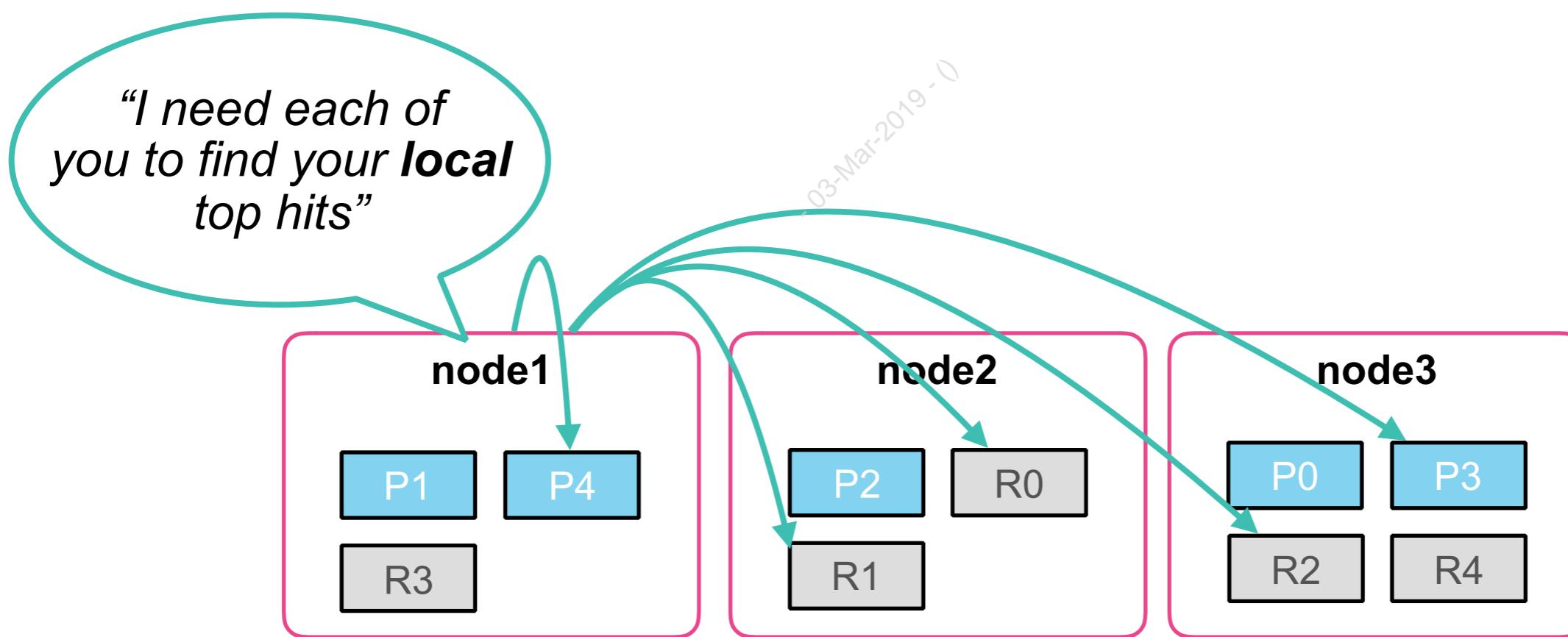
Anatomy of a Search

- 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



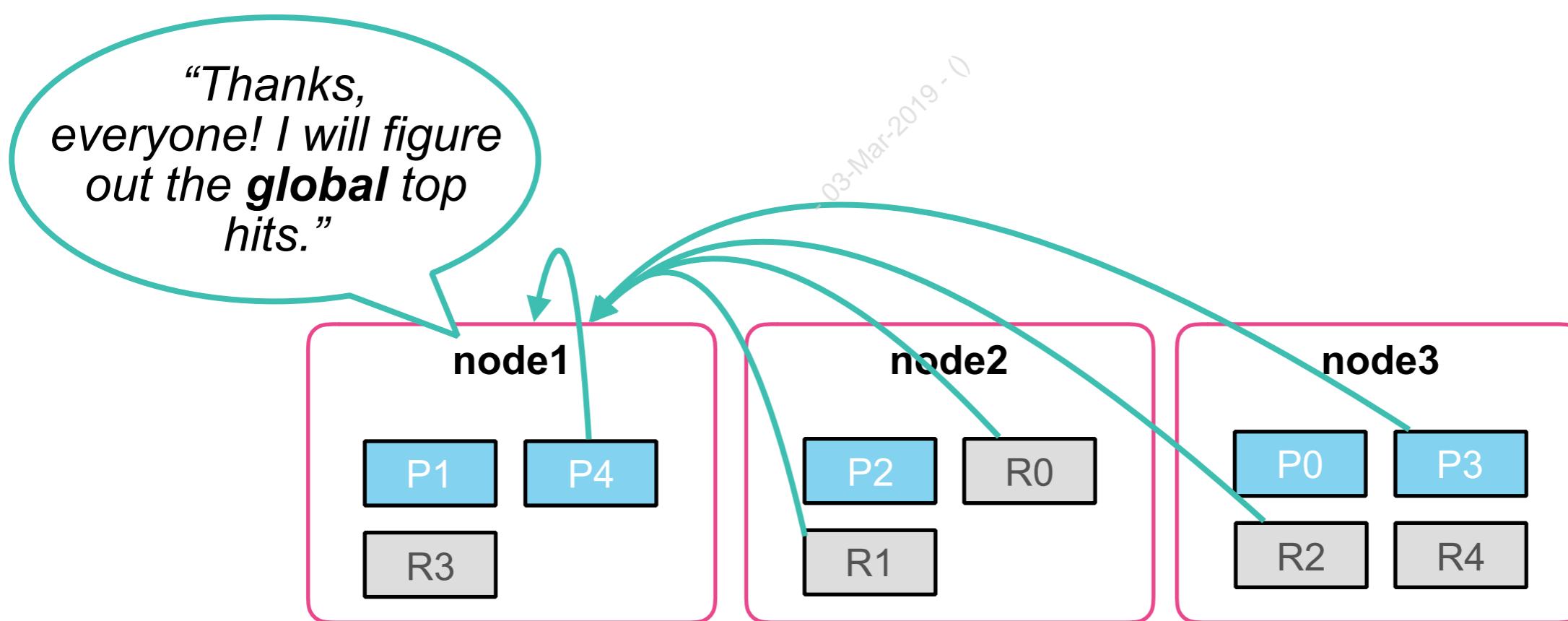
The Query Phase

- 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**



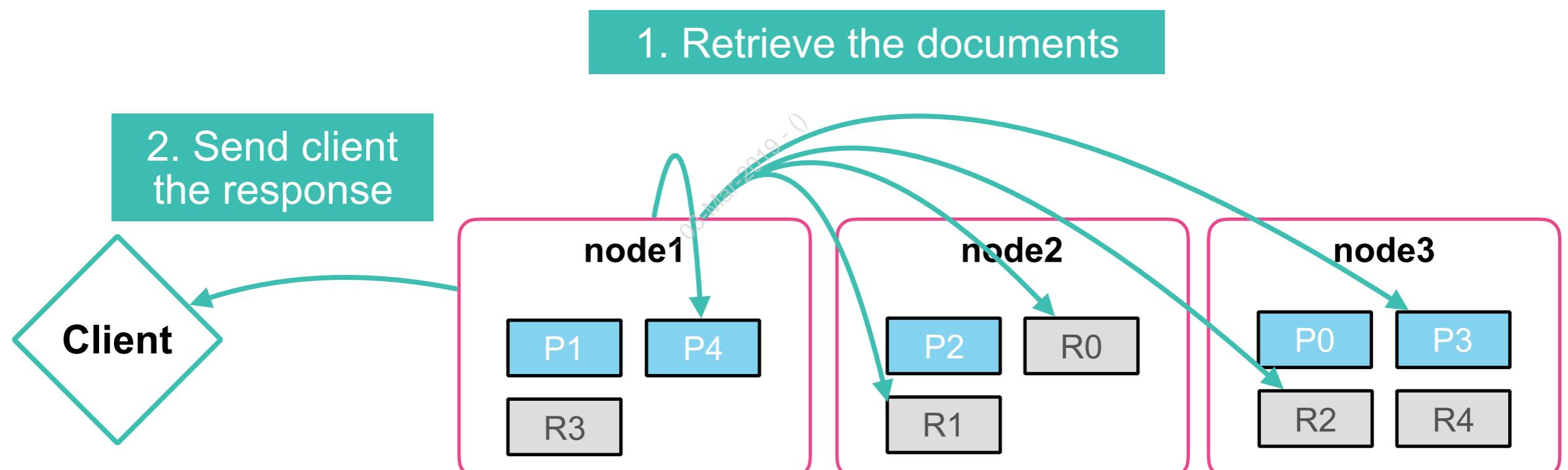
The Query Phase

- 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



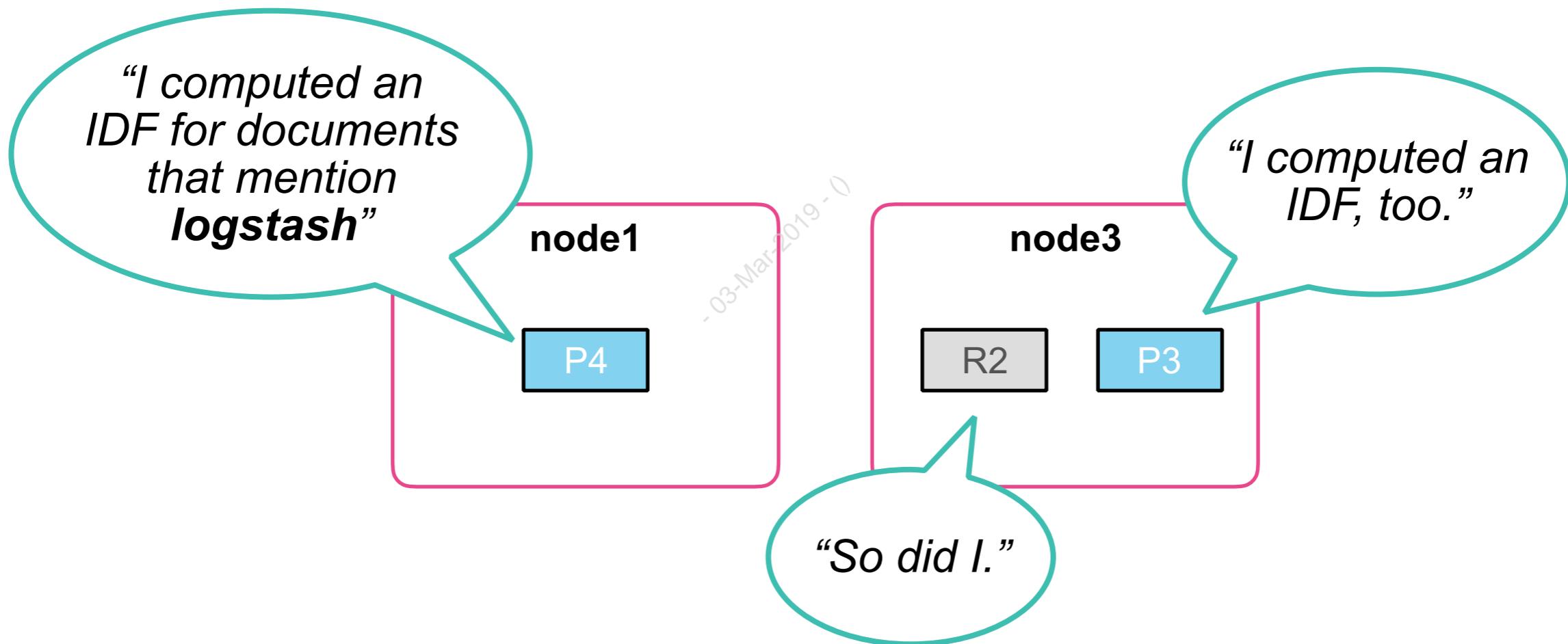
Is `_score` calculated locally or globally?

- Recall that a document's `_score` is based on *term frequency* (TF) and *inverse document frequency* (IDF)
 - TF is 100% accurate because it is document-level computation
- With the default query-then-fetch behavior, IDF is not 100% accurate:
 - because it is *calculated locally on each shard*
- For IDF to be 100% accurate:
 - its value would have to be computed *globally* before a shard can compute the `_score` of each hit
- Why not compute IDF globally for every search?
 - Because that would be expensive!



Is _score calculated locally or globally?

- A hit's score is calculated from the TF and IDF values
 - TF is a local value
 - IDF is a global value, but **each shard uses its local IDF instead**



So relevance is broken!?

- No - not at all!
 - Using local IDF is rarely a problem, especially with large datasets
 - If your documents are well distributed across shards, the local IDF between shards will be the same
- If you want to use a *global IDF*, set **search_type** to **dfs_query_then_fetch** in your query
 - A pre-query retrieves the local IDF from each shard first to compute the global IDF
 - However, you rarely need to use it in production

```
GET blogs/_search?search_type=dfs_query_then_fetch
```

Useful when testing or
debugging with small datasets



How many shards do I need?

Q3-Mar-2019 - 0

Determining the Number of Shards

- Now that we have discussed what shards are and how they are allocated across an Elasticsearch cluster, let's visit one of our most frequently asked questions...
 - How many shards does my index need?***
- ...and you are going to love the answer:

~03-Mar-2019 - 0

It depends!

- ***Heavy indexing?***
 - Use a higher number of primary shards so as to scale the indexing across more nodes
- ***Heavy search traffic?***
 - Increase the number of replicas (and add more nodes if necessary)
- ***Large dataset?***
 - Allow for enough primary shards to keep each shard under 10-40GB
- ***Small dataset?***
 - Nothing wrong with a one-shard index!
- We cover this topic in more detail in ***Engineer II***

Chapter Review

·03-Mar-2019·0

Summary

- Elasticsearch subdivides the data of your index into multiple pieces called ***shards***
- Each shard has one (and only one) ***primary*** and zero or more ***replicas***
- A search consists of a ***query phase*** and a ***fetch phase***

~03-Mar-2019~ Ø



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:** By default, the size of a document is considered when determining which shard of the index to route the document to.
4. **True or False:** An index operation has to be executed on the primary shard first before being synced to replicas.



Lab 8

03-Mar-2019

Understanding Shards



Chapter 9

Troubleshooting Elasticsearch

03-Mar-2019 - 0

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- Understanding Configuration Settings
- Elasticsearch Responses
- Cluster Health and Shard Allocation
- Diagnosing Health Issues

~03-Mar-2019~0



Understanding Configuration Settings

.03Mar2019.0

Configuration Areas

- Settings are configured in Elasticsearch at three levels
 - index
 - node
 - cluster
- Let's take a look at each of these configuration areas

~03-Mar-2019~ Ø



Index Settings

- ***Index settings*** include:
 - number of shards, replicas, refresh rates, read-only, etc.
 - typically configured using the REST APIs

```
PUT my_tweets
{
  "settings": {
    "number_of_shards": 3
  }
}
```

when the index is created

```
PUT my_tweets/_settings
{
  "index.blocks.write": false
}
```

change settings of an
existing index

Node Settings

- ***Node settings*** include:
 - file paths, labels, network interfaces (settings specific to the node)
 - typically configured in **elasticsearch.yml** or command line
- Settings in **elasticsearch.yml** (as with any YML file) can be flat:

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

- or indented:

```
cluster:  
  name: my_cluster
```

Cluster Settings

- ***Cluster settings*** include:
 - logging levels, index templates, scripts, shard allocation, etc
 - typically configured using the REST APIs

```
PUT _cluster/settings
{
  "persistent": {
    "discovery.zen.minimum_master_nodes": 2
  }
}
```

Can be **persistent** (survives restarts) or
transient (will not survive a full cluster restart)

Precedence of Settings

1. *Transient settings* take precedence over all
 2. *Persistent settings* take precedence over 3 and 4
 3. *Command-line settings* take precedence over 4
 4. *Config file settings* (e.g., `elasticsearch.yml`)
-
- Note that not all settings can be updated via the API
 - *static* settings can only be set in `elasticsearch.yml` or from the command line
 - *dynamic* settings can be updated on a live cluster using the *Cluster Update API*

Example of Dynamic Settings

- Suppose you are having issues with a node joining a cluster
 - You can dynamically change the logging level for the transport module:

```
PUT /_cluster/settings
{
  "transient" : {
    "logger.org.elasticsearch.transport.TransportService.tracer" : "TRACE"
  }
}
```

Change it back to “**INFO**” once you are done
debugging the issue

Elasticsearch Responses

,03-Mar-2019, 0

Elasticsearch Responses

- As discussed, Elasticsearch uses the REST APIs
- There 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
Unable to connect	networking issue or cluster down	check network status and cluster health
connection unexpectedly closed	node died or network issue	retry (with risk of creating a duplicate document)
4xx error	client error	fix bad request before retrying
429 error	Elasticsearch is too busy	retry (ideally with linear or exponential backoff)
5xx error	internal server error in Elasticsearch	look into Elasticsearch logs to see what is the issue



