



Elasticsearch Engineer

An Elastic training course

elastic.co/training

Welcome to Elastic virtual training



- 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:
 - use a supported web browser: Chrome or Firefox
 - open this page in an "incognito" or "private" window
 - disable any ad blockers, script blockers, proxy or VPN
- In case of problems, try the following steps in order:
 - refresh this web page
 - try another web browser
 - as a last resort, restarting your computer sometimes helps

Welcome to Elastic training

- Visit **learn.elastic.co** and log in
 - follow instructions from registration email to get access
- Go to "**My Enrollments**" and click on today's training
- Download the course files from the "**Content**" tab
- The .zip file contains:
 - a PDF file of the slides used in the course
 - a copy of the lab instructions
 - datasets and scripts used in the labs
- Click on "**Virtual Link**" to access the Lab Environment

About Elastic training

- Environment
- Introductions
- Code of conduct
 - www.elastic.co/community/codeofconduct

Elasticsearch Engineer: Agenda

- **Module 1: Getting started**
- Module 2: Data modeling
- Module 3: You know, for search
- Module 4: Data processing
- Module 5: Aggregations
- Module 6: The one about shards
- Module 7: Data management
- Module 8: Cluster management

Getting Started

Module 1

Topics

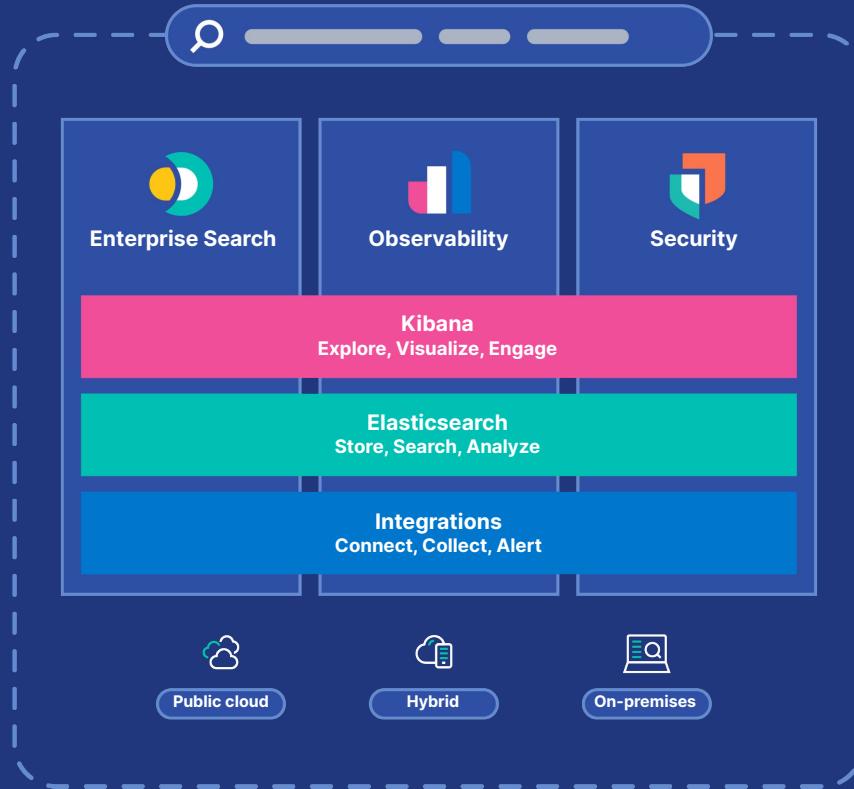
- Introduction to the Elastic Stack
- Data In
- Information Out

Introduction to the Elastic Stack

Module 1 Lesson 1

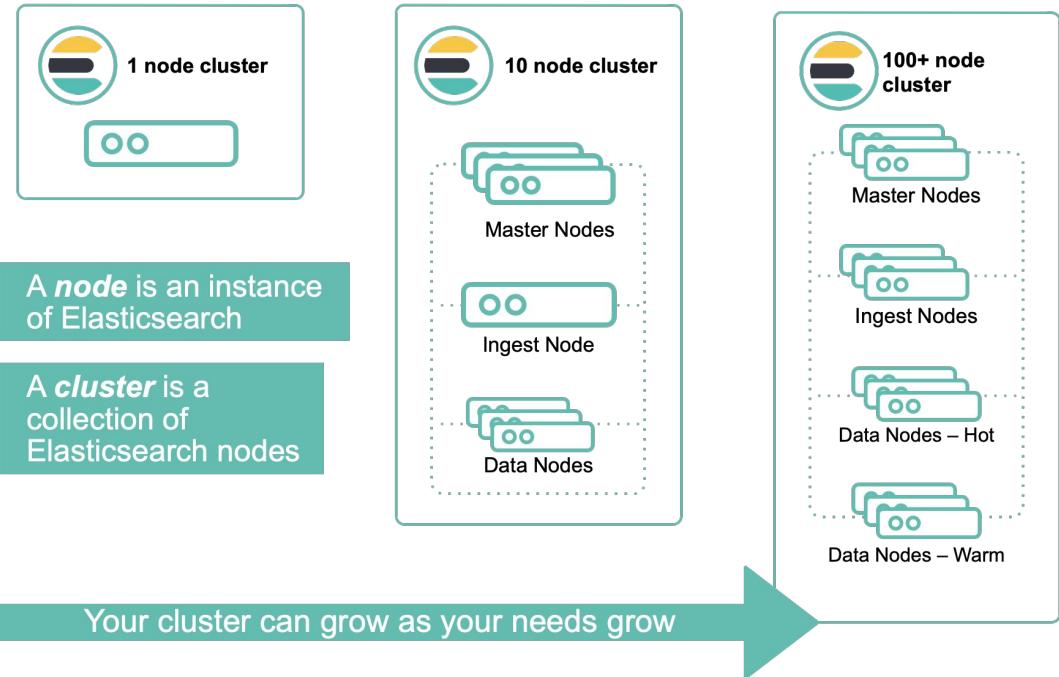


The Elastic Stack and the Elastic Search Platform



Elasticsearch

- Elasticsearch is a ***search*** and ***analytics*** engine for all types of data
- Distributed, highly available and scalable



Elasticsearch is a document store

- Elasticsearch is a distributed **document store**
- A **document** is a serialized JSON object that is stored in Elasticsearch under a unique document ID

A JSON object ...

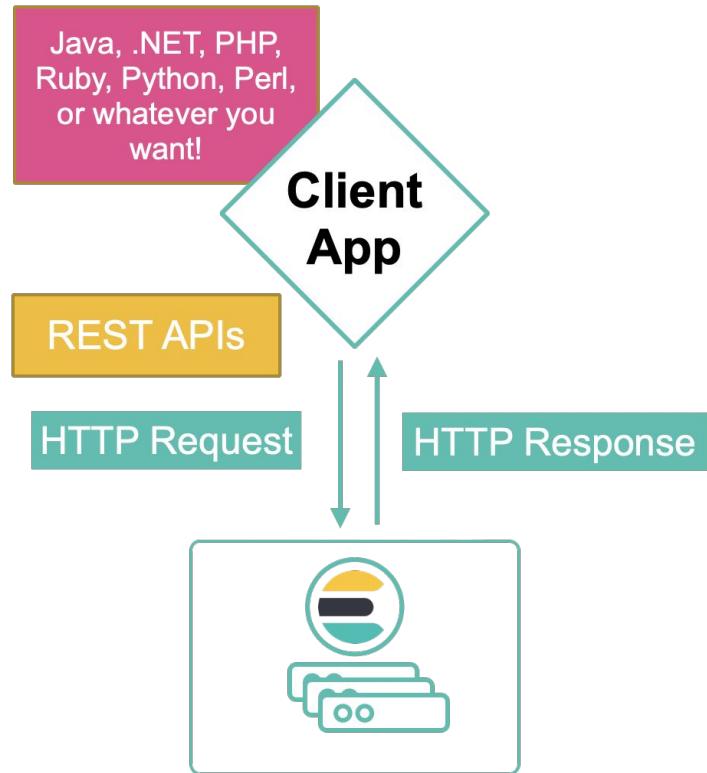
```
{  
  "title": "You Know, for Search",  
  "author_first_name": "Shay",  
  "author_last_name": "Banon",  
  "post_date": "2010-02-08T19...",  
  "body_110n": "ElasticSearch is an  
open source, distributed, RESTful,  
search engine which is built...",  
  ...  
}
```

... is stored in
Elasticsearch as a
document



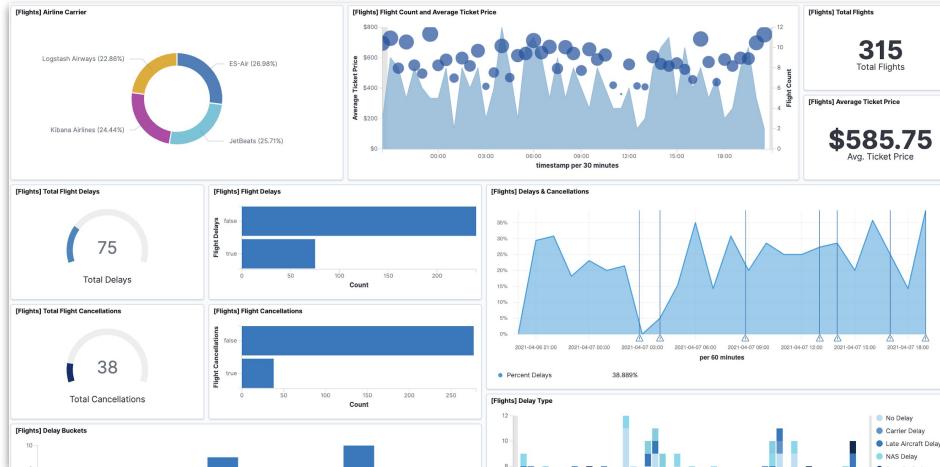
Easily used by any language

- Elasticsearch provides ***REST APIs*** to communicate with a cluster over HTTP
 - enables you to write your applications in any language
 - use a ***language client*** to interact with Elasticsearch using language-native features



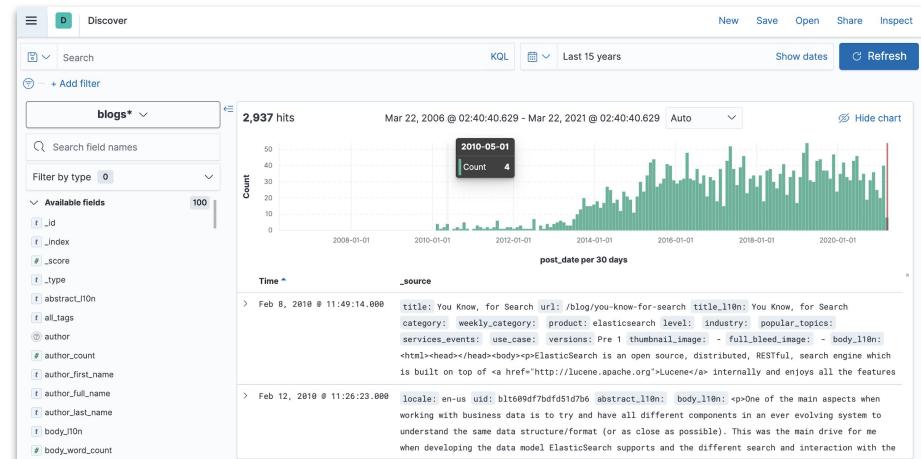
Kibana

- Kibana is a frontend application that sits on top of the Elastic Stack
- It provides **search** and **data visualization** capabilities for data indexed in Elasticsearch



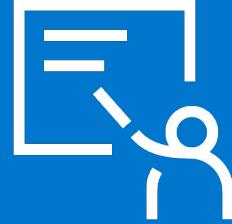
Explore and query your data with Kibana

- Start with **Discover**
 - Create a ***data view*** to access your data
- Explore the fields in your data
- Explore popular values
- Use the query bar and filters to see subsets of your data



Summary: Introduction to the Elastic Stack

Module 1 Lesson 1



Summary

- The ***Elastic Stack*** is a collection of products with ***Elasticsearch*** at the heart
- Elasticsearch is ***distributed*** and ***scales horizontally***
- The ***REST APIs*** enable you to write your client applications in any language
- ***Kibana*** is an analytics and visualization platform
- ***Beats*** are single purpose data shippers
- ***Logstash*** is a server side data processing pipeline

Quiz

1. What component in the Elastic Stack stores data?
2. What do we call an instance of Elasticsearch?
3. **True or False:** to query the data in Elasticsearch, you have to write your own application

Lab Environment



Lab environment

- Visit Strigo using the link that was shared with you
- Click on "**My Lab**" on the left



Lab environment

- Your lab environment has two Elasticsearch clusters
 - each with its own Kibana instance
- Click on ***Lab Instructions*** to start

You can pop out
your labs in a new
window

Elasticsearch Engineer Lab Guide

Lab 1.1: Getting Started

Objective:

In this lab, you will index some sample data that ships with Kibana and view some of the other features of using Kibana.

Table of contents

- Lab 1.1: Getting Started
- Lab 1.2: Data In
- Lab 1.3: Information Out
- Lab 2.1: Search Options
- Lab 2.2: Searching with the Query DSL
- Lab 2.3: Aggregations
- Lab 2.4: Advanced Search

Introduction to the Elastic Stack

Lab 1.1

Index some sample data and get
comfortable navigating Kibana.



Data In

Module 1 Lesson 2



Documents are JSON objects

- Imagine a database table that contains a collection of blogs:

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:

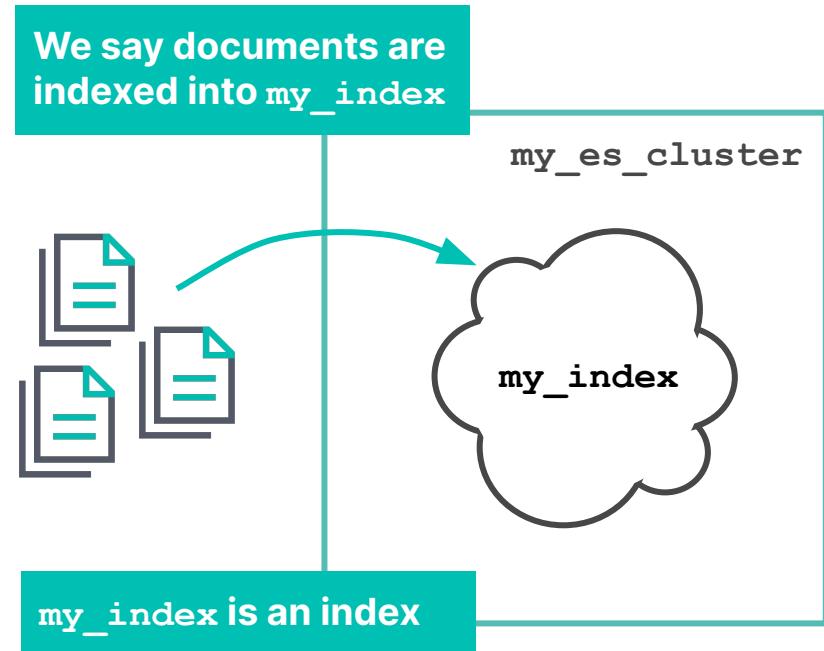
A document consists of **fields**...

```
{  
  "title": "You Know, for Search",  
  "author_first_name": "Shay",  
  "author_last_name": "Banon",  
  "post_date": "2010-02-08T19...",  
  "body_110n": "ElasticSearch is an open  
source, distributed, RESTful, search engine  
which is built...",  
  ...  
}
```

... and **values**

Documents are indexed into an index

- In Elasticsearch, a document is **indexed** into an **index**
 - we use index as a verb and a noun
- An index is a **logical** way of grouping data
 - An index can be thought of as an **optimized** collection of documents



Index a document

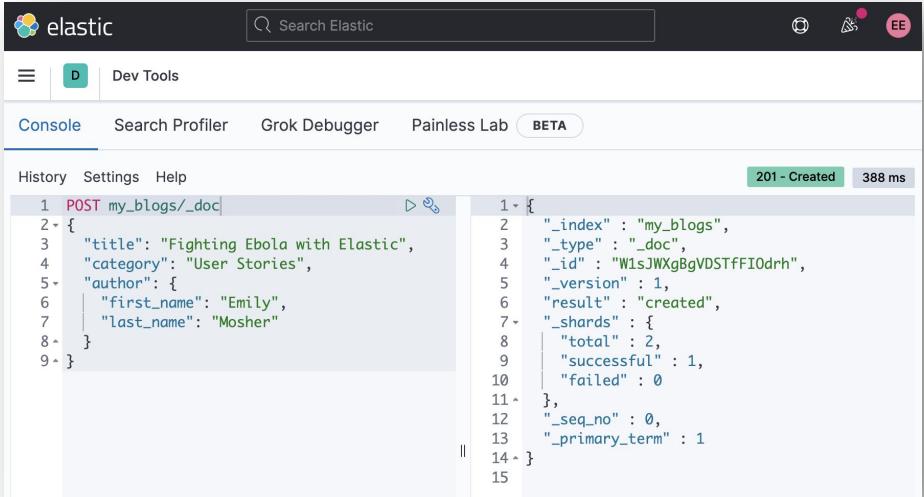
- Send a REST request
- Use **POST** and add the document in the request body
 - Elasticsearch generates the ID for you
 - you can also specify your own ID



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

Console

- Using curl all the time can be a bit tedious
- Kibana has **Console**, a developer tool for creating and submitting Elasticsearch requests more easily



The screenshot shows the Elasticsearch Dev Tools interface with the 'Console' tab selected. A POST request is being typed into the input field:

```
1 POST my_blogs/_doc| ↴ ⏷
2 { ↴ ⏷
3   "title": "Fighting Ebola with Elastic",
4   "category": "User Stories",
5   "author": {
6     "first_name": "Emily",
7     "last_name": "Mosher"
8   }
9 }
```

The response pane displays the successful creation of a document:

```
1 { ↴ ⏷
2   "_index": "my_blogs",
3   "_type": "_doc",
4   "_id": "W1sJWXgBqVDSTfFI0drh",
5   "_version": 1,
6   "result": "created",
7   "_shards": {
8     "total": 2,
9     "successful": 1,
10    "failed": 0
11  },
12  "_seq_no": 0,
13  "_primary_term": 1
14 }
```

Details at the bottom right: 201 - Created, 388 ms.

Retrieve a document

- Use the **Get API** with the document's unique ID:

document resource

document ID

request

```
GET blogs/_doc/QCsiEHkBtIxkzh9Ad3ru
```

response

```
{  
  ...  
  "_id" : "QCsiEHkBtIxkzh9Ad3ru",  
  "_source" : {  
    "locale" : "en-us",  
    "title" : "Becoming an Elastic Certified Engineer pays dividends",  
    "content" : """<p>The headline says it all...  
  
```

Bulk API

- Use the **Bulk API** to index many documents in a single API call
 - greatly increases the indexing speed
 - useful if you need to index a data stream such as log events

request

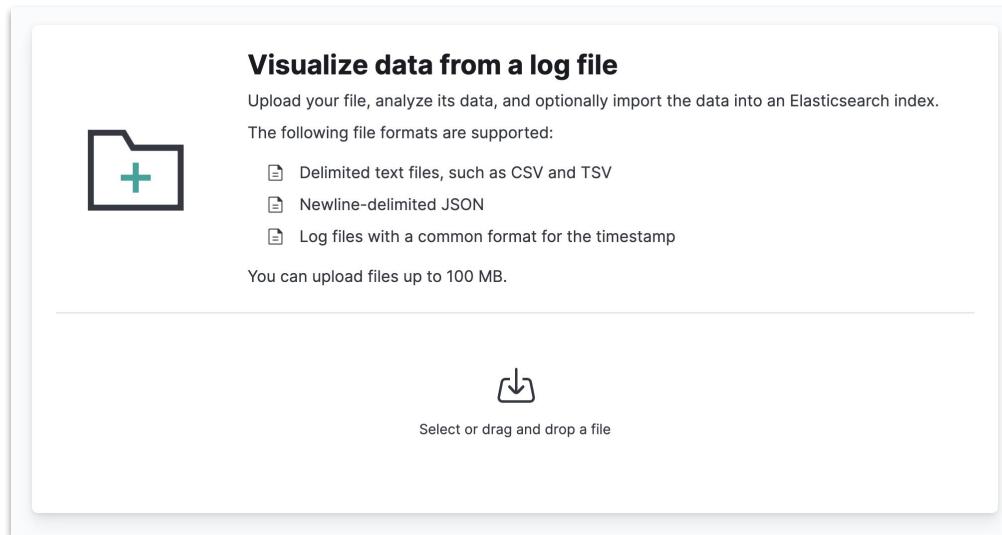
```
POST comments/_bulk
{"index" : {}}
{"title": "Tuning Go Apps with Metricbeat", "category": "Engineering"}
{"index" : {"_id": "unique_doc_id"}}
{"title": "Searching for a needle", "category": "User Stories"}
{"create" : {"_id": "unique_doc_id"}}
 {"title": "Searching for a needle in a haystack"}
 {"update" : {"_id": "unique_doc_id"}}
 {"doc": {"title": "Searching for a needle in a haystack"}}
 {"delete": {"_id": "unique_doc_id"}}
```

The actions are specified in the request body using a newline delimited JSON (NDJSON) structure

Four actions:
create, index,
update, and
delete

Upload a file in Kibana

- Quickly upload a log file or delimited CSV, TSV, or JSON file to start analyzing it



Integrations

- Ship your data from all your sources



Elastic Agent



Beats



Logstash



Web crawler



Content connectors



Elastic language client

- Use Cloud-native integrations to ship data from AWS, Azure or GCS



Microsoft Azure



Google Cloud

Lab Datasets

Understanding data

- Most data can be categorized into:
 - **(relatively) static data:** data set that may grow or change, but slowly or infrequently, like a catalog or inventory of items
 - **time series data:** event data associated with a moment in time that (usually) grows rapidly, like log files or metrics
- Elastic Stack works well with either type of data
 - examples of both types of data will be used in your lab exercises
 - throughout this course, think about what type of data set each tool we introduce can be used with

Static dataset: blogs

- Our static dataset is a collection of Elastic blog posts
- The **blogs** dataset:
 - grows slowly
 - may be updated
 - has old data that may be accessed as frequently as new data



16 MARCH 2021 NEWS

Elastic named a Leader in the 2021 GigaOm Radar on Cloud Observability

By Asawari Samant

[Read more →](#)



16 MARCH 2021 ENGINEERING

Detecting Cobalt Strike with memory signatures

By Joe Desimone

Signature-based detection — especially in-memory scanning — can be a valuable detection strategy. In this blog, learn how to detect Cobalt Strike regardless of configuration or stealth features enabled with an effective false positive rate of zero...

[Read more →](#)



15 MARCH 2021 NEWS

Elastic Cloud Value Calculator: Understand the business value of Elastic...

By Ben Pruden

A deep dive into the Elastic Cloud Value Calculator calculations and assumptions. Use it to understand the details behind each benefit group, or leverage it to create your own version custom to your situation...

[Read more →](#)

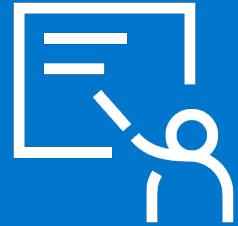
Time series dataset: logs

- Our time series dataset is the web access log files from elastic.co/blog
- Could also be referred to as ***event based data***
- The **web_traffic** dataset:
 - grows quickly
 - contains data that can be read-only
 - older data may eventually be archived

```
{  
  "user_agent" : "Mozilla/5.0 (compatible;  
                 inoreader.com; 7 subscribers)",  
  "request" : "/blog/intro-endpoint-security",  
  "content_type" : "text/html; charset=utf-8",  
  "is_https" : true,  
  "response" : 200,  
  "verb" : "GET",  
  "geoip_location_lat" : 42.683,  
  "geoip_location_lon" : 23.3175,  
  "@timestamp" : "2021-04-05T04:27:34.000Z",  
  "bytes_sent" : 26754,  
  "runtime_ms" : 1170  
}
```

Summary: Data In

Module 1 Lesson 2



Summary

- Most of our users' data falls into one of two categories:
 - **static data**
 - **time series data**
- A **document** is a serialized JSON object that is stored in Elasticsearch under a unique ID
- Elasticsearch creates a data structure called **inverted index** to support fast searches
- Documents can be **indexed** into an **index** in many ways:
 - curl, Console, Bulk API, Data Uploader, Beats, Agent, Logstash

Quiz

1. **True or False:** A data set of an inventory of items that grows slowly would be categorized as a static data set
2. What method of ingest would you use if you wanted to index:
 - a. a few CSV files?
 - b. metrics from 50 Nginx servers?
3. **True or False:** The Bulk API provides an efficient way to index multiple documents in a single POST request.

Data In

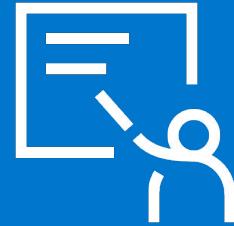
Lab 1.2

Index the blogs dataset using the file uploader in Kibana and the Bulk API



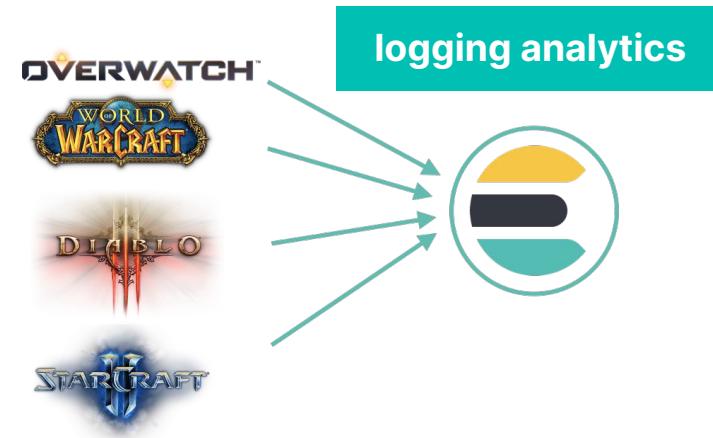
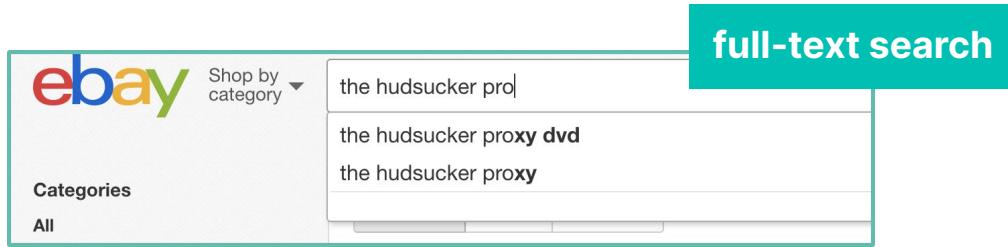
Information Out

Module 1 Lesson 3



Search

- Search is used to solve different problems



Different use cases, different challenges...

- **Full-text search:**
 - Typically uses human generated, error-prone data
 - Often uses free-form text fields for anybody to type anything
- **Logging analytics:**
 - Need to analyze HUGE amounts of data in real-time
 - Ingest load can vary
- **Operational metrics:**
 - Collect data from MANY different sources with different data formats

Basic structure of search

- In Elasticsearch, search breaks down into two basic parts:
 - Queries**
 - Which **documents** meet a specific set of criteria?
 - Aggregations**
 - Tell me something about a **bunch of documents**

Which blog titles are about “community”?

In which month did our blogs have the most views?

In which month did blogs about “community” have the most views?

Basic structure of a search request

request

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

response

```
{
  "took": 3,
  ...
  "hits": {
    "total": {...},
    "max_score": 5.76,
    "hits": [
      ...
    ],
    "aggregations": {
      "top_authors": {
        "buckets": [
          ...
        ]
      }
    }
  }
}
```

Queries find documents that match

Aggregations summarize the matched documents

hits = top 10 documents about community

buckets = top 10 authors who wrote about community

Queries

Query options

- There are several query languages to choose from:

- **KQL**
- **Lucene**

Used in Kibana's query bar

- **Query DSL**

Gives access to all Elasticsearch options
Most flexible

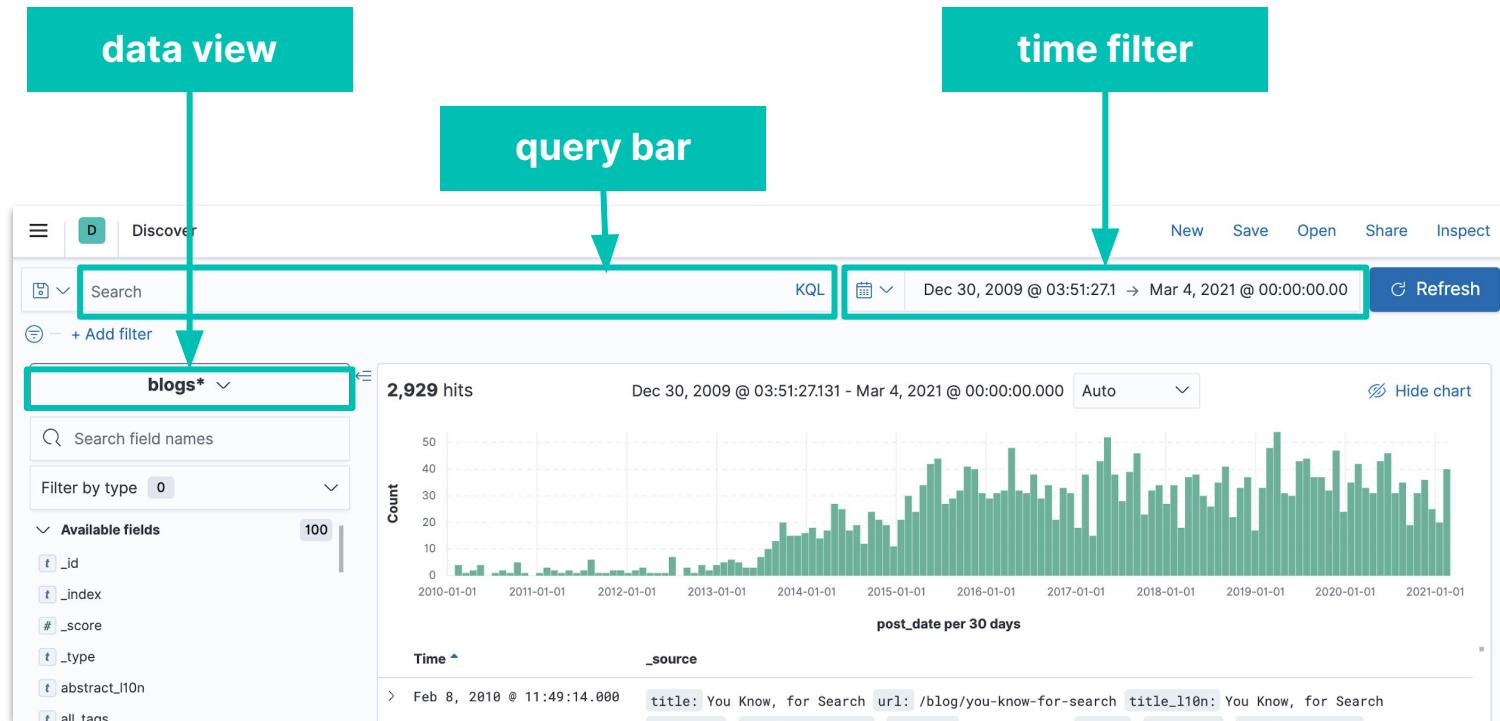
- **Elasticsearch SQL**

Search indices using the familiar SQL syntax

- **EQL**

For security data, threat hunting

Searching in Kibana



Query DSL

- A complete and robust query language for Elasticsearch
 - full-text search
 - aggregations
 - sort, paginate, manipulate responses

DSL = Domain Specific Language

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

The match_all query

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

request

GET blogs/_search

index or indices for your search

can be a comma separated list
or use wildcard (*)

response

```
{  
  "took" : 1,  
  "timed_out" : false,  
  "_shards" : { ... },  
  "hits" : {  
    "total" : { ... },  
    "max_score" : 1.0,  
    "hits" : [ ... ]  
  }  
}
```

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

total = the number of documents that matched this query

hits = array containing the documents that hit the search criteria

Let's search for terms

- Suppose you would like to search for blogs about "**community team**"
- Let's search for "**community team**" in the "**title**" field
- What do you think is required for a document to be a hit?

request

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

match uses **or** logic and is case-insensitive

- By default, the `match` query uses "**or**" logic if multiple terms appear in the search query
 - any document with the term "`community`" **or** "`team`" in the "`title`" field will be a hit
- By default, the match query is **case-insensitive**
- We'll cover queries in detail in module 3

request

```
GET blogs/_search
{
  "query": {
    "match": {
      "title": [
        "community team"
      ]
    }
  }
}
```

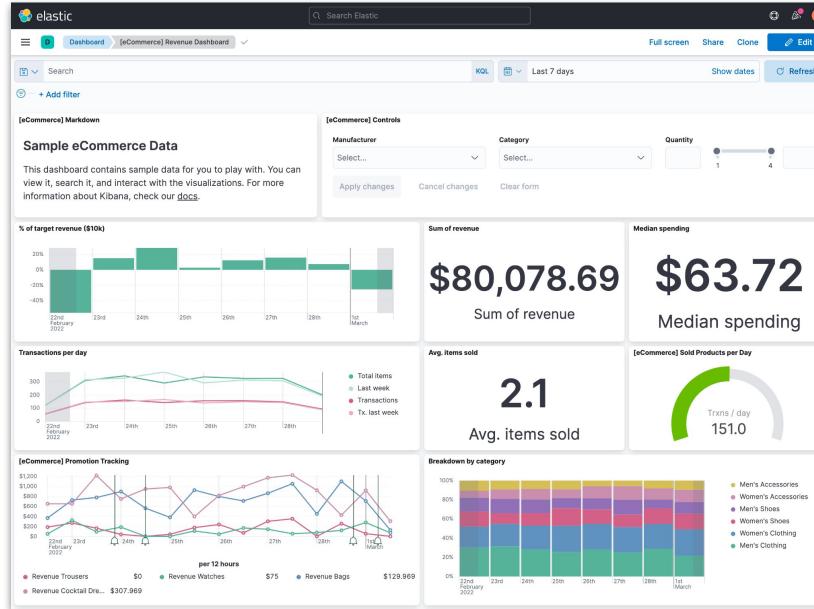
response

```
"title" : "Meet the team behind the Elastic Community Conference"
"title" : "Introducing Endgame Red Team Automation"
"title" : "Welcome Insight.io to the Elastic Team"
"title" : "Welcome Prelert to the Elastic Team"
...
...
```

Aggregations

Aggregations

- Every visualization on a Kibana dashboard is powered by aggregations



Let's aggregate the data

- Suppose you want to know who has written the most blogs
- You can use an aggregation for that

request

```
GET blogs/_search
{
  "size": 0,
  "aggregations": {
    "my_terms_agg": {
      "terms": {
        "field": "authors.full_name.keyword"
      }
    }
  }
}
```

response

```
"buckets" : [
  {
    "key" : "Shay Banon",
    "doc_count" : 204
  },
  {
    "key" : "シャイ バノン",
    "doc_count" : 204
  },
  {
    "key" : "Clinton Gormley",
    "doc_count" : 200
  },
  ...
]
```

Metric aggregations

- Aggregations can summarize your data
- For example: what is the number of unique blog authors?

request

```
GET blogs/_search
{
  "size": 0,
  "aggregations": {
    "my_cardinality_agg": {
      "cardinality": {
        "field": "authors.full_name.keyword"
      }
    }
  }
}
```

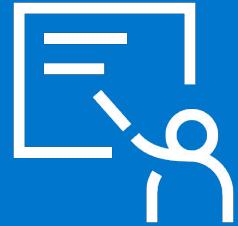
response

```
...
"aggregations" : {
  "my_cardinality_agg" : {
    "value" : 743
  }
}
...
```

- We'll cover aggregations in detail in module 4

Summary: Information Out

Module 1 Lesson 3



Summary

- Use ***Discover*** to learn about your dataset such as
 - what type of fields it has
 - what are the range of values found in each field
- Use the ***search APIs*** to retrieve or find documents in indices
- The ***query DSL*** enables you to search and aggregate over your data with a single request

Quiz

1. What two query languages are supported by the Kibana query bar?
2. **True or False:** The match query uses ***and*** logic by default
3. What is the default number of hits returned by a query using the search API?

Information Out

Lab 1.3

Query data with Discover, SQL and
the query DSL



More resources

- Intro to the Elastic Stack
 - www.elastic.co/elastic-stack/
- How to ingest data
 - www.elastic.co/integrations/
- Kibana
 - www.elastic.co/kibana/
- KQL
 - www.elastic.co/guide/en/kibana/current/kuery-query.html
- Query DSL
 - www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html

Elasticsearch Engineer: Agenda

- Module 1: Getting started
- **Module 2: Data modeling**
- Module 3: You know, for search
- Module 4: Data processing
- Module 5: Aggregations
- Module 6: The one about shards
- Module 7: Data management
- Module 8: Cluster management

Data Modeling

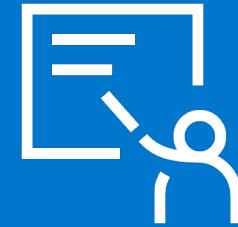
Module 2

Topics

- Strings in Elasticsearch
- Overview of Mappings
- Text Analysis
- Types and Parameters

Strings in Elasticsearch

Module 2 Lesson 1



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 called ***text analysis*** which occurs when your string fields are indexed

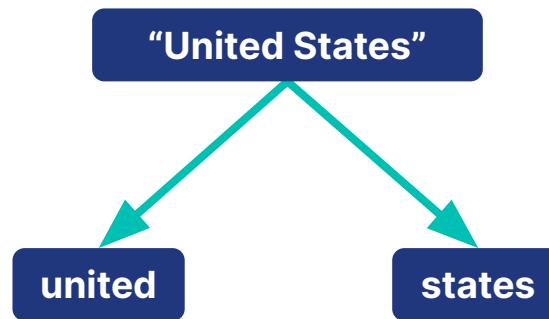
```
GET blogs/_search
{
  "query": {
    "match": {
      "content": "united states"
    }
  }
}
```

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

These two
queries return
the same hits

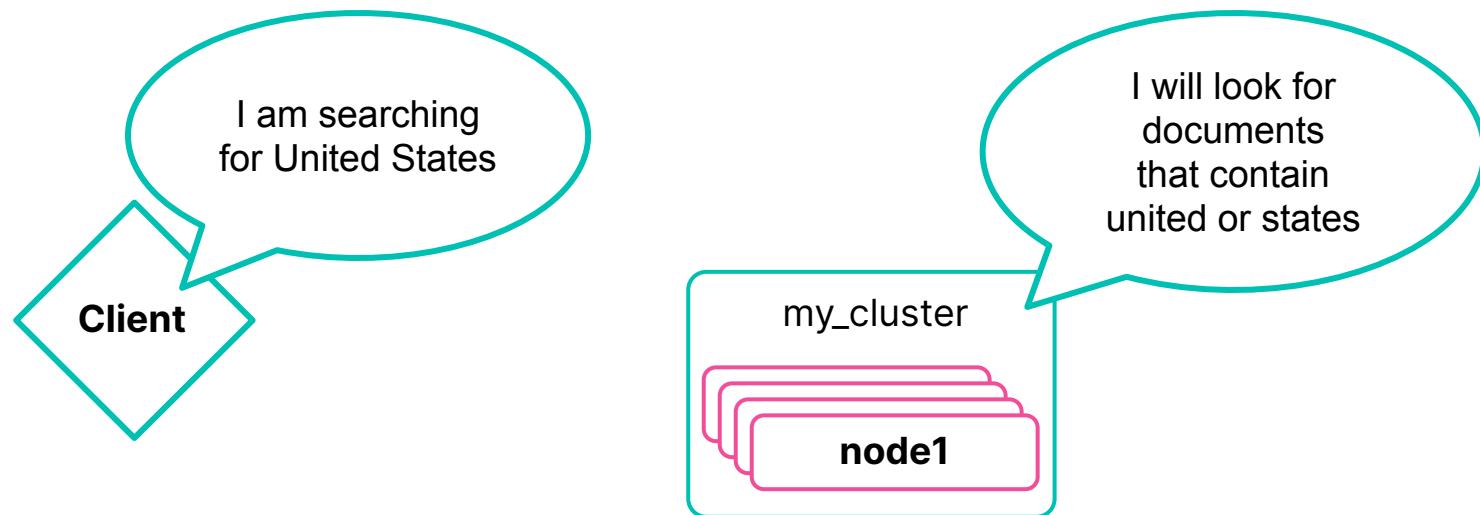
Analysis makes text searchable

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



Your query string is analyzed too

- This makes searches case-insensitive by default



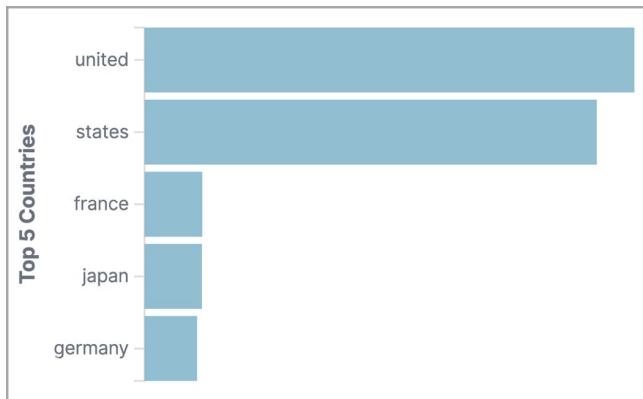
Analyzers

- Text analysis is done by an *analyzer*
- By default, Elasticsearch applies the **standard** analyzer
- There are many other analyzers, including:
 - whitespace, stop, pattern, language-specific analyzers, and more
- The built-in analyzers work great for many use cases
 - but you may need to define your own custom analyzers
 - we'll cover those in an upcoming lesson

Text and keyword

Some strings do not need to be analyzed

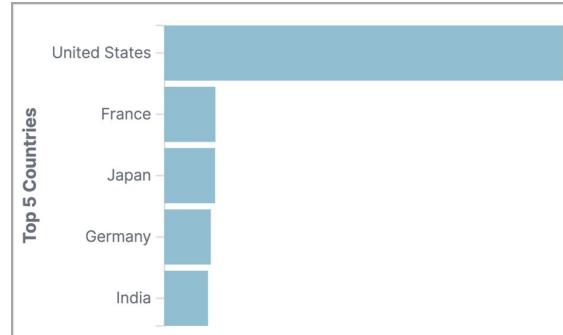
- Text analysis is great for full text search
 - enables you to search for individual words in a case-insensitive manner
- However, it is not great for things like aggregations
 - when you often want to see the original strings



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

Keyword vs. text

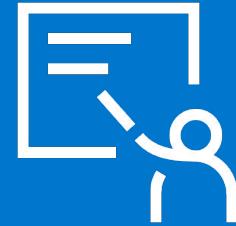
- Elasticsearch has two kinds of string data types:
 - text***, for **full-text search**
 - text fields are **analyzed**
 - keyword***, for **aggregations, sorting and exact searches**
 - keyword fields are **not analyzed**
 - the original strings, as they occur in the documents



Aggregate on a
keyword field instead

Summary: Strings in Elasticsearch

Module 2 Lesson 1



Summary

- Elasticsearch uses two string data types:
 - **Text** for full-text search
 - **Keyword** for exact searches, aggregations and sorting
- Text strings are ***analyzed***
- Keyword strings are not analyzed

Quiz

1. **True or False:** Text strings are lowercase by default
2. **True or False:** Keyword strings are lowercased by default

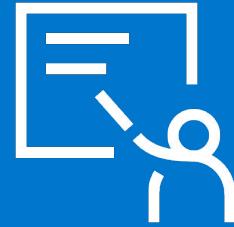
Strings in Elasticsearch

Lab 2.1



Mapping

Module 2 Lesson 2



What is a mapping?

- A ***mapping*** is a ***per-index*** schema definition that contains:
 - names of fields
 - data types of fields
 - how the field should be indexed and stored
- 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

Elasticsearch data types for fields

- Simple types:
 - **text**: for full-text (analyzed) strings
 - **keyword**: for exact value strings and aggregations
 - **date** and **date_nanos**: string formatted as dates, or numeric dates
 - numbers: **byte**, **short**, **integer**, **long**, **float**, **double**, **half_float**
 - **boolean**
 - **geo** types
- Hierarchical types: like **object** and **nested**
- www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html

Defining a mapping

- In many use cases, you will need to define your own mapping
- Defined in the **mappings** section of an index. You can:
 - define a mapping at index creation:

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

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

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

Why have we not defined a mapping yet?

- When you index a document with unmapped fields, Elasticsearch dynamically **creates** or **updates** the mapping for those fields
 - a new mapping is defined if one does not exist
 - fields not already defined in a mapping are added

```
POST my_blogs/_doc
{
  "username": "kimchy",
  "comment": "Search is something that any application should have",
  "details": {
    "created_at": "2010-04-23T15:48:50",
    "version": 2.1,
    "employee": true
  }
}
```

Dynamic mapping

- Our example doc produces this mapping...

request

```
GET my_blogs/_mapping
```

```
"my_blogs" : {
    "mappings" : {
        "properties" : {
            ...
            "details" : {
                "properties" :
                    "created_at" : {
                        "type" : "date"
                    },
                    "employee" : {
                        "type" : "boolean"
                    },
                    "version" : {
                        "type" : "float"
                    } } },
        "username" : {
            "type" : "text",
            "fields" : {
                "keyword" : {
                    "type" : "keyword",
                    "ignore_above" : 256
                }
            } } } } }
```

Multi-fields

Text and keyword in mapping

- 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 will give you both by default:
 - it will analyze any string as type **text**
 - and it will create a **keyword multi-field**

Understanding multi-fields

- Multi-fields enable you to index a field in multiple ways

```
POST my_index/_doc
{
  "country_name": "United States"
}
```

country_name is analyzed

united

states

country_name.keyword is not analyzed

United States

Multi-fields in the mapping

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

request

```
GET my_index/_mapping
```

response

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

Do you always need text and keyword?

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

Mapping optimizations

Dynamic mapping is rarely optimal

- For example, the default for an integer is **long**
 - not always appropriate for the content
- A more tailored type can help save on memory and speed

```
"status_code": 200
```



```
"status_code": {  
    "type": "short"  
}
```

Can you change a mapping?

- **No** – not without *reindexing* your documents
 - adding new fields is possible
 - all other mapping changes require reindexing
- **Why not?**
 - if you could switch a field's data type, all the values that were already indexed before the switch would become unsearchable on that field
- Invest the time to create a great mapping before you go to production

Fixing mappings

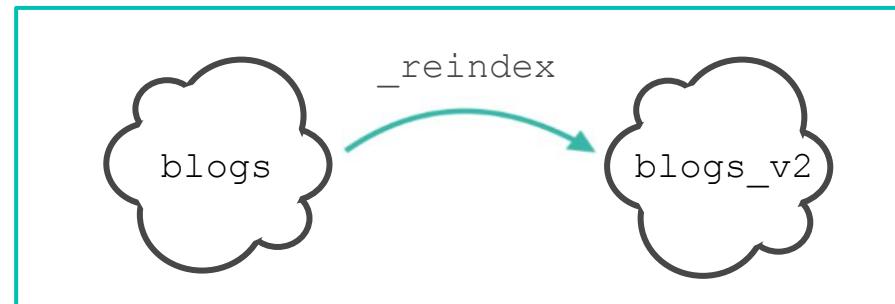
- Create a new index with the updated mapping

```
PUT blogs_v2
{
  "mappings": {
    "properties": {
      "publish_date": {
        "type": "date"
      }
    }
  }
}
```

The Reindex API

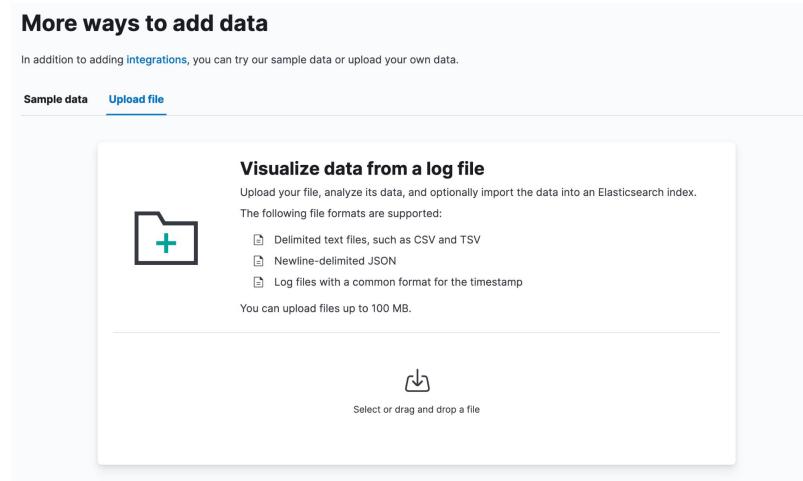
- To populate the new index, use the **reindex API**
 - reads data from one index and indexes them into another
 - use it to modify your mappings

```
POST _reindex
{
  "source": {
    "index": "blogs"
  },
  "dest": {
    "index": "blogs_v2"
  }
}
```



Defining your own mapping

- Use Kibana's **file uploader**, which does an excellent job of guessing data types
 - and allows you to customize the mapping before index creation



Defining your own mapping manually

- If not using the file uploader, to define an explicit mapping, follow these steps:
 1. **Index a sample document** that contains the fields you want defined in the mapping (into a dummy index)
 2. **Get the dynamic mapping** that was created automatically by Elasticsearch
 3. **Modify** the mapping definition
 4. **Create your index** using your custom mapping

Step 1: Index a sample document

- Start by indexing a document into a dummy index
 - Use values that will map closely to the data types you want

```
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"
}
```

Step 2: Get the dynamic mapping

- **GET** the mapping, then copy-paste it into **Console**
 - in Kibana's file uploader, this is in the **Advanced** section after **Import**

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

Step 3: Edit the mapping

- Define the mappings according to your use case:
 - **keyword** might work well for **category**
 - **content** may only need to be **text**

```
"mappings": {  
    "properties": {  
        "@timestamp": {  
            "type": "date"  
        },  
        "content": {  
            "type": "text"  
        },  
        "category": {  
            "type": "keyword"  
        }  
    }  
}
```

Step 4: Create a new index with the mapping

- `new_blogs` is now a new index with our explicit mappings
- Documents can now be indexed

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

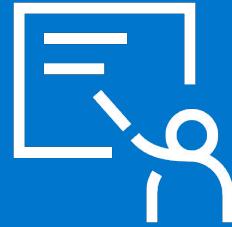
Great mapping means great performance

- Fast index speed
- Optimized storage space
- Improved searches

Summary:

Mapping

Module 2 Lesson 2



Summary

- **A *mapping*** is Elasticsearch's data schema
- A mapping is defined ***per index***
- If you do not define an explicit mapping, Elasticsearch will ***dynamically*** map the fields in your documents
- You cannot change the mapping of a field after the index has been created, but you can add new fields to a mapping
- Creating a custom mapping will produce savings in search and index speed, as well as memory and storage requirements

Quiz

1. **True or False:** A mappings is defined per index
2. **True or False:** You can change a field's data type from **integer** to **long** because those two types are compatible
3. Give two reasons why creating your own custom mapping is advisable

Mapping

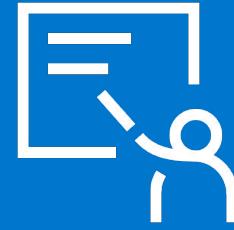
Lab 2.2

Define a custom mapping for the blogs index.



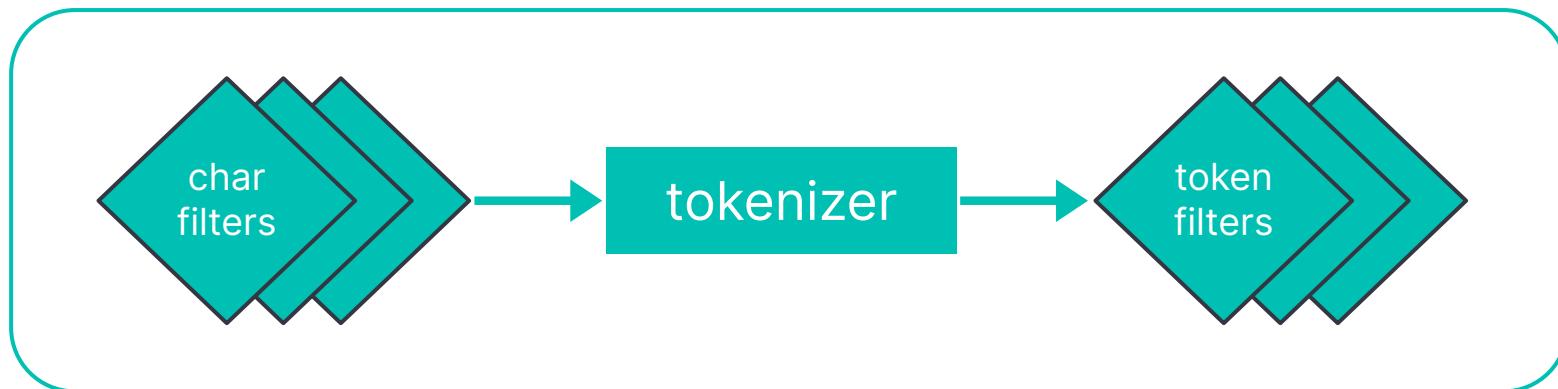
Text analysis

Module 2 Lesson 3



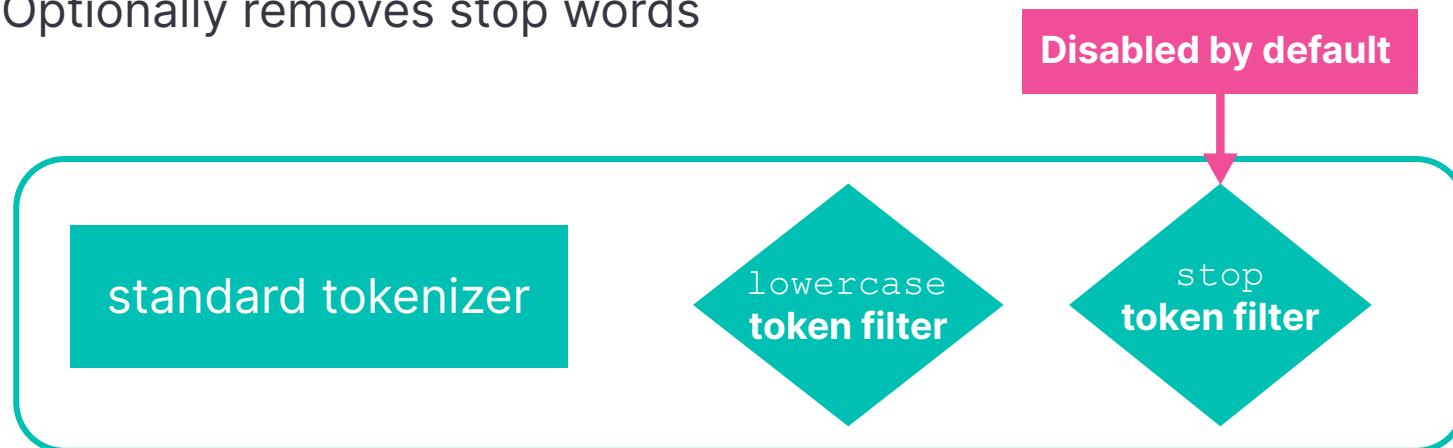
Anatomy of an analyzer

- An analyzer consists of three parts:
 - zero or more ***character filters***
 - exactly one ***tokenizer***
 - zero or more ***token filters***



The standard analyzer

- The default analyzer
- No character filters
- Uses the standard tokenizer
- Lowercases all tokens
- Optionally removes stop words

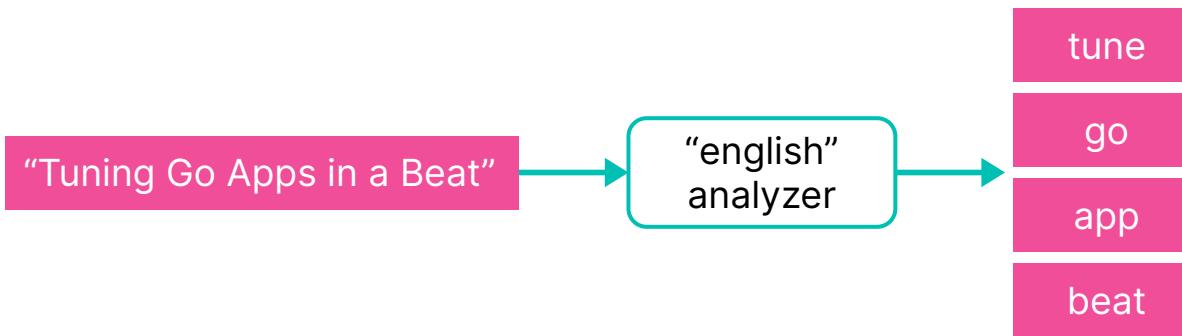


The Analyze API

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

request

```
GET _analyze
{
  "analyzer": "english",
  "text": "Tuning Go Apps in a Beat"
}
```



Custom analyzers

The use case for custom analyzers

- Many Elastic blogs talk about "**X-Pack**"
 - broken into separate tokens by the standard analyzer:

request

```
POST _analyze
{
  "text": [
    "We are excited to introduce
    the world to X-Pack."
  ],
  "analyzer": "standard"
}
```

A query for **X-Pack** returns all
blogs containing **X** or **Pack**

response

```
{
  "token": "x",
  "start_offset": 42,
  "end_offset": 43,
  "type": "<ALPHANUM>",
  "position": 8
},
{
  "token": "pack",
  "start_offset": 44,
  "end_offset": 48,
  "type": "<ALPHANUM>",
  "position": 9
}
```