Understanding the Index Response Body

- 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 the case of failure
Elasticsearch will return
the reason 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"  
      } } , ... }
```

All shards were available in the cluster when the request was received, but 2 shards failed



Cluster Health and Shard Allocation

103-Mar-2019 - 0

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
- ***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
- ***index health***
 - status of the **worst shard** in that index
- ***cluster health***
 - status of the **worst index** in the cluster



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
 - e.g., more replicas assigned than nodes in the cluster; a node failed; brand new index

What is the
cluster health?

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

03-Mar-2019 - 0

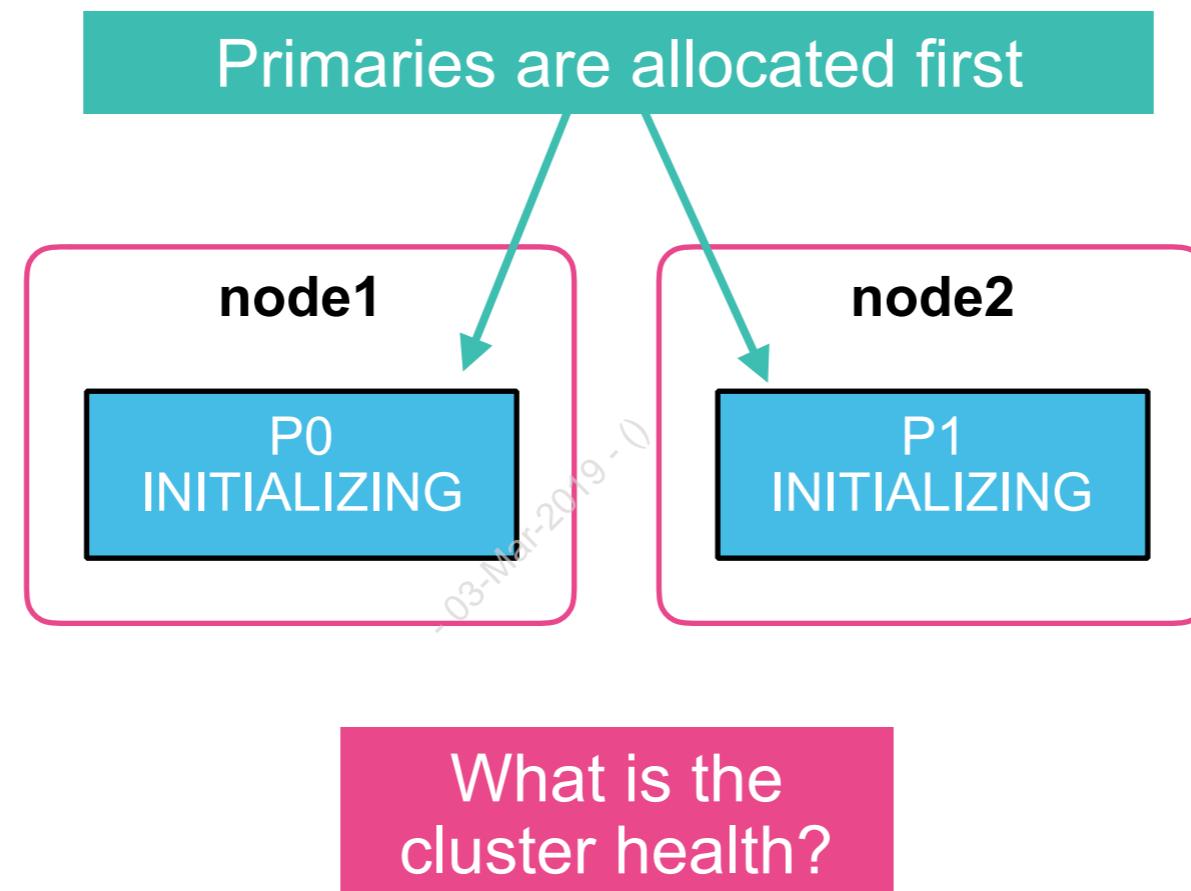
node1

node2

Four shards total, all in
the UNASSIGNED state

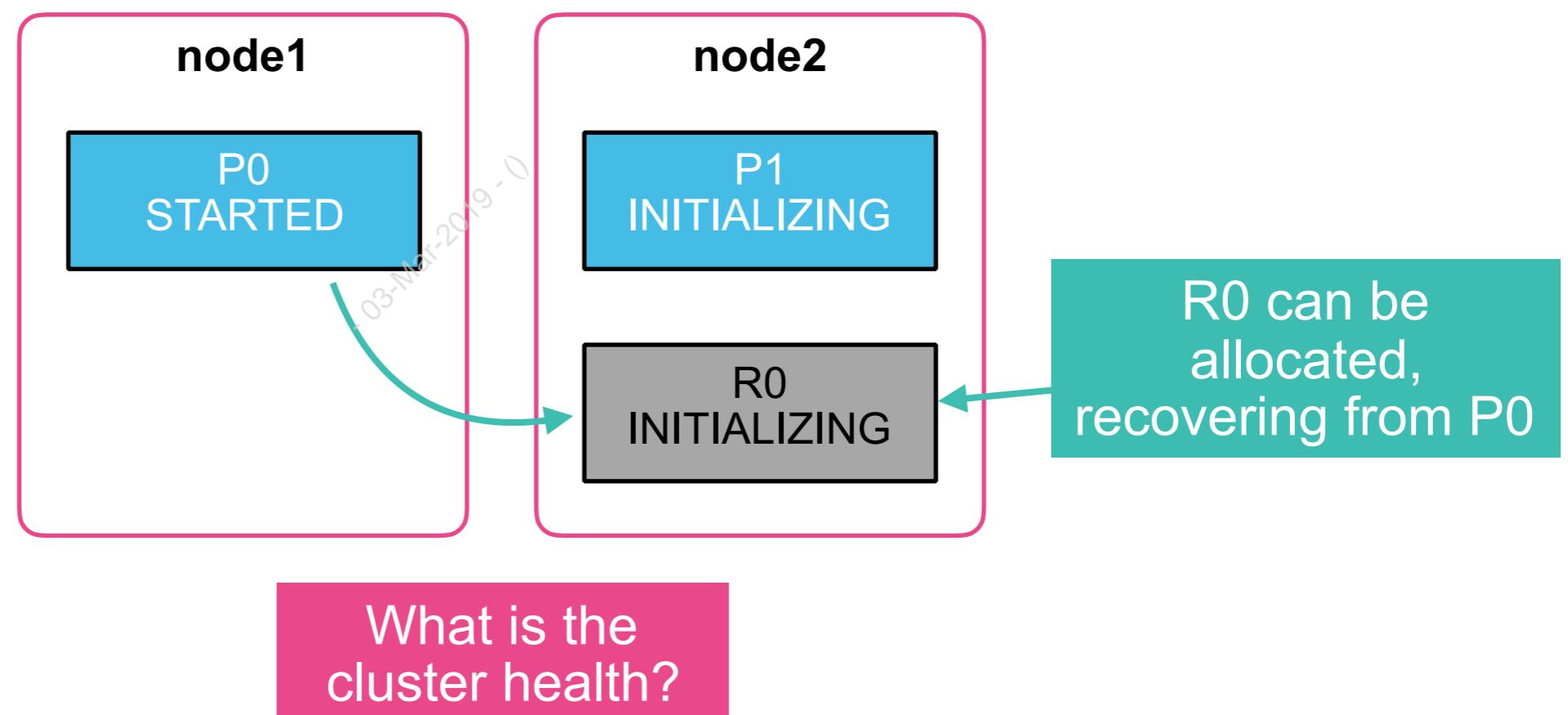
INITIALIZING

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



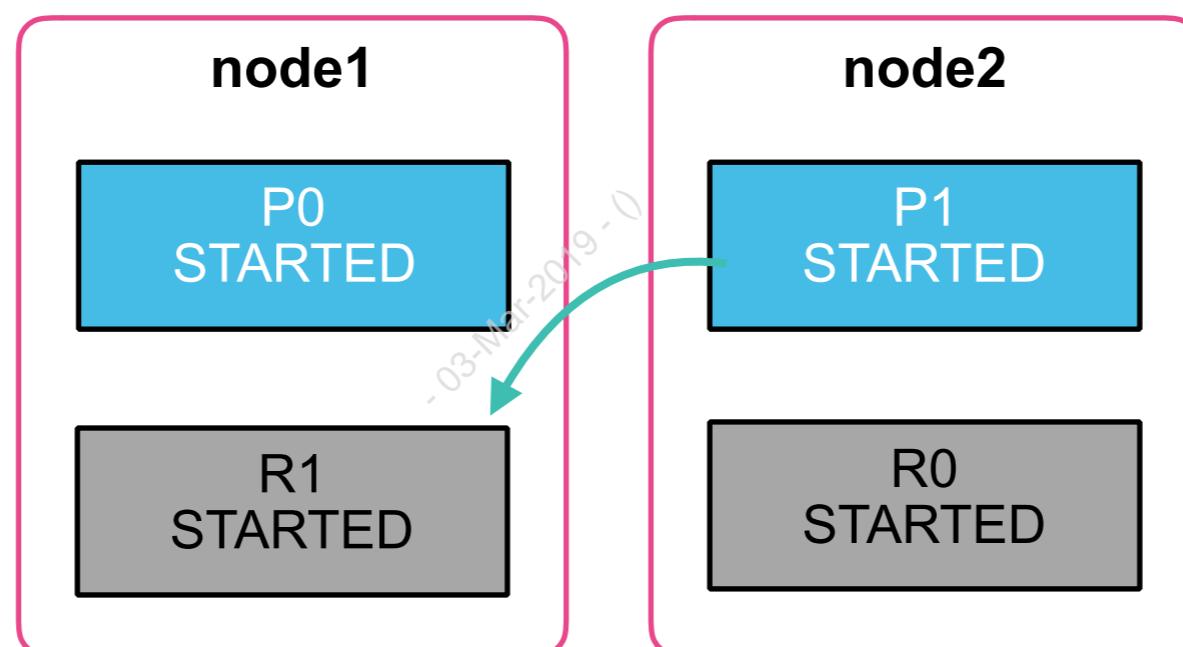
STARTED

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



STARTED

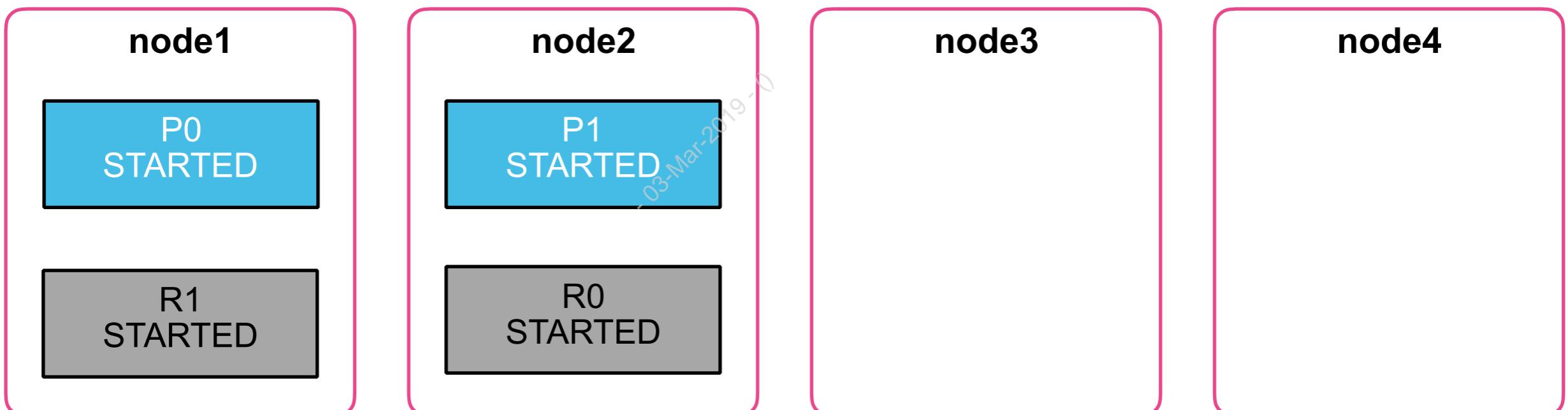
- Eventually, each primary and each replica will move from **INITIALIZING** to **STARTED**
 - the new index is fully allocated and distributed on the cluster



What is the
cluster health?

Adding Nodes

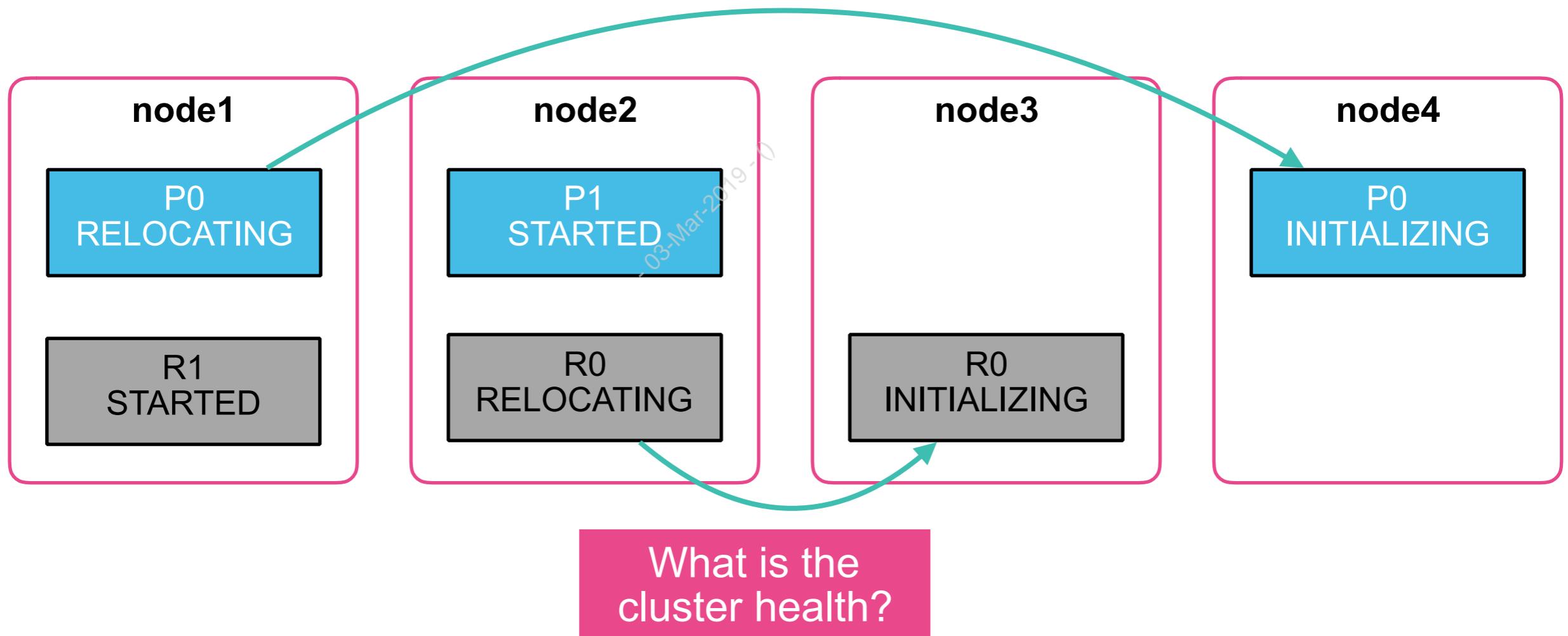
- Suppose we add a couple of nodes to our cluster:
 - the master node will attempt to balance the cluster by distributing shards to the new nodes



What is the
cluster health?

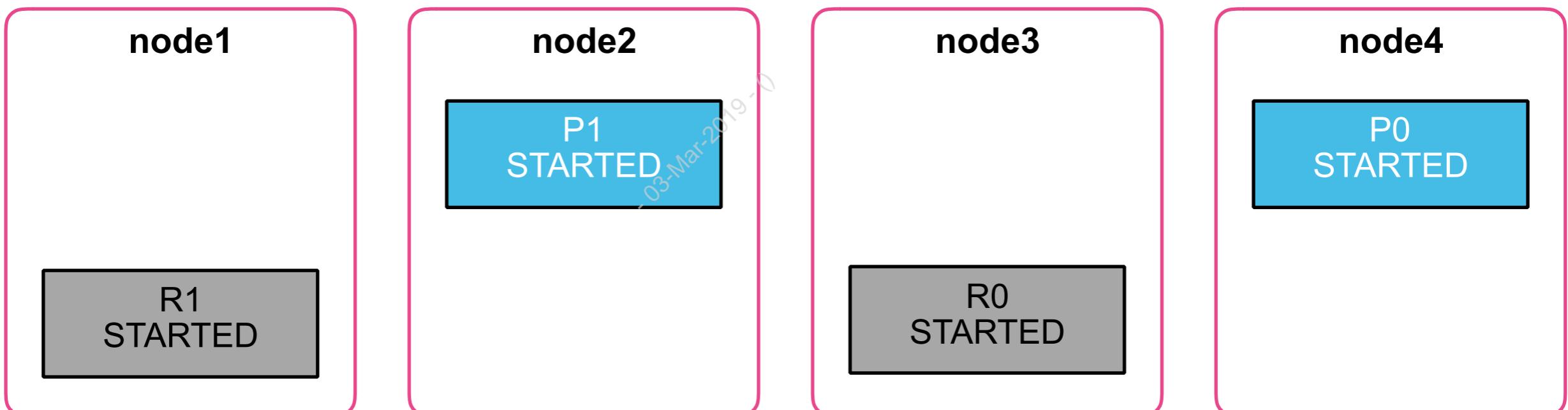
RELOCATING

- 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

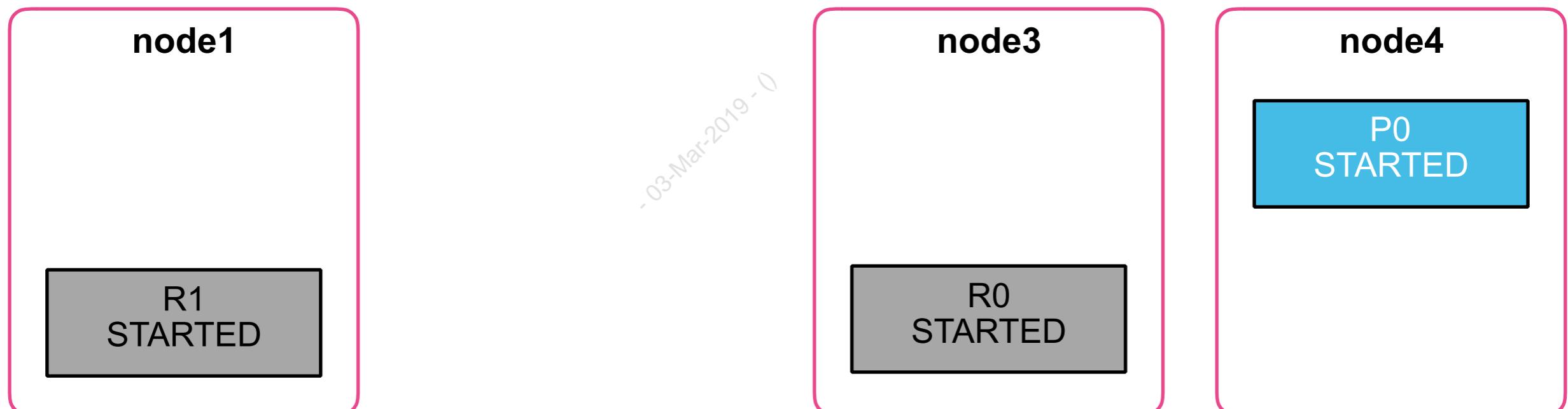
- After the relocated shard is started, the original shard gets removed:



What is the
cluster health?

Removing Nodes

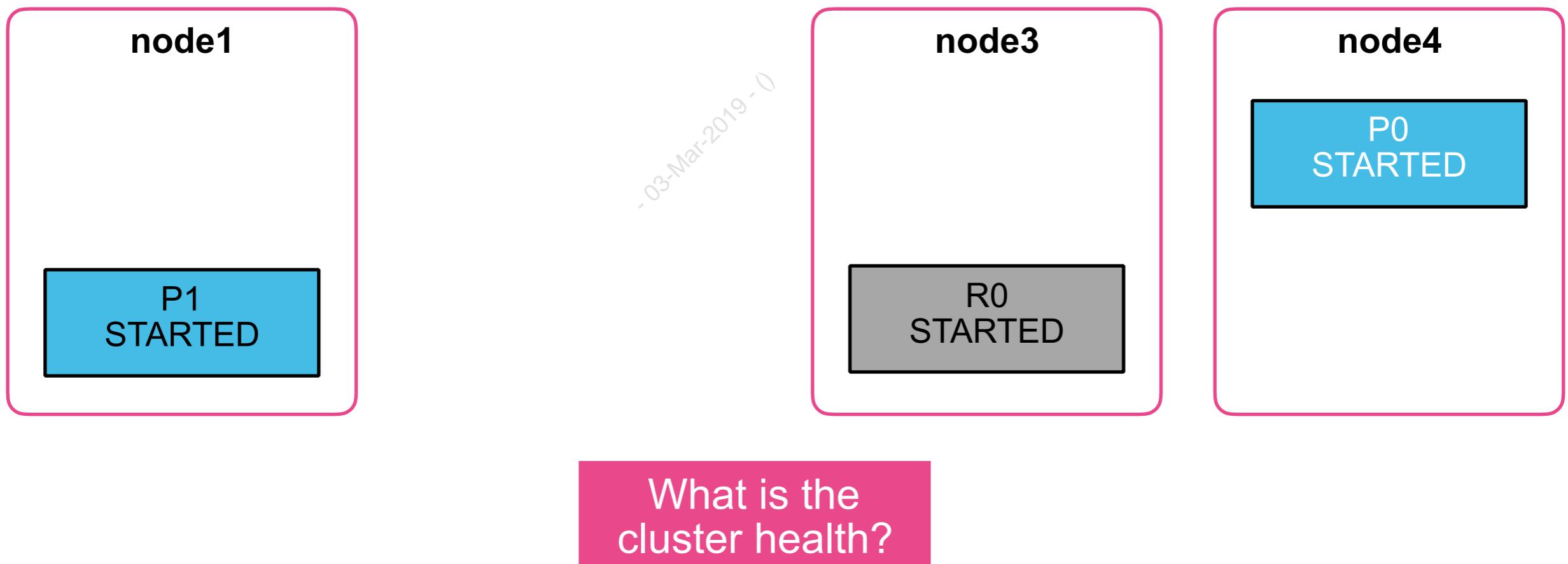
- Suppose we remove **node2** from the cluster (or it fails somehow):
 - no data is lost, because **R1** is a copy of **P1**



What is the
cluster health?

Shard Promotion

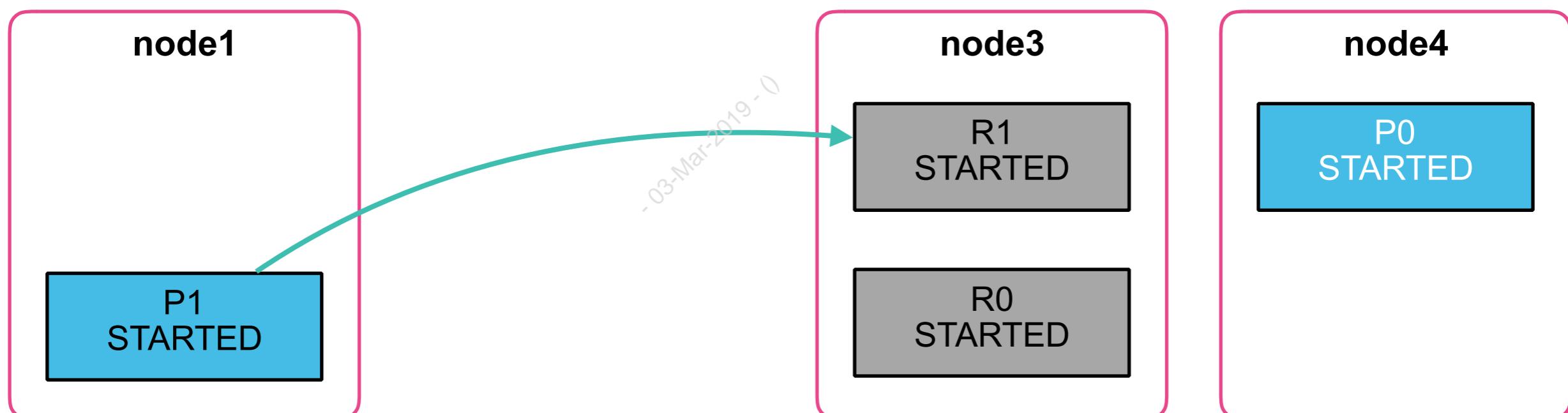
- R1 gets promoted to a primary shard, becomes P1:
 - replica promotion is really fast



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?



Diagnosing Health Issues

-03-Mar-2019-0

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 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 is **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 problem index (or indices)

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

set “**level**” equal to “**indices**” to view 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  
    },  
    ...  
  }  
}
```

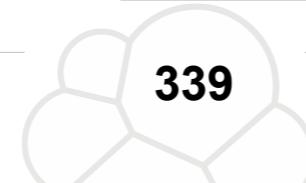
Health of a Specific Index

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

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

shard 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  
}
```



Shard Level

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

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

03-Mar-2019 - 0

helpful, but it is often
too many details

```
{  
  "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 details of the first unassigned shard that 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
```

we either have a node down or improper permission settings

```
{  
  "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",  
  ...  
}
```

shard 1 is a primary shard that is unassigned



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 details for just
this particular shard

Chapter Review

~03-Mar-2019~

Summary

- Transient settings take precedence over persistent settings, which take precedence over command-line settings, which take precedence over `elasticsearch.yml` settings
- **Shard allocation** is the process of assigning a shard to a node in the cluster
- A cluster has a **health** that contains various statistics and the status of the cluster
- A cluster's **health status** is either green, yellow, or red, depending on the current shard allocation
- A cluster's **routing table** contains which nodes are hosting which indices and shards
- Elasticsearch provides the **Cluster Allocation API** to help you locate any **UNASSIGNED** shards



Quiz

1. **True or False:** A *transient* setting survives a cluster restart.
2. Why should you bother checking the “_shards” section of a response?
3. What are the 4 states of a shard?
4. **True or False:** A cluster with a yellow status is missing indexed documents.
5. If your cluster has an unassigned primary shard, what is the cluster’s health status?

Lab 9

-03-Mar-2019-10

Troubleshooting Elasticsearch

Chapter 10

Improving Search Results

·03-Mar-2019·0

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- Our blog search application is missing some features that users would expect in any good search application, like:
 - the ability to **page** through a large result set
 - changing the **sort** order of the results
 - having the search terms **highlighted** in the response
 - recognizing search terms that are **spelled incorrectly**
 - searching for the title or author of a blog
- In this chapter, we will discuss how to provide some of these common (and expected) search features

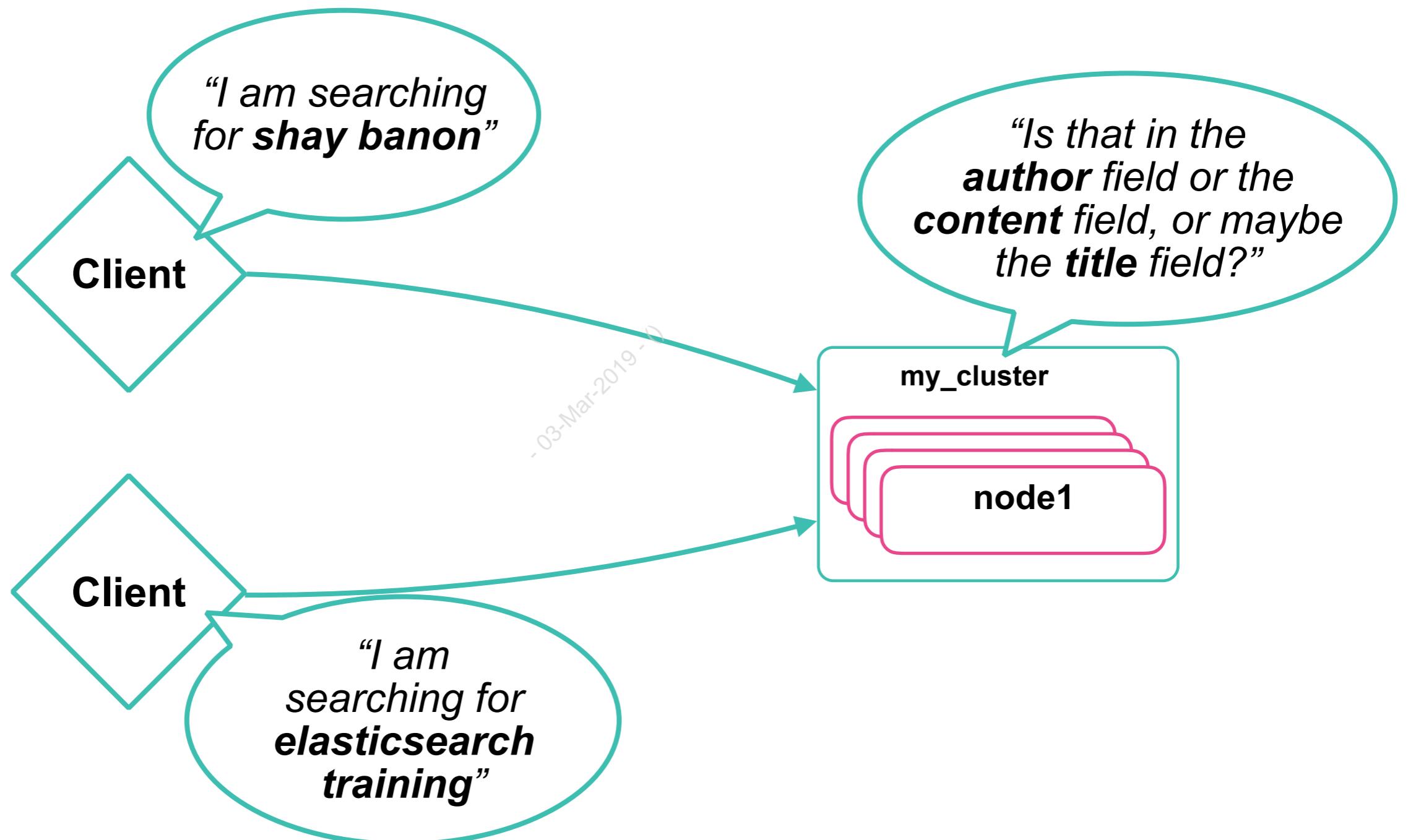


Multi-field Searches

-03-Mar-2019-

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...

~03-Mar-2019~0



The multi_match Query

- The **multi_match** query provides a convenient shorthand for running a **match** query against multiple fields
 - By default, the **_score** from the best field is used (a **best_fields** search)

```
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": 12.121704

"author": "Tim Roes"

"title": "Making Kibana Accessible"

"content": " Inclusivity and diversity are primary ... Our CEO, **Shay Banon**, ... "

"_score": 10.22507

"author": "Livia Froelicher"

"title": "Where in the World Is Elasticsearch - July 14, 2014"

"content": " Last week was awesome ... amazing community — **Shay Banon** (@kimchy) ...
Our own crew with **Shay Banon**, ... "

"_score": 9.988037

"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 Elastic **Shay Banon**, ... "



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"
      ],
      "type": "best_fields"
    }
  }
}
```

boost title by 2

8-Mar-2019 →

```
"_score": 16.307825
"author": "Steven Schuurman"
"title": "The Next Chapter: Shay Banon to Succeed Steven Schuurman as Elastic CEO"

"_score": 12.121704
"author": "Tim Roes"
"title": "Making Kibana Accessible"

"_score": 12.121704
"author": "Livia Froelicher"
"title": "Where in the World Is Elasticsearch - July 14, 2014"
```



Is there a best practice for boosting?

- Well, it depends!
 - Experiment and analyze your search results to find that 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"
      ],
      "type": "best_fields"
    }
  }
}
```

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

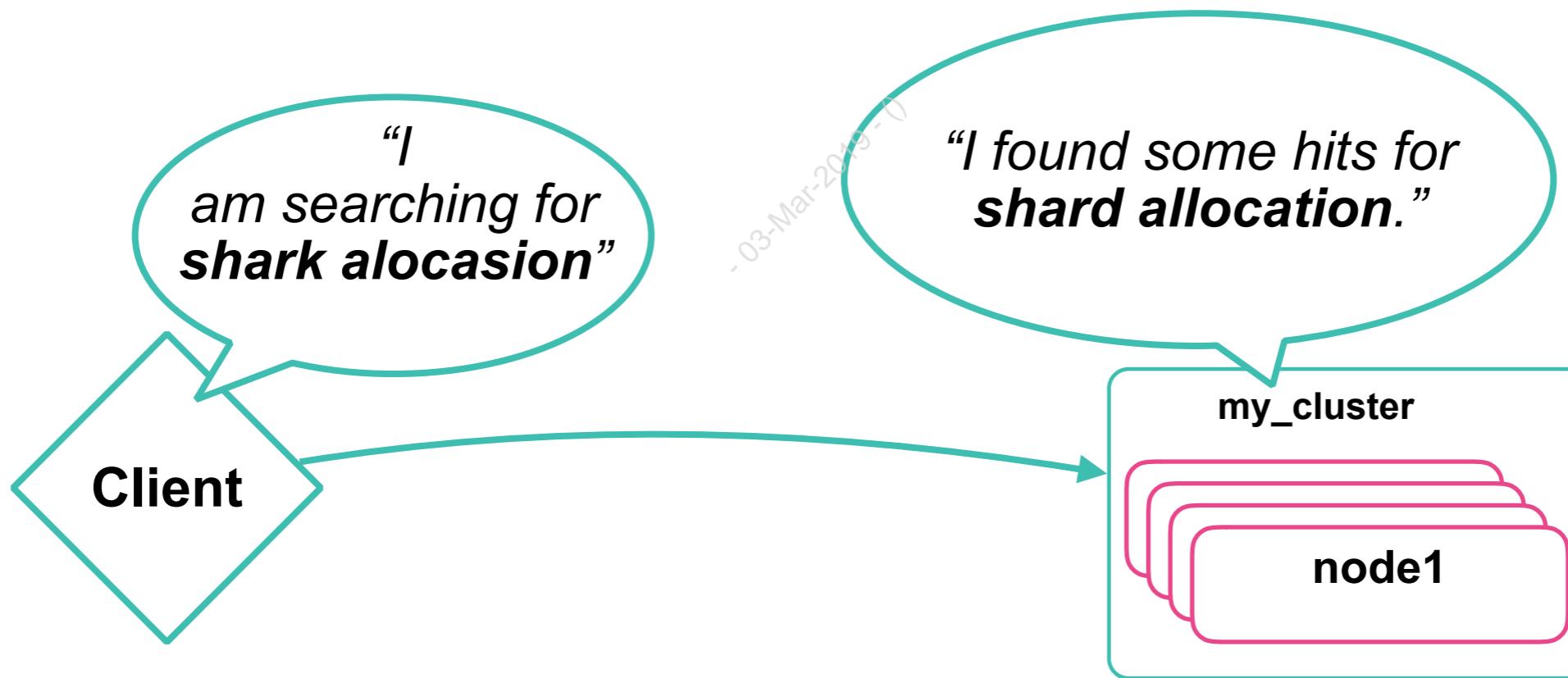


Mispelt Werds

„03-Mar-2010“

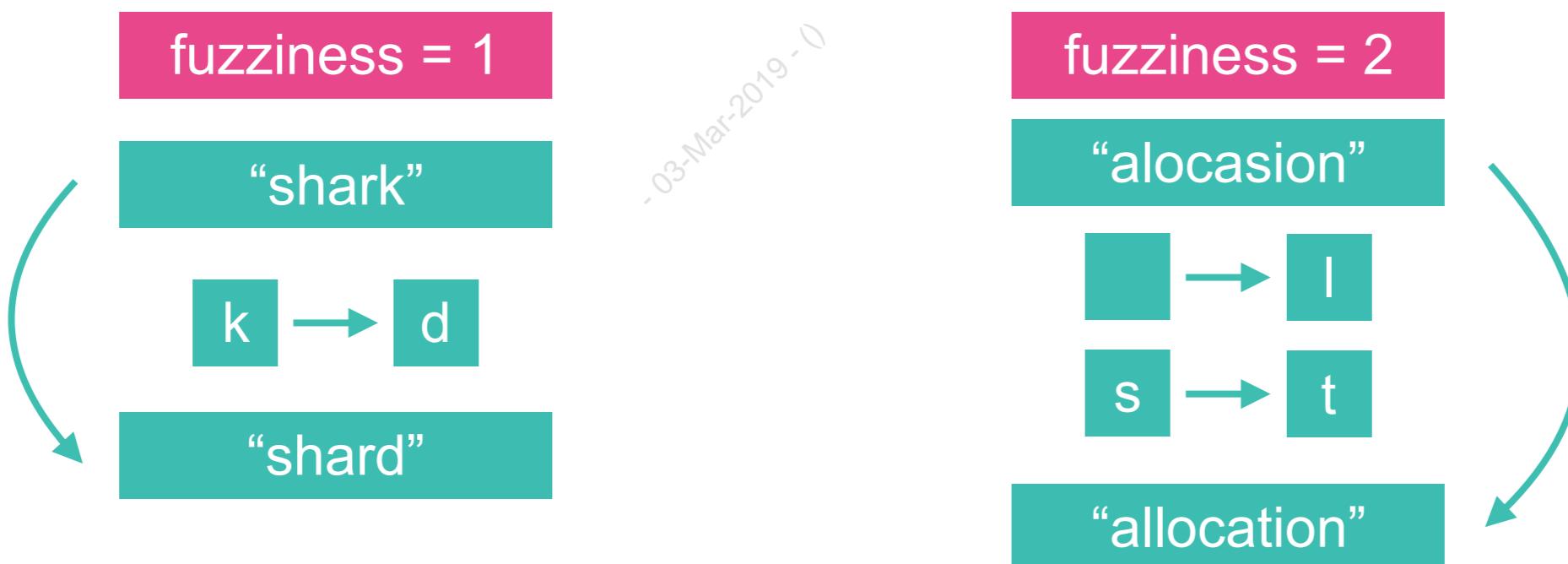
Mispelt Werds

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



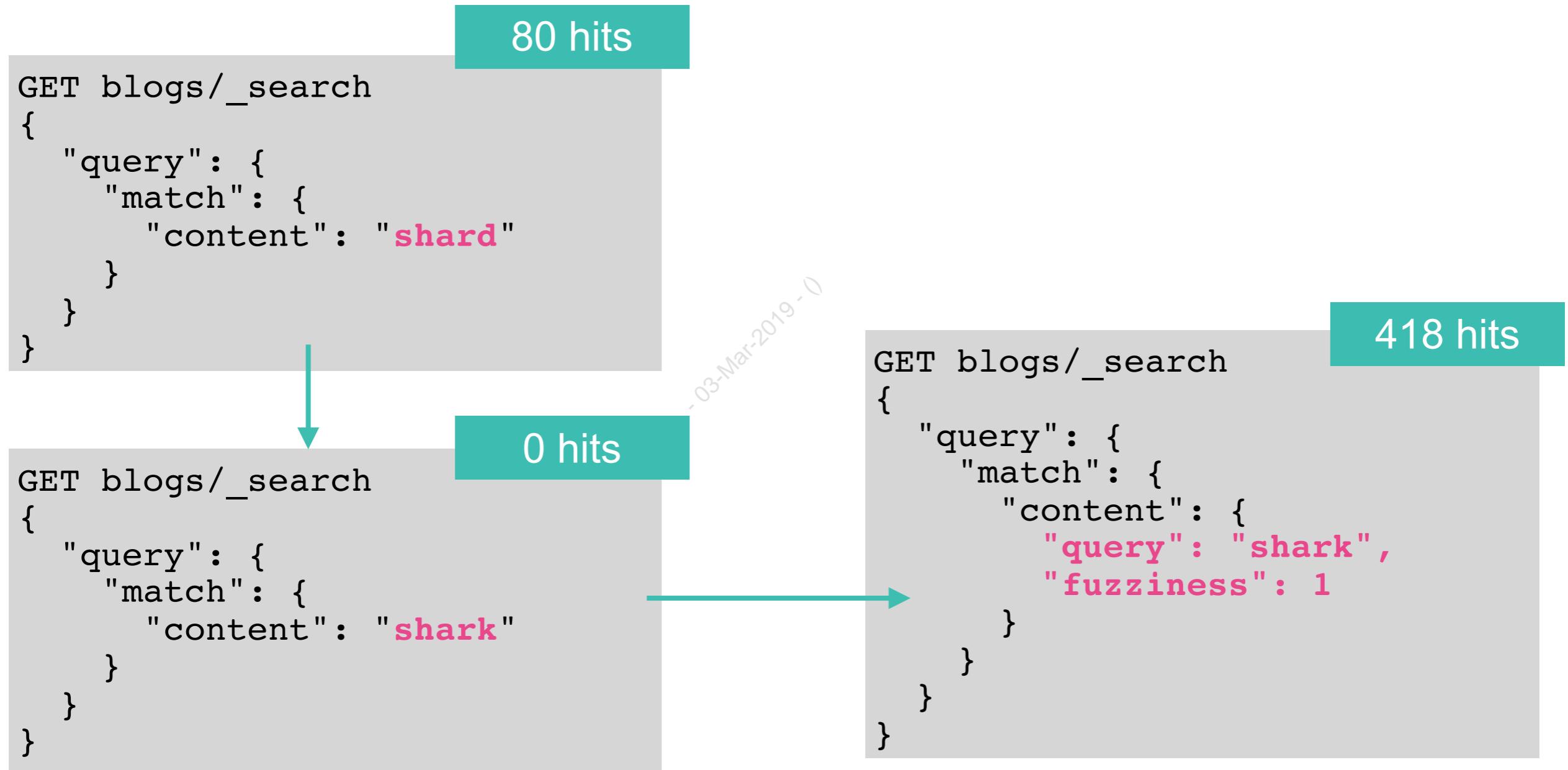
What is Fuzziness?

- **Fuzzy matching** treats two words that are “*fuzzily*” similar as if they were the same word
 - Fuzziness is something that can be assigned a value
 - 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”



Multiple terms in the query

- If the query has multiple terms, the fuzziness value is *applied to each term*

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

A “fuzziness” of 1 results in
“monitering datu” matching
“monitoring data”



Choosing the Fuzziness

- Be aware of the side effect of **fuzziness** and short terms
 - if you set **fuzziness** to 2, then "hi" matches "jim", "tim", "uri", "phil", "cj", "ali" and so on

```
GET blogs/_search
{
  "size": 100,
  "_source": "author",
  "query": {
    "match": {
      "author": {
        "query": "hi world",
        "fuzziness": 2
      }
    }
  }
}
```



86 hits, including...

"Loggy D. Wood"

"Jim Goodwin"

"Tim Sullivan"

"Uri Cohen"

"Phil Verdemato"

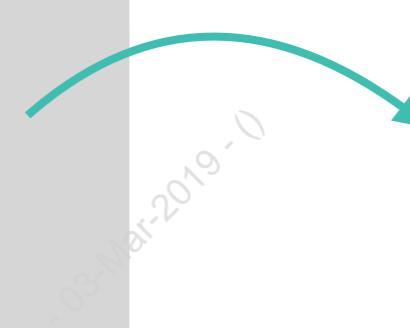
"CJ Cenizal"

"Ali Beyad"

If you are not sure what fuzziness to use...

- **fuzziness** can be set to “auto”
 - generates an edit distance based on the *length of the query terms*
 - “auto” is actually the preferred way to use fuzziness

```
GET blogs/_search
{
  "size": 100,
  "_source": "author",
  "query": {
    "match": {
      "author": {
        "query": "hi world",
        "fuzziness": "auto"
      }
    }
  }
}
```



0 hits with “auto”, which kind of makes sense with this example

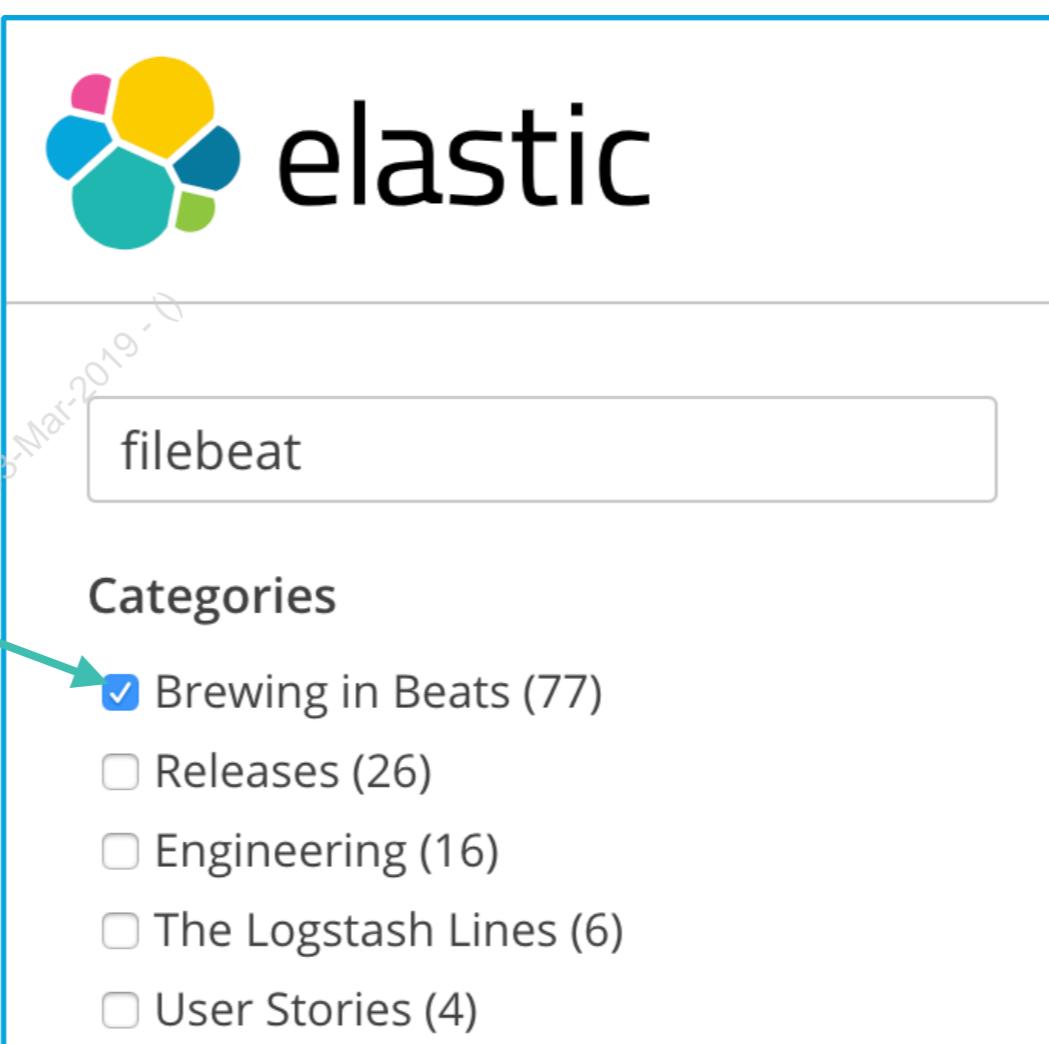


Searching for Exact Terms

03-May-2019 - 0

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



A “category” filter will require exact matches

The screenshot shows a user interface for searching blog posts. At the top is the Elastic logo. Below it, the word "filebeat" is displayed in a search bar. Underneath the search bar is the heading "Categories". A list of categories follows, each preceded by an input checkbox. The "Brewing in Beats" checkbox is checked, while the others are unchecked. The categories listed are: Brewing in Beats (77), Releases (26), Engineering (16), The Logstash Lines (6), and User Stories (4). A watermark "03-Mar-2019-D" is visible diagonally across the interface.