Defining a custom analyzer

- In index settings, define:
 - a ***mapping*** character filter to remove the hyphen
 - a custom analyzer that uses the character filter
- The custom **my_content_analyzer** analyzer changes **X-Pack** into **XPack**

```
PUT blogs_test
{
  "settings": {
    "analysis": {
      "char_filter": {
        "my_filter": {
          "type": "mapping",
          "mappings": [
            "X-Pack => XPack"
          ]
        }
      },
      "analyzer": {
        "my_content_analyzer": {
          "type": "custom",
          "char_filter": [
            "my_filter"
          ],
          "tokenizer": "standard",
          "filter": [
            "lowercase"
          ]
        }
      }
    }
  }
  ...
}
```

Test the analyzer

- Let's make sure "X-Pack" is analyzed as expected:

request

```
POST blogs_test/_analyze
{
  "text": ["We love X-Pack"],
  "analyzer": "my_content_analyzer"
}
```

- Note that you can run _analyze against an index to test custom analyze defined in that index

response

```
{
  "token": "we",
  "start_offset": 0,
  "end_offset": 2,
  "type": "<ALPHANUM>",
  "position": 0
},
{
  "token": "love",
  "start_offset": 3,
  "end_offset": 7,
  "type": "<ALPHANUM>",
  "position": 1
},
{
  "token": "xpack",
  "start_offset": 8,
  "end_offset": 14,
  "type": "<ALPHANUM>",
  "position": 2
}
```

Applying custom analyzers to fields

- Configure a field to use the custom analyzer in the mapping:

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

Order matters in filters

```
GET _analyze
{
  "tokenizer": "whitespace",
  "filter": ["lowercase", "stop"],
  "text": "To Be Or Not To Be"
}
```

[]

```
GET _analyze
{
  "tokenizer": "whitespace",
  "filter": ["stop", "lowercase"],
  "text": "To Be Or Not To Be"
}
```

to
be
or
not
to
be

Summary: Text analysis

Module 2 Lesson 3



Summary

- An analyzer consists of:
 - zero or more character filters
 - one tokenizer
 - zero or more token filters
- Custom analyzers are defined per index
- Apply an analyzer to text fields in the mapping
- Use the **_analyze** API to test your analyzers

Quiz

1. **True or False:** When defining an analyzer, token filters are applied in the order they are listed
2. **True or False:** It is possible to define multiple tokenizers in an analyzer

Text analysis

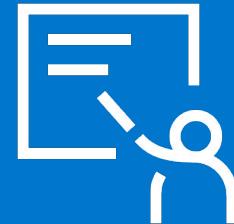
Lab 2.3

Configure custom analyzers for the
blogs index



Types and parameters

Module 2 Lesson 4



Mapping parameters

Mapping parameters

- In addition to the **type**, fields in a mapping can be configured with additional parameters
 - for example to set the analyzer for a text field:

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

- www.elastic.co/guide/en/elasticsearch/reference/current/mapping-params.html

Date formats

- Use **format** to set the **date format** used for date fields
 - defaults to ISO 8601
- Choose from dozens of built-in date formats or define your own

```
"properties": {  
    "my_date_field" : {  
        "type": "date",  
        "format" : "dd/MM/yyyy||epoch_millis"  
    }  
}
```

Coercing data

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

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

All three documents can be indexed and the ratings values will be **coerced** into numbers

Coercing data

- You can ***disable coercion*** if you want Elasticsearch to reject documents that have unexpected values:

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

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



Not storing doc values

- By default, Elasticsearch creates a **doc values** data structure for many fields during indexing
 - doc values enable you to aggregate/sort on those fields
 - but take up disk space
- Fields that won't be used for aggregations or sorting:
 - set **doc_values** to **false**

```
"url" : {  
    "type": "keyword",  
    "doc_values" : false  
}
```

Not indexing a field

- By default, for every field, Elasticsearch creates a data structure that enables fast queries
 - ***inverted index*** (e.g. text, keyword) or ***BKD tree*** (e.g. geo and numeric)
 - takes up disk space
- Set **index** to **false** for fields that do not require fast querying
 - fields with doc values still support slower queries

```
"display_name": {  
    "type": "keyword",  
    "index": false  
}
```

Disabling a field

- A field that won't be used at all and should just be stored in _source:
 - set **enabled** to **false**

```
"display_name": {  
    "enabled": false  
}
```

The `copy_to` parameter

- Consider a document with three location fields:

```
POST locations/_doc
{
  "region_name": "Victoria",
  "country_name": "Australia",
  "city_name": "Surrey Hills"
}
```

- You could use a `bool/multi_match` query to search all three fields
- Or* you could copy all three values to a single field during indexing using `copy_to...`

The copy_to parameter

```
"properties": {  
    "region_name": {  
        "type": "keyword",  
        "index": "false",  
        "copy_to": "locations_combined"  
    },  
    "country_name": {  
        "type": "keyword",  
        "index": "false",  
        "copy_to": "locations_combined"  
    },  
    "city_name": {  
        "type": "keyword",  
        "index": "false",  
        "copy_to": "locations_combined"  
    },  
    "locations_combined": {  
        "type": "text"  
    }  
}
```

During indexing, the values will be copied to the **locations_combined** field

The copy_to parameter

- The **locations_combined** field is not stored in the `_source`
 - but it is indexed, so you can query on it:

request

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

response

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

Dynamic data

Use case for dynamic templates

- Manually defining a mapping can be tedious when you:
 - have documents with a ***large number of fields***
 - or ***don't know the fields*** ahead of time
 - or want to ***change the default mapping*** for certain field types
- Use ***dynamic templates*** to define a field's mapping based on one of the following:
 - the field's ***data type***
 - the ***name*** of the field
 - the ***path*** to the field

Dynamic template example

- Map any string field with a name that starts with **ip*** as type IP:

```
PUT my_index
{
  "mappings": {
    "dynamic_templates": [
      {
        "strings_as_ip": {
          "match_mapping_type": "string",
          "match": "ip*",
          "mapping": {
            "type": "ip"
          }
        }
      }
    ]
  }
}
```

request

```
POST my_index/_doc
{
  "ip_adress": "157.97.192.70"
}

GET my_index/_mapping
```

response

```
"properties" : {
  "ip_adress" : {
    "type" : "ip"
  }
}
```

Summary: Types and parameters

Module 2 Lesson 4



Summary

- Elasticsearch data types control how fields are processed, stored, and queried
- ***Mapping parameters*** enable you to define how Elasticsearch indexes the fields in your documents
- ***Dynamic templates*** make it easier to set up your own mappings by defining defaults for fields based on their JSON type, name, or path

Quiz

1. What **mapping parameter** would you use to change a field's date format?
2. **True or False:** The `copy_to` parameter adds a new field to the `_source` that is returned with the hits
3. **True or False:** If you set `enabled` to `false`, you can no longer query on that field, but still aggregate on it

Types and parameters

Lab 2.4

Iteratively improve and optimize the mapping for the blogs index



More resources

- Mapping
 - www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html
- Multi-fields
 - www.elastic.co/guide/en/elasticsearch/reference/master/multi-fields.html
- Analyzers
 - www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html
- Date formats
 - www.elastic.co/guide/en/elasticsearch/reference/current/mapping-date-format.html
- Dynamic templates
 - www.elastic.co/guide/en/elasticsearch/reference/current/dynamic-templates.html

Elasticsearch Engineer: Agenda

- Module 1: Getting started
- Module 2: Data modeling
- **Module 3: You know, for search**
- Module 4: Data processing
- Module 5: Aggregations
- Module 6: The one about shards
- Module 7: Data management
- Module 8: Cluster management

You know, for search

Module 3

Topics

- Searching with the Query DSL
- More queries
- Developing search applications

Searching with the Query DSL

Module 3 Lesson 1



Query DSL

- In module 1, you've encountered the Query DSL
- A search language for Elasticsearch
 - query
 - aggregate
 - sort, paginate, manipulate responses
 - DSL = Domain Specific Language

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

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 "**community**" **or** "**team**" in the **title** field is a hit

request

```
GET blogs/_search
{
  "query": {
    "match": {
      "title": [
        "community team"
      ]
    }
  }
}
```

response

```
"title" : "Meet the team behind the Elastic Community Conference"
"title" : "Introducing Endgame Red Team Automation"
"title" : "Welcome Insight.io to the Elastic Team"
"title" : "Welcome Prelert to the Elastic Team"
...
...
```

Search for all the terms

- Change the **or** logic of the `match` query into **and** using the `operator` parameter

Only 1 match!

```
GET blogs/_search
{
  "query": {
    "match": {
      "title": {
        "query": "community team",
        "operator": "and"
      }
    }
  }
}
```

Change the structure of the query to
use optional parameters

The `minimum_should_match` parameter

- 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
- Two of the search terms must occur in the title of a document for it to be a match

```
GET blogs/_search
{
  "query": {
    "match": {
      "title": {
        "query": "elastic community team",
        "minimum_should_match": 2
      }
    }
  }
}
```

match does not care about the order of terms

- Even with the **and** operator, the order of the terms does not matter
 - nor how far apart the terms are

request

```
GET blogs/_search
{
  "query": {
    "match": {
      "title": {
        "query": "community team",
        "operator": "and"
      }
    }
  }
}
```

response

```
"title" : "Meet the team behind the
Elastic Community Conference"
```

The `match_phrase` query

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

```
GET blogs/_search
{
  "query": {
    "match_phrase": {
      "title": "community team"
    }
  }
}
```

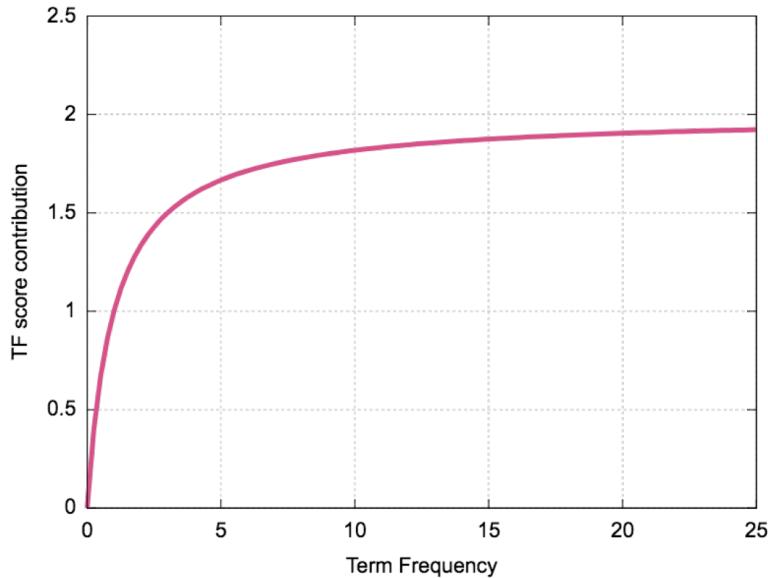
Returns no hits as there are no documents with these two terms with this exact phrase in the title

The response

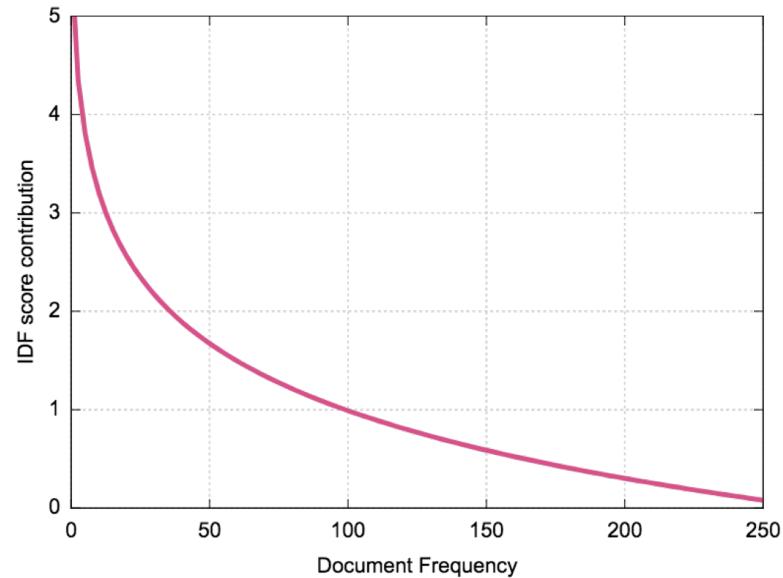
Score!

- Elasticsearch calculates a **score** for each document that is a hit
 - represents how relevant a document is in regards to the specific query
- Elasticsearch's default scoring algorithm is **BM25**
- Three elements determine 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

TF and IDF



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



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

Query response

- By default the query response will return:
 - the top 10 documents that match the query
 - sorted by `_score` in descending order

```
GET blogs/_search
{
  "from": 0,
  "size": 10,
  "sort": [
    "_score": {
      "order": "desc"
    }
  ],
  "query": {
    ...
  }
}
```

Default settings made explicit

Changing the response

- Set **from** and **size** to paginate through the search results
- Set **sort** to sort on one or more fields instead of score

```
GET blogs/_search
{
  "from": 100,
  "size": 50,
  "sort": [
    {
      "publish_date": {
        "order": "asc"
      }
    },
    "_score"
  ],
  "query": {
    ...
  }
}
```

Retrieve **50** hits,
starting from hit **100**

Sort primarily on **publish_date**, in **ascending** order.
If two documents have the same publish_date, sort them on **_score**

Retrieve selected fields

- By default, each hit in the response includes the document's `_source`
 - the original document that was indexed
- Use `fields` to only retrieve specific fields

```
GET blogs/_search
{
  "_source": false,
  "fields
```

Summary: Searching with the Query DSL

Module 3 Lesson 1



Summary

- The Query DSL enables you to:
 - query
 - aggregate
 - sort, paginate, manipulate responses
- The match query defaults to the **or** operator, but you can change it into **and**
- Use the **match_phrase** query if you care about the order and position of search terms
- Elasticsearch will calculate a score for each hit, using the **BM25** algorithm

Quiz

1. Which two Query DSL parameters can you use to paginate through search results?
2. **True or False:** Search results are always sorted by score.
3. What is the default number of hits returned from a query using the Query DSL?

Searching with the Query DSL

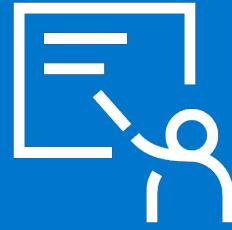
Lab 3.1

Search data in Elasticsearch



More queries

Module 3 Lesson 2



Queries

- The Query DSL offers many types of queries

Full-text queries

- match
- match_phrase
- multi_match
- query_string
- ...

Term-level queries

- term
- range
- exists
- fuzzy
- regexp
- wildcard
- ...

Many more

- script
- percolate
- span_queries
- geo_queries
- nested
- ...

Searching for numbers, dates and IPs

- So far you've used the **match** query
 - a **full-text query**
 - great for searching words in bodies of text
- How would you query for dates, numbers and IPs?
 - For example:
find all blogs that have been written since 2020
- Use the **range** query

The range query

- Use the following parameters to specify a range:
 - **gt** - greater than
 - **gte** - greater than or equal to
 - **- less than**
 - **- less than or equal to**
- Ranges can be open ended

```
GET blogs/_search
{
  "query": {
    "range": {
      "publish_date": {
        "gte": "2020-01-01",
        "lte": "2021-12-31"
      }
    }
  }
}
```

Find all blogs that were published in 2020 or 2021

Date math

- Use **date math** to express relative dates in range queries

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

If now = 2021-10-19T11:56:22	
now-1h	2021-10-19T10:56:22
now+1h+30m	2021-10-19T13:26:22
now/d+1d	2021-10-20T00:00:00
2022-01-15 +1M	2022-02-15T00:00:00

range query with date math example

```
GET blogs/_search
{
  "query": {
    "range": {
      "publish_date": {
        "gte": "now-1y"
      }
    }
  }
}
```

Find all blogs that were published in the past year

Searching multiple fields

- How would you query multiple fields at once?
 - For example:

find blogs that mention "Agent" in the title or content fields
- Use the **multi_match** query:

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

Multi_match and scoring

- By default, the best scoring field will determine the score
 - set **type** to **most_fields** to let the score be the sum of the scores of the individual fields instead:

```
GET blogs/_search
{
  "query": {
    "multi_match": {
      "type": "most_fields",
      "query": "agent",
      "fields": [
        "title",
        "content"
      ]
    }
  }
}
```

The more fields contain the word **agent**, the higher the score

Multi_match and phrases

- You can search for phrases with the multi_match query
 - set **type** to **phrase**:

```
GET blogs/_search
{
  "query": {
    "multi_match": {
      "type": "phrase",
      "query": "elastic agent",
      "fields": [
        "title",
        "content"
      ]
    }
  }
}
```

Find blogs that contain the **phrase** "elastic agent" in the title or content fields

Combining queries

- Suppose you want to write the following query:
 - ***find blogs about “Agent” written in English***
- This search is actually a combination of two queries:
 - “Agent” needs to be in the **content** or **title** field
 - and “en-us” in the **category** field
- How can you combine these two queries?
 - by using Boolean logic and the **bool** query...

The `bool` query

- The `bool` query combines one or more boolean clauses:
 - **must**
 - **filter**
 - **must_not**
 - **should**
- Each of the clauses is optional
- Clauses can be combined
- Any clause accepts one or more queries

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

The must clause

- Any query in a **must** clause must match for a document to be a hit
- Every query contributes to the score

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "content": "agent"
          }
        },
        {
          "match": {
            "locale": "en-us"
          }
        }
      ]
    }
  }
}
```

This **must** clause has **two** match queries

The filter clause

- Filters are like **must** clauses: any query in a **filter** clause has to match for a document to be a hit
- But, queries in a **filter** clause do not contribute to the score

Filters are great for **yes/no** type queries. In this example, a blog is either written in English or it is not. That should not influence the score.

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "content": "agent"
          }
        }
      ],
      "filter": [
        {
          "match": {
            "locale": "en-us"
          }
        }
      ]
    }
  }
}
```

The must_not clause

- Use **must_not** to exclude documents that match a query
- Queries in a **must_not** clause do not contribute to the score

This query finds all blogs that mention **agent** in the **content** field that are **not** written in English

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "content": "agent"
          }
        }
      ],
      "must_not": [
        {
          "match": {
            "locale": "en-us"
          }
        }
      ]
    }
  }
}
```

The should clause

- Use **should** to boost documents that match a query
- Queries in a **should** clause contribute to the score
- But, documents that do not match the queries in a **should** clause are returned as hits too

This query finds all blogs that mention **agent** in the **content** field. Documents that are **written in English** will get a higher score

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "content": "agent"
          }
        }
      ],
      "should": [
        {
          "match": {
            "locale": "en-us"
          }
        }
      ]
    }
  }
}
```

Query vs filter context

- Use filters as much as possible
- Skipping score calculation is **faster**
- Frequently used filters can be **cached**

If you frequently filter for a specific time range,
that filter can be **cached**

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "content": "agent"
          }
        }
      ],
      "filter": [
        {
          "range": {
            "publish_date": {
              "gt": "2020"
            }
          }
        }
      ]
    }
  }
}
```

Query vs filter context

Query context

```
"match": {"title": "community"}
```

How well does
the document
match this
query?

Filter context

```
"bool": {  
  "filter": [  
    { "match": {"title": "community"} }  
  ]  
}
```

Does the
document match
this query?

```
"hits" : {  
  "total" : {  
    "value" : 28,  
    "relation" : "eq"  
  },  
  "max_score" : 6.1514335,  
  "hits" : [
```

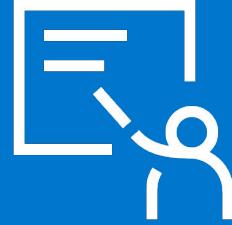
The query
contributes to
ranking

```
"hits" : {  
  "total" : {  
    "value" : 28,  
    "relation" : "eq"  
  },  
  "max_score" : 0.0,  
  "hits" : [
```

The query *does not*
contribute to ranking

Summary: More queries

Module 3 Lesson 2



Summary

- The **Query DSL** offers many types of queries
- Use the **range** query to query for numbers, dates and IPs
- The **multi_match** and **bool** queries can query multiple fields at once
- Queries in a **query context** contributes to ranking while queries in a **filter context** do not

Quiz

1. **True or False:** The Query DSL can only be used for full text queries.
2. **True or False:** If a document matches a filter clause, the score is increased by a factor of 2.
3. Would the following query be best searched in a query context or filter context?

```
"match" : { "locale": "en-us" }
```

More queries

Lab 3.2

Become comfortable writing queries
using the Elasticsearch Query DSL



Developing search applications

Module 3 Lesson 3



Highlighting

Highlighting results

- It's common to highlight the matched terms in search results

elastic certified

All (4,787) Documentation (4,647) Blog (92) Webinar (7) General (41)

SORT BY Relevance ▾

Showing results 1 - 20 for: "elastic certified"

Preparing for the Elastic Certified Observability Engineer Exam | Elastic Videos WEBINAR

<https://www.elastic.co/webinars/preparing-for-the-observability-engineer-exam>
On-demand webinar Preparing for the **Elastic Certified** Observability Engineer Exam See when this webinar starts in my time zone Hosted by Rich Raposa Sr. Certification Manager **Elastic** Highlights **Elastic Certified** Observability Engineers have a clear view of what's happening across their entire



Elastic Certification FAQ | Elastic Training

<https://www.elastic.co/training/certification/faq>
prepare for each exam. Each webinar shows what the exam environment and tasks look like, answers questions about the exam, and in general, gets you comfortable with our testing system so you can stay focused: Preparing for the **Elastic Certified** Engineer exam
Preparing for the **Elastic Certified**

The highlight clause

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

request

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

Highlight **kibana** in
the title

response

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

Highlight wraps ****
**** tags around terms

Changing the tags

- Use `pre_tags` and `post_tags` to change the tags

request

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

response

```
"highlight": {
  "title": [
    "<es-hit>Kibana</es-hit> 5.6.1 released"
  ]
}
```

Search templates

Use case for search templates

- The search bar in your application uses a complex search request
OR
- You have several applications sending the same complex search request to one Elasticsearch cluster

Q community team|

request

```
GET my_blogs/_search/template
{
  "id": "search_bar",
  "params": {
    "blog_query": "community team"
  }
}
```

Store a search template

- Use **search templates** to pre-render search requests
 - to store a search template, use the `_scripts` endpoint

request

```
PUT _scripts/my_search_template
{
  "script": {
    "lang": "mustache",
    "source": {
      "query": {
        "match": {
          "{{my_field}}": "{{my_value}}"
        }
      }
    }
  }
}
```

Name of the search template

The only option for search templates

Names of your parameters

Using a stored search template

- Use the `_search/template` endpoint to search with a stored template
 - passing in the necessary parameter values

request

```
GET blogs/_search/template
{
  "id": "my_search_template", → Name of the search template
  "params": {
    "my_field": "title",
    "my_value": "shay banon" → Define your parameters
  }
}
```

Defining defaults for parameters

request

- You can specify a default value for a parameter, in case it is not defined

```
PUT _script/my_search_template
{
  "script": {
    "lang": "mustache",
    "source": {
      "query": {
        "range": {
          "@timestamp": {
            "gte": "{{start}}",
            "lte": "{{end}} {{^end}} now{{/end}}"
          }
        }
      }
    }
  }
}
```

```
GET blogs/_search/template
{
  "id": "my_search_template",
  "params": {
    "start": "2021-01-01"
  }
}
```

is executed as

```
GET blogs/_search
{
  "query": {
    "range": {
      "@timestamp": {
        "gte": "2021-01-01",
        "lte": "now"
      }
    }
  }
}
```

If `end` is not specified
use default value

Using conditionals

- The syntax is: {{#var}} {{/var}}

```
PUT _scripts/my_search_template
{
  "script": {
    "lang": "mustache",
    "source":
    """
    { "query": { "bool": {
      "must": [ {
        "match": { "content": "{search_term}" } }
      {#search_date}}
      ,
      "filter": [ {
        "range": {
          "@timestamp": {
            "gte": "{start}",
            "lt": "{end}" } } }
      {/search_date}
      } } }
    """
  }
}
```

request

```
GET blogs/_search/template
{
  "id": "my_search_template",
  "params": {
    "search_term": "logstash"
  }
}
```

is rendered as

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

If only **search_term** is specified

Using conditionals

request

```
GET blogs/_search/template
{
  "id": "my_search_template",
  "params": {
    "search_term": "logstash",
    "search_date": {
      "start": "2021-01-01",
      "end": "2021-05-01"
    }
  }
}
```

is rendered as

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "content": "logstash" } }
      ],
      "filter": [
        {
          "range": {
            "@timestamp": {
              "gte": "2021-01-01",
              "lte": "2021-05-01"
            }
          }
        }
      ]
    }
  }
}
```

If both `search_term` and `search_date` are specified

Async search

Async search

- Search asynchronously
- Useful for slow queries and aggregations
 - monitor the progress
 - retrieve partial results as they become available

request

```
POST blogs/_async_search?wait_for_completion_timeout=0s
{
  "query": {
    "match": {
      "title": "community team"
    }
  }
}
```

The body is identical to a regular `_search` request

`wait_for_completion_timeout`
defaults to 1s

Async search response

response

```
{  
  "id" : "Fk0tWm1LM1hmVHA2bGNvMHF6alRhM3ccZWZ0Uk9NcFVUR3VDTzc3OENmYUcyQToyMDYyMQ==",  
  "is_partial" : true,  
  "is_running" : true,  
  "start_time_in_millis" : 1649075069466,  
  "expiration_time_in_millis" : 1649507069466,  
  "response" : {  
    ...  
    "hits" : {  
      "total" : {  
        "value" : 0,  
        "relation" : "gte"  
      },  
      "max_score" : null,  
      "hits" : [ ]  
    }  
  }  
}
```

The **id** can be used to retrieve the results later

is_partial indicates whether the current set of results are partial

is_running indicates whether the query is still running

You can retrieve the results until **expiration_time_in_millis** (defaults to 5 days)

Retrieve the results

- Use the **id** to retrieve search results
- The response will tell you whether
 - the query is still running (**is_running**)
 - the results are partial (**is_partial**)

request

```
GET/_async_search/Fk0tWm1LM1hmVHA2bGNvMHF6alRhM3ccZWZ0Uk9NcFVUR3VDT..  
.
```

Summary: Developing search applications

Module 3 Lesson 3



Summary

- **Search templates** enable you to decouple application code from query logic
- Search templates can be stored using the **_scripts** endpoint
- Use the **_search/template** endpoint to search with a stored template
- Use **async search** for long running searches

Quiz

- **True or False:** To store a search template, use the `_template` endpoint.
- In what templating language are search templates written?
- How long do async searches wait for results by default, before starting to return hits?

Developing search applications

Lab 3.3



Search with search templates

Run async searches

More resources

- Query DSL
 - www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html
 - www.elastic.co/guide/en/elasticsearch/reference/current/search-search.html
 - www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html
- Async search
 - www.elastic.co/guide/en/elasticsearch/reference/current/async-search.html
- Search templates
 - www.elastic.co/guide/en/elasticsearch/reference/current/search-template.html

Elasticsearch Engineer: Agenda

- Module 1: Getting started
- Module 2: Data modeling
- Module 3: You know, for search
- **Module 4: Data processing**
- Module 5: Aggregations
- Module 6: The one about shards
- Module 7: Data management
- Module 8: Cluster management

Data Processing

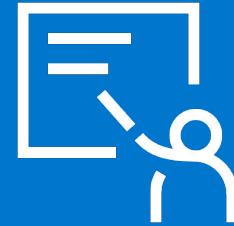
Module 4

Topics

- Changing Data
- Enriching Data
- Painless Scripting and Runtime Fields

Changing Data

Module 4 Lesson 1



Processors

Processors

- **Processors** can be used to transform documents before being indexed into Elasticsearch
- There are different ways to deploy processors:
 - Beats processors
 - Logstash
 - Ingest node pipelines

```
{  
    "content": "This  
blog...",  
    "locales":  
    "de-de,fr-fr",  
    ...  
}
```

Processors

```
{  
    "content": "This blog...",  
    "number_of_views": 123,  
    "locales":  
    ["de-de", "fr-fr"],  
    "content_length": 10389,  
    ...  
}
```

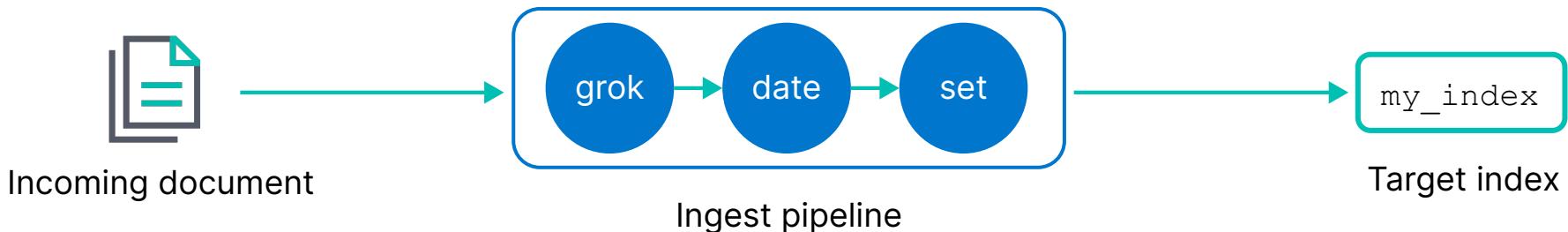
Processors

- Beats processors, Logstash filters, and ingest pipelines all have their own set of processors
 - several ***commonly used processors*** are in all three tools:

Manipulate fields	Manipulate values	Special operations
<ul style="list-style-type: none">setremoverenamedot_expander...	<ul style="list-style-type: none">split/joingrokdissectgsub...	<ul style="list-style-type: none">csv/jsongeoipuser_agentscriptpipeline...