Category	Count
Brewing in Beats	77
Releases	26
Engineering	16
The Logstash Lines	6
User Stories	4



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": {
    "match": {
      "category": "Brewing in Beats"
    }
  }
}
```

We get back all categories containing “*in*”

"Brewing in Beats"

"This week in
Elasticsearch and
Apache Lucene"

"Kurrently in Kibana"



Using the keyword Field

- If you need ***exact matches*** in your queries, you should index the string as **keyword**
 - then search on the **keyword** field

```
GET blogs/_search
{
  "query": {
    "match": {
      "category.keyword": "Brewing in Beats"
    }
  }
}
```

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

“Brewing in Beats”



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": "filebeat"
          }
        },
        {
          "filter": {
            "match": {
              "category.keyword": "Brewing in Beats"
            }
          }
        }
      ]
    }
  }
}
```

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

No need for category to contribute to the `_score`



Will users enter a category exactly?

- No way! In our blogs use case, we would not expect our users to type a category exactly
 - we would need to *provide a UI* for selecting a category

We probably can't count on users typing in a category exactly, so we provided an UI

03 Mar 2019 - 0

The screenshot shows a search interface with the Elasticsearch logo at the top. On the left, there's a sidebar titled "Categories" containing two items: "Brewing in Beats (93)" and "Engineering (1)". A teal callout box with the text "We probably can't count on users typing in a category exactly, so we provided an UI" has arrows pointing to both the sidebar title and the first category item. The main search results area is visible on the right.

Brewing in Beat

Categories

Brewing in Beats (93)

Engineering (1)



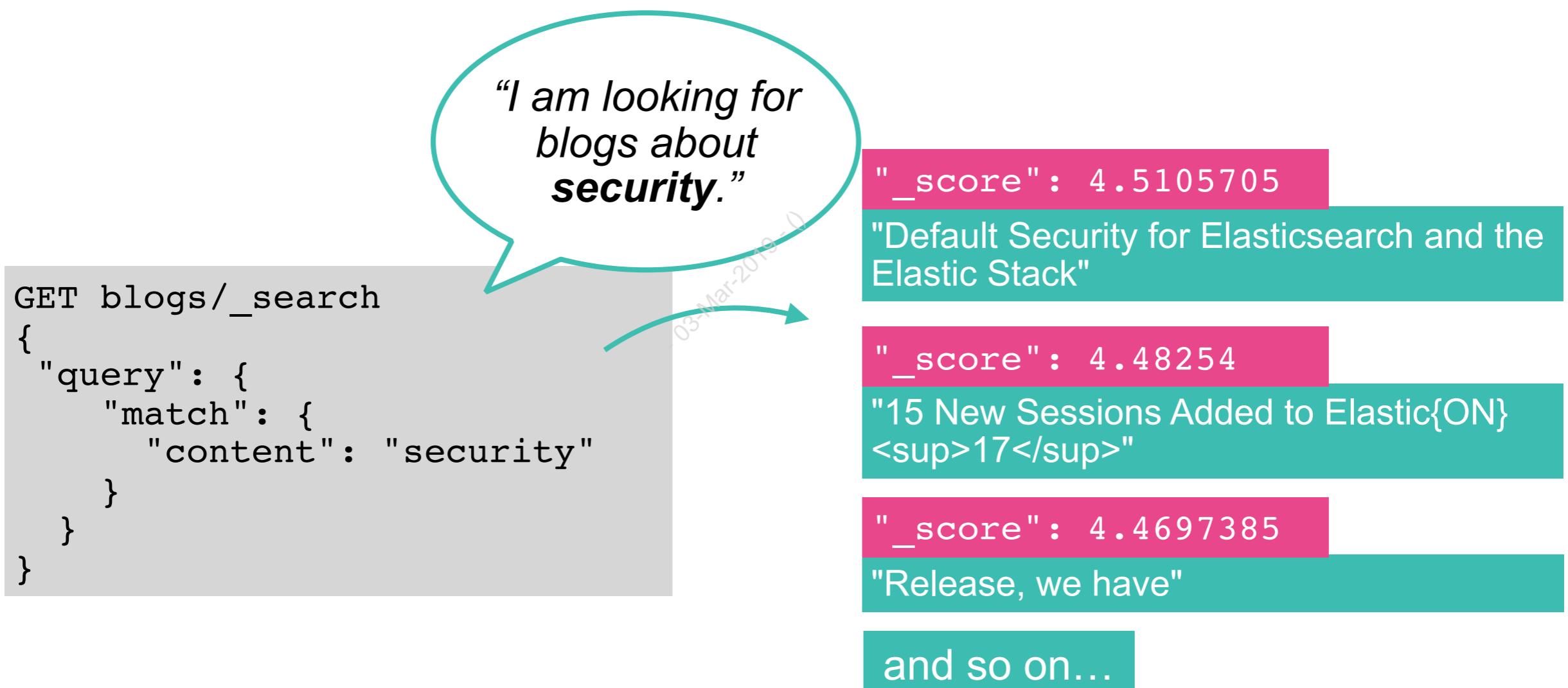
Sorting

·03-Mar-2019·



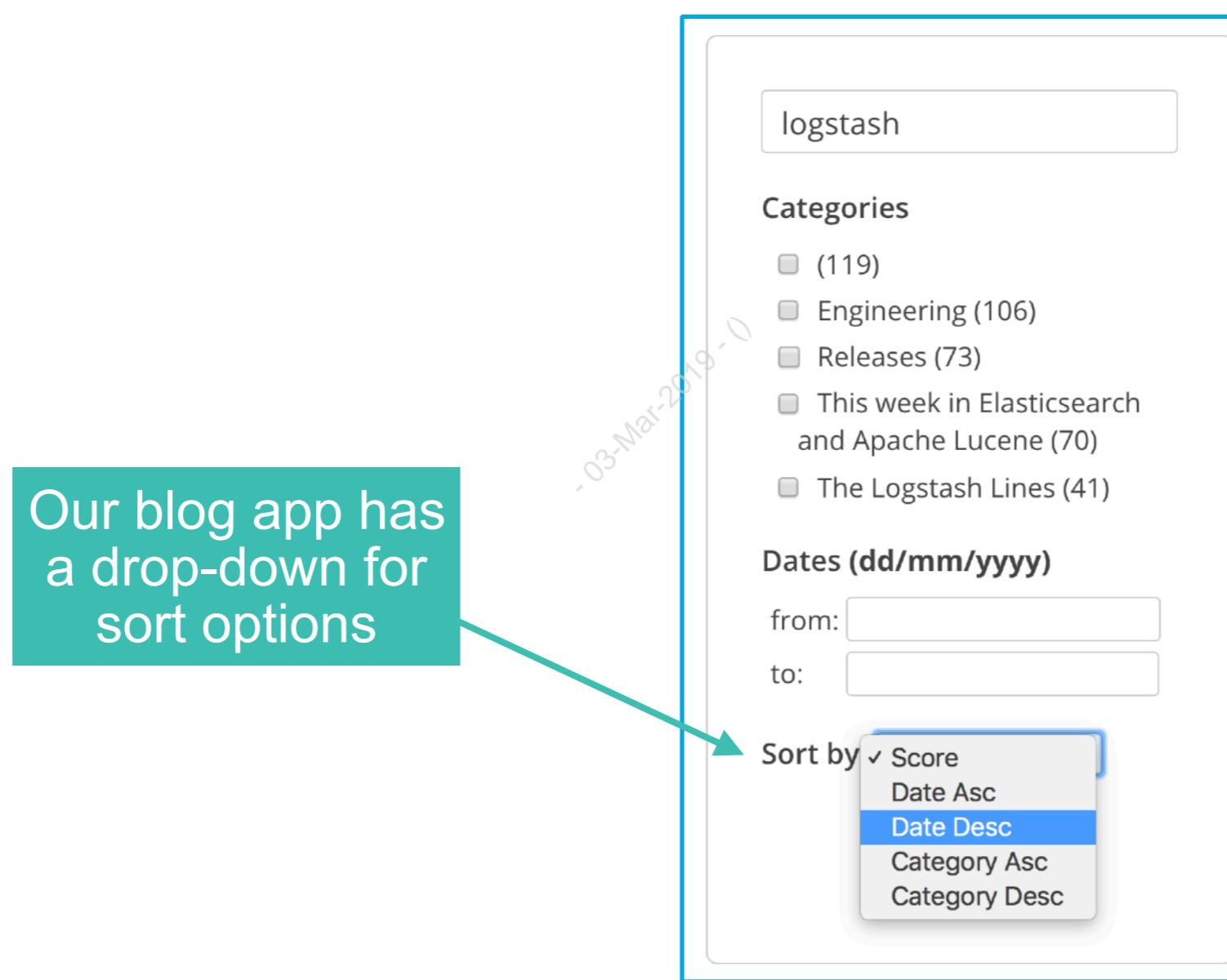
Sorting Results

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



Let's provide other sort options

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



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

~03-Mar-2019~0

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

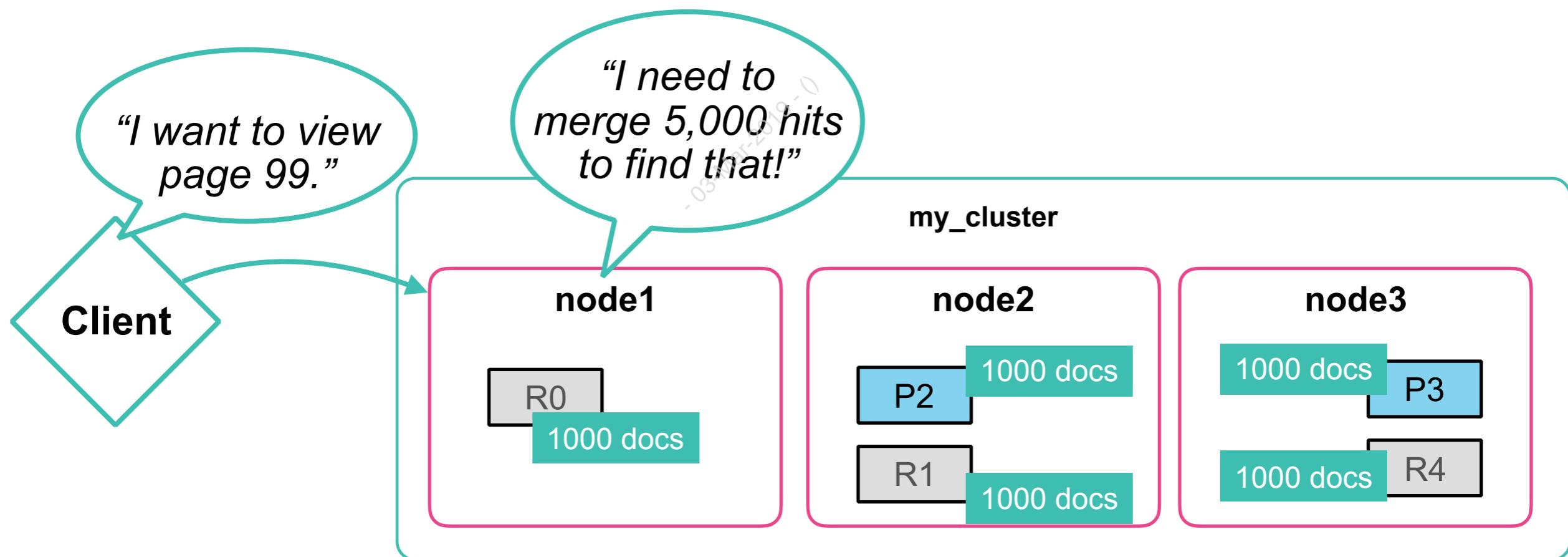
```
GET blogs/_search
{
  "from": 10,
  "size": 10,
  "query": {
    "match": {
      "content": "elasticsearch"
    }
  }
}
```

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



Be Careful with Deep Pagination

- Suppose you want hits from 990 to 1,000
 - That requires a 1,000 documents *from each shard* to be considered!
 - To avoid overwhelming a cluster, *from + size* can not be more than `index.max_result_window` (which defaults to 10,000)



The `search_after` Parameter

- If you have a lot of pages, a better option for retrieving subsequent pages is to use the `search_after` parameter
 - The first page does not need a `search_after`

```
GET blogs/_search
{
  "size": 10,
  "query": {
    "match": {
      "content": "elasticsearch"
    }
  },
  "sort": [
    {"_score": {"order": "desc"}},
    {"_id": {"order": "asc"}}
  ]
}
```

Run the initial search...and keep track of the “`sort`” block from the last document returned

Add `_id` to avoid any “ties” in sorting

The search_after Parameter

- For the subsequent query, set the **search_after** parameter to **the last sort values** from the previous result set:

```
GET blogs/_search
{
  "size": 10,
  "query": {
    "match": {
      "content": "elasticsearch"
    }
  },
  "sort": [
    {"_score": {"order": "desc"}},
    {"_id": {"order": "asc"}}
  ],
  "search_after": [
    0.55449957,
    "1346"
]
}
```

"I want the next 10 hits."

The previous hit had a
_score=0.55449957 and _id = 1346



Highlighting

~03-Mar-2019~

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)

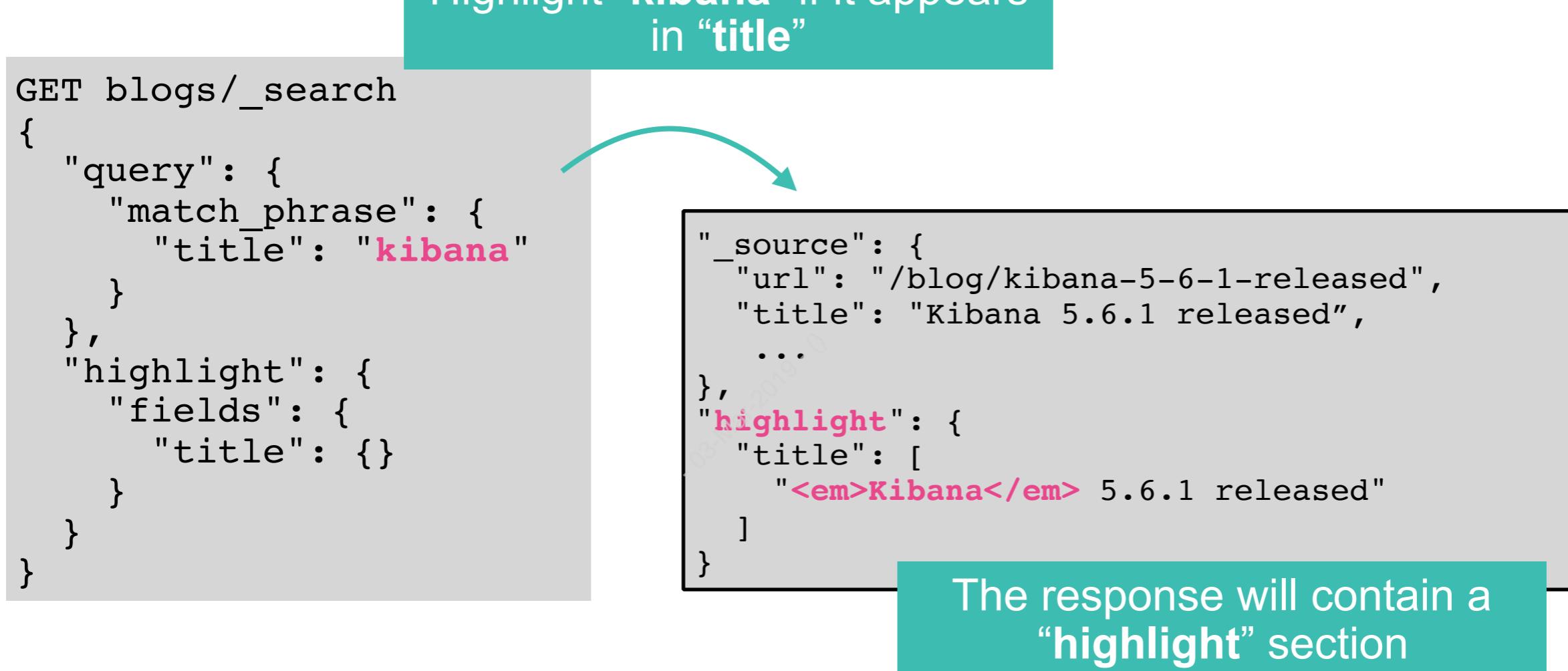
Logstash 6.1.0 Released [_score: 10.546306]
December 13, 2017 [Releases]
Logstash 6.1.0 has launched!, Read on for what's new in Logstash 6.1.0., We're proud to announce a great new way to extend Logstash functionality in 6.1.0., scripting via the Logstash Ruby filter., We've been working on a full rewrite 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:



Highlighting multiple fields

- Note: you have to query a field if you want it highlighted in the results (or set `require_field_match: false`)

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

```
"highlight": {
  "title": [...]
}
"content" not highlighted
```

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

```
"highlight": {
  "title": [...],
  "content": [...]
}
"content" is highlighted
```

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"
  ]
}
```



Chapter Review

·03-Mar-2019·0

Summary

- The **multi_match** query provides a convenient shorthand for running a **match** query against multiple fields
- **Fuzzy matching** treats two words that are “**fuzzily**” similar as if they were the same word
- If you need searches for **exact text**, we typically index the text as **keyword**
- 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 query



Quiz

1. What is the behavior of “**best_fields**” in a “**multi_match**” query?
2. What is the “**fuzziness**” edit distance from “**the beatles**” and “**te beetles**” (assuming **standard** analyzed text)?
3. What two parameters are used to implement paging of search results?
4. 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"
    }
  }
}
```



Lab 10

Improving Search Results

~03-Mar-2019



Chapter 11

Aggregating Data

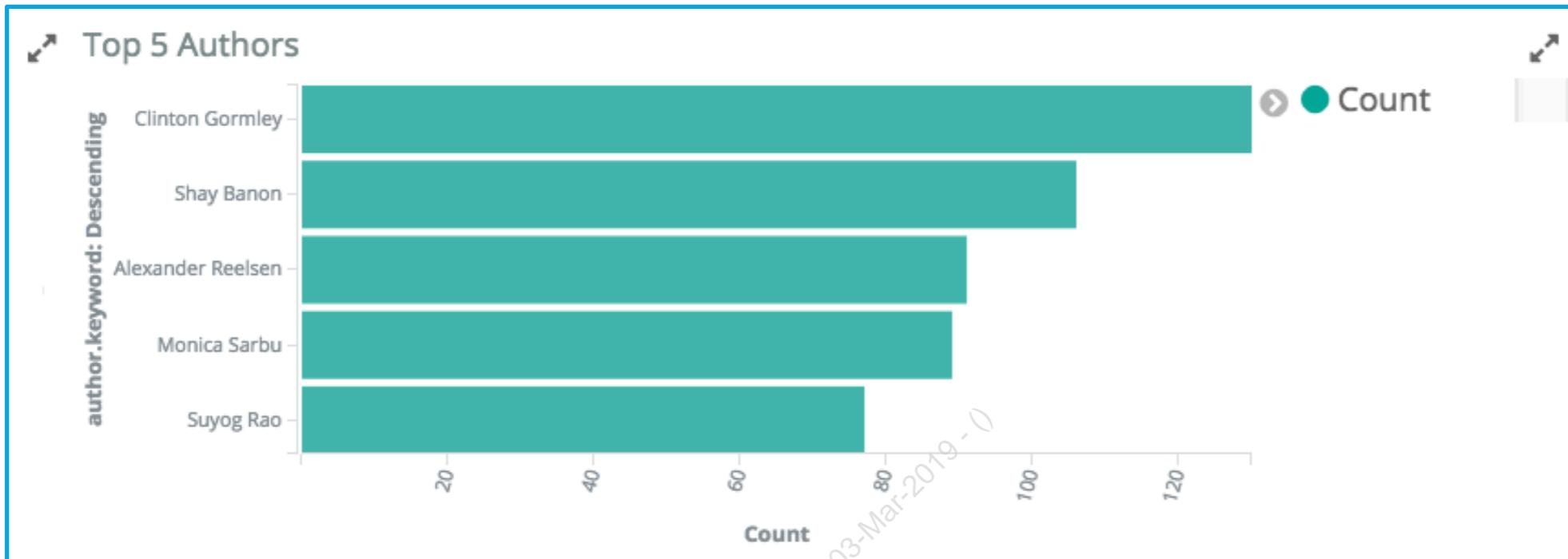
03-Mar-2019

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- This chapter discusses aggregations and how they are used to analyze and visualize your data



Terms Per Category

Engineering: category	: category	Releases: category	
title.stop: Descending	Count	title.stop: Descending	Count
elasticsearch	164	elasticsearch	121
released	70	where	93
kibana	52	world	93
elastic	44	elastic	80
logstash	43	released	66

What are Aggregations?

What is an Aggregation?

- We have been focusing on search, but Elasticsearch has another powerful capability known as aggregations
- **Aggregations** are a way to perform analytics on your indexed data

What movies have “star” in the title?

*That
is a search
query...*

*What
is the average ticket price
of a movie in the U.S?*

*That
is an aggregation...*

03-Mar-2019 - 0



You ask different questions with aggs:

- In **search**, we have been asking for a subset of the original dataset:
 - Results are **hits** that match our search parameters

“What are the **ERROR**'s in our log file?”

- In **aggregations**, we analyze the data by asking questions about the dataset:
 - Results are (typically) **computed values**

“How many 404 errors occurred each day last month?”

“What is the average amount of time visitors spend on our home page?”

“What is the most visited page on our website?”