Ingest node pipelines

- ***Ingest node pipelines***
 - perform common transformations on your data before indexing
 - consist of a series of ***processors***
 - are executed on ***ingest*** nodes



Create pipelines

- Use Kibana ***Ingest Pipelines UI*** to create and manage pipelines
 - view a list of your pipelines and drill down into details
 - edit or clone existing pipelines
 - delete pipelines

The screenshot shows the 'Create pipeline' interface in the Kibana Ingest Pipelines UI. At the top, there's a breadcrumb navigation: Stack Management > Ingest Pipelines > Create pipeline. To the right of the breadcrumb is a link to 'Create pipeline docs'. Below the header, there are two main sections: 'Name' and 'Description'. The 'Name' section includes a text input field and an optional checkbox for 'Add version number'. The 'Description' section includes a text input field and an optional note about it being 'optional'. Below these sections is a heading 'Processors' followed by a large button labeled 'Add your first processor'. Underneath this button, there's a sub-instruction: 'Use processors to transform data before indexing.' with a 'Learn more' link. At the bottom of the page are three buttons: a green 'Create pipeline' button with a checkmark icon, a 'Cancel' button, and a 'Show request' link.

Example of a pipeline

- Add a field to a document using the `set` processor

Configure processor X

[Set documentation](#)

Processor

Set

Sets the value of a field.

Field

number_of_views

Field to insert or update.

Value

0

Value for the field.

Override

If enabled, overwrite existing field values. If disabled, only update `null` fields.

Ignore empty value

If `value` is `null` or an empty string, do not update the field.

Adds `number_of_views` if the field does not already exist

If field already exists overwrites value to 0 (default)

Test the pipeline

- Before you save the pipeline, make sure it works

Test pipeline: [Add documents](#)

- You can write the `_source` of any document or explicitly select one that is in an index

Test pipeline

Documents Output

Provide documents for the pipeline to ingest. [Learn more.](#)

▼ Add a test document from an index

Provide the document's index and document ID. To explore your existing data, use [Discover](#).

Index

blogs

Document ID

yDwlsX8B5XZxY9n85ytr

Add document

✓ Document added

Documents

Clear all

```
{  
  "_id": "yDwlsX8B5XZxY9n85ytr",  
  "_index": "blogs",  
  "_source": {  
    "content": "<h2>About RTE</h2><p>At the core of the power sys  
    "url": "/blog/elasticsearch-at-rte-blackout-prevention-throug  
    "category": [  
      | "blt26ff0a1ade01f60d"  
    ],  
  },  
}
```

Test the pipeline

- Alternatively use APIs

request

```
POST GET _ingest/pipeline/test/_simulate
{
  "docs": [
    {
      "_source": {
        "url": "/blog/elasticsearch-at-rte-blackout-prevention-through-weather-prediction",
        "publish_date": "2017-11-27T17:34:25.000Z"
      }
    }
  ]
}
```

response

```
"docs" : [
  {
    "doc" : {
      "_index" : "_index",
      "_id" : "_id",
      "_source" : {
        "number_of_views" : "0",
        "publish_date" :
        "2017-11-27T17:34:25.000Z",
        "url" :
        "/blog/elasticsearch-at-rte-blackout-prevention-through-weather-prediction"
      },
      "_ingest" : {
        "timestamp" :
        "2022-03-23T10:51:05.764937697Z"
      }
    }
  ]
}
```

Use the pipeline

Apply a pipeline to documents in indexing requests

```
POST my_index/_update_by_query?pipeline=my_pipeline
```

Set a default pipeline

```
PUT my_index
{
  "settings": {
    "default_pipeline": "my_pipeline"
  }
}
```

Set a pipeline with Update by Query or Reindex APIs

```
POST _reindex
{
  "source": {
    "index": "my_index"
  },
  "dest": {
    "index": "new_index",
    "pipeline": "my_pipeline"
  }
}
```

Dissect processor

- The **dissect** processor extracts structured fields out of a single text field within a document

```
1.2.3.4 [30/Apr/1998:22:00:52 +0000] \"GET /english/images/montpellier/18.gif\"
```



```
%{clientip} [%{@timestamp}] \"%{verb} %{request}\"
```



```
"_source": {  
    "request": "/english/images/montpellier/18.gif",  
    "verb": "GET",  
    "@timestamp": "30/Apr/1998:22:00:52 +0000",  
    "clientip": "1.2.3.4"  
}
```

Pipeline processor

- Create a pipeline that ***references other pipelines***
 - can be used with conditional statements

```
PUT _ingest/pipeline/blogs_pipeline
{
  "processors" : [
    {
      "pipeline" : { "name": "inner_pipeline" }
    },
    {
      "set" : {
        "field": "outer_pipeline_set",
        "value": "outer_value",
      }
    }
  ]
}
```

Updating Documents

Changing data

- You can change index settings and modify the `_source` using various Elasticsearch APIs:
 - `_reindex`
 - `_update_by_query`
- You have already seen some of these APIs in the labs

The Reindex API

- The **Reindex API** indexes source documents into a destination index
 - source and destination indices must be different
- To reindex only a subset of the source index:
 - use **max_docs**
 - and/or add a **query**

```
POST _reindex
{
  "max_docs": 100,
  "source": {
    "index": "blogs",
    "query": {
      "match": {
        "category": "Engineering"
      }
    },
    "dest": {
      "index": "blogs_fixed"
    }
}
```

Reindex from a remote cluster

- Remote hosts have to be explicitly allowed in `elasticsearch.yml` using the `reindex.remote.whitelist` property

```
POST _reindex
{
  "source": {
    "remote": {
      "host": "http://otherhost:9200",
      "username": "user",
      "password": "pass"
    },
    "index": "remote_index",
  },
  "dest": {
    "index": "local_index"
  }
}
```

Update by Query

- To change all the documents in an existing index use the ***Update by Query API***
 - reindexes every document into the same index
 - update by Query has many of the same features as Reindex

request

```
POST blogs/_update_by_query
{
  "query": {
    "match": { "category" : "Engineering" }
  }
}
```

The Delete by Query API

- Use the ***Delete by Query API*** to delete documents that match a specified query
 - deletes every document in the index that is a hit for the query

request

```
POST blogs_fixed/_delete_by_query
{
  "query": {
    "match": {
      "author.full_name.keyword": "Clinton Gormley"
    }
  }
}
```

Summary: Changing Data

Module 4 Lesson 1



Summary

- You can copy documents from one index to another using the ***Reindex*** API
- The ***Update By Query*** API and the ***Delete by Query*** API will update or delete a collection of documents
- ***Ingest pipelines*** use Elasticsearch Ingest APIs to perform common transforms to your data before indexing
- Use Kibana ***Ingest Pipelines UI*** to manage your pipelines
- Pipelines can also be defined using Beats or Logstash

Quiz

1. **True or False:** In a *Reindex* request, you can add a query clause to only reindex a subset of the documents.
2. **True or False:** The processors of an ingest pipeline are executed in parallel.
3. What does the following configuration do?

```
PUT blogs_fixed
{
  "settings": {
    "default_pipeline": "blogs_pipeline"
  }
}
```

Changing Data

Lab 4.1

Reindex documents

Define ingest pipeline



Enriching Data

Module 4 Lesson 2



Enrichment use cases

- Add zip codes based on geo location
- Enrichment based on IP range
- Currency conversion
- Denormalizing data
- Threat Intel Enrichment

Denormalize your data

- At its heart, Elasticsearch is a flat hierarchy and trying to force relational data into it can be very challenging
- Documents should be modeled so that search-time operations are as cheap as possible
- Denormalization gives you the most power and flexibility
 - Optimize for reads
 - No need to perform expensive joins

Denormalize your data

- Denormalizing your data refers to “**flattening**” your data
 - storing redundant copies of data in each document instead of using some type of relationship
- There are various ways to denormalize your data
 - Outside Elasticsearch
 - Write your own application-side join
 - Logstash filters (eg, JDBC driver)
 - Inside Elasticsearch
 - Enrich processor in ingest node pipelines

Denormalize blogs

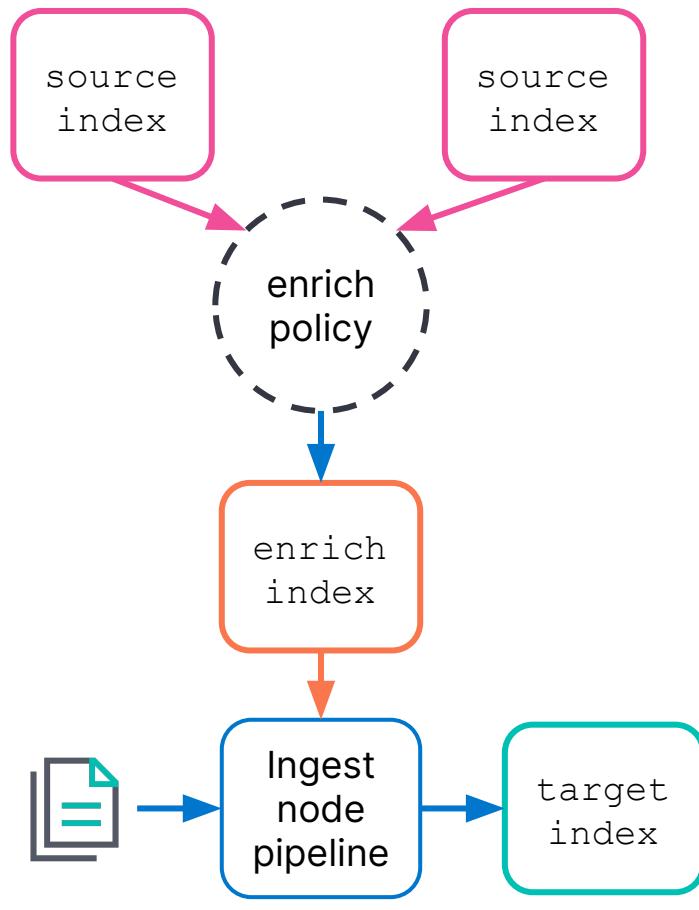
- In the **blogs** index, there is a **category** field with ID values
- There is an associated **categories** index with **uid** and **title** fields

blogs		categories	
title	category	uid	title
How finding time for cooking and culture helps forge better connections at work	bltc253e0851420b088	blt26ff0a1ade01f60d	User Stories
Querying a petabyte of cloud storage in 10 minutes	blt1d90b8e0edce3ea9	bltfaae4466058cc7d6	Releases
Elastic for Education: How we support students and educators	blt26ff0a1ade01f60d	bltc253e0851420b088	Culture
		blt0c9f31df4f2a7a2b	News
		blt1d90b8e0edce3ea9	Engineering

The Enrich Processor

Enrich your data

- Use the `enrich` processor to add data from your existing indices to incoming documents during ingest
- There are several steps to enriching your data
 - Step 1: Set up an enrich policy
 - Step 2: Create an enrich index for the policy
 - Step 3: Create an ingest pipeline with an enrich processor
 - Step 4: Set up your index to use the pipeline



Step 1: Set up an enrich policy

Once created, you can't update or change an enrich policy

```
PUT _enrich/policy/cat_policy → policy name  
{  
  "match": { → policy type can be match or geo_match  
  "indices": "categories", → one or more source indices  
  "match_field": "uid", → match field from the source indices  
  "enrich_fields": ["title"] → enrich fields from the source indices to add to incoming docs  
}
```

Step 2: Create an enrich index for the policy

- Execute the enrich policy to create the enrich index for your policy

```
POST _enrich/policy/cat_policy/_execute
```

- An enrich policy uses enrich data from the policy's source indices to create a streamlined system index called the enrich index
 - the processor uses this index to match and enrich incoming documents

Step 3: Create ingest pipeline with enrich processor

```
PUT /_ingest/pipeline/categories_pipeline
{
  "processors" : [
    {
      "enrich" : {
        "policy_name": "cat_policy",
        "field" : "category.uid", →
        "target_field": "cat_title"
      }
    },
    ...
  ]
}
```

the field in the input document
that matches the policy's
`match_field`

Set `max_matches > 1` if the field in
the input document is an array

Step 4: Use the pipeline

- Finally update each document with the enriched data

```
POST blogs_fixed/_update_by_query?pipeline=categories_pipeline
```

**Set the pipeline as a default_pipeline
if you want to enrich incoming
documents**

Summary: **Enriching Data**

Module 4 Lesson 2



Summary

- Elasticsearch is not designed to work with relational data
- Denormalize your data whenever possible
- Use the `enrich` processor in an ingest node pipeline to denormalize data
 - Set up and execute an enrich policy first

Quiz

1. **True or False:** Denormalizing your data enables you to quickly and easily search across multiple fields in a single query without joining datasets.
2. How do you create an enrich index?
3. **True or False:** You can update the enrich index.

Enriching Data

Lab 4.2

Enrich an index



Painless Scripting and Runtime Fields

Module 4 Lesson 3



Scripting

- Wherever scripting is supported in the Elasticsearch APIs, the syntax follows the same pattern
- Elasticsearch compiles new scripts and stores the compiled version in a cache

```
"script": {  
    "lang": "...",  
    "source": "...",  
    "params": { ... }  
}
```

Scripts may be **inline or stored**

Use **"source"** for inline script or
"id" for stored script

Pass **"params"** instead of
hard-coding values

Painless scripting

- **Painless** is a performant, secure scripting language designed specifically for Elasticsearch
- Painless is the default language
 - you don't need to specify the language if you're writing a Painless script
- Use Painless to
 - create Kibana scripted fields
 - process reindexed data
 - create runtime fields which are evaluated at query time

Example of a Painless script

- Painless has a Java-like syntax (and can contain actual Java code)
- Fields of a document can be accessed using a **Map** named **doc**

```
GET blogs/_search
{
  "script_fields": {
    "second_part_of_url": {
      "script": {
        "source": "doc['url'].value.splitOnToken('/') [2]"
      }
    }
  }
}
```

Painless Lab

- The **Painless Lab** is an interactive code editor that lets you test and debug Painless scripts in real-time

The screenshot shows the Painless Lab interface. At the top, there's a navigation bar with 'Dev Tools' and 'Painless Lab'. Below it, a tab bar includes 'Console', 'Search Profiler', 'Grok Debugger', and 'Painless Lab' (which is currently selected and has a 'BETA' badge). The main area contains two panes: a code editor on the left and an output panel on the right.

Code Editor Content:

```
1 boolean isInCircle(def posX, def posY, def circleX, def circleY, def radius) {  
2     double distanceFromCircleCenter = Math.sqrt(Math.pow(circleX - posX, 2) + Math.pow(circleY  
    - posY, 2));  
3     return distanceFromCircleCenter <= radius;  
4 }  
5  
6 boolean isOnCircle(def posX, def posY, def circleX, def circleY, def radius, def thickness,  
    def squashY) {  
7     double distanceFromCircleCenter = Math.sqrt(Math.pow(circleX - posX, 2) + Math.pow(  
        (circleY - posY) / squashY, 2));  
8     return (  
9         distanceFromCircleCenter >= radius - thickness  
10        && distanceFromCircleCenter <= radius + thickness  
11    );  
12 }
```

Output Panel Content:

The output panel displays a grid of asterisks ('*') and dots ('.') representing the execution results of the script. The grid is approximately 10 columns wide and 10 rows high, with some patterns appearing in the middle.

Runtime Fields

"Schema on read" with runtime fields

- Ideally, your schema is defined at index time ("schema on write")
- However, there are situations, where you may want to define a schema on read:
 - to fix errors in your data
 - to structure or parse your data
 - to change the way Elasticsearch returns data
 - to add new fields to your documents

... without having to reindex your data

Creating a runtime field

- Configure:
 - a name for the field
 - a type
 - a custom label (optional)
 - a value (optional)
 - a format (optional)

Create field
Index pattern: *kibana_sample_data_ecommerce*

Name Type

Set custom label
Create a label to display in place of the field name in Discover, Maps, and Visualize. Useful for shortening a long field name. Queries and filters use the original field name.

Set value
Set a value for the field instead of retrieving it from the field with the same name in `_source`.

Set format
Set your preferred format for displaying the value. Changing the format can affect the value and prevent highlighting in Discover.

[Show advanced settings](#)

[Cancel](#) [Save](#)

Runtime field and Painless tips

- Avoid runtime fields if you can
 - They are computationally expensive
 - Fix data at ingest time instead
- Avoid errors by checking for **null** values
- Use the **Preview** pane to validate your script

Edit field 'my_field'

Index pattern: kibana_sample_data_ecommerce

Name	Type
my_field	Keyword

Set custom label
Create a label to display in place of the field name in Discover, Maps, and Visualize. Useful for shortening a long field name. Queries and filters use the original field name.

Set value
Set a value for the field instead of retrieving it from the field with the same name in `_source`.

Define script
`1 emit("Hello world");`

[Cancel](#) [Save](#)

Preview

From: kibana_sample_data_ecommerce

Document ID: dRc8Ln0BJ0fqGiUjNVg4

Filter fields

my_field	Hello world
category	[{"Men's Clothing"]
currency	EUR
customer_first_name	Eddie
customer_full_name	Eddie Underwood
customer_gender	MALE
customer_id	38
customer_last_name	Underwood

Show more

Mapping a runtime field

- `runtime` section defines the field *in the mapping*
- Use a Painless script to `emit` a field of a given type

```
PUT blogs/_mapping
{
  "runtime": {
    "day_of_week": {
      "type": "keyword",
      "script": {
        "source": "emit(doc['publish_date'].value.dayOfWeekEnum.getDisplayName(TextStyle.FULL, Locale.ROOT))"
      }
    }
  }
}
```

Searching a runtime field

- You access runtime fields from the search API like any other field
 - Elasticsearch sees runtime fields no differently

```
GET blogs/_search
{
  "query": {
    "match": {
      "day_of_week": "Monday"
    }
  }
}
```

Runtime fields in a search request

- `runtime_mappings` section defines the field *at query time*
- Use a Painless script to `emit` a field of a given type

```
GET blogs/_search
{
  "runtime_mappings": {
    "day_of_week": {
      "type": "keyword",
      "script": {
        "source": "emit(doc['@timestamp'].value.dayOfWeekEnum.getDisplayName(TextStyle.FULL, Locale.ROOT))"
      }
    }
  },
  "aggs": {
    "my_agg": {
      "terms": {
        "field": "day_of_week"
      }
    }
  }
}
```

Summary: Painless Scripting and Runtime Fields

Module 4 Lesson 3



Create scripts and runtime fields

Summary

- **Painless** is the scripting language used in Elasticsearch
- Kibana provides a Painless lab to test your Painless scripts
- **Runtime fields** allow for creating arbitrary non-indexed data fields
 - runtime fields are evaluated at query time

Quiz

- Which scripting languages are safe to use in Elasticsearch?
- **True or False:** A runtime field can be added to a mapping dynamically.
- **True or False:** You need to first define runtime fields in the mappings before using it in queries.

Painless Scripting and Runtime Fields

Lab 4.3



Elasticsearch Engineer: Agenda

- Module 1: Getting started
- Module 2: Data modeling
- Module 3: You know, for search
- Module 4: Data processing
- **Module 5: Aggregations**
- Module 6: The one about shards
- Module 7: Data management
- Module 8: Cluster management

Aggregations

Module 5

Topics

- Metrics and bucket Aggs
- Combining aggs
- More aggregations
- Transforming data

Metrics and bucket aggs

Module 5 Lesson 1

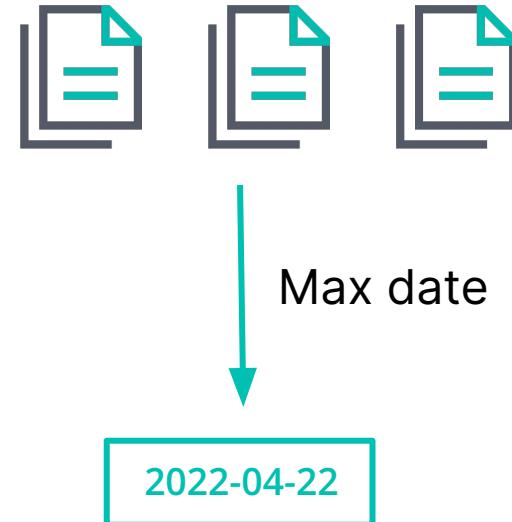


Aggregations

- Help you summarize your data
- Elasticsearch organizes aggregations into 3 categories
 - ***Metrics aggregations***
 - ***Bucket aggregations***
 - ***Pipeline aggregations***

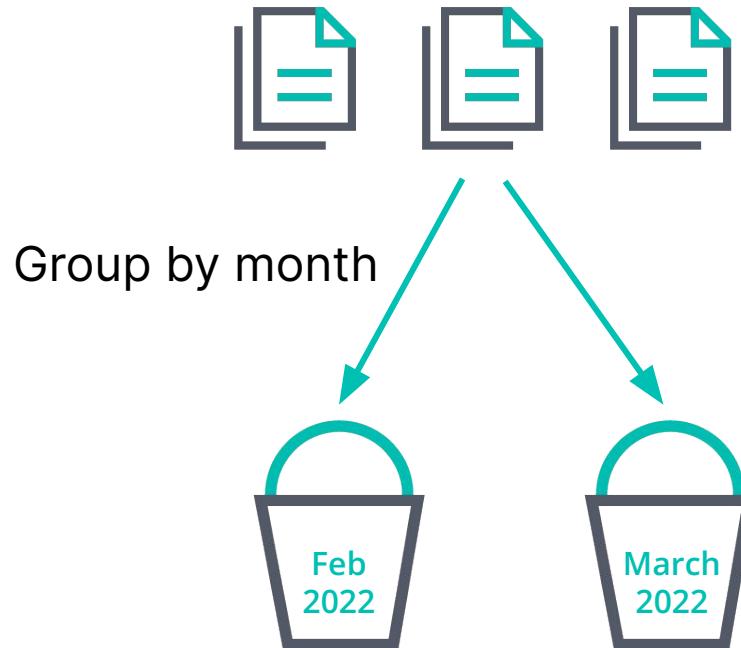
Metrics aggregations

- Compute values extracted from the documents
- Examples:
 - min/max/sum/avg
 - weighted_avg
 - stats
 - percentiles
 - geo_centroid
 - top_hits



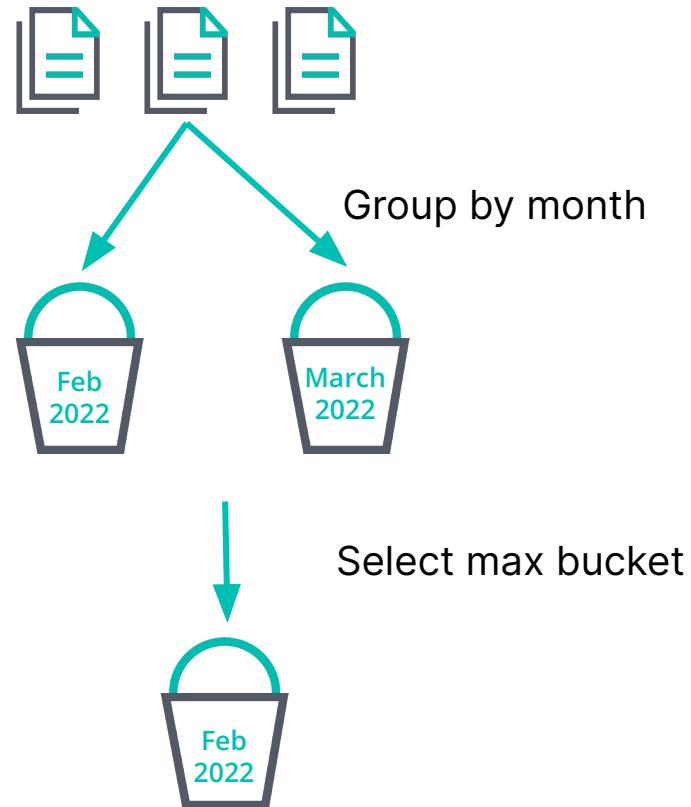
Bucket aggregations

- Group documents according to certain criterion
- Examples:
 - date_histogram
 - terms
 - filter
 - range
 - global



Pipeline aggregations

- Work on output produced from other aggregations
- Examples:
 - bucket min/max/sum/avg
 - cumulative_sum
 - moving_aggs
 - bucket_sort



Basic structure of aggregations

- An aggregation summarizes your data as metrics, statistics, or other analytics
- Run aggregations as part of a search request

```
GET blogs/_search
{
  "aggs": {
    "my_agg_name": {
      "AGG_TYPE": {
        ...
      }
    }
  }
}
```

The "aggregation" clause or
"aggs" for short

The results of this aggregation will
return in a field you name

There are many aggregation types

Aggregation results

- Aggregation results are in the response's "aggregations" object

request

```
GET blogs/_search
{
  "aggs": {
    "blogs_by_month": {
      "date_histogram": {
        "field": "publish_date",
        "calendar_interval": "month"
      }
    }
  }
}
```

response

```
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : { ... },
  "hits" : {
    ...
    "hits" : [ → Top 10 hits
      ...
      ]
    },
    "aggregations" : {
      "blogs_by_month" : {
        ...
      }
    }
}
```

even if you don't include "query" in the request

Return only aggregation results

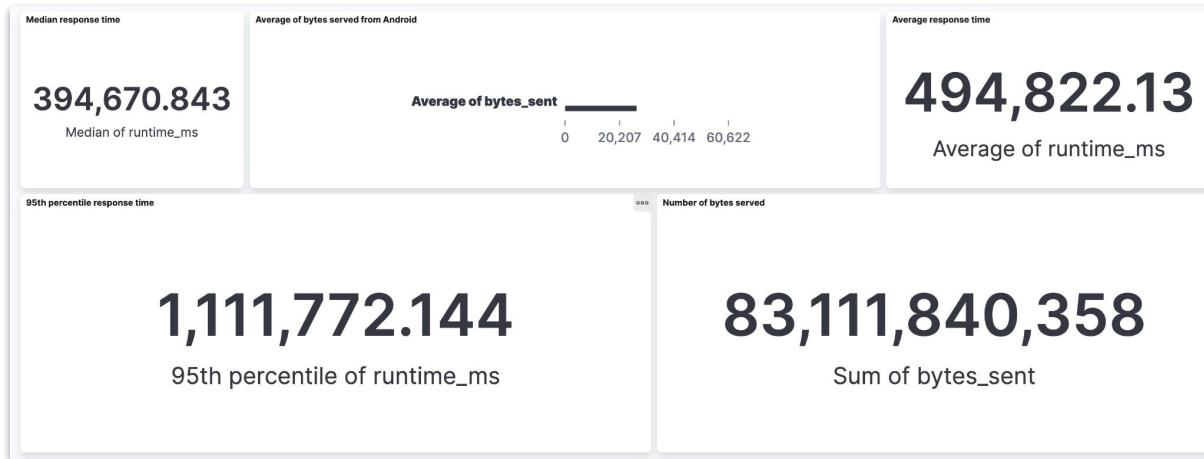
- To return only aggregation results, set "**size**" to 0
 - faster responses
 - smaller payload

Frequently used search requests with "size" set to 0 can be cached in the **shard request cache**

```
GET blogs/_search
{
  "track_total_hits": true,
  "size": 0,
  "aggs": {
    "blogs_by_month": {
      "date_histogram": {
        "field": "publish_date",
        "calendar_interval": "month"
      }
    }
  }
}
```

Metrics questions

- What is the number of bytes served from all blogs?
- What is the average of bytes served from Android devices?
- What is the average response time?
- What is the median response time?
- What is the 95 percentile?



Metrics aggregations

- Use metrics aggregations to compute numeric values extracted from your data

request

```
GET blogs/_search?size=0
{
  "aggs": {
    "no_of_authors": {
      "cardinality": {
        "field":
          "authors.full_name.keyword"
      }
    }
  }
}
```

response

```
"aggregations" : {
  "no_of_authors" : {
    "value" : 743
  }
}
```

Some metrics aggregations return a single value, some return many

Buckets questions

- How many requests reach our system every day?
- How many requests took between 0-200, 200-500, 500+ ms?
- What are the most viewed blogs on our website?
- Which are the 5 most popular blog categories?



Bucket aggregations

- Use bucket aggregations to group your data

request

```
GET blogs/_search?size=0
{
  "aggs": {
    "blogs_by_month": {
      "date_histogram": {
        "field": "publish_date",
        "calendar_interval": "month"
      }
    }
  }
}
```

Use `calendar_interval` or
`fixed_interval`

response

Bucket aggregation results return in a `buckets` array



```
"aggregations" : {
  "blogs_by_month" : {
    "buckets" : [
      {
        "key_as_string" : "2010-02...",
        "key" : 1264982400000,
        "doc_count" : 4
      },
      {
        "key_as_string" : "2010-03...",
        "key" : 1267401600000,
        "doc_count" : 1
      },
      ...
    ]
  }
}
```

Each bucket has a `doc_count`

Summary: Metrics and bucket aggs

Module 5 Lesson 1



Summary

- Aggregations are summaries of data
- Aggregations are part of a search request
- Set "**size**" to 0 for faster results
- ***Bucket aggregations*** group your data in buckets
- ***Metrics aggregations*** compute values from your data
- ***Pipeline aggregations*** work on the output of another aggregation

Quiz

1. Is `date_histogram` a **bucket**, **metrics**, or **pipeline** aggregation?
2. **True** or **False**: You can choose arbitrary names to identify aggregation clauses
3. Which aggregation would you use to put logging events into buckets by log level (“error”, “warn”, “info”, etc.)?

Metrics and bucket aggs

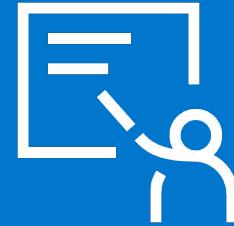
Lab 5.1



Write both metric and bucket aggregations using the Query DSL.

Combining aggs

Module 5 Lesson 2



Reducing the scope of an aggregation

- By default, aggregations are performed on all documents in the index
- Combine with a query to reduce the scope

Get the number of authors
for french blogs

```
GET blogs/_search?size=0
{
  "query": {
    "term": {
      "locale.keyword": {
        "value": "fr-fr"
      }
    }
  },
  "aggs": {
    "no_of_authors": {
      "cardinality": {
        "field": "authors.full_name.keyword"
      }
    }
  }
}
```

Run multiple aggregations

- You can specify multiple aggregations in the same request

request

```
GET blogs/_search?size=0
{
  "aggs": {
    "no_of_authors": {
      "cardinality": {
        "field": "authors.full_name.keyword"
      },
      "first_name_stats": {
        "string_stats": {
          "field": "authors.first_name.keyword"
        }
      }
    }
  }
}
```

response

```
"aggregations" : {
  "first_name_stats" : {
    "count" : 6188,
    "min_length" : 2,
    "max_length" : 41,
    "avg_length" : 5.479476...,
    "entropy" : 4.802759740211148
  },
  "no_of_authors" : {
    "value" : 743
  }
}
```

Run sub-aggregations

- Bucket aggregations support bucket or metric sub-aggregations

request

```
GET blogs/_search?size=0
{
  "aggs": {
    "blogs_by_month": {
      "date_histogram": {
        "field": "publish_date",
        "calendar_interval": "month" },
      "aggs": {
        "no_of_authors": {
          "cardinality": {
            "field": "authors.full_name.keyword" }
      } } } } }
```

response

```
"aggregations" : {
  "blogs_by_month" : {
    "buckets" : [
      {
        "key_as_string" : "2010-02...",
        "key" : 1264982400000,
        "doc_count" : 4,
        "no_of_authors" : {"value" : 2}
      },
      {
        "key_as_string" : "2010-03...",
        "key" : 1267401600000,
        "doc_count" : 1,
        "no_of_authors" : {"value" : 2}
      },
      ...
    ] } }
```

Combined aggregation questions

- What is the sum of bytes per day?
- What is the number of bytes served daily and median bytes size?
- What is the number of bytes served monthly per OS?

Calculating a single metric per bucket

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

request

```
GET kibana_sample_data_logs/_search
{
  "size": 0,
  "aggs": {
    "requests_per_day": {
      "date_histogram": {
        "field": "@timestamp",
        "calendar_interval": "day"
      },
      "aggs": {
        "daily_number_of_bytes": {
          "sum": {
            "field": "bytes"
          }
        }
      }
    }
  }
}
```

response

```
" aggregations" : {
  "requests_per_day" : {
    "buckets" : [
      {
        "key_as_string" :
"2022-01-09T00:00:00.000Z",
        "key" : 1641686400000,
        "doc_count" : 249,
        "daily_number_of_bytes" : {
          "value" : 1531493.0
        }
      }
    ]
  }
}
```

What is the sum of bytes per bucket?

Calculating multiple metrics per bucket

- You can calculate multiple metrics on bucket aggregations

request

```
GET kibana_sample_data_logs/_search
{
  "size": 0,
  "aggs": {
    "requests_per_day": {
      "date_histogram": {
        "field": "@timestamp",
        "calendar_interval": "day"
      },
      "aggs": {
        "daily_number_of_bytes": {
          "sum": {
            "field": "bytes"
          }
        },
        "median_bytes": {
          "percentiles": {
            "field": "bytes",
            "percents": [50]
          }
        }
      }
    }
  }
}
```

response

```
"aggregations" : {
  "requests_per_day" : {
    "buckets" : [
      {
        "key_as_string" : "2022-01-09T00:00:00.000Z",
        "key" : 1641686400000,
        "doc_count" : 249,
        "daily_number_of_bytes" : {
          "value" : 1531493.0
        },
        "median_runtime" : {
          "values" : {
            "50.0" : 6193.0
          }
        }
      }
    ]
  }
}
```

What is the sum of bytes served daily and median bytes size?

Sorting by a metric

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

```
"aggs": {  
    "requests_per_day": {  
        "date_histogram": {  
            "field": "@timestamp",  
            "calendar_interval": "day",  
            "order": {  
                "daily_number_of_bytes": "desc"  
            }  
        },  
        "aggs": {  
            "daily_number_of_bytes": {  
                "sum": {  
                    "field": "bytes"  
                }  
            },  
            "median_bytes" : {  
                "percentiles": {  
                    "field": "bytes",  
                    "percents": [50]  
                }  
            }  
        }  
    }  
}
```

“Find the day with the maximum bytes served.”

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

Sub-buckets

- What question is answered by the following terms aggregation nested inside a date_histogram aggregation?

```
GET kibana_sample_data_logs/_search
{
  "size": 0,
  "aggs": {
    "logs_by_month": {
      "date_histogram": {
        "field": "@timestamp",
        "calendar_interval": "month"
      },
      "aggs": {
        "machine_os": {
          "terms": {
            "field": "machine.os.keyword",
            "size": 10
          }
        }
      }
    }
  }
}
```

The “terms” agg is
sub-bucketed inside the
“date_histogram”

Sub-buckets

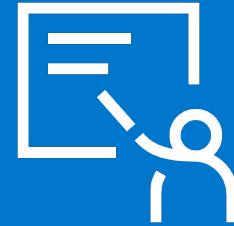
- The log events are bucketed **by month**, then
 - within each month the events are further bucketed **by machine.os**

What are the most popular operating systems, per month?

```
"aggregations" : {
  "logs_by_month" : {
    "buckets" : [
      {
        "key_as_string" :
"2022-01-01T00:00:00.000Z",
        "key" : 1640995200000,
        "doc_count" : 5316,
        "machine.os" : {
          "doc_count_error_upper_bound" : 0,
          "sum_other_doc_count" : 0,
          "buckets" : [
            {
              "key" : "win xp",
              "doc_count" : 1077
            },
            {
              "key" : "win 8",
              "doc_count" : 1075
            },
            {
              "key" : "win 7",
              "doc_count" : 1059
            }
          ]
        }
      }
    ]
  }
}
```

Summary: Combining aggs

Module 5 Lesson 2

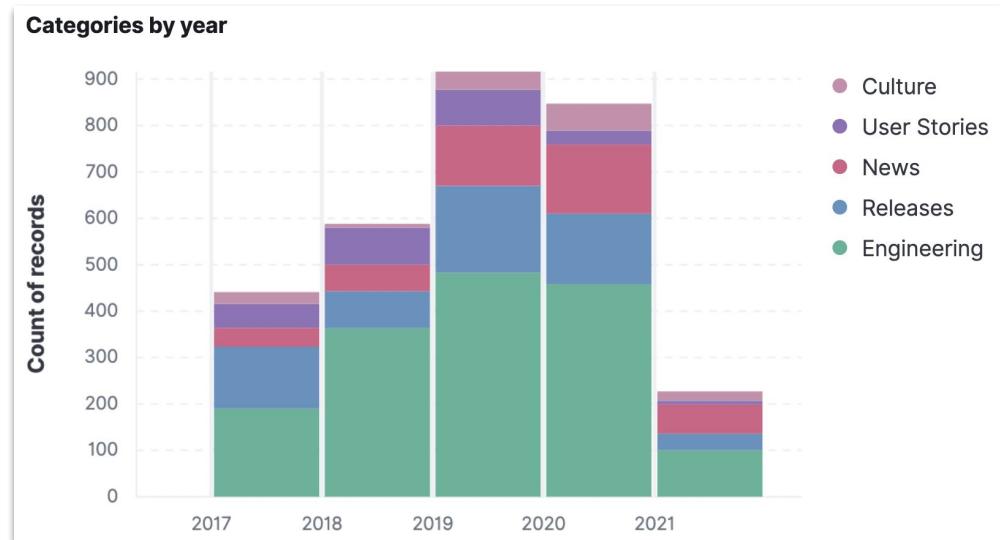


Summary

- You can combine an aggregation with a query to reduce the scope of the aggregation
- An aggregation can be a combination of metric and bucket aggregations

Quiz

1. Write an aggregation to answer the question: “Who wrote the most blogs?”
2. What aggregations would you use to build this visualization?



Combining aggs

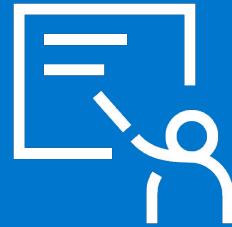
Lab 5.2

Combine multiple metric and bucket aggregations via Query DSL



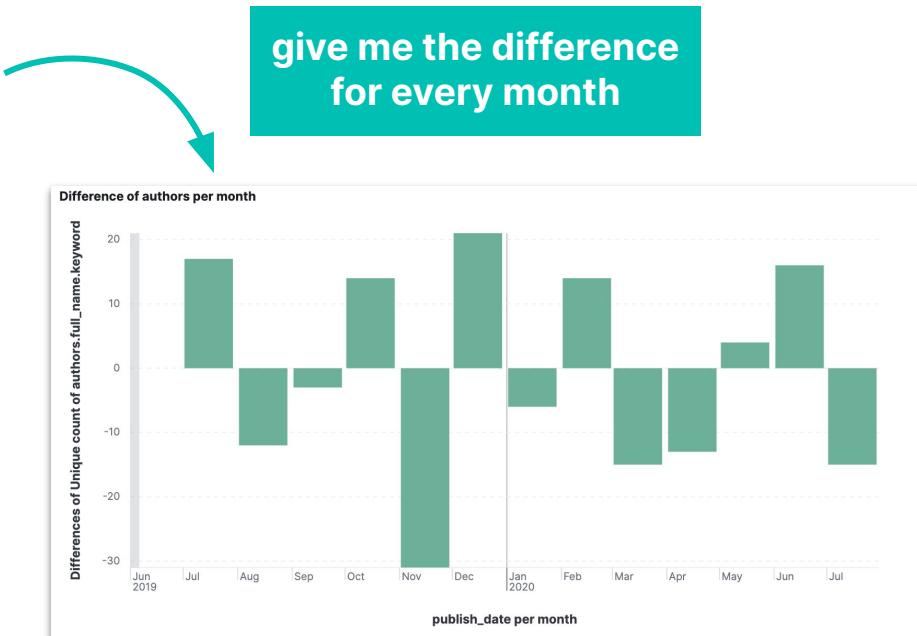
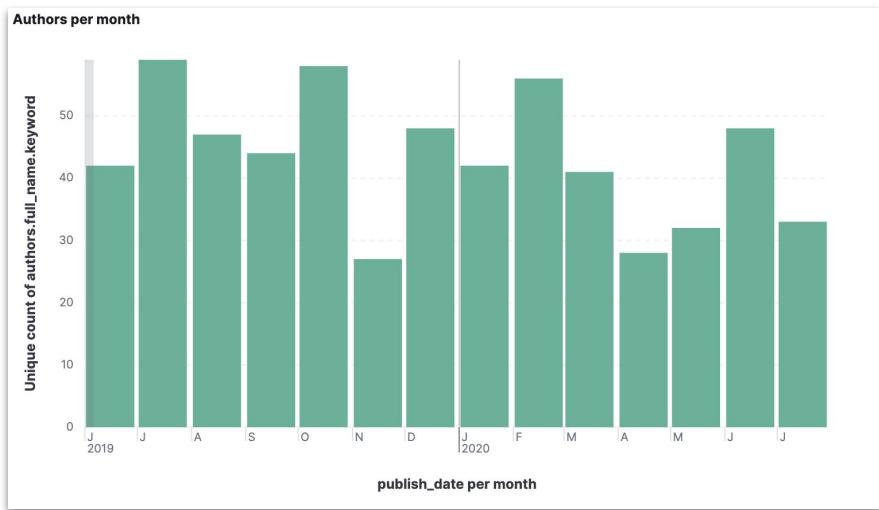
More aggregations

Module 5 Lesson 3



Pipeline aggregations

- Use pipeline aggregations to use output from another aggregation



Pipeline aggregations

request

```
"aggs": {  
  "blogs_by_month": {  
    "date_histogram": {  
      "field": "publish_date",  
      "calendar_interval": "month" },  
    "aggs": {  
      "no_of_authors": {  
        "cardinality": {  
          "field": "authors.full_name.keyword" } },  
      "diff_author_ct": {  
        "derivative": {  
          "buckets_path": "no_of_authors" } }  
    }  
  }  
}
```

response

```
"aggregations" : {  
  "blogs_by_month" : {  
    "buckets" : [  
      ...  
      { "key_as_string" : "2019-11...",  
        "key" : 1572566400000,  
        "doc_count" : 25,  
        "no_of_authors" : { "value" : 27 },  
        "diff_author_ct": { "value" : -31.0 },  
      },  
      { "key_as_string" : "2019-12...",  
        "key" : 1575158400000,  
        "doc_count" : 92,  
        "no_of_authors" : { "value" : 48 },  
        "diff_author_ct": { "value" : 21.0 },  
      },  
      ...  
    ] } }
```

$$48 - 27 = 21$$

top_hits aggregation

Motivation for top_hits aggregation

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

request

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

Alexander wrote 81 blogs about
Logstash filters, but which blogs are
the most relevant?

response

```
"buckets" : [
  {
    "key" : "Alexander Reelsen",
    "doc_count" : 81
  },
  {
    "key" : "Suyog Rao",
    "doc_count" : 80
  },
  {
    "key" : "Monica Sarbu",
    "doc_count" : 78
  },
  {
    "key" : "Dmitry Soshnikov",
    "doc_count" : 75
  }
]
```

Example of top_hits aggregation

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

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

The top_hits aggregation response

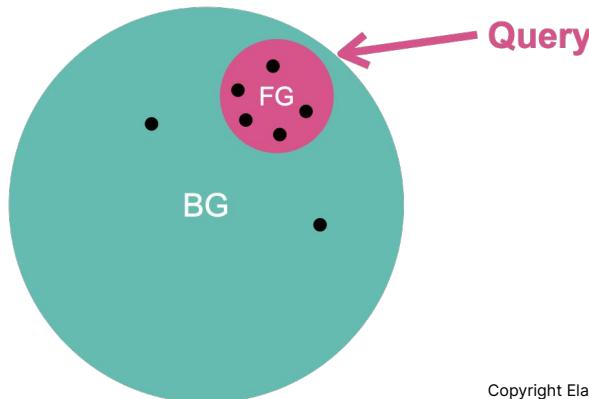
- Notice the top 5 hits from each bucket are returned in the “aggregations” clause of the response

```
"buckets": [
  {
    "key": "Alexander Reelsen",
    "doc_count": 81,
    "logstash_top_hits": {
      "hits": [
        ...
        "hits": [
          {
            "_index": "blogs",
            "_id": "wALTjn8BZCXjek5PSm6Y",
            "_score": 5.2558136,
            "_source": {
              "url": "/blog/2015-01-07-this-week-in-elasticsearch",
              "publish_date": "2015-01-07T20:45:06.000Z",
              "category": [],
              "locale": "en-us",
              "title": "This Week in Elasticsearch - January 07, 2015",
              ...
            }
          }
        ]
      }
    }
  }
]
```

Significant aggregations

The significant aggregations

- Terms aggregation with a noise filter
 - the terms aggregation returns **commonly common** terms
- Low frequency terms in the background data may pop out as high frequency terms in the foreground data
 - the significant aggregations find **uncommonly common** terms in your dataset



Some use cases:

- recommendation
 - fraud detection
 - defect detection
- and more...

Let's start with a terms aggregation

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

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

The terms aggregation response

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

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

```
"buckets" : [
  {
    "key" : "Monica Sarbu",
    "doc_count" : 145,
    "content_terms" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 49511,
      "buckets" : [
        {
          "key" : "a",
          "doc_count" : 145
        },
        {
          "key" : "to",
          "doc_count" : 145
        }
        {
          "key" : "the",
          "doc_count" : 142
        },
        ...
      ]
    }
  }
]
```

The terms aggregation response

- Try the same aggregation with **significant_text**:

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

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

The significant_text aggregation response

```
"buckets" : [  
    {  
        "key" : "brewing",  
        "doc_count" : 60,  
        "score" : 8.368918988781472,  
        "bg_count" : 92  
    },  
    {  
        "key" : "beat",  
        "doc_count" : 94,  
        "score" : 7.5808297768414405,  
        "bg_count" : 241  
    },  
    {  
        "key" : "metricset",  
        "doc_count" : 57,  
        "score" : 6.618709869203329,  
        "bg_count" : 104  
    },  
    ...  
]
```

It appears Monica is an expert
on Beats!

There are two significant aggregations

- There are two different ***significant aggregations***
 - **`significant_terms`** (should be used with keyword fields)
 - **`significant_text`** (should be used with text fields)
- We recommend that you use a ***significant aggregation*** as an ***inner aggregation*** of the sampler aggregation
 - it limits the analysis to a small selection of top-matching documents (e.g 200)
 - it will typically improve speed, memory use, and quality of results
- www.elastic.co/blog/significant-terms-aggregation

Summary: More aggregations

Module 5 Lesson 3



Summary

- Pipeline aggregations work on the output of another aggregation
- The `top_hits` aggregation returns the most relevant documents for each bucket
- The significant aggregations find uncommonly common terms in your dataset

Quiz

Explain which aggregations answer the questions below:

1. What are the most significant terms of the 3 authors who wrote the most blogs?
2. Which year saw the highest number of blog authors?

More aggregations

Lab 5.3

Create more complex aggregations



Transforming data

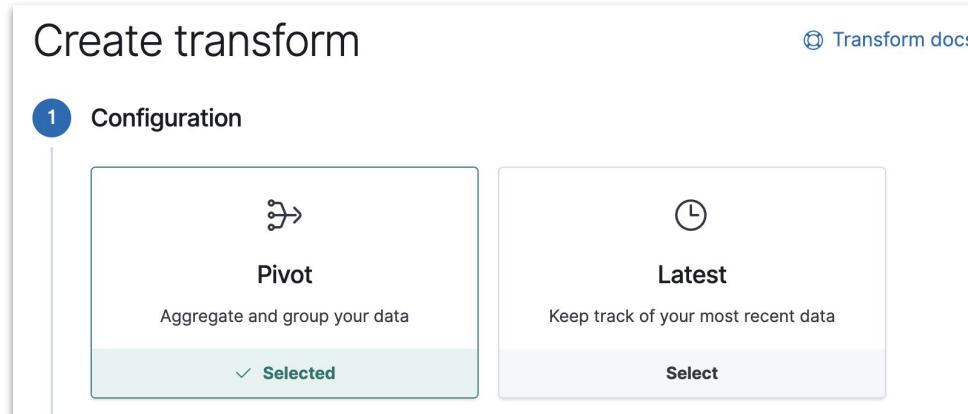
Module 5 Lesson 4



Transforms

Transforming data

- Transforms enable you to convert existing Elasticsearch indices into summarized indices using aggregations
 - pivot your event-centric data into entity-centric indices
 - find the latest document among all the documents that have a certain unique key

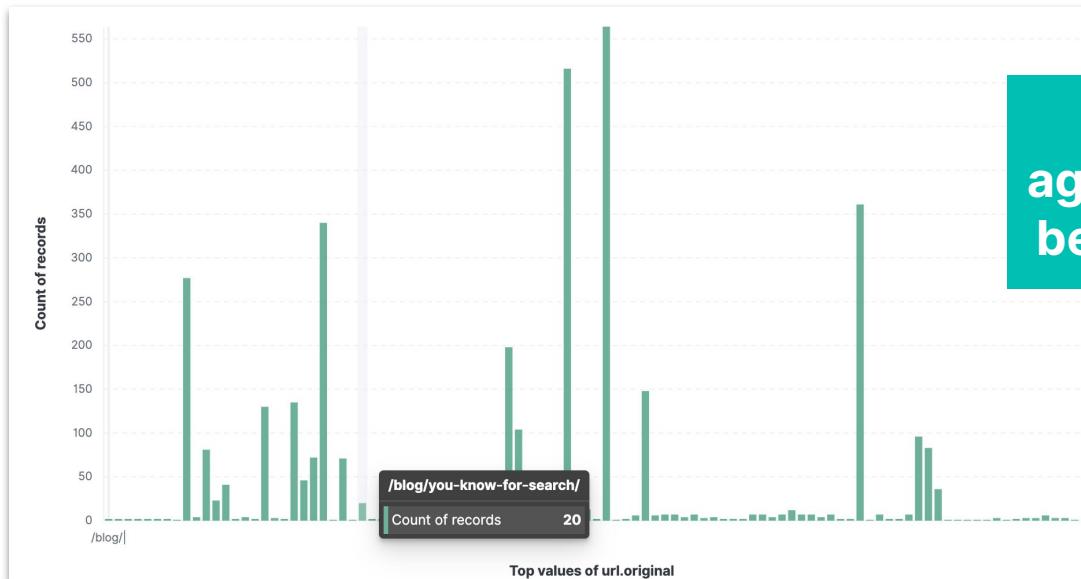


Cluster-efficient aggregations

- Elasticsearch aggregations are a powerful and flexible feature that enable you to summarize and retrieve complex insights about your data
- But if you run a complex aggregation on a lot of data you may run out of memory, for example
 - you need a complete feature index rather than a top-N set of items
 - you need to sort aggregation results by a pipeline aggregation
 - you want to create summary tables to optimize queries
- Data transforms can help!

An example

- The **blogs** index is a collection of all the blogs
 - The **web_traffic** index is a record of visitors to our blogs
 - How many visitors visited each blog?



Using a normal aggregation may not be the best solution

Example: transform `web_traffic`

- Transform `web_traffic` using pivot
 - group by url
- Choose aggregations of stats you want to collect for each url
 - count of docs = number of visits
 - avg (`runtime_ms`) = average time to load page
- You can even edit runtime mappings directly in the UI

Group by
url.original

Add a group by field ...

Aggregations
@timestamp.value_count
runtime_ms.avg

Add an aggregation ...

Transform preview

Columns Sort fields

url.original	@timestamp.value_count	runtime_ms.avg
/blog/exciting-times-ahead	75	533.333333333334
/blog/exciting-times-ahea...	1	0
/blog/exciting-times-ahea...	2	500
/blog/the-benefits-of-cl...	133	593.984962406015
/blog/the-benefits-of-cl...	1	0

Check the preview
of your transform

Transforms settings

- Transforms can run in **Continuous mode**
 - a checkpoint is generated each time a transform ingests and transforms new source data
 - configure the frequency to set the interval between checks (max 1 hour)
- Configure **Retention policy** to identify out of date documents in the destination index

2 Transform details

Transform ID
web_traffic_transform

Transform description
Description (optional)

Destination index
my_traffic_transform_index

Create Kibana index pattern

Continuous mode

Retention policy

> Advanced settings

Destination index

- Create the **destination index** in advance with custom settings for optimal performance
 - use **Preview transforms API** to see the **generated_dest_index**
 - optimize index mapping & settings
 - disable **_source** to save storage space
 - use **index sorting** if you have more than one group by field

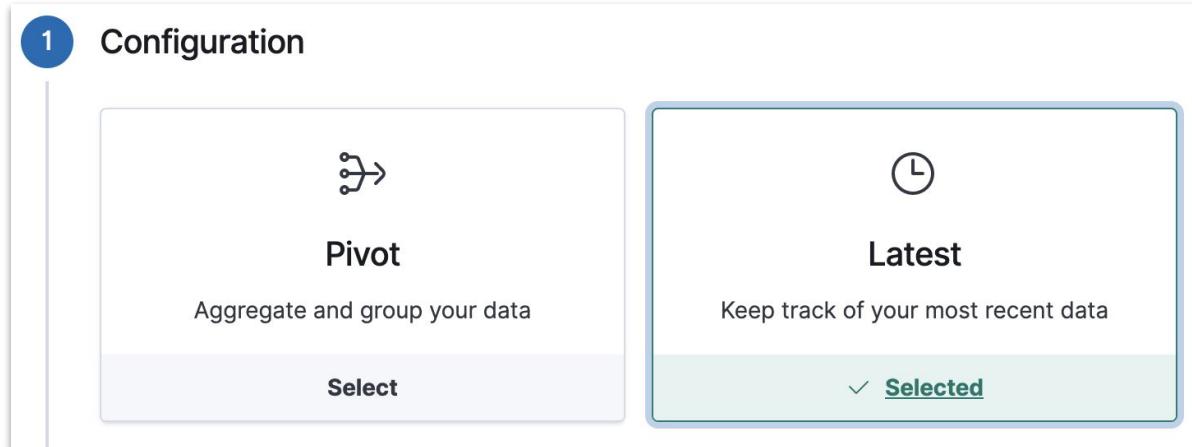
response

```
"preview" : [ ... ],
"generated_dest_index" : {
  "mappings" : {
    "_meta" : { ... },
    "properties" : {
      "runtime_ms.avg" : {
        "type" : "double"
      },
      "runtime_ms.avg" : {
        "type" : "double"
      },
      "url.original" : {
        "type" : "keyword"
      },
      "@timestamp" : {
        "type" : "object"
      }
    }
  }
}
```

...

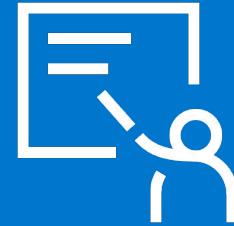
Latest

- Use the latest transforms to copy the most recent documents into a new index
- For example
 - keep track of the latest purchase for each customer
 - latest event for each host



Summary: Transforming Data

Module 5 Lesson 4



Summary

- Use transforms to index summaries of time series data
 - ***pivot***: to collect results of complex bucket and **metrics aggs**
 - ***latest***: to collect most recent documents of **bucket aggs**
- Transforms can run continuously to keep track of the most recent results
- Tune the settings to throttle continuous transforms if search is using too much resources
- Optimize the destination index settings and mappings to improve performance and reduce storage space

Quiz

1. When should you use transforms?
2. **True or False:** Data transforms are always expensive and should be used with caution
3. Activity logs are collected from multiple servers to an index.

What transforms type can you use to answer the following question?

“Which of my servers have been idle during the last 10 mins?”

Transforming Data

Lab 5.4

Perform a pivot transform.