Aggs in our Blog Search App

aggregations

search criteria

performance

Categories

- Engineering (110)
- User Stories (48)
- (36)
- Releases (30)
- News (21)

Dates (dd/mm/yyyy)

from:

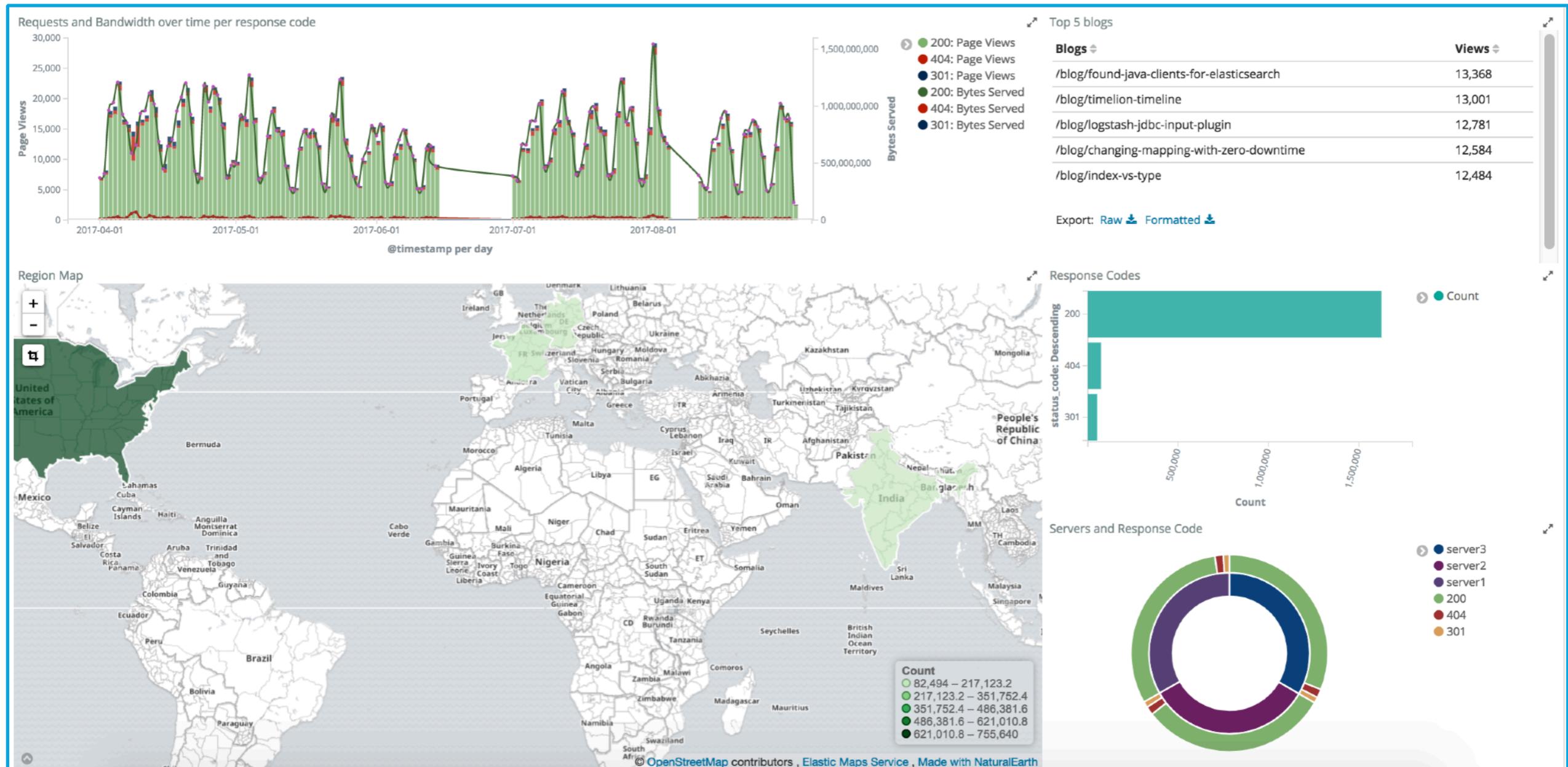
to:

Sort by:

Score 



Kibana Visualizations



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
average response
size of our blog
page?*

```
GET logs_server*/_search
{
  "aggs": {
    "average_response_size": {
      "avg": {
        "field": "response_size"
      }
    }
  }
}
```

The “**avg**” aggregation
computes the average of a
numeric field

In this agg, the average of
“**response_size**” is calculated
over all documents

Viewing the Results

- The results will have an “**aggregations**” section with the results of all the “**aggs**” in your search

```
"hits": {  
  "total": 1751476,  
  "max_score": 1,  
  "hits": [  
    ...  
  ]  
},  
"aggregations": {  
  "average_response_size": {  
    "value": 54426.026095124435  
  }  
}
```

We get the top 10 hits...

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

The average response size over all docs

If you just want the aggregation value:

- 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 since the “fetch” phase can be skipped

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "average_response_size": {
      "avg": {
        "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": {
    "average_response_size": {
      "value": 54426.026095124435
    }
  }
}
```



The Scope of an Agg

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

*What is the
average response size
where the language
code is French?*

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

```
"hits": {
  "total": 99672,
  "max_score": 0,
  "hits": []
},
"aggregations": {
  "average_french_response": {
    "value": 51120.747261016135
  }
}
```

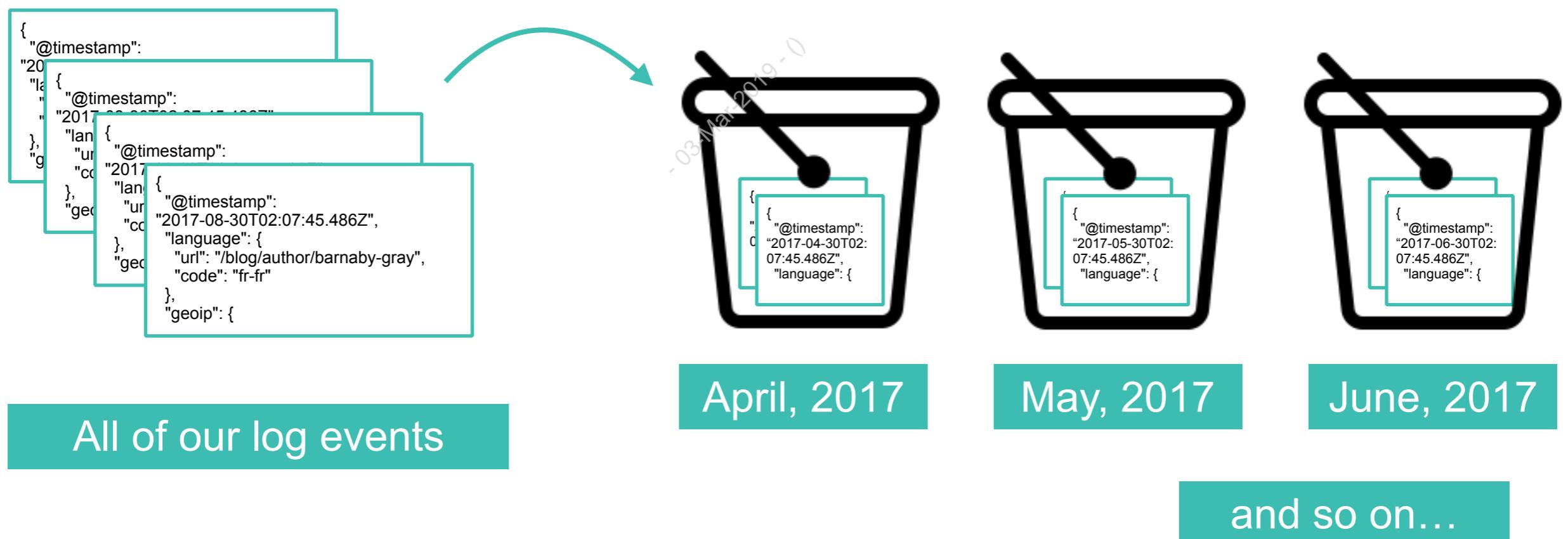
The average **size** is
computed over 99,672
docs (instead of all docs)

Buckets and Metrics

· 03-Mar-2019 · 0

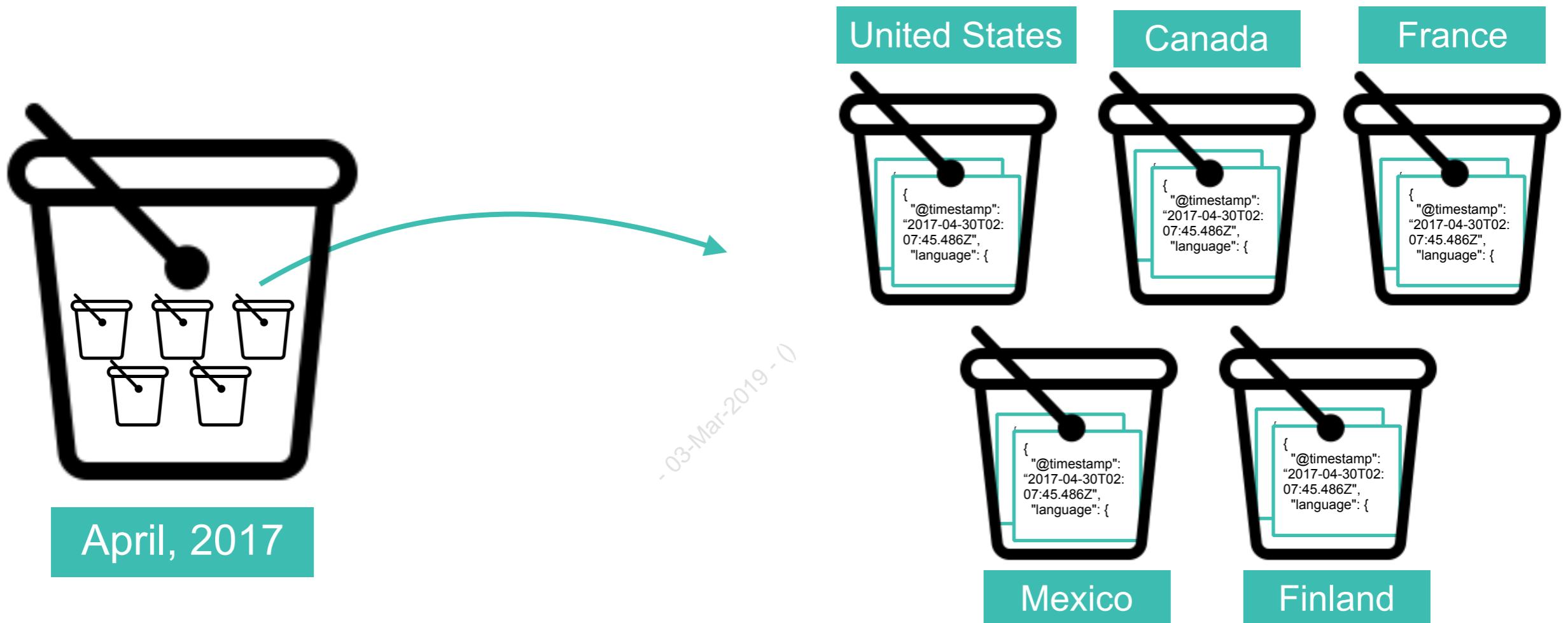
Buckets

- A *bucket* is simply a collection of documents that meet a criterion
 - Buckets are a key element of aggregations
- For example, suppose we want to analyze log traffic by month:



Buckets can be Nested

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



A bucket can consist of nested buckets within it

Logs from April, 2017, bucketed by country



Metrics

- **Metrics** compute numeric values based on your dataset
 - Either from fields
 - Or from values generated by custom scripts
- Most metrics are mathematical operations that output a single value:
 - **avg, sum, min, max, cardinality**
- Some metrics output multiple values:
 - **stats, percentiles, percentile_ranks**

An Agg can be Metrics + Buckets

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



Let's answer some
aggregation questions...

-03-Mar-2019-

The max Aggregation

- What is the largest value of “response_size”?

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

“What is the maximum value of the response_size field?”

```
"aggregations": {
  "max_response_size": {
    "value": 270740
  }
}
```

The stats Aggregation

- The **stats** metric aggregation gives you common statistics of a field in a single aggregation request:
 - count, min, max, avg and sum

*What can you
tell me about the
“**response_size**” field?*

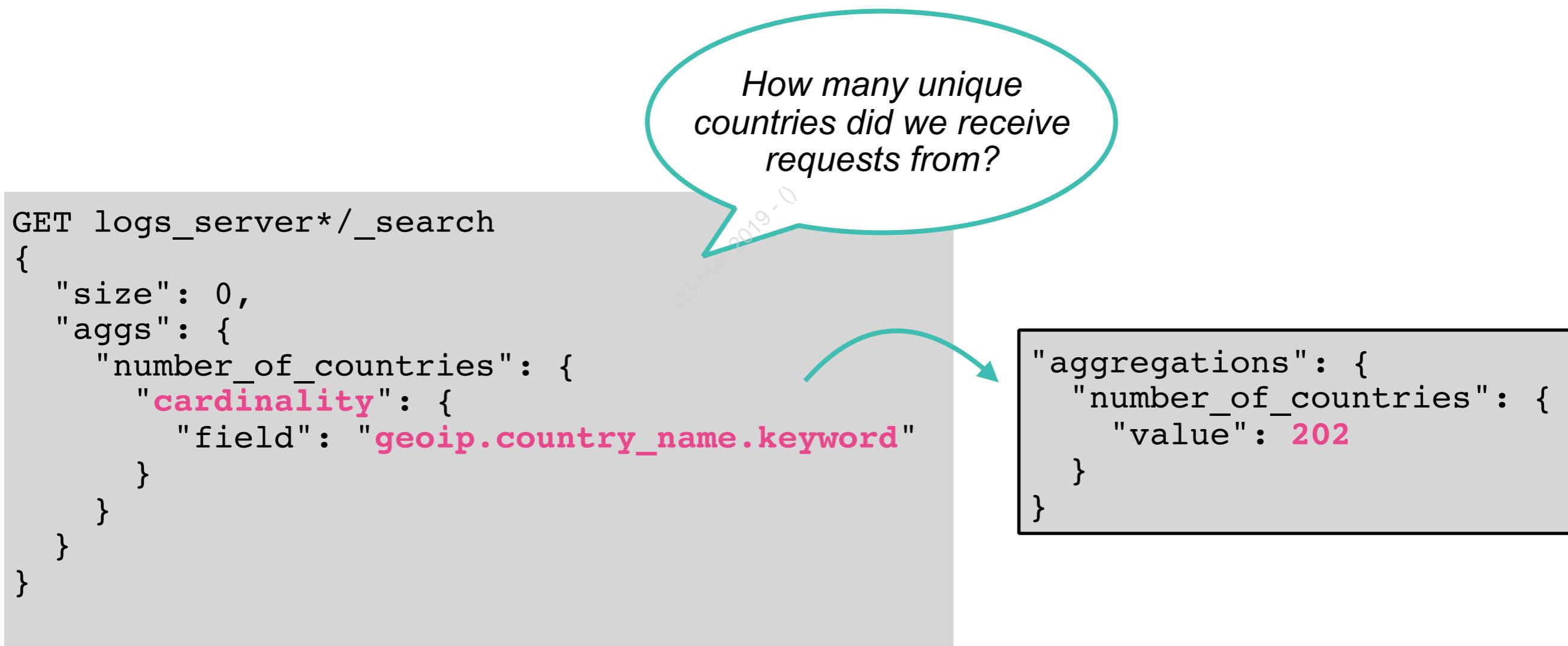
```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "size_stats": {
      "stats": {
        "field": "response_size"
      }
    }
  }
}
```

```
"aggregations": {
  "size_stats": {
    "count": 1751285,
    "min": 98,
    "max": 270740,
    "avg": 54426.026095124435,
    "sum": 95315483110
  }
}
```



The cardinality Aggregation

- The result may not be exactly precise for large datasets
 - based on the HyperLogLog++ algorithm
 - close to accurate up until **precision_threshold** (max of 40,000)

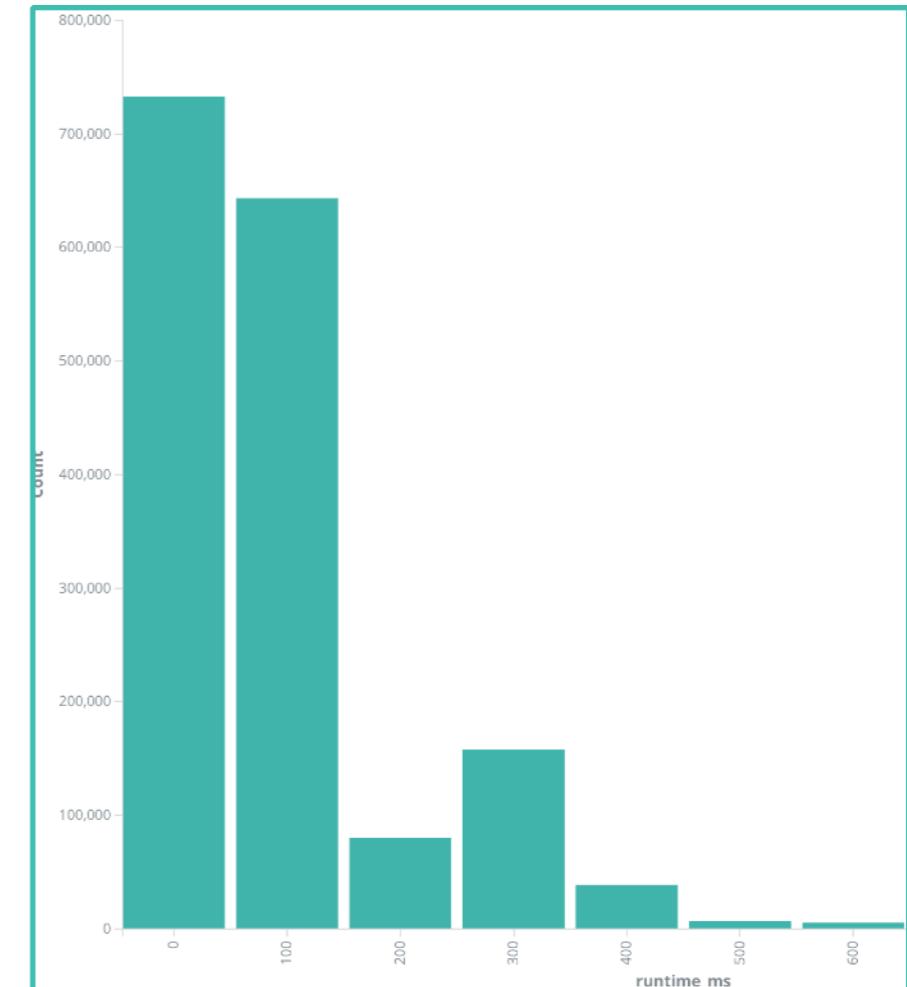


The histogram Aggregation

- The *histogram* bucket aggregation builds a histogram on a given “field” using a specified “interval”:
 - A histogram is like a bar chart that shows how many values fit into various intervals

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

A bucket is created for every 100 milliseconds



The API result:

- The number of buckets is dynamic and depends on the input values and the interval

80,161 log events had a runtime between 200 ms (inclusive) and 300 ms (exclusive)

03-Mar-2019 - 0