Elasticsearch engineer: agenda

- Module 1: Getting started
- Module 2: Data modeling
- Module 3: You know, for search
- Module 4: Data processing
- Module 5: Aggregations
- **Module 6: The one about shards**
- Module 7: Data management
- Module 8: Cluster management

The one about shards

Module 6

Topics

- Understanding shards
- Scaling Elasticsearch
- Distributed operations

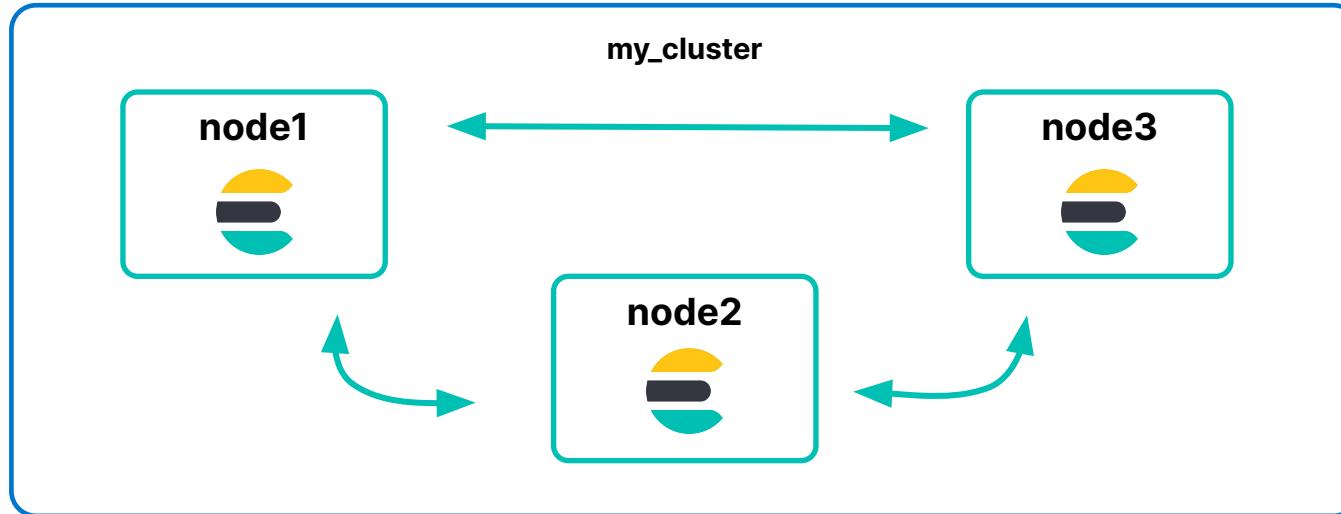
Understanding shards

Module 6 Lesson 1



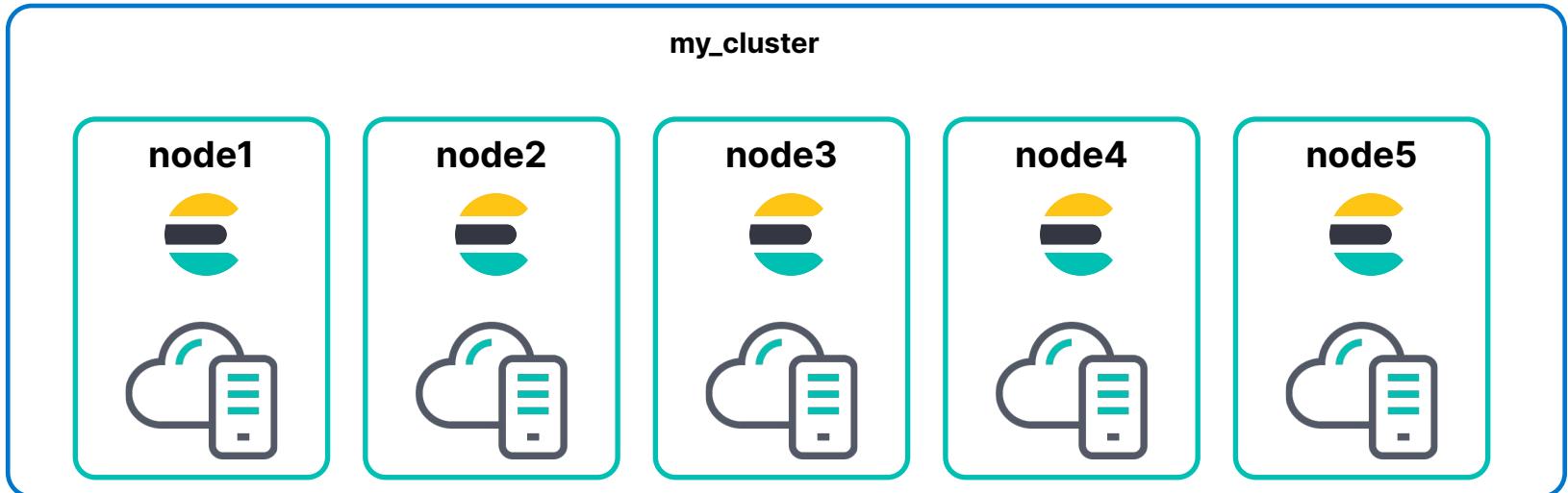
The cluster

- The largest unit of scale in Elasticsearch is a cluster
- A cluster is made of 1 or more nodes
 - nodes communicate with each other and exchange information



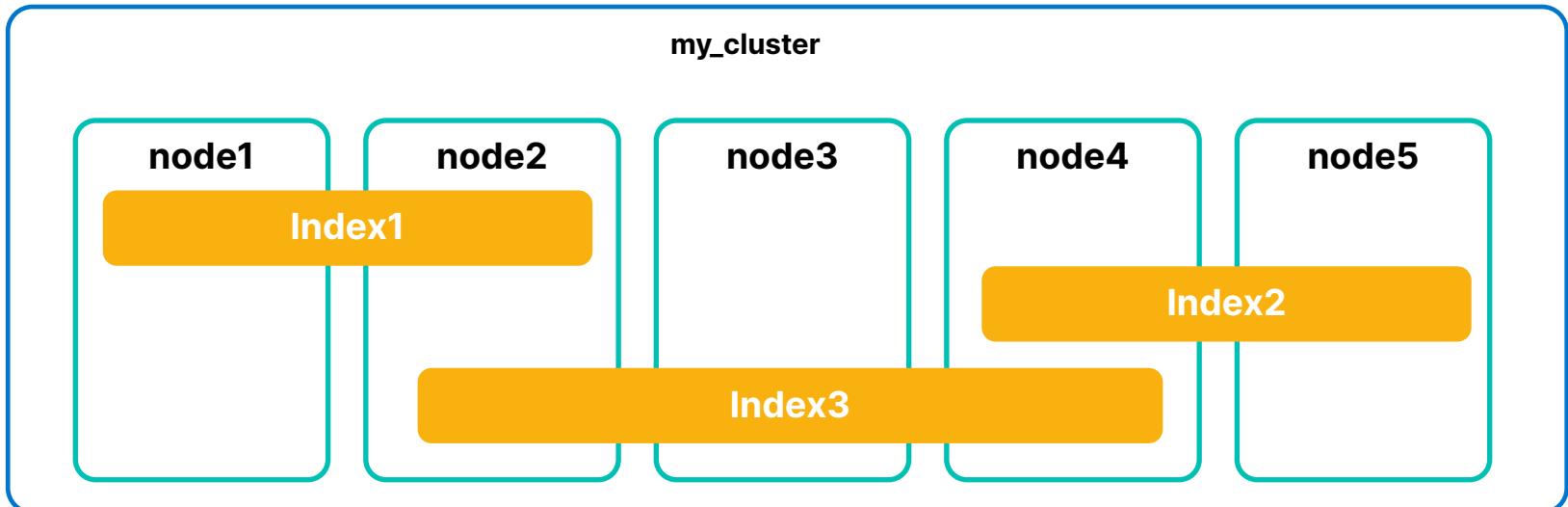
The node

- A node is an instance of a Elasticsearch
 - a node is typically deployed 1-to-1 to a host
 - to scale out your cluster, add more nodes



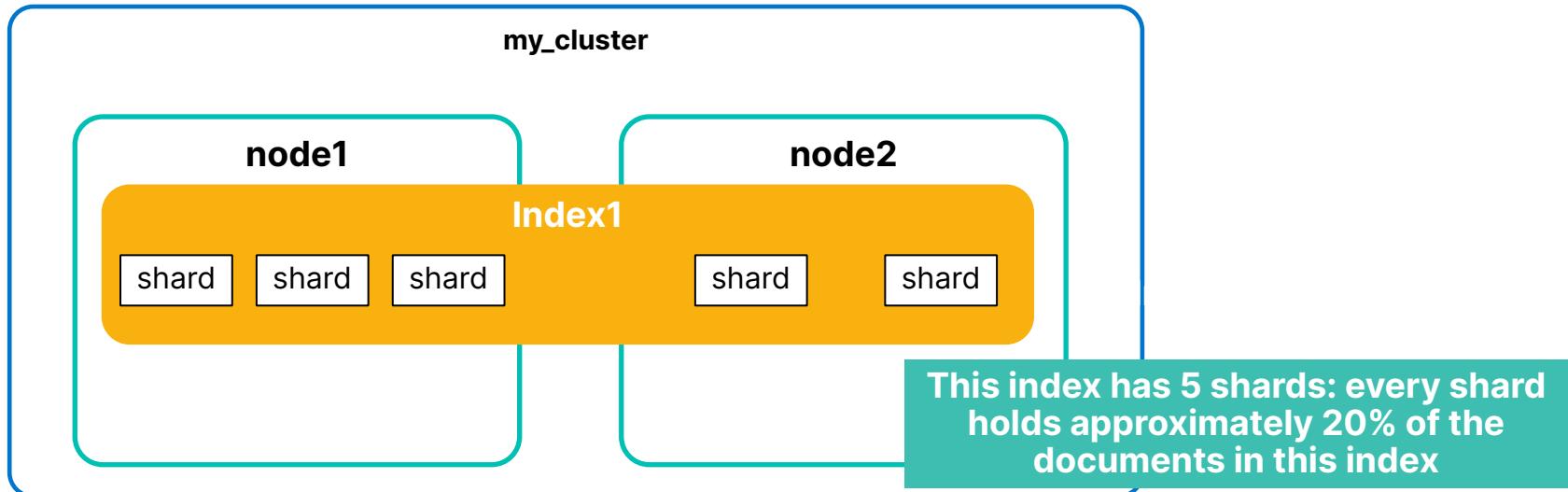
The index

- An index is a collection of documents that are related to each other
 - the documents stored in Elasticsearch are distributed across nodes



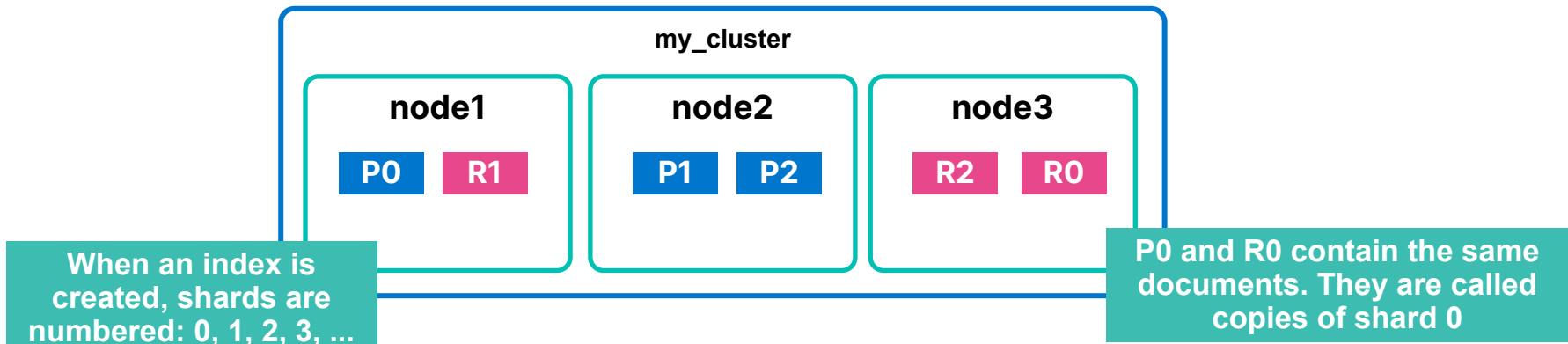
The shard

- An index distributes documents over one or more shards
- Each **shard**:
 - is an instance of Lucene
 - contains all the data of any one document



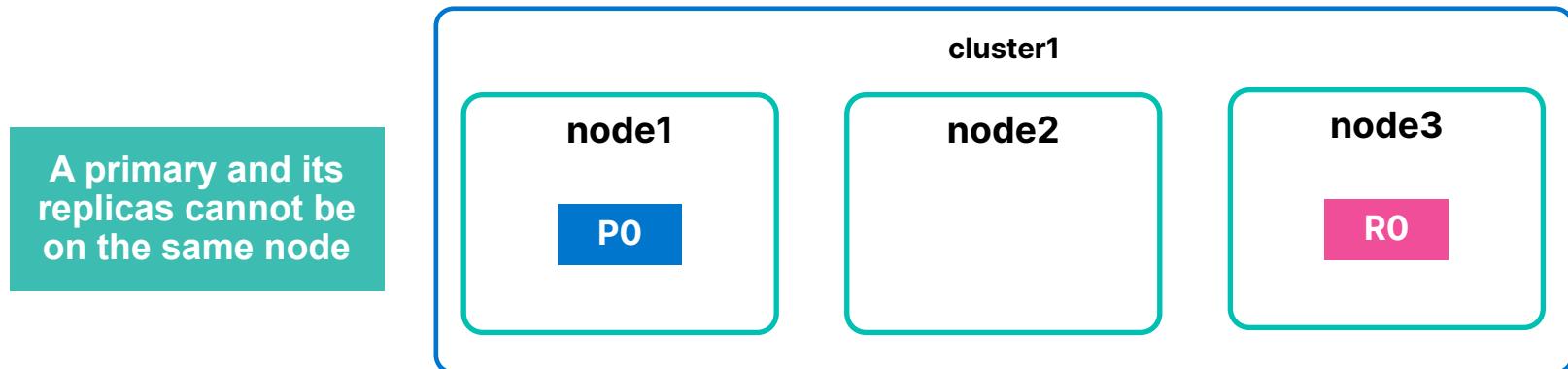
Primary vs. replica

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



The shards of the blogs data set

- Our blogs index has the default number of ***one primary*** shard with ***one replica***
 - you can not increase the number of primary shards after an index is created
 - the number of replicas is dynamic

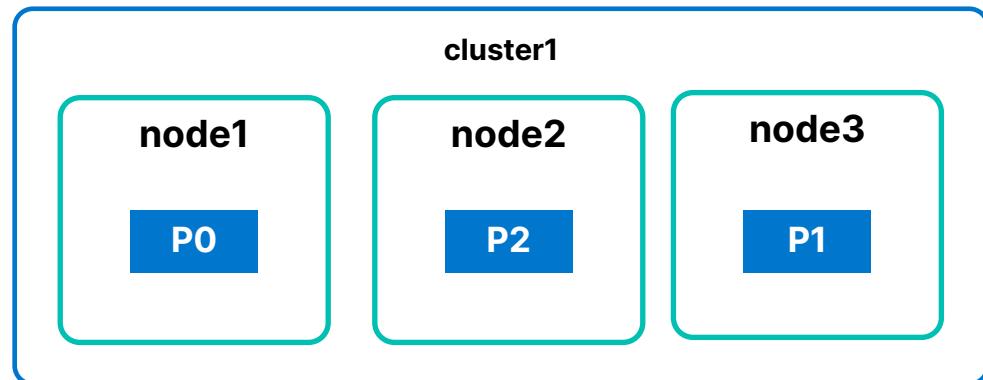


Configuring the number of primaries

- Specify the number of primary shards *when you create the index*
 - default is 1
 - use the `number_of_shards` setting

request

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



Why only one primary?

- **Oversharding** is one of the most common problems users encounter
 - too many small shards consume resources
- A shard typically holds **tens of gigabytes**
- If more shards are needed:
 - creating multiple indices make it easy to scale
 - otherwise, the **Split API** enables you to increase the number of shards

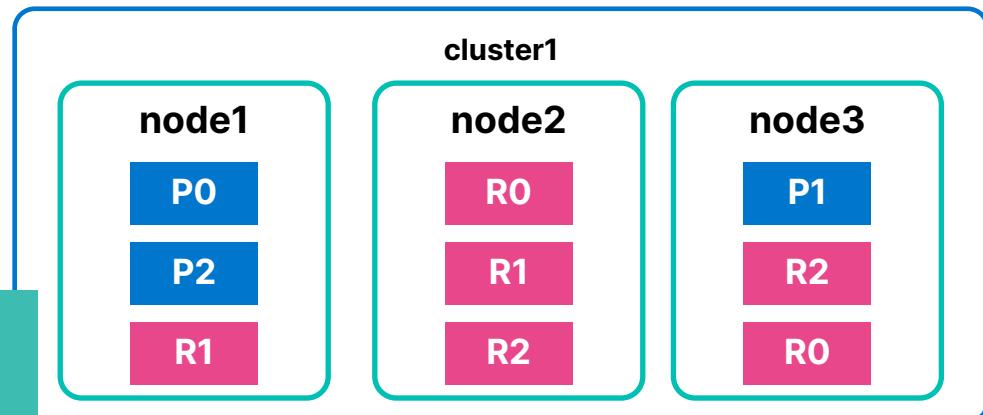
Configuring the number of replicas

- The default number of replicas per primary is 1
 - specify the number of replica sets when you create the index
 - use the `number_of_replicas` setting
 - can be changed at any time

request

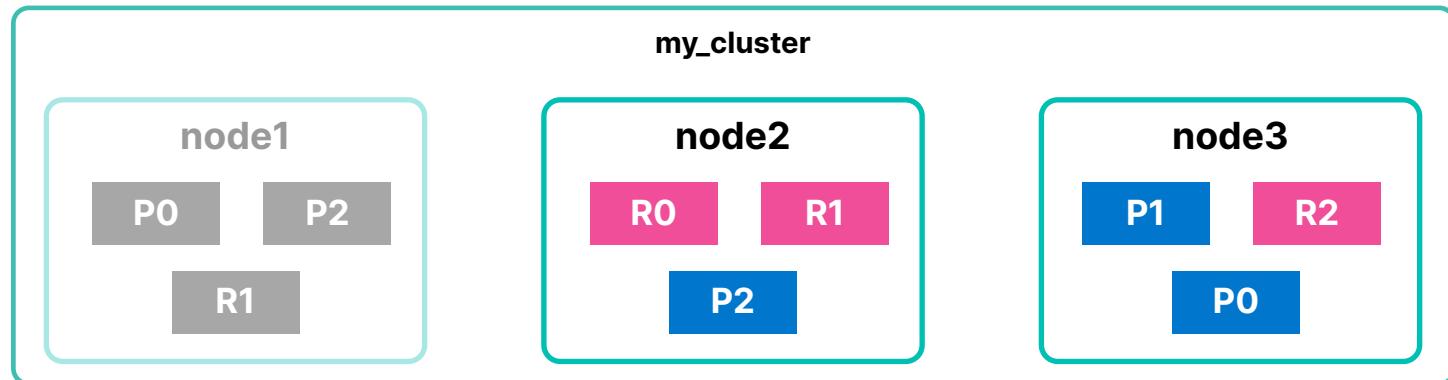
```
PUT my_new_index/_settings
{
  "number_of_replicas": 2
}
```

3 primaries + 2 replica sets =
9 total shards



Why create replicas?

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

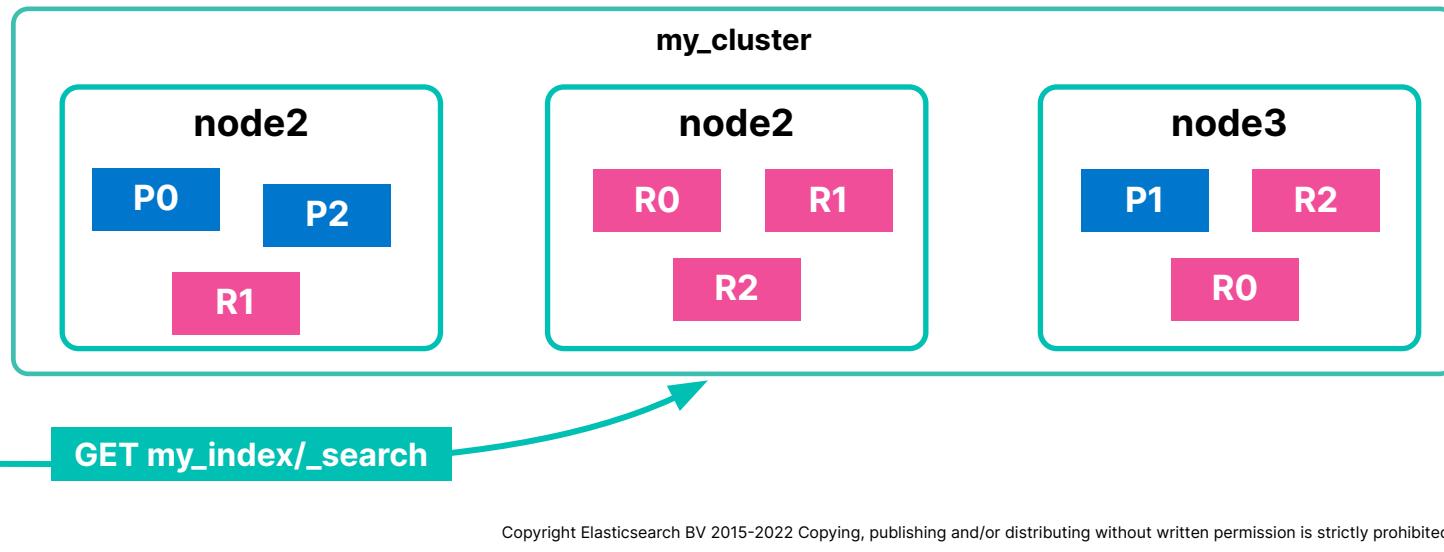


If node1 fails, its data is still available on other nodes

P0 and P2 were lost so an R0 and an R2 will be promoted

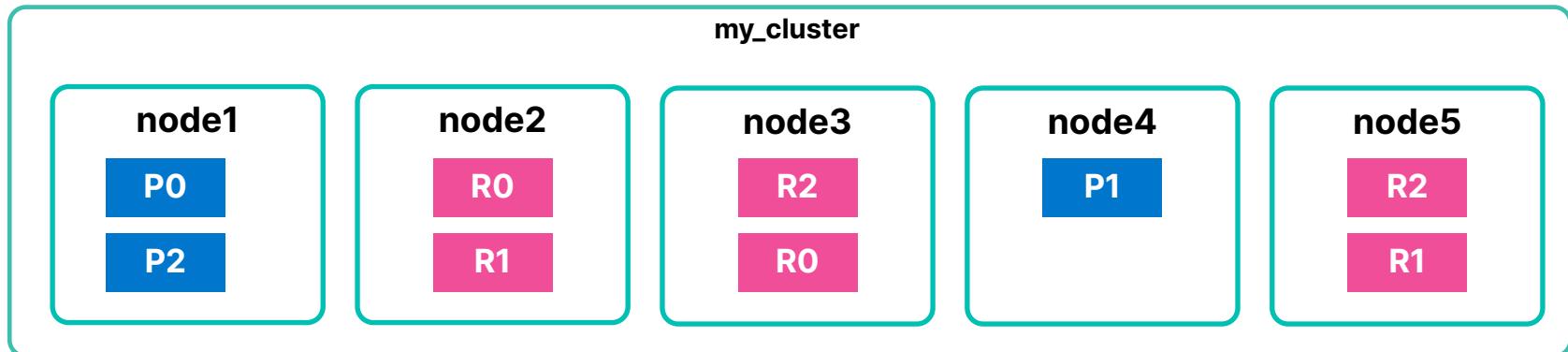
Why create replicas?

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



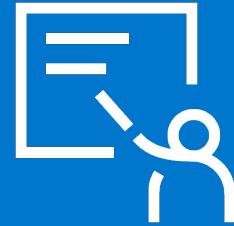
Scaling Elasticsearch

- Adding nodes to a cluster will trigger a ***redistribution of shards***
 - and the creation of replicas (if enough nodes exist)



Summary: Understanding shards

Module 6 Lesson 1



Summary

- Elasticsearch subdivides the data of your index into multiple pieces called ***shards***
- Each shard copy has one (and only one) ***primary*** and zero or more ***replicas***
- ***Oversharding*** is one of the most common problems users encounter
- Replicas are promoted to primaries as needed

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:** Each shard should hold no more than one or two gigabytes
3. **True or False:** Elasticsearch automatically migrates shards to rebalance the cluster as the cluster grows

Understanding shards

Lab 6.1

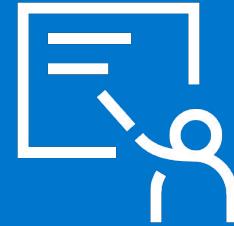


Create an index with multiple shards.

Analyze the behavior of those shards.

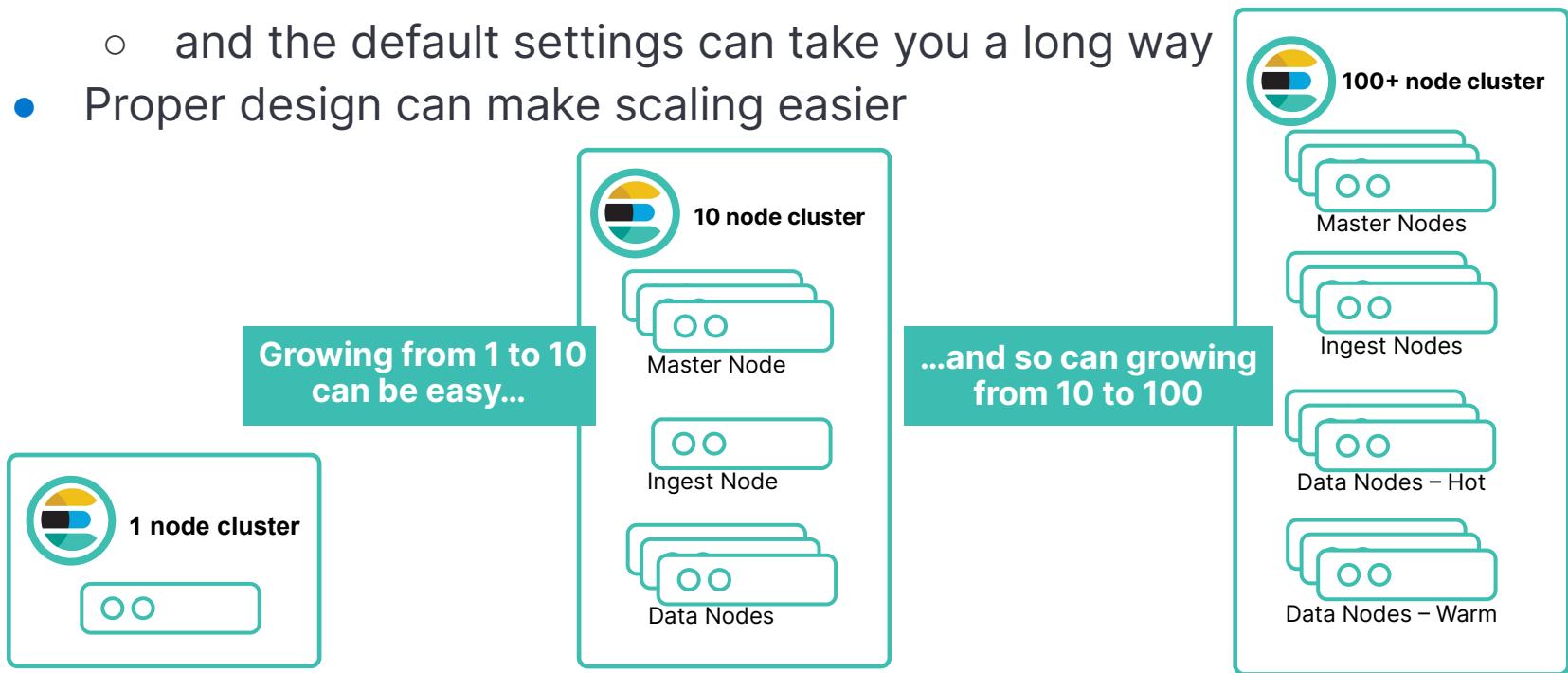
Scaling Elasticsearch

Module 6 Lesson 2



Designing for scale

- Elasticsearch is built to scale
 - and the default settings can take you a long way
- Proper design can make scaling easier



One shard...

... does not scale very well:

request

```
PUT my_index
{
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 0
  }
}
```

node1

P0

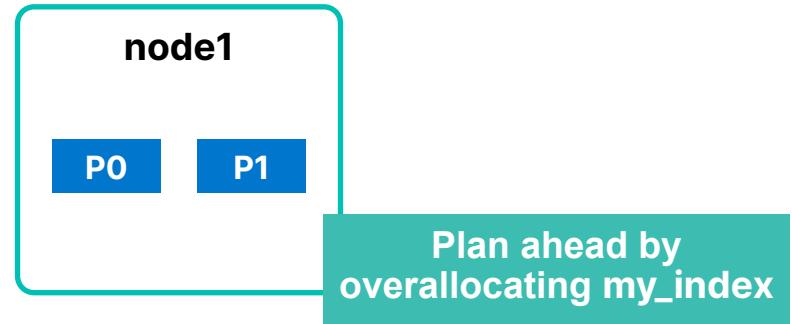
Adding a second node would
not change anything

Two shards...

... can scale if we add a node:

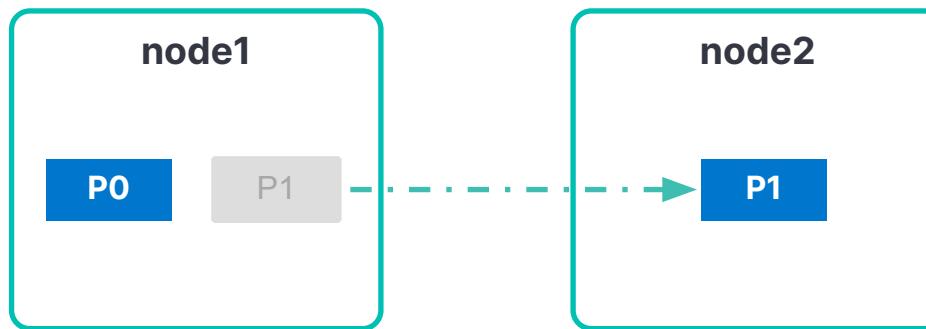
request

```
PUT my_index
{
  "settings": {
    "number_of_shards": 2,
    "number_of_replicas": 0
  }
}
```



Balancing of shards

- Elasticsearch ***automatically balances shards***:
 - **node1** is now responsible for half the amount of data
 - write throughput has doubled (twice the disk IO available)
 - the memory pressure on **node1** is less than before
 - searches now use the resources of both **node1** and **node2**



Shard overallocation

- If you expect your cluster to grow, then plan for that by overallocating shards:
 - ***number of shards > number of nodes***

request

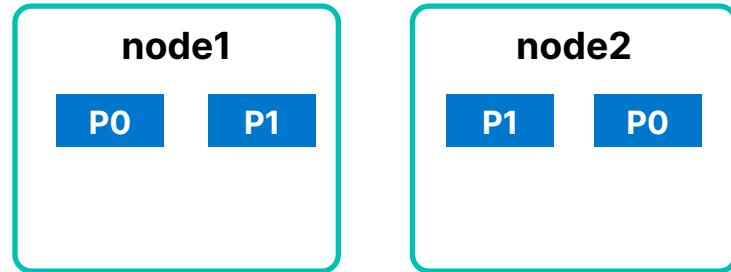
```
PUT my_index
{
  "settings": {
    "number_of_shards": 4,
    "number_of_replicas": 0
  }
}
```



Shard overallocation

- Overallocating shards works well for static data, but not for time-series data
 - for time-series data, ***create multiple indices***
- ***1 index with 4 shards*** is similar to ***2 indices each with 2 shards***
 - the end result is 4 shards in both scenarios

```
PUT 2021-07-01-logs
{
  "settings": {
    "number_of_shards": 2
  }
}
PUT 2021-07-02-logs
{
  "settings": {
    "number_of_shards": 2
  }
}
```



Too much overallocation

- A little overallocation is good
- A ***kajillion shards*** is not good:
 - each shard comes at a cost (Lucene indices, file descriptors, memory, CPU)
- A shard typically holds ***at least*** tens of gigabytes
 - depends on the use case
 - a 100 MB shard is probably too small

Do not overshard

- **Business requirements**
 - 1GB per day
 - 6 months retention
 - ~180GB
- **Common scenario**
 - 3 different logs
 - 1 index per day each
 - 5 shards (default before 7.0)
 - 6 months retention
 - ~2700 shards

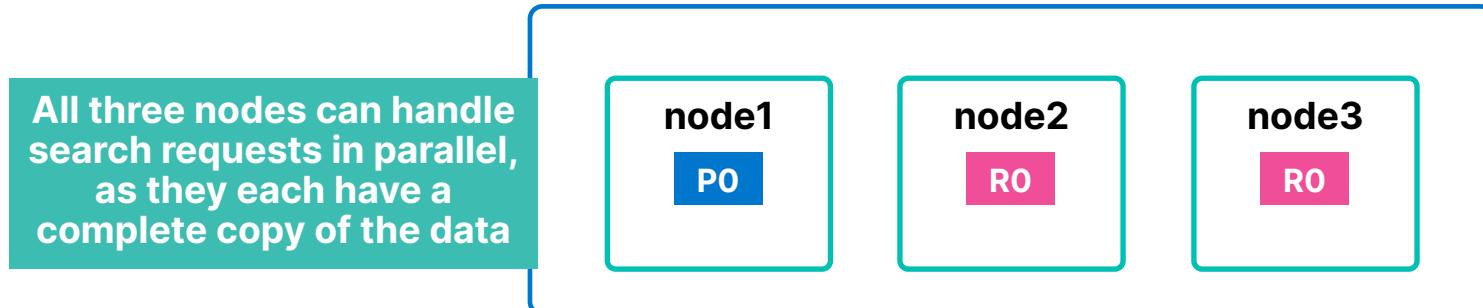
We could easily have
this data on 10 shards

Too many shards for
no good reason!

Scaling for reads

Scaling for reads

- Queries and aggregations scale with replicas
- For example, have one primary and as many replicas as you have additional nodes
 - use `auto_expand_replicas` setting to change the number of replicas automatically as you add/remove nodes



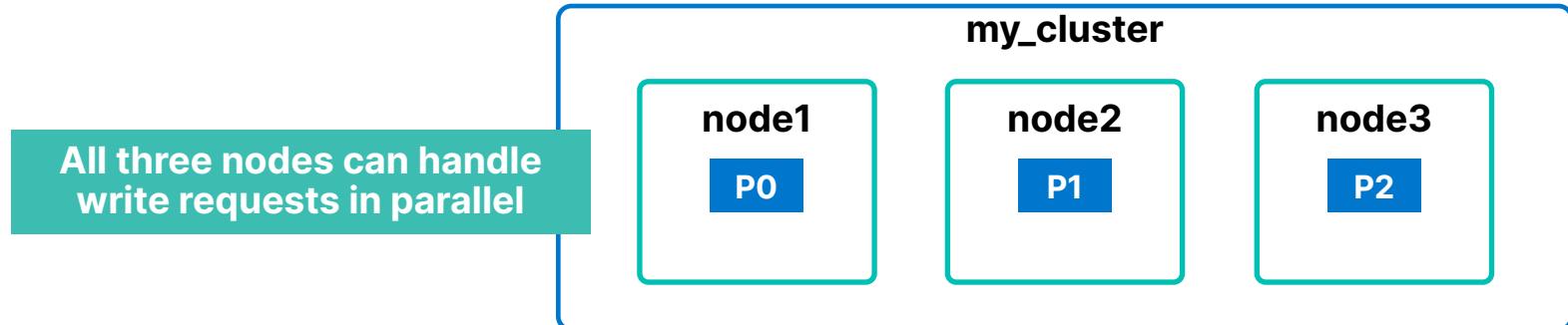
Optimizing for read throughput

- Create flat, denormalized documents
- Query the smallest number of fields
 - consider `copy_to` over `multi_match`
- Map identifiers as keyword instead of as a number
 - term queries on keyword fields are very fast
- Force merge read-only indices
- Limit the scope of aggregations (upcoming topic)
- Use filters, as they are cacheable

Scaling for writes

Scaling for writes

- Write throughput scales by *increasing number of primaries*
 - having many primary shards allows Elasticsearch to "fan out" the writes, so each shard does less work
 - maximize throughput by using disks on all machines
- When an index is done with writes, you can **shrink** it

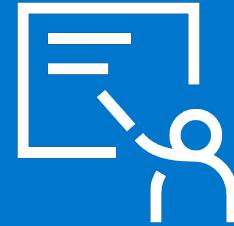


Optimizing for write throughput

- Use `_bulk` API to minimize the overhead of HTTP requests
- Parallelize your write requests
- ***Disable refreshing every second:***
 - set `index.refresh_interval` to -1 for very large writes (then back to default when finished indexing)
 - set `index.refresh_interval` to 30s to increase indexing speed but affect search as little as possible
- ***Disable replicas***, then re-enable after very large writes
 - every document also needs to be written to every replica
- Use ***auto-generated IDs***:
 - Elasticsearch won't check whether a doc ID already exists

Summary: **Scaling Elasticsearch**

Module 6 Lesson 2



Summary

- If you expect your cluster to grow, then plan for that by ***overallocating*** shards
- A little overallocation is good. A ***kajillion shards*** is not!
- You can ***scale the read workload*** of your cluster by adding more nodes and increasing the number of replicas of your indices
- You can ***scale the write workload*** of your cluster by adding more nodes and increasing the number of primaries of your indices

Quiz

1. If you have a two node cluster, under what circumstances would you create an index with more than two primary shards?
2. **True or False:** To maximize read throughput you need to divide your data over as many primaries as possible
3. **True or False:** Using auto-generated IDs is more efficient than providing your own IDs

Scaling Elasticsearch

Lab 6.2



Optimize an index for writes and reads.
Add one node to the cluster.

Distributed operations

Module 6 Lesson 3

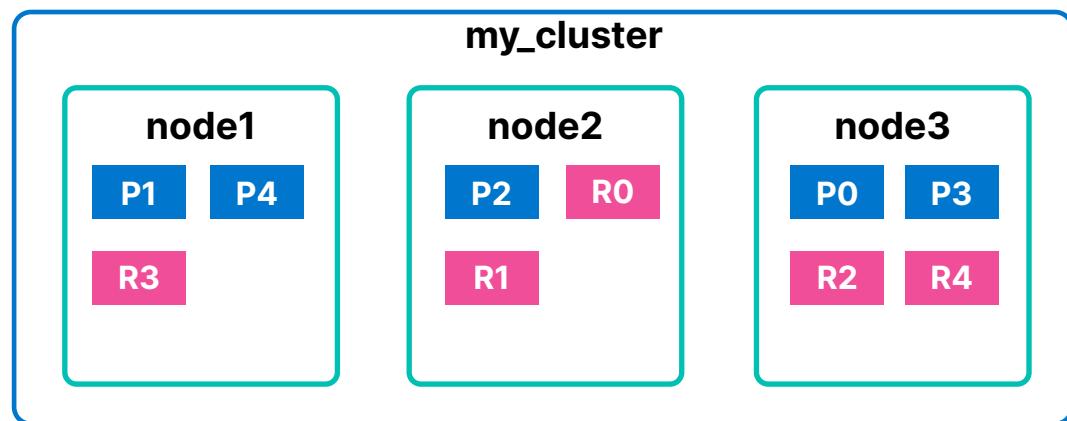


Write operations

How data goes in

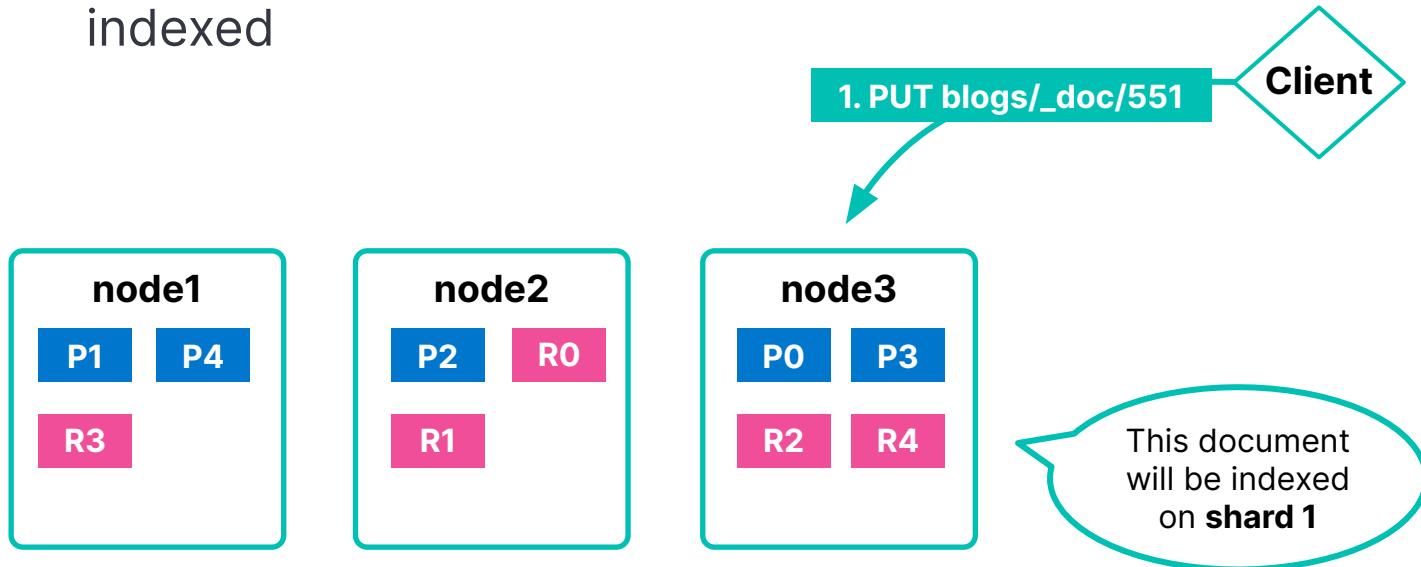
- Let's take a look at the details of how a document is indexed into a cluster
 - suppose we index the following document into our blogs index, which currently has **5 primary shards with 1 replica**

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



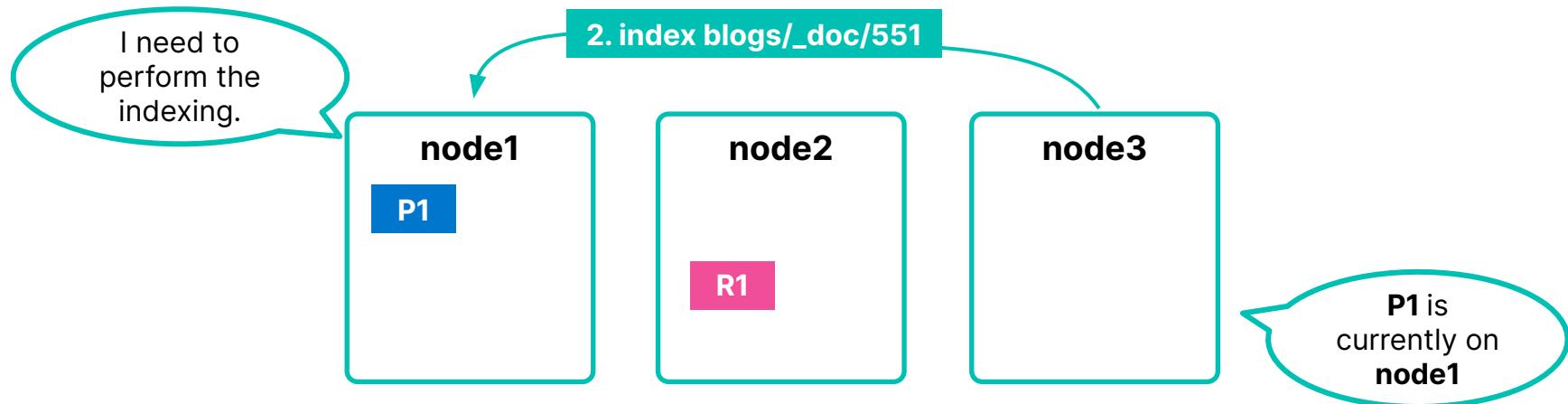
Document routing

- The index request is sent to a chosen **coordinating** node
- This node will determine on which shard the document will be indexed



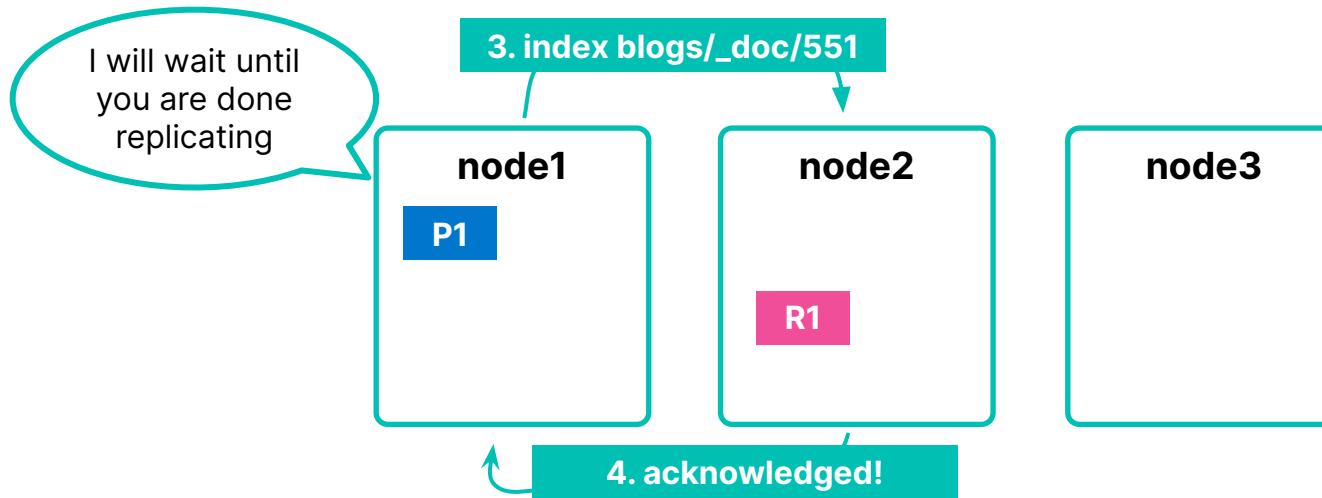
Write operations on the primary shard

- When you index, delete, or update a document, the **primary shard** has to perform the operation first
 - node3 will forward the indexing request to node1



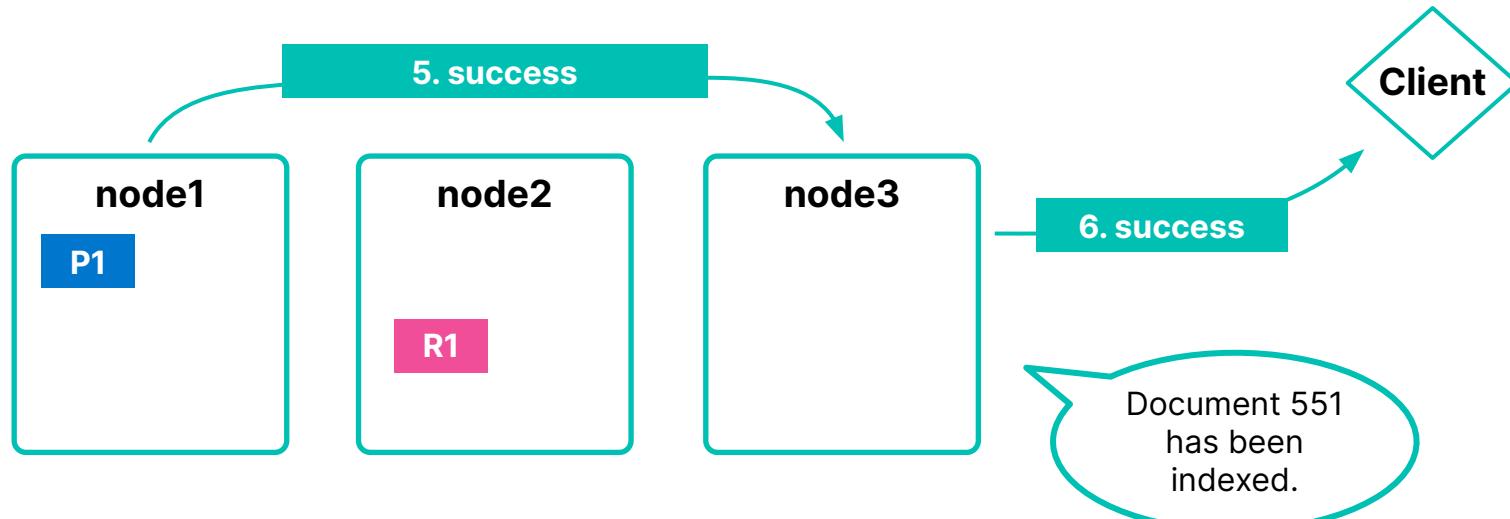
Replicas are synced

- **node1** indexes the new document, then forwards the request (in parallel) to all replica shards
 - **P1** has one replica (**R1**) that is currently on **node2**



Client response

- **node1** lets the coordinating node (**node3**) know that the write operation is successful on every shard
 - and **node3** sends the details back to the client application



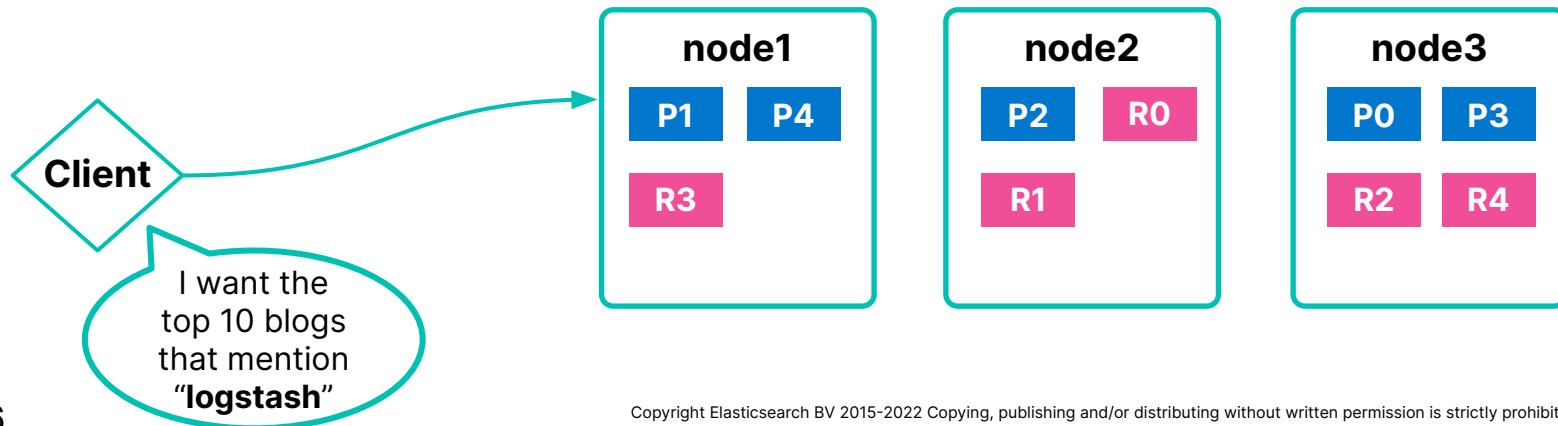
Updates and deletes

- **Updating or deleting** is similar to indexing 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

Search operations

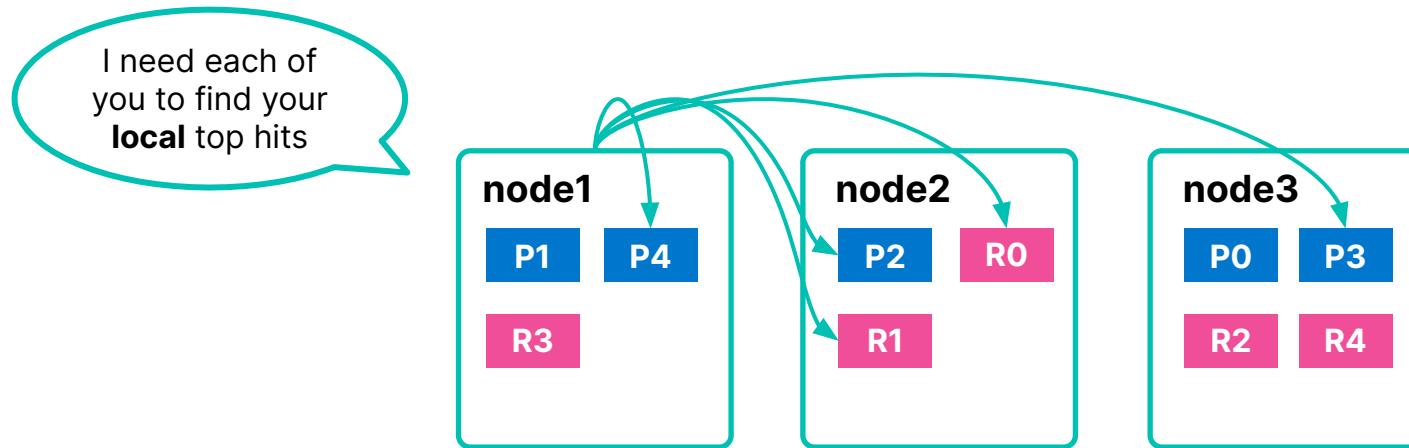
Anatomy of a search

- Distributed search is a challenging task
 - we have to search for hits in a copy of every shard of 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 scatter phase

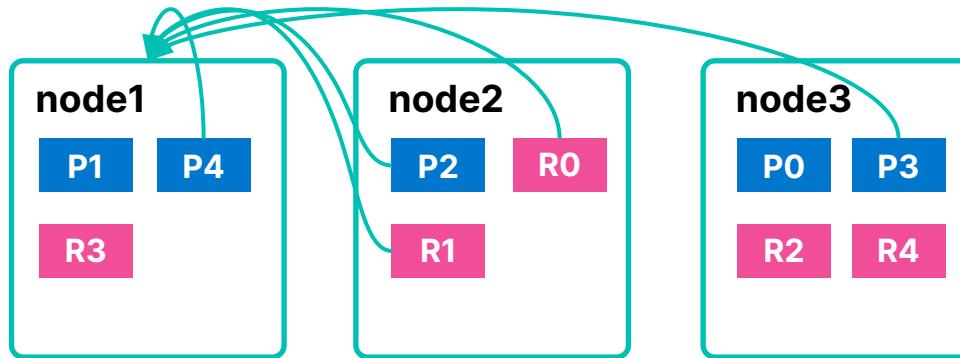
- The initial part of a search is referred to as the **scatter phase**
 - the query is broadcast to a **shard copy** of every shard in the index
 - each shard executes the query **locally**



The scatter phase

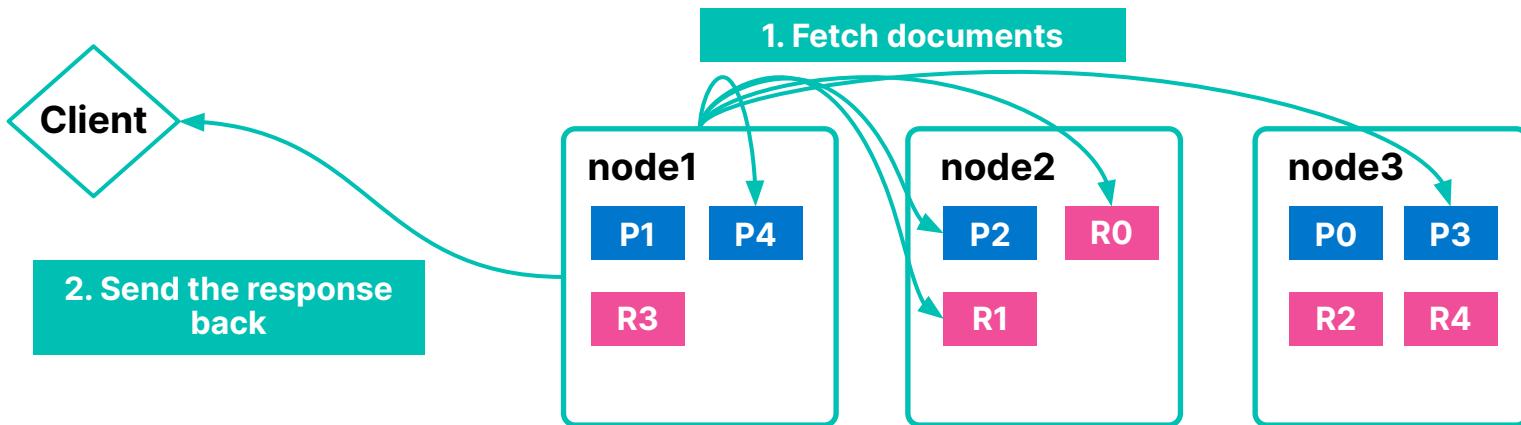
- Each shard returns the **doc IDs** and **sort values** of its top hits to the coordinating node
- The coordinating node **merges these values** to create a **globally sorted list of results**

Thanks,
everyone! I will
figure out the
global top hits.



The gather phase

- Once the coordinating node has determined the doc IDs of the top 10 hits, it can **fetch** the documents' `_source`
 - then returns the top documents to the client



Summary: Distributed operations

Module 6 Lesson 3



Summary

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

Quiz

1. **True or False:** An index operation has to be executed on the primary shard first before being synced to replicas
2. **True or False:** A search operation can be redirected to a replica shard

Distributed operations

Lab 6.3



Run a search query on different shards.