```
" aggregations": {  
   "runtime_histogram": {  
     "buckets": [  
       {  
         "key": 0,  
         "doc_count": 732687  
       },  
       {  
         "key": 100,  
         "doc_count": 643341  
       },  
       {  
         "key": 200,  
         "doc_count": 80161  
       },  
       {  
         "key": 300,  
         "doc_count": 157539  
       },  
       {  
         "key": 400,  
         "doc_count": 38759  
       },  
       {  
         "key": 500,  
         "doc_count": 10000  
       },  
       {  
         "key": 600,  
         "doc_count": 10000  
       },  
       {  
         "key": 700,  
         "doc_count": 10000  
       },  
       {  
         "key": 800,  
         "doc_count": 10000  
       },  
       {  
         "key": 900,  
         "doc_count": 10000  
       },  
       {  
         "key": 1000,  
         "doc_count": 10000  
       },  
       {  
         "key": 1100,  
         "doc_count": 10000  
       },  
       {  
         "key": 1200,  
         "doc_count": 10000  
       },  
       {  
         "key": 1300,  
         "doc_count": 10000  
       },  
       {  
         "key": 1400,  
         "doc_count": 10000  
       },  
       {  
         "key": 1500,  
         "doc_count": 10000  
       },  
       {  
         "key": 1600,  
         "doc_count": 10000  
       },  
       {  
         "key": 1700,  
         "doc_count": 10000  
       },  
       {  
         "key": 1800,  
         "doc_count": 10000  
       },  
       {  
         "key": 1900,  
         "doc_count": 10000  
       },  
       {  
         "key": 2000,  
         "doc_count": 10000  
       },  
       {  
         "key": 2100,  
         "doc_count": 10000  
       },  
       {  
         "key": 2200,  
         "doc_count": 10000  
       },  
       {  
         "key": 2300,  
         "doc_count": 10000  
       },  
       {  
         "key": 2400,  
         "doc_count": 10000  
       },  
       {  
         "key": 2500,  
         "doc_count": 10000  
       },  
       {  
         "key": 2600,  
         "doc_count": 10000  
       },  
       {  
         "key": 2700,  
         "doc_count": 10000  
       },  
       {  
         "key": 2800,  
         "doc_count": 10000  
       },  
       {  
         "key": 2900,  
         "doc_count": 10000  
       },  
       {  
         "key": 3000,  
         "doc_count": 10000  
       },  
       {  
         "key": 3100,  
         "doc_count": 10000  
       },  
       {  
         "key": 3200,  
         "doc_count": 10000  
       },  
       {  
         "key": 3300,  
         "doc_count": 10000  
       },  
       {  
         "key": 3400,  
         "doc_count": 10000  
       },  
       {  
         "key": 3500,  
         "doc_count": 10000  
       },  
       {  
         "key": 3600,  
         "doc_count": 10000  
       },  
       {  
         "key": 3700,  
         "doc_count": 10000  
       },  
       {  
         "key": 3800,  
         "doc_count": 10000  
       },  
       {  
         "key": 3900,  
         "doc_count": 10000  
       },  
       {  
         "key": 4000,  
         "doc_count": 10000  
       },  
       {  
         "key": 4100,  
         "doc_count": 10000  
       },  
       {  
         "key": 4200,  
         "doc_count": 10000  
       },  
       {  
         "key": 4300,  
         "doc_count": 10000  
       },  
       {  
         "key": 4400,  
         "doc_count": 10000  
       },  
       {  
         "key": 4500,  
         "doc_count": 10000  
       },  
       {  
         "key": 4600,  
         "doc_count": 10000  
       },  
       {  
         "key": 4700,  
         "doc_count": 10000  
       },  
       {  
         "key": 4800,  
         "doc_count": 10000  
       },  
       {  
         "key": 4900,  
         "doc_count": 10000  
       },  
       {  
         "key": 5000,  
         "doc_count": 10000  
       },  
       {  
         "key": 5100,  
         "doc_count": 10000  
       },  
       {  
         "key": 5200,  
         "doc_count": 10000  
       },  
       {  
         "key": 5300,  
         "doc_count": 10000  
       },  
       {  
         "key": 5400,  
         "doc_count": 10000  
       },  
       {  
         "key": 5500,  
         "doc_count": 10000  
       },  
       {  
         "key": 5600,  
         "doc_count": 10000  
       },  
       {  
         "key": 5700,  
         "doc_count": 10000  
       },  
       {  
         "key": 5800,  
         "doc_count": 10000  
       },  
       {  
         "key": 5900,  
         "doc_count": 10000  
       },  
       {  
         "key": 6000,  
         "doc_count": 10000  
       },  
       {  
         "key": 6100,  
         "doc_count": 10000  
       },  
       {  
         "key": 6200,  
         "doc_count": 10000  
       },  
       {  
         "key": 6300,  
         "doc_count": 10000  
       },  
       {  
         "key": 6400,  
         "doc_count": 10000  
       },  
       {  
         "key": 6500,  
         "doc_count": 10000  
       },  
       {  
         "key": 6600,  
         "doc_count": 10000  
       },  
       {  
         "key": 6700,  
         "doc_count": 10000  
       },  
       {  
         "key": 6800,  
         "doc_count": 10000  
       },  
       {  
         "key": 6900,  
         "doc_count": 10000  
       },  
       {  
         "key": 7000,  
         "doc_count": 10000  
       },  
       {  
         "key": 7100,  
         "doc_count": 10000  
       },  
       {  
         "key": 7200,  
         "doc_count": 10000  
       },  
       {  
         "key": 7300,  
         "doc_count": 10000  
       },  
       {  
         "key": 7400,  
         "doc_count": 10000  
       },  
       {  
         "key": 7500,  
         "doc_count": 10000  
       },  
       {  
         "key": 7600,  
         "doc_count": 10000  
       },  
       {  
         "key": 7700,  
         "doc_count": 10000  
       },  
       {  
         "key": 7800,  
         "doc_count": 10000  
       },  
       {  
         "key": 7900,  
         "doc_count": 10000  
       },  
       {  
         "key": 8000,  
         "doc_count": 10000  
       },  
       {  
         "key": 8100,  
         "doc_count": 10000  
       },  
       {  
         "key": 8200,  
         "doc_count": 10000  
       },  
       {  
         "key": 8300,  
         "doc_count": 10000  
       },  
       {  
         "key": 8400,  
         "doc_count": 10000  
       },  
       {  
         "key": 8500,  
         "doc_count": 10000  
       },  
       {  
         "key": 8600,  
         "doc_count": 10000  
       },  
       {  
         "key": 8700,  
         "doc_count": 10000  
       },  
       {  
         "key": 8800,  
         "doc_count": 10000  
       },  
       {  
         "key": 8900,  
         "doc_count": 10000  
       },  
       {  
         "key": 9000,  
         "doc_count": 10000  
       },  
       {  
         "key": 9100,  
         "doc_count": 10000  
       },  
       {  
         "key": 9200,  
         "doc_count": 10000  
       },  
       {  
         "key": 9300,  
         "doc_count": 10000  
       },  
       {  
         "key": 9400,  
         "doc_count": 10000  
       },  
       {  
         "key": 9500,  
         "doc_count": 10000  
       },  
       {  
         "key": 9600,  
         "doc_count": 10000  
       },  
       {  
         "key": 9700,  
         "doc_count": 10000  
       },  
       {  
         "key": 9800,  
         "doc_count": 10000  
       },  
       {  
         "key": 9900,  
         "doc_count": 10000  
       },  
       {  
         "key": 10000,  
         "doc_count": 10000  
       }  
     ]  
   }  
 }
```



The min_doc_count Parameter

- You can use `min_doc_count` to raise the minimum count to throw out smaller buckets

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "runtime_histogram": {
      "histogram": {
        "field": "runtime_ms",
        "interval": 100,
        "min_doc_count": 1000
      }
    }
  }
}
```

Only return buckets with at least 1,000 documents

Nesting a Metric in a Bucket

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

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "runtime_histogram": {
      "histogram": {
        "field": "runtime_ms",
        "interval": 100,
        "min_doc_count": 1000
      },
      "aggs": {
        "average_runtime": {
          "avg": {
            "field": "runtime_ms"
          }
        }
      }
    }
  }
}
```

"What is the average runtime for each bucket?"

```
"aggregations": {
  "runtime_histogram": {
    "buckets": [
      {
        "key": 0,
        "doc_count": 732687
        "average_runtime": {
          "value": 42.194726213
        }
      },
    ]
  }
},
```



The date_histogram Aggregation

- Puts documents in buckets based on a given interval

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

How many requests did we receive each month?



The date_histogram Aggregation

- The bucket names are a “key” based on the time interval
 - also represented as a “key_as_string” field in the response

```
"aggregations": {  
  "logs_by_month": {  
    "buckets": [  
      {  
        "key_as_string": "2017-03-01T00:00:00.000Z",  
        "key": 1488326400000,  
        "doc_count": 255  
      },  
      {  
        "key_as_string": "2017-04-01T00:00:00.000Z",  
        "key": 1491004800000,  
        "doc_count": 467961  
      },  
      {  
        "key_as_string": "2017-05-01T00:00:00.000Z",  
        "key": 1493596800000,  
        "doc_count": 393111  
      },  
      ...  
    ]  
  }  
}
```

March, 2017 bucket

“doc_count” is the default metric of buckets



Metrics on a date_histogram

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

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "logs_by_month": {
      "date_histogram": {
        "field": "@timestamp",
        "interval": "month"
      },
      "aggs": {
        "average_size": {
          "avg": {
            "field": "response_size"
          }
        }
      }
    }
  }
}
```

What was the average response size for each month?

```
"aggregations": {
  "logs_by_month": {
    "buckets": [
      {
        "key_as_string": "2017-03-01T00:00:00.000Z",
        "key": 1488326400000,
        "doc_count": 255,
        "average_size": {
          "value": 62199.87450980392
        }
      }
    ]
  }
}
```

The terms Aggregation

„03-Mar-2019“

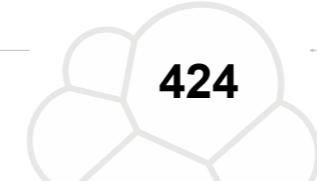
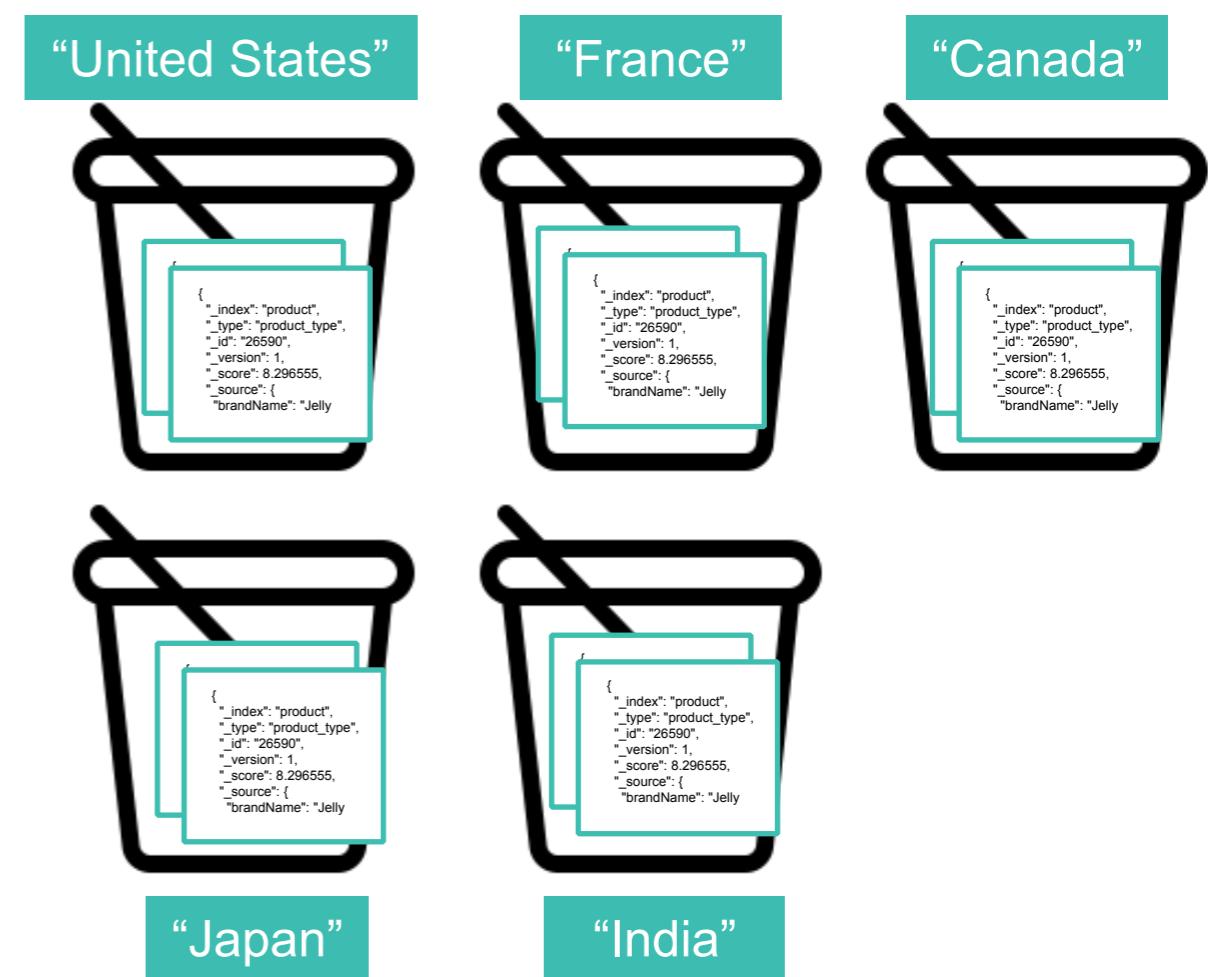
Terms Aggregation

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



03-Mar-2019 - 0

Start by using “**terms**” to put the logs into distinct buckets by “**country**”



Simple Example of terms

- 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
      }
    }
  }
}
```

“What are the **unique country names** that we receive requests from?”

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



The Output of our terms Aggregation:

- Notice each bucket has a “key” that represents the distinct value of “field”,
- and “doc_count” for the number of docs in the bucket

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

```
"aggregations": {  
  "country_name_terms": {  
    "doc_count_error_upper_bound": 16207,  
    "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 Agg is Handy!

- You can answer a lot of questions using **terms**
 - or even just to get a feel for what your data looks like

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "top_10_urls": {
      "terms": {
        "field": "originalUrl.keyword",
        "size": 10
      }
    }
  }
}
```

“Find the top 10 most-visited URLs.”

```
"buckets": [
  {
    "key": "/blog/feed",
    "doc_count": 58696
  },
  {
    "key": "/blog/found-java-clients-for-elasticsearch",
    "doc_count": 13368
  },
  {
    "key": "/blog/timelion-timeline",
    "doc_count": 13001
  },
  ...
]
```



Understanding the Output of terms

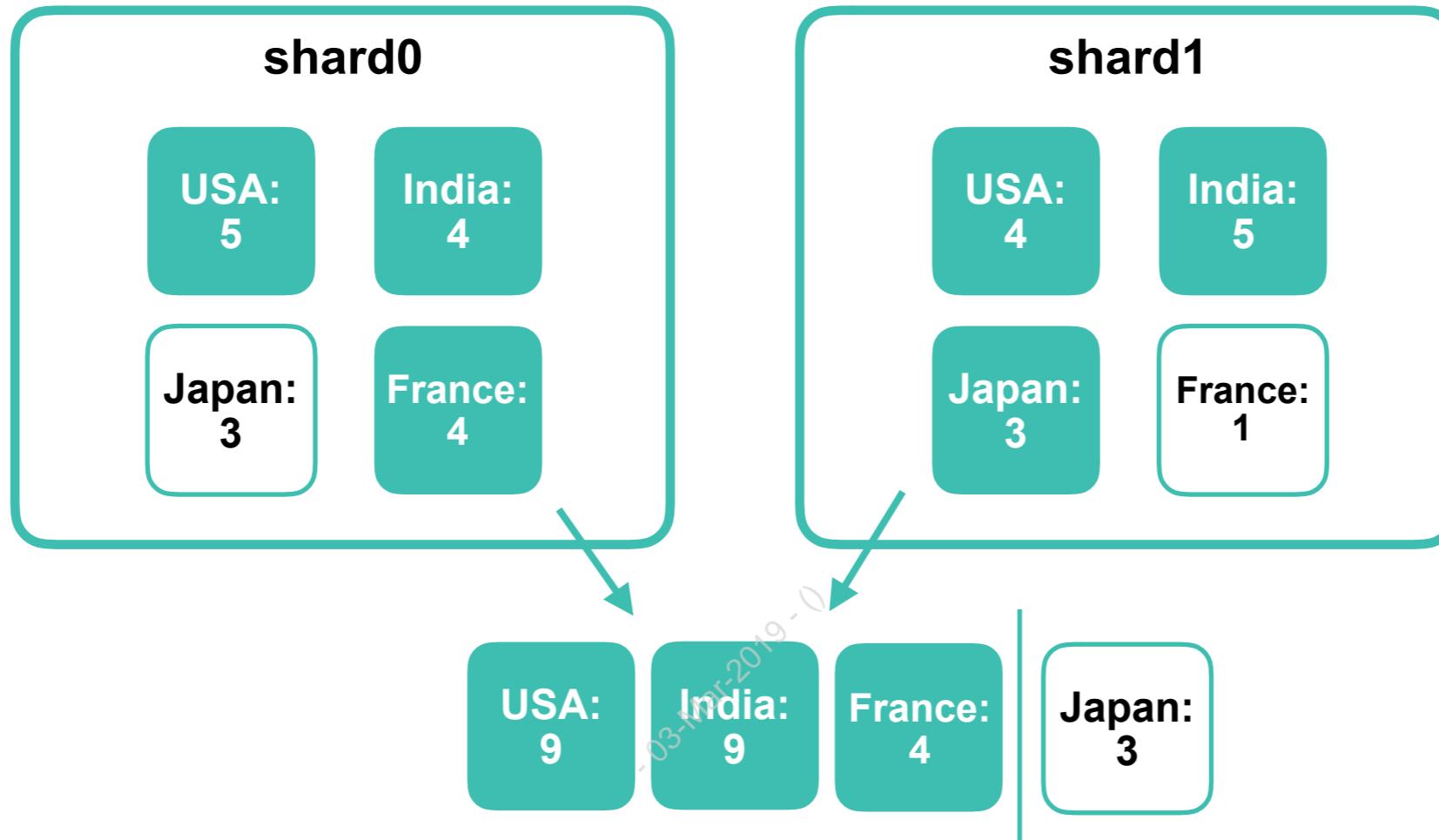
- There are two special values returned in a terms aggregation:
 - “**“doc_count_error_upper_bound”**”: maximum number of missing documents that could potentially have appeared in a bucket
 - “**“sum_other_doc_count”**”: number of documents that do not appear in any of the buckets

```
"aggregations": {  
    "top_10_urls": {  
        "doc_count_error_upper_bound": 5830,  
        "sum_other_doc_count": 1553455,  
        "buckets": [  
            {  
                "key": "/blog/feed",  
                "doc_count": 58696  
            },  
            {  
                "key": "/blog/found-java-clients-for-elasticsearch",  
                "doc_count": 13368  
            },  
            {  
                "key": "/blog/elastic-mapreduce-is-now-amazon-mr",  
                "doc_count": 13368  
            },  
            {  
                "key": "/blog/announcing-the-new-elasticsearch-node",  
                "doc_count": 13368  
            }  
        ]  
    }  
}
```

The value 58,696 may be off by as much as 5,830



Why are terms aggs not always precise?



If **shard_size=3**, the top 3 terms from each shard are determined, which can lead to inaccuracy
(Japan is 6 and should be in the top 3)

The shard_size Parameter

- **shard_size** tells Elasticsearch to gather more top terms from each shard to improve accuracy
 - `shard_size = (size x 1.5) + 10` (by default)

~03-Mar-2019~0



Increasing the shard_size Parameter

- Increasing **shard_size** improves accuracy at a cost of performance

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "top_10_urls": {
      "terms": {
        "field": "originalUrl.keyword",
        "size": 10,
        "shard_size": 500
      }
    }
  }
}
```

```
"aggregations": {
  "top_10_urls": {
    "doc_count_error_upper_bound": 492,
    "sum_other_doc_count": 1553455,
    "buckets": [
      {
        "key": "/blog/feed",
        "doc_count": 58696
      },
      ...
    ]
  }
}
```

That lowered the error upper bound from 5830 to 492

Enabling show_term_doc_count_error

- Shows you the upper-bound error *for each bucket*:

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "top_10_urls": {
      "terms": {
        "field": "originalUrl.keyword",
        "size": 10,
        "show_term_doc_count_error": true
      }
    }
  }
}
```

Even though the error upper bound is 5830...

```
"aggregations": {
  "top_10_urls": {
    "doc_count_error_upper_bound": 5830,
    "sum_other_doc_count": 1553455,
    "buckets": [
      {
        "key": "/blog/feed",
        "doc_count": 58696,
        "doc_count_error_upper_bound": 0
      },
    ]
  }
},
```

...this bucket is precise



Let's answer *more*
aggregation questions...

-03-Mar-2019-

Nesting Buckets

- What question are we answering here with this nested **terms** agg inside a **date_histogram** agg?

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

The “**terms**” agg is nested inside the “**date_histogram**”



Nesting Buckets

- The log events are bucketed *by month*,
 - then within each month the events are bucketed *by country*:

```
"aggregations": {  
  "logs_by_month": {  
    "buckets": [  
      {  
        "key_as_string": "2017-03-01T00:00:00.000Z",  
        "key": 1488326400000,  
        "doc_count": 255,  
        "country_name": {  
          "doc_count_error_upper_bound": 0,  
          "sum_other_doc_count": 40,  
          "buckets": [  
            {  
              "key": "United States",  
              "doc_count": 163  
            },  
            {  
              "key": "Canada",  
              "doc_count": 13  
            },  
            {  
              "key": "France",  
              "doc_count": 7  
            },  
            {  
              "key": "Other",  
              "doc_count": 11  
            }  
          ]  
        }  
      }  
    ]  
  }  
}
```



Bucket Sorting

- You can specify the sorting of **terms**, **histogram**, and **date_histogram** using “order”:
 - **_count** sorts by their **doc_count** (default in **terms**)
 - **_key** sorts alphabetically (default in **histogram** and **date_histogram**)
- You can also sort by a metric value in a nested aggregation

~03-Mar-2019 - 0

Sort by _key Example

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

The default for `date_histogram` is
`_key` ascending



Sorting by a Metric

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "logs_by_month": {
      "date_histogram": {
        "field": "@timestamp",
        "interval": "month",
        "order": {
          "average_runtime": "asc"
        }
      },
      "aggs": {
        "average_runtime": {
          "avg": {
            "field": "runtime_ms"
          }
        }
      }
    }
  }
}
```

“Find the month with the shortest average runtime.”

This output will be sorted by the results of the nested metric

Chapter Review

·03-Mar-2019·0

Summary

- You have two new tools in your Elasticsearch toolbox:
 - Metrics Aggregations like min, max, avg, stats
 - Bucket Aggregations like date_histogram and terms
- **Metrics** compute numeric values based on your dataset
- A **bucket** is a collection of documents that meet a criterion
- An **aggregation** can be thought of as a unit-of-work that builds analytic information over a set of documents
- Aggregations can be nested within other aggregations
- The **terms** aggregation dynamically creates buckets for every unique term it encounters of a specified field



Quiz

1. **Query or Aggregation:** “What is the box office revenue of all movies directed by Steven Spielberg?”
2. **Query or Aggregation:** “What are the nearby restaurants that serve pizza?”
3. What aggregation would you use to put logging events into buckets by log level (“error”, “warn”, “info”, etc.)?
4. How can you increase the accuracy of a **terms** aggregation?
5. What aggregation(s) would you use to answer the question “How many unique visitors came to our website today?”



Lab 11

Aggregating Data

.03-Mar-2019-Q



Chapter 12

Best Practices

,03-Mar-2019, 0

- 1 Elastic Stack Overview
- 2 Getting Started with Elasticsearch
- 3 CRUD Operations
- 4 Querying Data
- 5 Text Analysis and Mappings
- 6 Custom Mappings
- 7 Node Types
- 8 Understanding Shards
- 9 Troubleshooting Elasticsearch
- 10 Improving Search Results
- 11 Aggregating Data
- 12 Best Practices



Topics covered:

- Index Aliases
- Index Templates
- Scroll Searches
- Cluster Backup

~03-Mar-2019~0



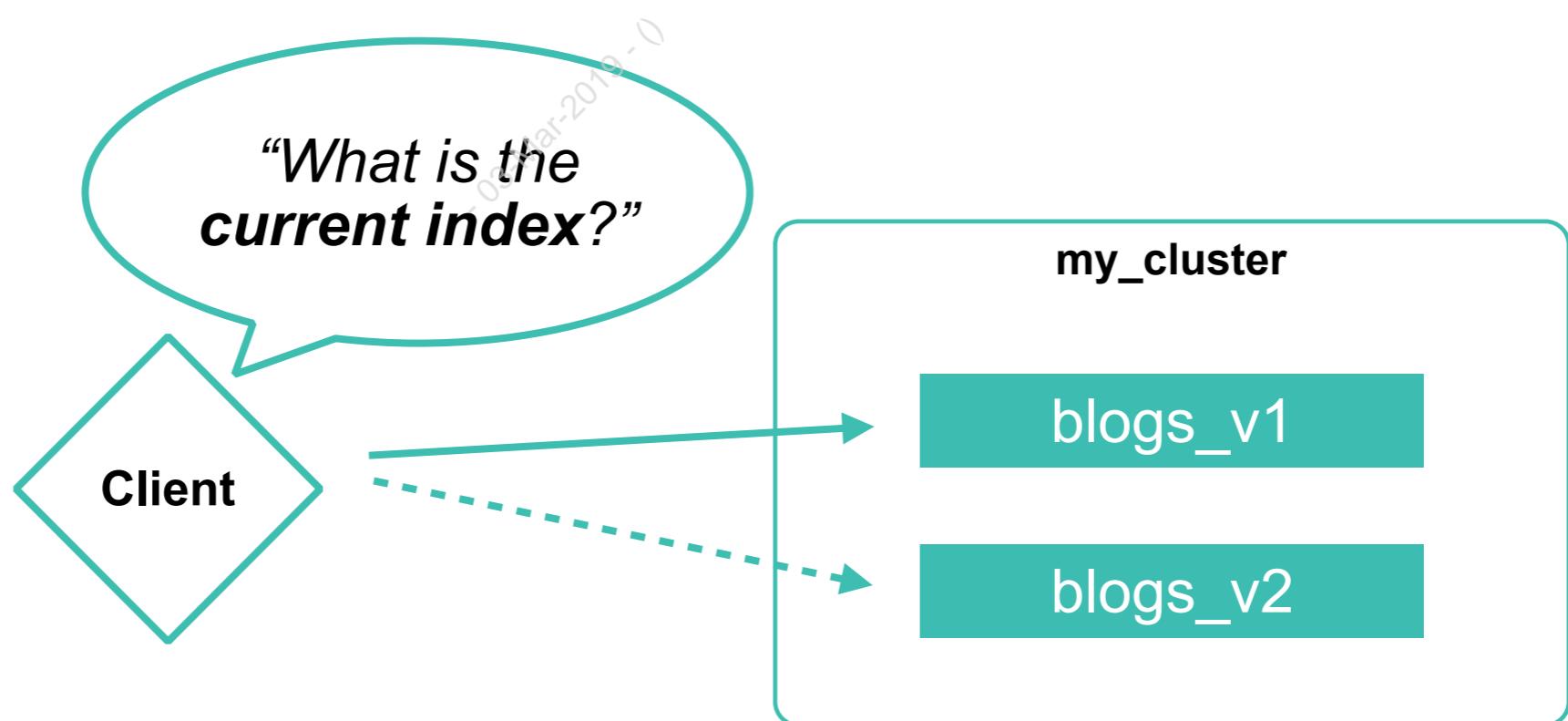
Index Aliases

· 03-Mar-2019 ·



Use Case for Aliases

- Our search page application communicates with the `blogs_v1` index in Elasticsearch:
 - If we want to update the mapping, like we did in Chapter 4 we need to reindex the data into a new index.
 - Then we need to update the application and redeploy
- An ***alias*** would simplify this greatly...



Index Aliases

- You can define an *alias* for an index
 - using the `_aliases` endpoint:

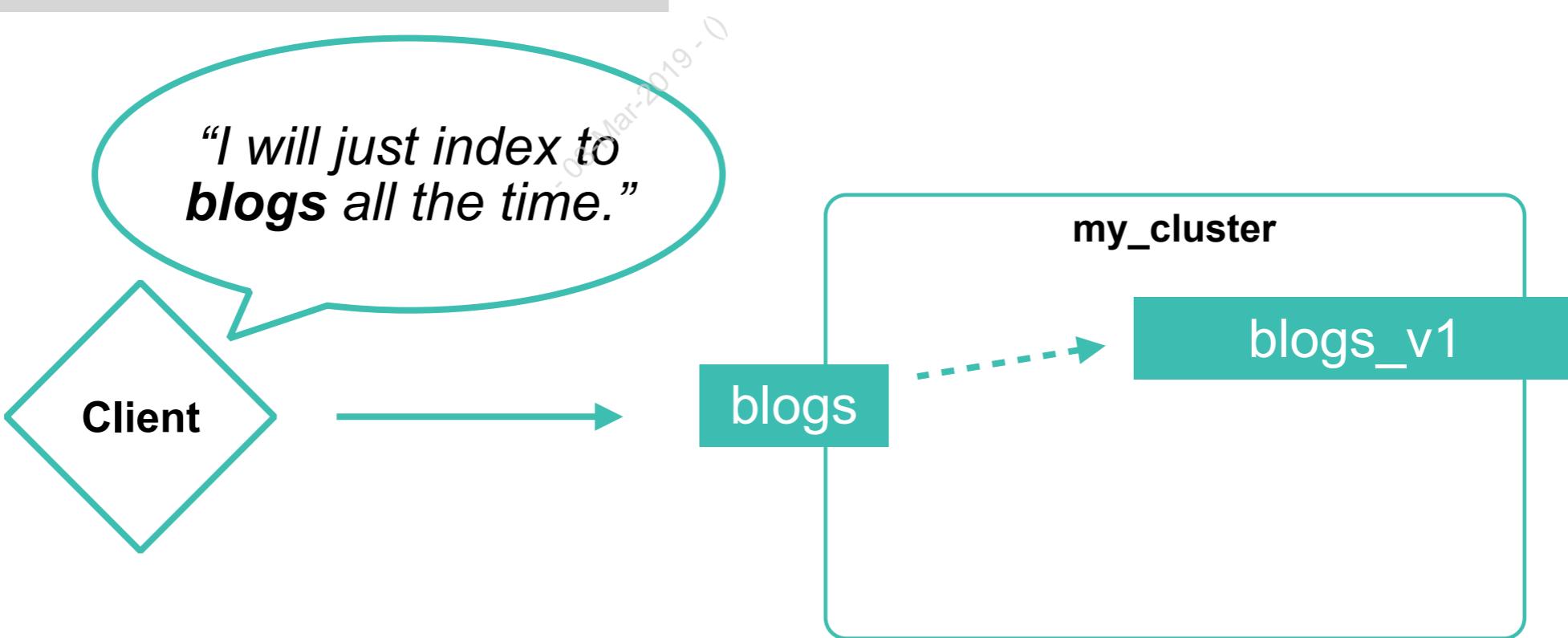
```
POST _aliases
{
  "actions": [
    {
      "add": {
        "index": "INDEX_NAME",
        "alias": "ALIAS_NAME"
      }
    },
    {
      "remove": {
        "index": "INDEX_NAME",
        "alias": "ALIAS_NAME"
      }
    }
  ]
}
```



Defining an Alias

```
POST _aliases
{
  "actions": [
    {
      "add": {
        "index": "blogs_v1",
        "alias": "blogs"
      }
    }
  ]
}
```

“blogs” is an alias for
“blogs_v1”

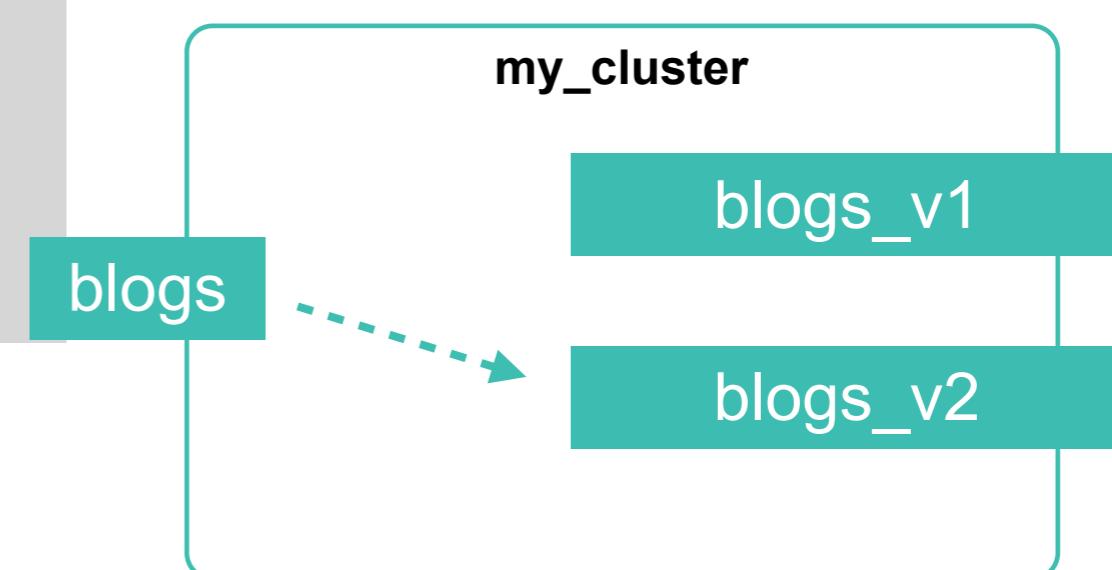


Changing an Alias

- When the log index rolls over to a new month, update the alias to point to the new, most-recent index:

```
POST _aliases
{
  "actions": [
    {
      "add": {
        "index": "blogs_v2",
        "alias": "blogs"
      }
    },
    {
      "remove": {
        "index": "blogs_v1",
        "alias": "blogs"
      }
    }
  ]
}
```

“blogs” is now an alias for “blogs_v2”



Filtered Aliases

- An alias can include a **filter**
 - useful for creating different views of the same index
 - be careful, it does not provide any security

```
POST _aliases
{
  "actions": [
    {
      "add": {
        "index": "blogs_v1",
        "alias": "blogs_engineering",
        "filter": {
          "match": {
            "category": "Engineering"
          }
        }
      }
    }
  ]
}
```

Any query on the **blogs_engineering** alias will only include blogs that are in the “Engineering” category



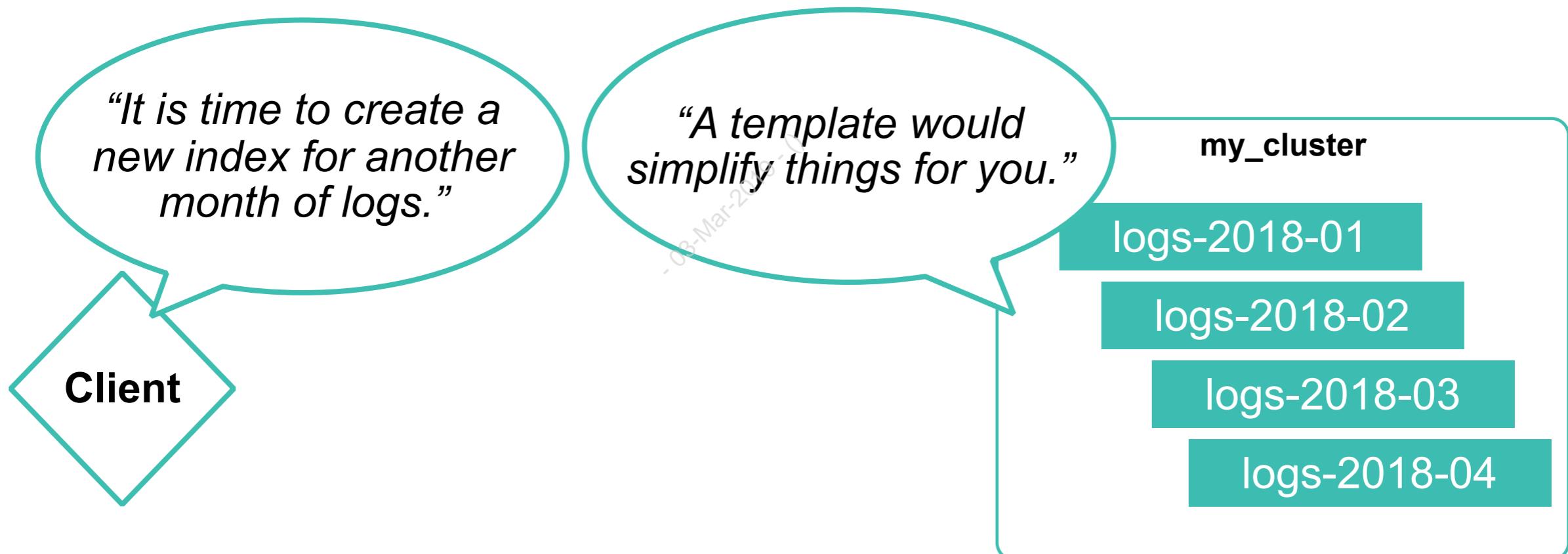
Index Templates

~03-Mar-2019



Index Templates

- ***Index templates*** allow you to define mappings and settings that will automatically be applied to newly-created indices
 - useful when you need to create multiple indices ***with the same settings and mappings***, like our monthly log indices:



Defining a Template

- Use the `_template` endpoint to add, view and delete templates

```
PUT _template/logs_template
{
  "index_patterns": "logs-*",
  "order": 1,
  "settings": {
    "number_of_shards": 4,
    "number_of_replicas": 1
  },
  "mappings": {
    "_doc": {
      "properties": {
        "@timestamp": {
          "type": "date"
        }
      }
    }
  }
}
```

name of the template

Apply this template to
any new index that
starts with “logs-”

Define the **settings** and
mappings you want for
new “logs-” indices



Test the Template

- You can test our template by simply creating an index whose name starts with “logs-”

```
PUT logs-2017-12
```

```
GET logs-2017-12
```

```
{  
  "logs-2017-12": {  
    "aliases": {},  
    "mappings": {  
      "_doc": {  
        "properties": {  
          "@timestamp": {  
            "type": "date"  
          }  
        }  
      }  
    },  
    "settings": {  
      "index": {  
        "creation_date": "1518672588315",  
        "number_of_shards": "4",  
        "number_of_replicas": "1",  
        "uuid": "2HFecwdXRHiIhwOQodI-BA",  
        "version": {  
          "created": "6010199"  
        },  
        "provided_name": "logs-2017-12"  
      }  
    }  
  }  
}
```

Multiple Templates

- You can have multiple index templates
 - They get merged and applied to the created index
 - You can provide an “**order**” value to control the merging process
- When a new index is created:
 1. The default settings and mappings are applied
 2. Settings and mappings from the lowest “**order**” template are applied
 3. Settings and mappings from the higher “**order**” templates are applied, overriding previous settings along the way
 4. The setting and mappings from the **PUT** command of the index are applied last and override all previous templates

Let's define a second template:

```
PUT _template/logs_2018_template
{
  "index_patterns": "logs-2018*",
  "order": 5, ←
  "settings": {
    "number_of_shards": 6,
    "number_of_replicas": 2
  }
}
```

This template has a higher “order” than “logs_template”



Now define a new index

- that matches both templates:

PUT logs-2018-06

GET logs-2018-06

```
{  
  "logs-2018-06": {  
    "aliases": {},  
    "mappings": {  
      "_doc": {  
        "properties": {  
          "@timestamp": {  
            "type": "date"  
          }  
        }  
      }  
    },  
    "settings": {  
      "index": {  
        "creation_date": "1518672972728",  
        "number_of_shards": "6",  
        "number_of_replicas": "2",  
        "uuid": "3Nzr_5odT16KqftZ2BsPvA",  
        "version": {  
          "created": "6010199"  
        },  
        "provided_name": "logs-2018-06"  
      }  
    }  
  }  
}
```

From “logs_template”

From
“logs_2018_template”



Scroll Searches

·03-Mar-2019·A

Use Case for Scroll Searches

- **Large result sets** can be expensive to both the cluster and the client
 - Suppose a client application wants to retrieve the access log events for a particular month, which could be millions of documents



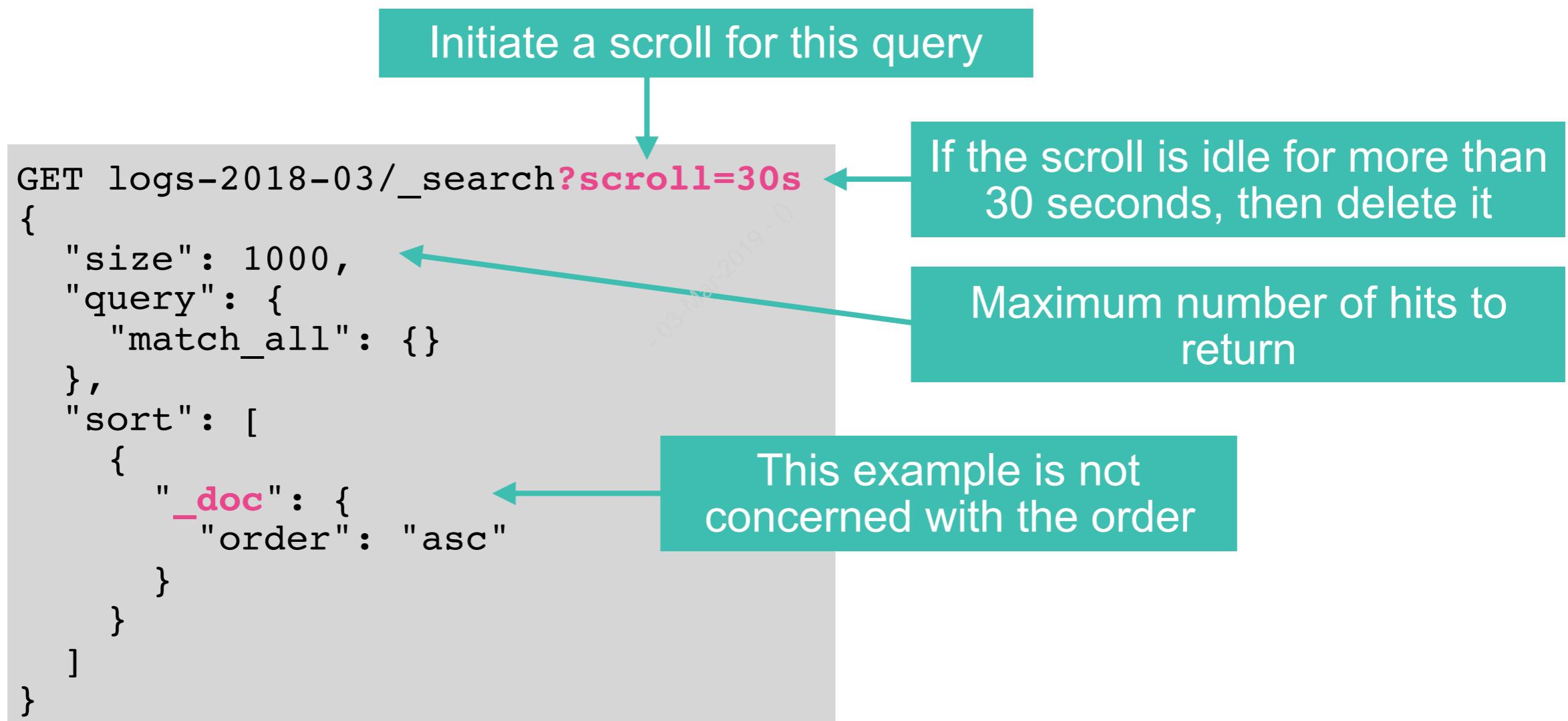
Scroll Searches

- The **Scroll API** allows you to take a snapshot of a large number of results from a single search request
 - Notice the word “**snapshot**”. Any changes made to documents after the scroll is initiated will not show up in your results
- Scroll searches involve the following steps...

~03-Mar-2019~0

1. Initiate the scroll

- To initiate a scroll search, add the **scroll** parameter to your search query
 - specifying how long Elasticsearch should keep the search context alive



2. Get the current _scroll_id

- The response will contain:
 - the first page of results,
 - and a `_scroll_id`
- Hang on to the `_scroll_id`! You need it for the next request

```
{  
  "_scroll_id": "DnF1ZXJ5VGh1bkZ1dGNoBQAAAAAAWOFn1wQVR3N3pxUjdLMn-  
  JLcUZpSDVkWWcAAAAAAAFjAAAAPFn1wQVR3N3pxUjdLMnJLcUZpSDVkWWcAAAAAAAFkRZ5V  
  kFUdzd6cVI3SzJyS3FGaUg1ZFln",  
  "took": 0,  
  "timed_out": false,  
  "_shards": {  
    "total": 5,  
    "successful": 5,  
    "skipped": 0,  
    "failed": 0  
  },  
  "hits": {  
    "total": 584721,  
    "max_score": null,  
    "hits": [...]  
  }  
}
```



3. Retrieve the next chunk of documents

- When you are ready for more documents, you just need to send a **GET** request, along with the ***prior_scroll_id***

Do not specify an index, type or query here -
just invoke **scroll**

```
GET _search/scroll
{
  "scroll": "30s",
  "scroll_id":
  "DnF1ZXJ5VGh1bkZldGNoBQAAAAAAWOFn1WQVR3N3pxUjdLMn-
  JLcUZpSDVkWWcAAAAAAAFAjAAAAWPFn1WQVR3N3pxUjdLMnJLcUZpSDVkWWcAAAAAA
  AAFkRZ5VkJFdzd6cVI3SzJyS3FGaUg1ZFln"
}
```

Always use the ***prior_scroll_id***

If the scroll is still alive, the next
set of hits is returned

4. Repeat

- Keep retrieving groups of documents until you get them all:

```
GET _search/scroll
{
  "scroll": "30s",
  "scroll_id": "DnF1ZXJ5VGh1bkZldGNoBQAAAAAAAAWOFn1WQVR3N3pxUjdLMn-
JLcUZpSDVkWWcAAAAAAAFAjAAAAWFFn1WQVR3N3pxUjdLMnJLcUZpSDVkWWcAAAAAA
AAFKRZ5VkJFUdzd6cVI3SzJyS3FGaUg1ZFln"
}
```

Always use the prior _scroll_id



5. Clearing a scroll

- If you are done using a scroll and want to manually close it, use a **DELETE** request:

```
DELETE _search/scroll/DnF1ZXJ5VGh1bkZldGNoBQAAAAAAAOFnlWQVR3N3pxUjdLMn-JLcUZpSDVkWWcAAAAAAAFjAAAAWPFnlWQVR3N3pxUjdLMnJLcUZpSDVkWWcAAAAAAAFkRZ5VkJUdzb6cVI3SzJyS3FGaUg1ZFln
```

03-Mar-2019 - 0



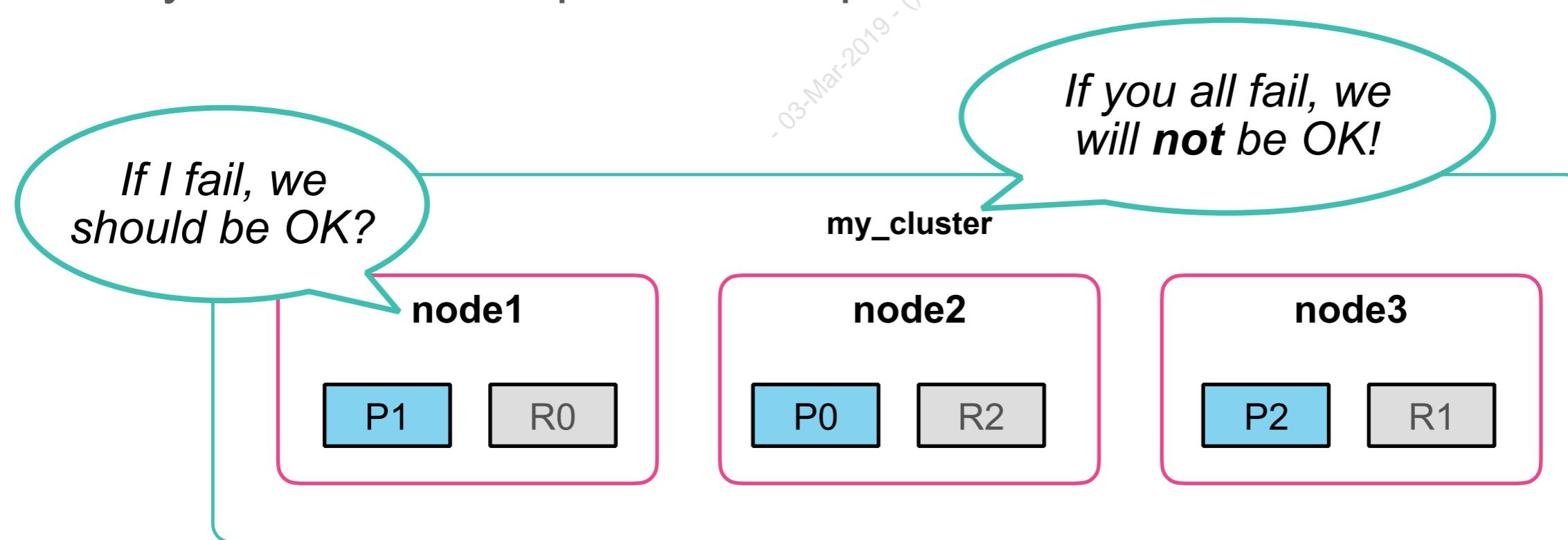
Cluster Backup

-03-Mar-2019-



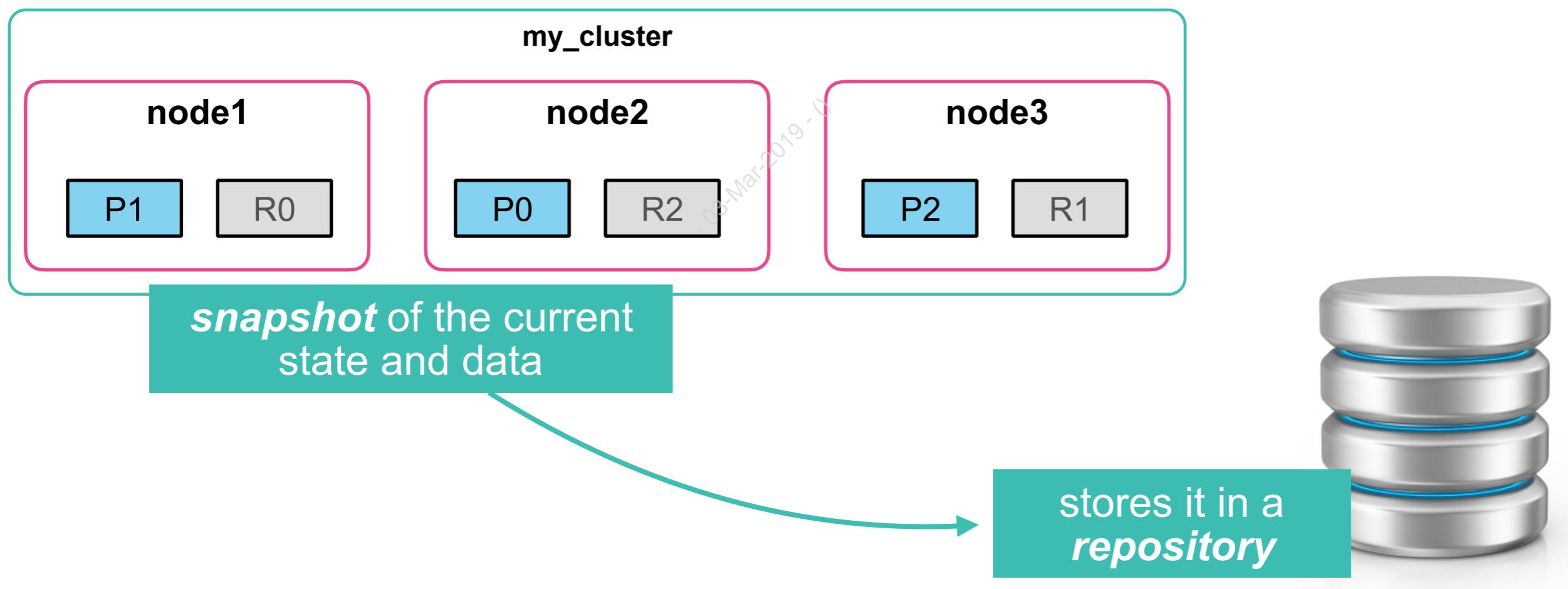
Cluster Backup

- We have talked a lot about *replica shards*
 - they provide replication of your documents
 - that is not the same as a backup!
- Replicas do not provide protection against catastrophic failure
 - you need a complete backup for those situations



Snapshot and Restore

- The ***Snapshot and Restore API*** provides a cluster backup mechanism
 - takes the current state and data in your cluster and saves it to a ***repository***



Repositories

- The backup process starts with the creation of a **repository**
 - You can configure multiple repositories per cluster
- Different types are supported:

Repository	Configuration type	
Shared file system	"type" : "fs"	
Read-only URL	"type" : "url"	
S3	"type" : "s3"	
HDFS	"type" : "hdfs"	
Azure	"type" : "azure"	
Google Cloud Storage	"type" : "gcs"	S3, HDFS, Azure and GCS require the appropriate plugin to be installed



Registering a Repository

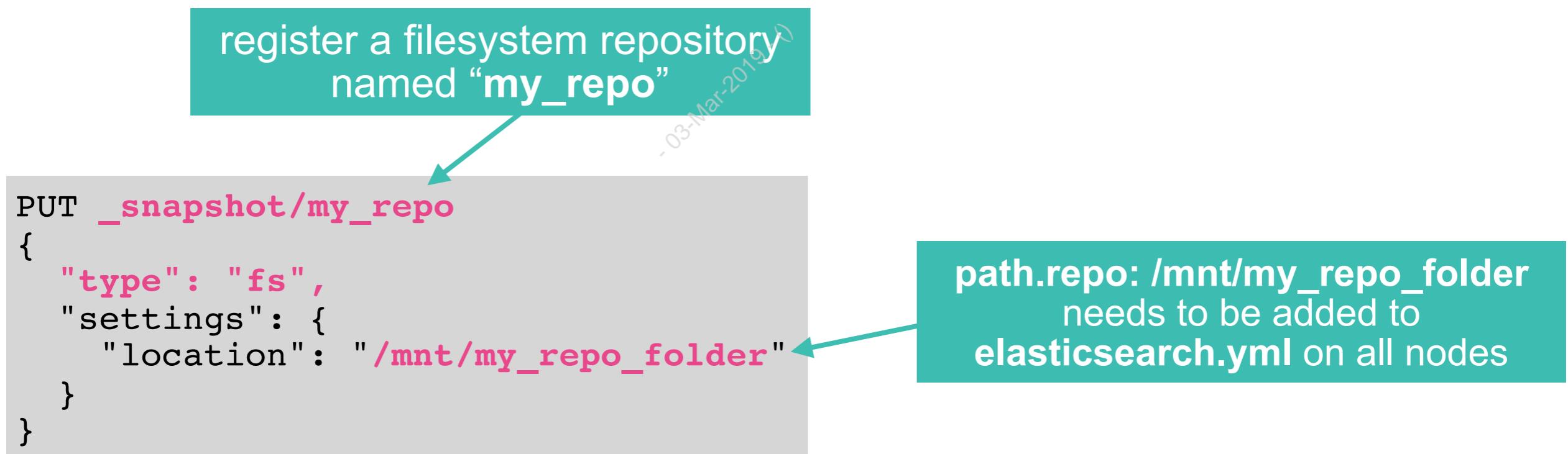
- Before a snapshot can be performed, the repository needs to be *registered*
 - using the `_snapshot` endpoint
 - the folder must be accessible from all nodes in the cluster
 - `path.repo` must be configured on all nodes for an “fs” repo

register a filesystem repository named “`my_repo`”

PUT `_snapshot/my_repo`

```
{  
  "type": "fs",  
  "settings": {  
    "location": "/mnt/my_repo_folder"  
  }  
}
```

`path.repo: /mnt/my_repo_folder`
needs to be added to
`elasticsearch.yml` on all nodes



File System Repository Settings

- Other settings for an “fs” repo include:

```
PUT _snapshot/my_repo
{
  "type": "fs",
  "settings": {
    "location": "/mnt/my_repo_folder",
    "compress": true, ←
    "max_restore_bytes_per_sec": "40mb",
    "max_snapshot_bytes_per_sec": "40mb"
  }
}
```

compress metadata files

throttles per-node snapshot and restore rates

S3 Repository Settings

- Your repository can be a bucket in S3
 - The **repository-s3** plugin needs to be installed on your cluster

```
$ bin/elasticsearch-plugin install repository-s3
```

- Set the “**type**” of repo to “**s3**”:

```
PUT _snapshot/my_s3_repo
{
  "type": "s3",
  "settings": {
    "bucket": "my_s3_bucket_name"
  }
}
```

The name of the bucket
in S3

- The docs show how to configure IAM permissions:
 - <https://www.elastic.co/guide/en/elasticsearch/plugins/current/repository-s3-repository.html>

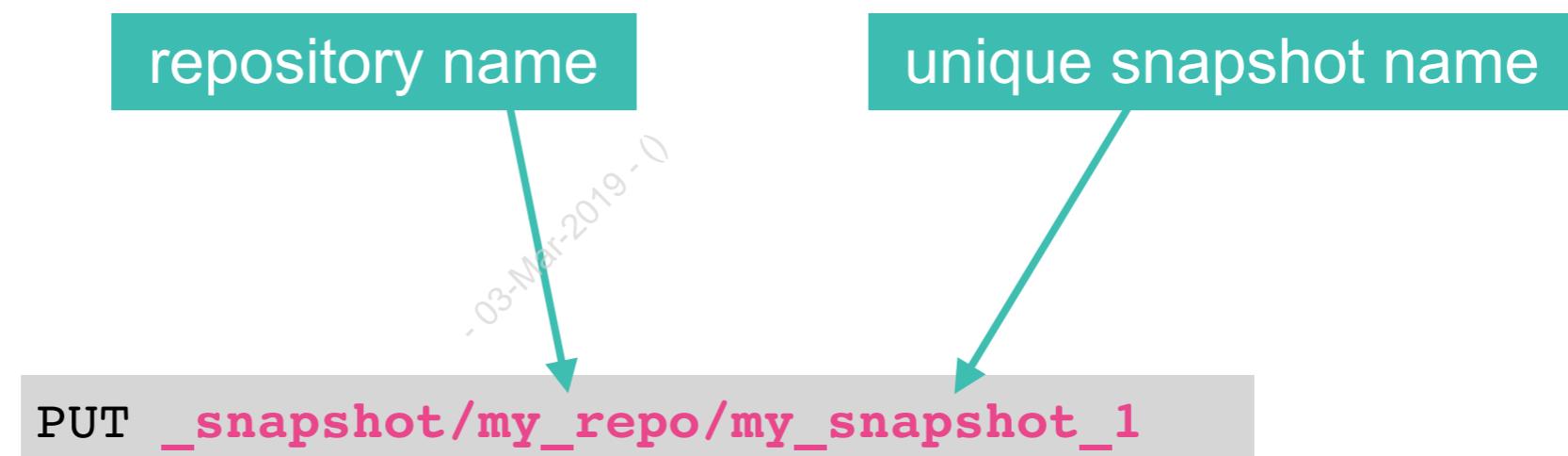


Taking a Snapshot

· 03-Mar-2019

Snapshotting All Indices

- Once the repository is configured, you can take a snapshot using the `_snapshot` endpoint:
 - snapshots are a “point-in-time” copy of the data
 - only changes since the last snapshot are copied



No specific indices were listed, so this snapshot includes ***all open indices***

Specifying Indices

- Add the “**indices**” parameter to pick specific indices for the snapshot:

```
PUT _snapshot/my_repo/my_logs_snapshot_1
{
  "indices": "logs-*",
  "ignoreUnavailable": true,
  "includeGlobalState": true
}
```

snapshot of all open
indices that start with
“**logs-**”

This snapshot also ignores any
unavailable indices, and includes
the global state metadata

Monitoring Progress

- Use the `_status` endpoint to view the progress of a snapshot:

```
GET _snapshot/my_repo/my_snapshot_2/_status
```

- You can also wait for the snapshot to complete (but it may take a while):

```
PUT _snapshot/my_repo/my_logs_snapshot_2?wait_for_completion=true
{
  ...
}
```

Managing Snapshots

- Get information about *all snapshots* in a repo:

```
GET _snapshot/my_repo/_all
```

- Get information about a *specific snapshot*:

```
GET _snapshot/my_repo/my_snapshot_1
```

- *Delete* a snapshot:

```
DELETE _snapshot/my_repo/my_snapshot_1
```



Restoring from a Snapshot

03-Mar-2019 - 0

Restoring from a Snapshot

- Use the `_restore` endpoint on the ID of the snapshot to restore all indices from a snapshot:

```
POST _snapshot/my_repo/my_snapshot_2/_restore
```

- You can also restore specific indices:

```
POST _snapshot/my_repo/my_snapshot_2/_restore
{
  "indices": "logs-*",
  "ignore_unavailable": true,
  "include_global_state": false
}
```



Renaming Indices

- You can restore old indices into new ones without replacing existing data
 - specify a “`rename_pattern`” and a “`rename_replacement`”:

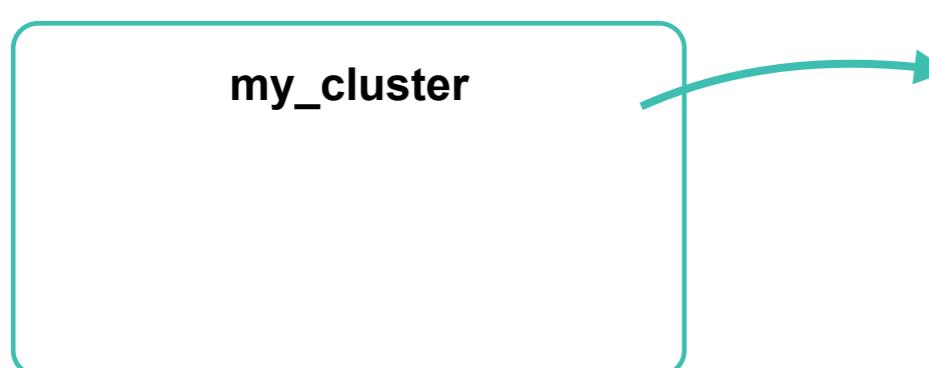
```
POST _snapshot/my_repo/my_snapshot_2/_restore
{
  "indices": "logs-*",
  "ignore_unavailable": true,
  "include_global_state": false,
  "rename_pattern": "logs-(.+)",
  "rename_replacement": "restored-logs-$1"
}
```

If an index starts with “`logs-*`”,
create a new index as
“`restored-logs-*`”

Restore to Another Cluster

- You can restore a snapshot made from one cluster into another cluster
 - you need to *register the repository* in the new cluster first

```
POST _snapshot/my_repo/snap1
```



```
POST _snapshot/my_repo/snap1/_restore
```

Chapter Review

·03-Mar-2019·0

Summary

- The ***Index Aliases API*** allows you to define an ***alias*** for an index
- ***Index templates*** allow you to define mappings and settings that will automatically be applied to newly-created indices
- The ***Scroll API*** allows you to take a snapshot of a large number of results from a single search request
- The ***Snapshot and Restore API*** provides a cluster backup mechanism

03-Mar-2019 10:10



Quiz

- True or False:** It is considered a best practice to use aliases for all of your production indexes.
- True or False:** A template of order 1 overrides the settings of a template of order 5.
- True or False:** Configuring all indices to have 2 or more replicas provides a reliable backup mechanism for a cluster.
- True or False:** You can take a snapshot of your entire cluster in a single REST request.



Lab 12

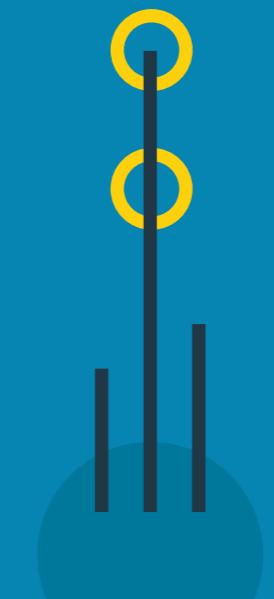
Best Practices

03-Mar-2019



Conclusions

,03-Mar-2019, 0



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>

~03-Mar-2019 - 0

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



03-Mar-2019-0

SPECIALIZATIONS



Engineer II

1 Elasticsearch Internals

2 Field Modeling

3 Fixing Data

4 Advanced Search & Aggregations

5 Cluster Management

6 Capacity Planning

7 Document Modeling

8 Monitoring and Alerting

9 From Dev to Production



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

03-Mar-2019-0

Thank you!

Please complete the online survey

-03-Mar-2019-

Quiz Answers

-03-Mar-2019-0

Chapter 1 Quiz Answers

1. True
2. scalable search, and to make it easy to use from any programming language
3. somewhat static data, and time-series data
4. Logstash would work well for that
5. Metricbeat would be a great option
6. Packetbeat is great for ingesting network traffic details



Chapter 2 Quiz Answers

1. **elasticsearch.yml, jvm.options, and log4j2.properties**
2. Using the transport protocol, on port 9300 by default
3. The new node pings the servers listed in the **discovery.zen.ping.unicast.hosts** property
4. The node will bind to the IP address of the local machine, allowing it to be reachable by the outside world
5. False
6. Proxy, Firewall, Elastic Security



Chapter 3 Quiz Answers

1. An index is a collection of shards that are distributed across nodes in the cluster
2. By default, this is true, but this behavior can be turned off (recommended for production clusters)
3. 10
4. The existing document is deleted, and the new document is indexed
5. True

~03-Mar-2019~0



Chapter 4 Quiz Answers

1. Multiple match terms use “and” or “or”, while multiple terms in match_phrase are “and” and position matters
2. Slop
3. You could use either the “and” operator, or specify a “minimum_should_match” value
4. You could use a bool query with a must match query for “scripting” in the content field and a filter for the “Engineering” category
5. False. A filter clause has no effect on a document’s score

Chapter 5 Quiz Answers

1. Text analysis (or just analysis)2
2. “value” would be a “long”
3. “value” would be mapped as both “text” and “keyword”
4. False
5. Character filter, tokenizer, token filter

~03-Mar-2019-0



Chapter 6 Quiz Answers

1. True
2. True
3. False

~03-Mar-2019~ Ø



Chapter 7 Quiz Answers

1. 2
2. You should not have only 2 master-eligible nodes! You will not be able to achieve high-availability without the possibility of a split-brain scenario.
3. Set node.master to true, and node.ingest and node.data to false

~03-Mar-2019 - 0



Chapter 8 Quiz Answers

1. 12. 4 primary shards, and 8 replicasQuery
2. False
3. False. The `_id` is used by default to compute the shard number
4. True

~03-Mar-2019~0

Chapter 9 Quiz Answers

1. False
2. You can have shards that failed during the query, yet the response code will show the query as a success
3. Unassigned, Initializing, Started, Relocating
4. False. Yellow has a potential for lost data because not all replicas exist, but no documents are missing
5. Must be red

~03-Mar-2019~0

Chapter 10 Quiz Answers

1. The score of each hit is the “best” score from all the scores calculated
2. A fuzziness of 1 would cause the two terms to match
3. “from” and “size”
4. The “category” field is “text”, so “User Stories” is analyzed to “user”, “stories”. The “category.keyword” field is “keyword”, so the search term is not analyzed and must be an exact match.

Chapter 11 Quiz Answers

1. actually both query and aggregation
2. Query
3. terms
4. increasing “shard_size”
5. cardinality

~03-Mar-2019~ Ø

Chapter 12 Quiz Answers

1. True - it's a good idea.
2. False. Higher order templates override lower order template settings
3. False! Replicas provide no backup mechanism of any kind
4. True

~03-Mar-2019~0

Elasticsearch Engineer I

Course: Elasticsearch Engineer I

Version 6.5.1

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

~03-Mar-2019~0