More resources

- How many shards?
 - www.elastic.co/blog/how-many-shards-should-i-have-in-my-elasticsearch-cluster
- Autoscale in Elastic Cloud:
 - www.elastic.co/blog/autoscale-your-elastic-cloud-data-and-machine-learning-nodes
- Resizing Elasticsearch shards:
 - www.elastic.co/blog/resizing-elasticsearch-shards-for-fun-and-profit
- Replica shard management:
 - www.elastic.co/blog/optimizing-costs-elastic-cloud-replica-shard-management
- Shards and scoring:
 - www.elastic.co/blog/practical-bm25-part-1-how-shards-affect-relevance-scoring-in-elasticsearch

More resources

- Tuning for writes:
 - www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-indexing-speed.html
- Tuning for reads:
 - www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-search-speed.html

Elasticsearch engineer: agenda

- Module 1: Getting started
- Module 2: Data modeling
- Module 3: You know, for search
- Module 4: Data processing
- Module 5: Aggregations
- Module 6: The one about shards
- **Module 7: Data management**
- Module 8: Cluster management

Data management

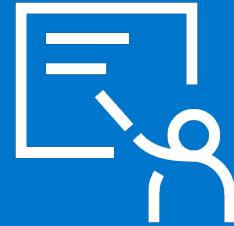
Module 7

Data management agenda

- Data management concepts
- Data streams
- Index lifecycle management
- Searchable snapshots

Data management concepts

Module 7 Lesson 1



Managing data

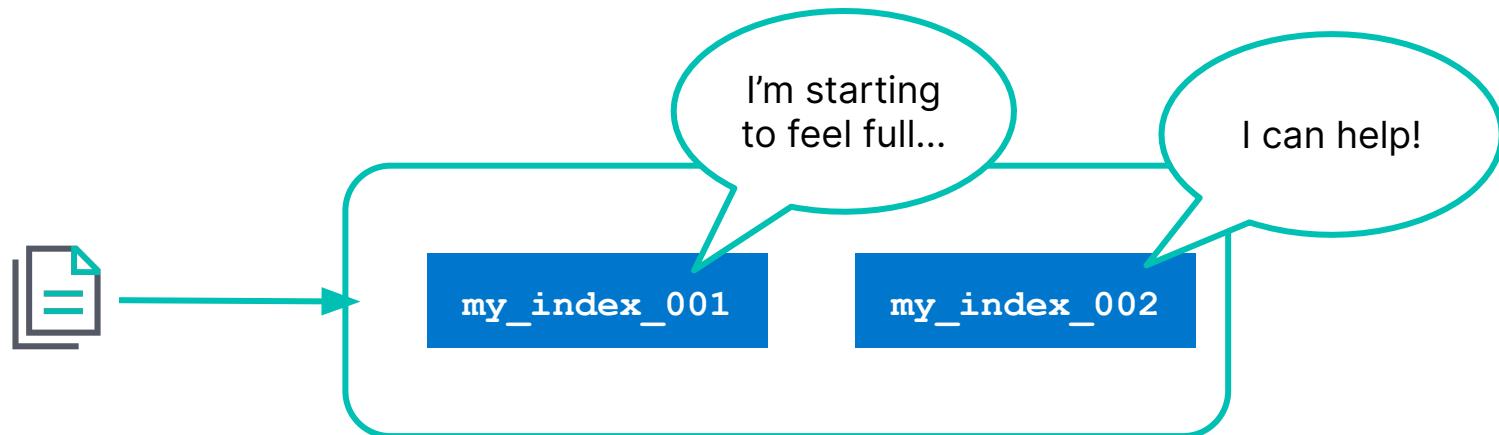
- Data management needs differ depending on the type of data you are collecting:

	Static data	Time series data
Data grows ...	slowly	fast
Updates ...	may happen	never happen
Old data is read...	frequently	infrequently

Index aliases

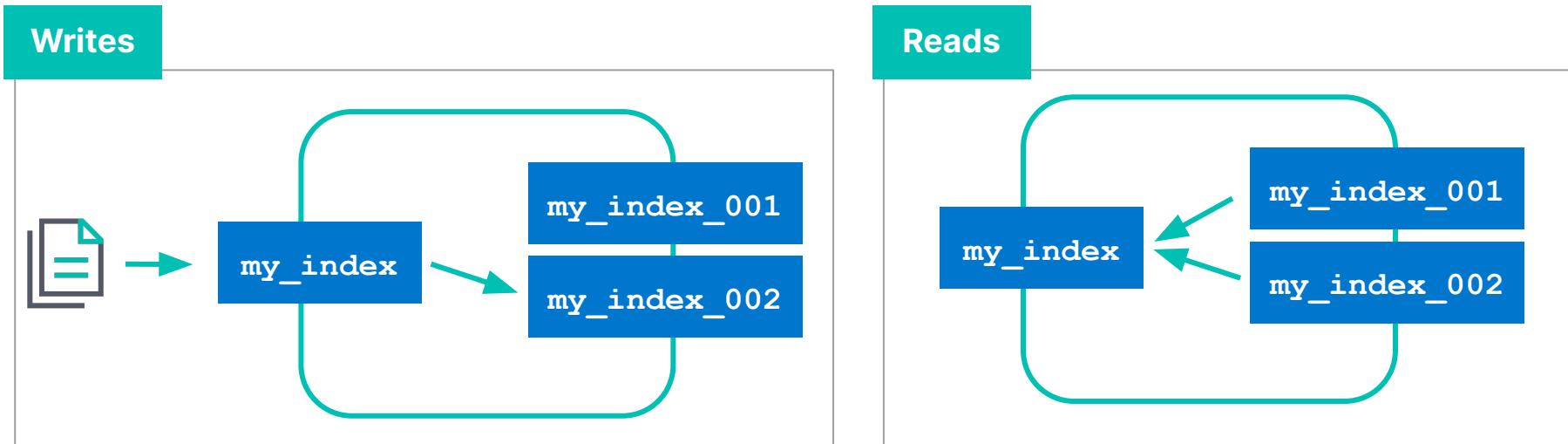
Scaling indices

- Indices scale by adding more shards
 - increasing the number of shards of an index is expensive
- **Solution:** create a new index



Using aliases

- Use ***index aliases*** to simplify your access to the growing number of indices:



An alias to multiple indices

- Use the `_aliases` endpoint to create an alias
 - specify the `write index` using `is_write_index`

```
POST _aliases
{
  "actions": [ {
    "add": {
      "index": "my_logs-*",
      "alias": "my_logs"
    }
  },
  {
    "add": {
      "index": "my_logs-2021-07",
      "alias": "my_logs",
      "is_write_index": true
    }
  } ]
}
```

Reads will search all matching indices

Write requests will go to this index

Index templates

What are index templates?

- If you need to create multiple indices with the same settings and mappings, use an ***index template***
 - templates match an ***index pattern***
 - if a new index matches the pattern, then the template is applied

Name

A unique identifier for this template.

Name

my_template

Index patterns

The index patterns to apply to the template.

Index patterns

my_index-* ×

Spaces and the characters \ / ? " < > | are not allowed.

Elements of an index template

- An index template can contain the following sections:
 - ***component templates***
 - ***settings***
 - ***mappings***
 - ***aliases***
- ***Component templates*** are reusable building blocks that can contain:
 - settings, mappings or aliases
 - components are reused across multiple templates

Defining an index template

- This **logs-template**:
 - overrides the default setting of 1 replica
 - for any new indices with a name that begins with **logs**:

```
PUT _index_template/_logs-template
{
  "index_patterns": [ "logs*" ],
  "template": {
    "settings": {
      "number_of_replicas": 0
    }
  }
}
```

Template name

Apply to any index that starts with “logs”

Applying an index template

- Create an index that matches the index pattern of one of your index templates:



Component template example

- A common setting across many indices may be to auto expand replica shards as more nodes become available
 - put this setting into a component template:

Create component template

1 Logistics 2 Index settings 3 Mappings 4 Aliases 5 Review

Index settings (optional)

Define the behavior of your indices.

Index settings

```
{  
  "index": {  
    | "auto_expand_replicas": "0-4"  
  }  
}
```

Component template example

- Use the component in an index template:

Edit template 'logs'

1 Logistics 2 Component templates 3 Index settings 4 Mappings 5 Aliases 6 Review template

Component templates (optional)

Component templates let you save index settings, mappings and aliases and inherit from them in index templates.

Components selected: 4

= logs-mappings	M	S	A
= data-streams-mappings	M	S	A
= logs-settings	M	S	A
= auto-rePLICAS	M	S	A

Search component templates Filter 3

.alerts-ecs-mappings	M	S	A
.alerts-observability.apm.alerts-mappings	M	S	A
.alerts-observability.logs.alerts-mappings	M	S	A
.alerts-observability.metrics.alerts-mappings	M	S	A
.alerts-observability.uptime.alerts-mappings	M	S	A

**M: mappings
S: index settings
A: aliases**

Copyright Elasticsearch BV 2015-2022 Copying, publishing and/or distributing without written permission is strictly prohibited

 elastic

Resolving template match conflicts

- One and only one template will be applied to a newly created index
- If more than one template defines a matching index pattern, the **priority** setting is used to determine which template applies
 - the highest priority is applied, others are not used
 - set a **priority** over 200 to override auto-created index templates
 - use the **_simulate** tool to test how an index would match

```
POST /_index_template/_simulate_index/my_index-test
```

Summary: Data management concepts

Module 7 Lesson 1



Summary

- ***Index aliases*** simplify reads and writes to multiple indices
- ***Index templates*** can be used to ensure new indices receive the same settings and mappings
- ***Rollovers*** can be used with aliases and templates to create new indices when an older one becomes full

Quiz

1. **True or False:** You must always use aliases for all of your production indices
2. **True or False:** If more than one template matches an index pattern, both templates will be applied

Data management concepts

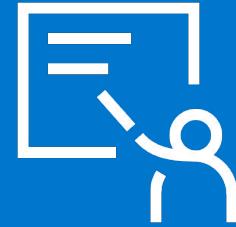
Lab 7.1



Create templates, aliases and run a
rollover.

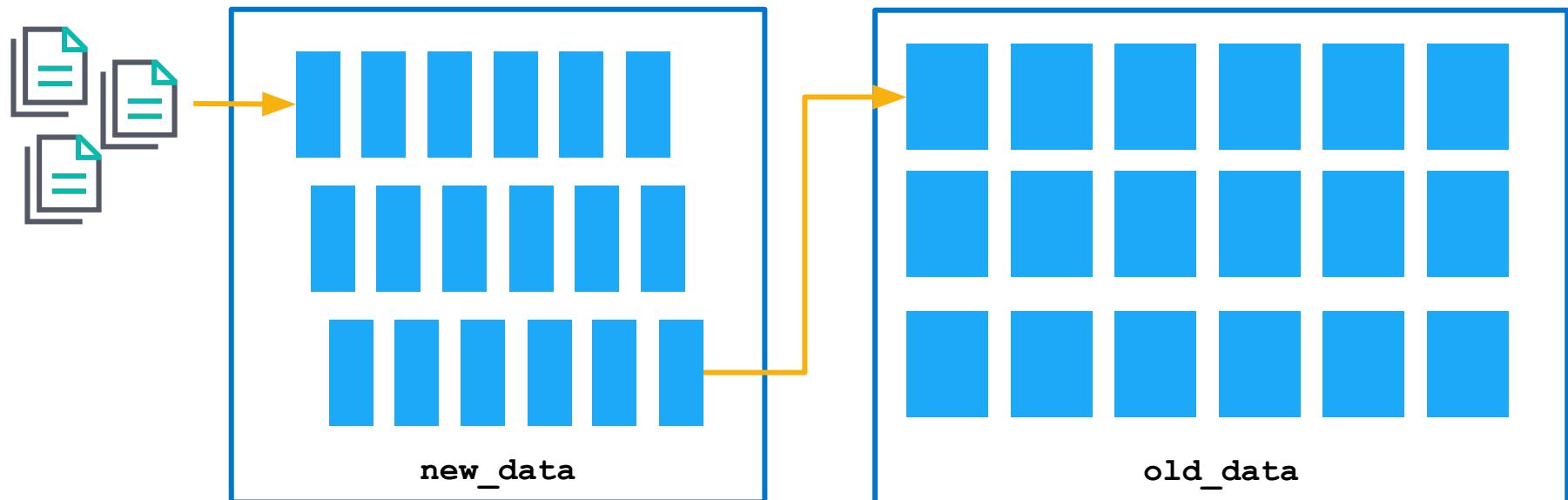
Data streams

Module 7 Lesson 2



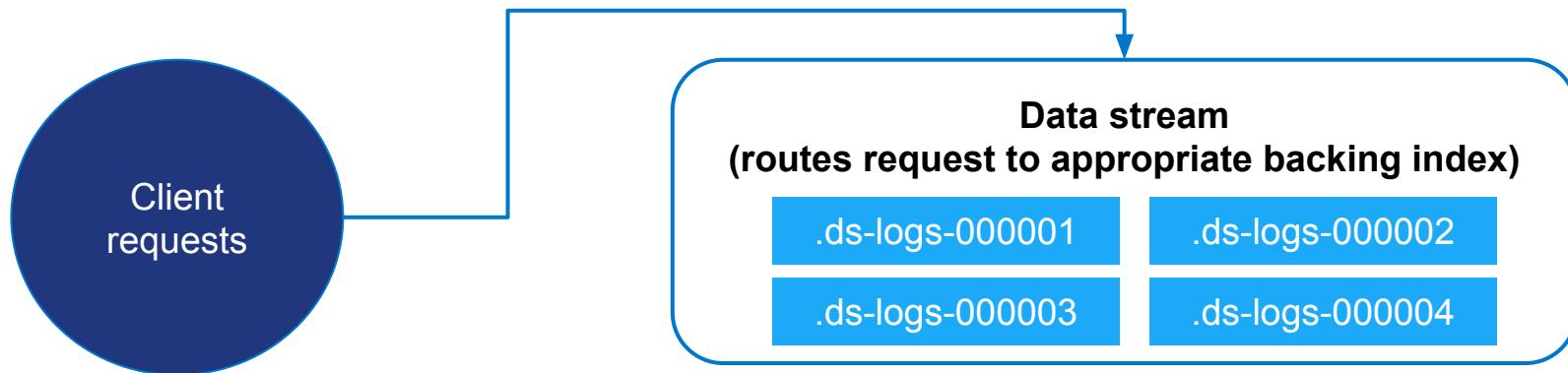
Time series data management

- Time series data typically *grows quickly* and is *almost never updated*



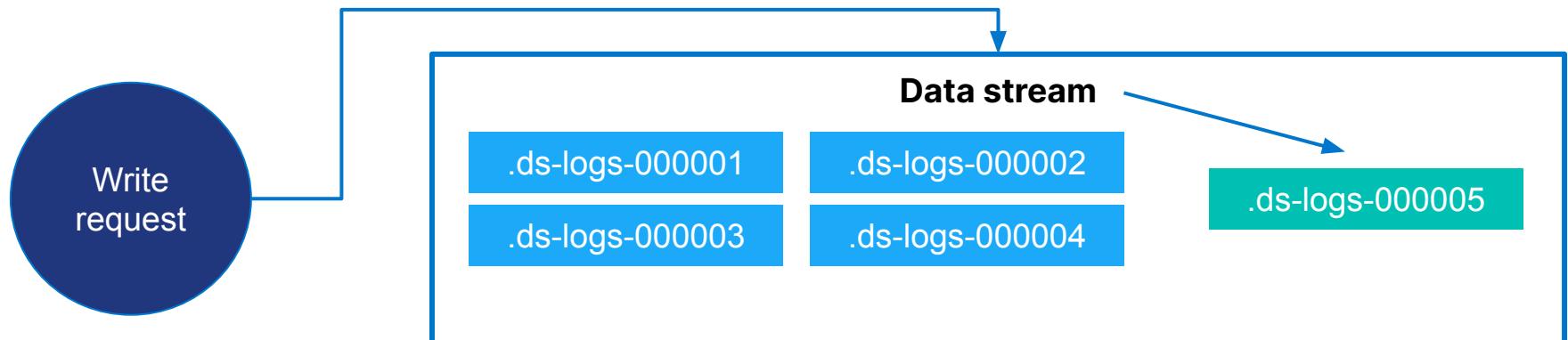
Data streams

- A ***data stream*** lets you store time-series data across multiple indices while giving you ***a single named resource for requests***
 - indexing and search requests are sent to the data stream
 - the stream routes the request to the appropriate ***backing index***



Backing indices

- Every data stream is made up of ***hidden backing indices***
 - with a single write index
- A ***rollover*** creates a new backing index
 - based on age or size
 - which becomes the stream's new ***write index***



Data stream naming conventions

- Data streams are named by:
 - ***type***, to describe the generic data type
 - ***dataset***, to describe the specific subset of data
 - ***namespace***, for user-specific details
- Each data stream should include **`constant_keyword`** fields for:
 - `data_stream.type`
 - `data_stream.dataset`
 - `data_stream.namespace`
- **`constant_keyword`** has the same value for all documents

`metrics-system.cpu-production`

`type`

`dataset`

`namespace`

Example use of data streams

- Log data separated by ***app*** and ***env***
- Each data stream can have ***separate*** lifecycles
- Different datasets can have different fields

logs-app1-prod

logs-app2-prod

logs-app1-dev

```
GET logs-*-*/_search
{
  "query": {
    "bool": {
      "filter": {
        "term": {
          "data_stream.namespace": "prod"
        }
      }
    }
  }
}
```

The appropriate constant_keyword allows for fast filters

Creating a data stream

- **Step 1:** create *component templates*
 - make sure you have an `@timestamp` field
- **Step 2:** create a data stream-enabled *index template*
- **Step 3:** create the *data stream* by indexing documents

Step 1: create component templates

```
PUT _component_template/my-mappings
{
  "template": {
    "mappings": {
      "properties": {
        "@timestamp": {
          "type": "date",
          "format": "date_optional_time||epoch_millis"
        }
      }
    }
  }
}
```

Step 2: create an index template

- Make sure it is data stream enabled:

```
PUT _index_template/my-index-template
{
  "index_patterns": ["my-data-stream"],
  "data_stream": { },
  "composed_of": [ "my-mappings" ],
  "priority": 500
}
```

Step 3: index data

- Use `POST <stream>/_doc` or `PUT <stream>/_create/<doc_id>`
 - if you use `_bulk`, you must use the `create` action

request

```
POST my-data-stream/_doc
{
  "@timestamp": "2099-05-06T16:21:15.000Z",
  "message": "192.0.2.42 -[06/May/2099:16:21:15] \"GET /images/bg.jp...\""
}
```

response

```
{
  "_index": ".ds-my-data-stream-2021.05.19-000001",
  "_type": "_doc",
  "_id": "8wI7hXkBRefR4b-NXph_",
  ...
}
```

Changing a data stream

- Changes should be made to the ***index template*** associated with the stream
 - new backing indices will get the changes when they are created
 - older backing indices can have limited changes applied
- Changes to static mappings still require a reindex
- Before reindexing, use the resolve API to check for conflicting names:

```
GET /_resolve/index/new-data-stream*
```

Reindexing a data stream

- Set up a new data stream template
 - use the ***data stream API*** to create an empty data stream:

```
PUT /_data_stream/new-data-stream
```

- Reindex with **op_type** of **create**
 - can also use single backing indices to preserve order

```
POST /_reindex
{
  "source": {
    "index": "my-data-stream"
  },
  "dest": {
    "index": "new-data-stream",
    "op_type": "create"
  }
}
```

Summary: Data streams

Module 7 Lesson 2



Summary

- A ***data stream*** is a collection of backing indices behind an alias
- Data streams are ideal for time series data that grows quickly
- A data stream is designed for data that is ***written once*** and ***never updated***
- To create a data stream:
 - create a component template or templates
 - create an index template that is data stream enabled
 - index documents to the new data stream

Quiz

1. **True or False:** When `_bulk` writing to a data stream, you should use the `create` action
2. How do you configure an index template to use data streams?
3. Where should you perform changes to a data stream's configuration?

Data streams

Lab 7.2

Create a new data stream and convert an alias.



Index lifecycle management

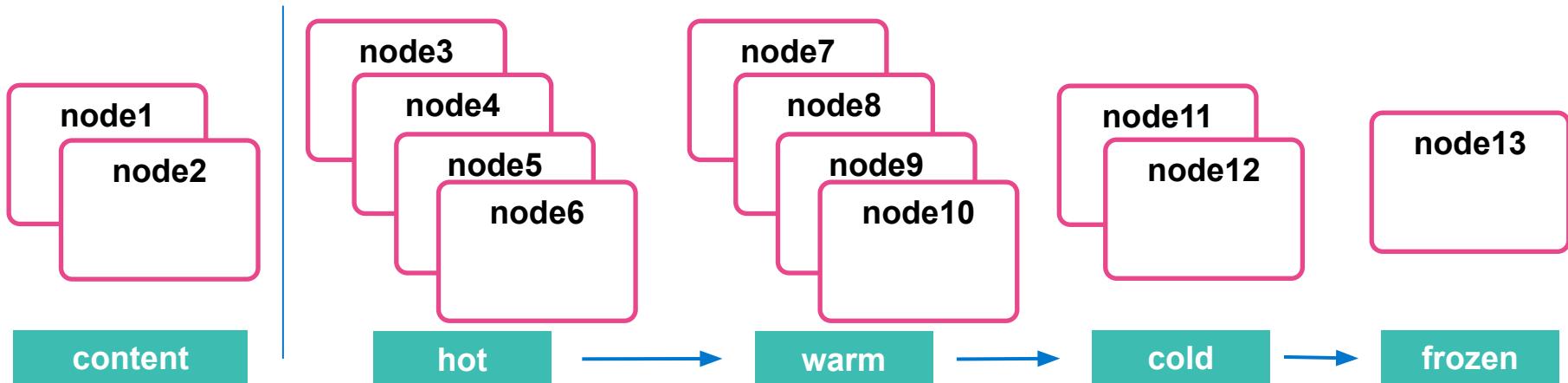
Module 7 Lesson 3



Data tiers

What is a data tier?

- A ***data tier*** is a collection of nodes with the same ***data role***
 - that typically share the same hardware profile
- There are ***five types of data tiers***:



Overview of the five data tiers

- The ***content tier*** is useful for static datasets
- Implementing a ***hot → warm → cold → frozen architecture*** can be achieved using the following data tiers :
 - **hot tier:** have the fastest storage for writing data and for frequent searching
 - **warm tier:** for read-only data that is searched less often
 - **cold tier:** for data that is searched sparingly
 - **frozen tier:** for data that is accessed rarely and never updated

Data tiers, nodes and indices

- Every node is ***all*** data tiers by default
 - change using the `node.roles` parameter
 - node roles are handled for you automatically on Elastic Cloud
- Data stream indices are created in the hot tier by default
- Move indices to colder tiers as the data gets older
 - define an ***index lifecycle management*** policy to manage this

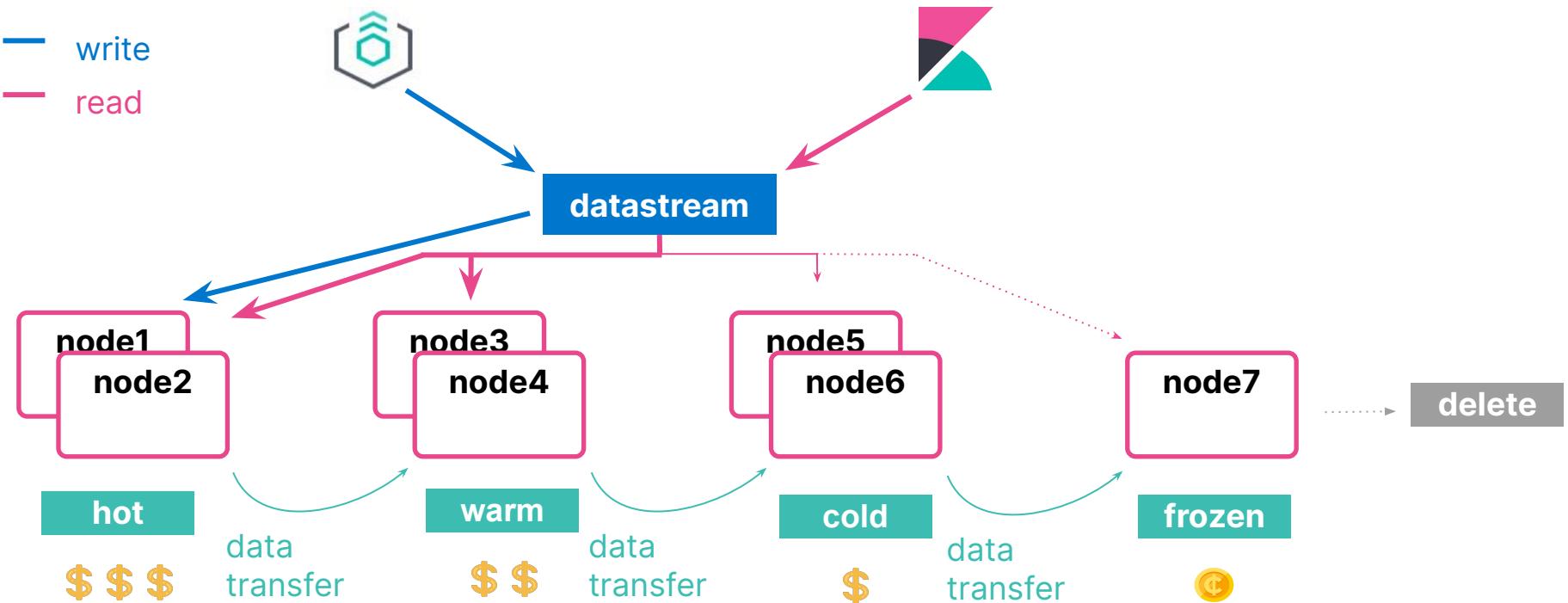
Configuring an index to prefer a data tier

- Set the ***data tier preference*** of an index using the **routing.allocation.include._tier_preference** property
 - data_content** is the default for all indices
 - you can update the property at any time
 - ILM can manage this setting for you (which we discuss next)

```
PUT logs-2021-03
{
  "settings": {
    "index.routing.allocation.include._tier_preference" : "data_hot"
  }
}
```

Index lifecycle management

Index lifecycle management (ILM)



ILM actions

- ILM consists of **policies** that trigger **actions**, such as:

Action	Description
rollover	create a new index based on age, size, or doc count
shrink	reduce the number of primary shards
force merge	optimize storage space
searchable snapshot	saves memory on rarely used indices
delete	permanently remove an index

ILM policy example

- During the ***hot phase*** you might:
 - create a new index every two weeks
- In the ***warm phase*** you might:
 - make the index read-only and move to warm for one week
- In the ***cold phase*** you might:
 - convert to a fully-mounted index, decrease the number of replicas, and move to cold for three weeks
- In the ***delete phase***:
 - the only action allowed is to delete the 28-days-old index
- Let's take a look at how to define this ILM policy...

Define the hot phase

- We want indices in the hot phase for 2 weeks:

```
PUT _ilm/policy/my-hwcd-policy
{
  "policy": {
    "phases": {
      "hot": {
        "actions": {
          "rollover": {
            "max_age": "14d"
          }
        }
      }
    }
  },
}
```

Hot phase Required

Store your most recent, most frequently-searched data in the hot tier. The hot tier provides the best indexing and search performance.

Rollover

Start writing to a new index when the current index reaches a certain size, document count, or age. Enables you to optimize performance and manage resource usage when working with time series data.

Note: How long it takes to reach the rollover criteria in the hot phase can vary. [Learn more](#)

Enable rollover

Maximum primary shard size

Maximum age

14

Rollover to a new index after 14 days

Define the warm phase

- We want the old index to move to the warm tier immediately and set the index as read-only:
 - ***data age*** is calculated ***from the time of rollover***

Move the warm tier
immediately after
rollover

```
"warm": {  
    "min_age": "0d",  
    "actions": {  
        "readonly": {}  
    }  
},
```

The screenshot shows the 'Warm phase' configuration in the Elasticsearch Settings interface. A yellow checkmark icon is positioned above the 'Warm phase' toggle switch. The toggle switch is turned on, indicated by a checked state. To the right of the switch is the text 'Move data into phase when:' followed by a text input field containing the value '0'. This input field is circled in red. Below the switch, a descriptive text explains: 'Move data to the warm tier when you are still likely to search it, but infrequently need to update it. The warm tier is optimized for search performance over indexing performance.' There is also a 'Advanced settings' link and a 'Replicas' section at the bottom.

Warm phase

Move data into phase when: 0

Move data to the warm tier when you are still likely to search it, but infrequently need to update it. The warm tier is optimized for search performance over indexing performance.

Advanced settings

Replicas

Set the number of replicas. Remains the same as the previous

Define the cold phase

- After one week of warm, move the index to the cold phase, and convert the index:

```
"cold": {  
    "min_age": "7d",  
    "actions": {  
        "searchable_snapshot" : {  
            "snapshot_repository" :  
                "my_snapshot"  
        }  
    }  
},
```

Move to cold 7 days after rollover

Cold phase Move data into phase when days old (7)

Move data to the cold tier when you are searching it less often and don't need to update it. The cold tier is optimized for cost savings over search performance.

Searchable snapshot Convert to a fully-mounted index that contains a complete copy of your data and is backed by a snapshot. You can reduce the number of replicas and rely on the snapshot for resiliency. [Learn more](#)

Convert to fully-mounted index

Snapshot repository my_snapshot (x)
Each phase uses the same snapshot repository.

Define the delete phase

- Let's delete the data four weeks ***after rollover***.
 - which means the documents lived for 14 days in hot
 - then 7 days in warm
 - then 21 days in cold

```
"delete": {  
    "min_age": "28d",  
    "actions": {  
        "delete": {}  
    }  
}
```

Delete phase [Remove](#)

Move data into phase when: days

Delete data you no longer need.

Delete 28 days from rollover

Applying the policy

- Create a component template
- Link your ILM policy using the setting:
 - **index.lifecycle.name**

```
PUT _component_template/my-ilm-settings
{
  "template": {
    "settings": {
      "index.lifecycle.name": "my_hwcd_policy"
    }
  }
}
```

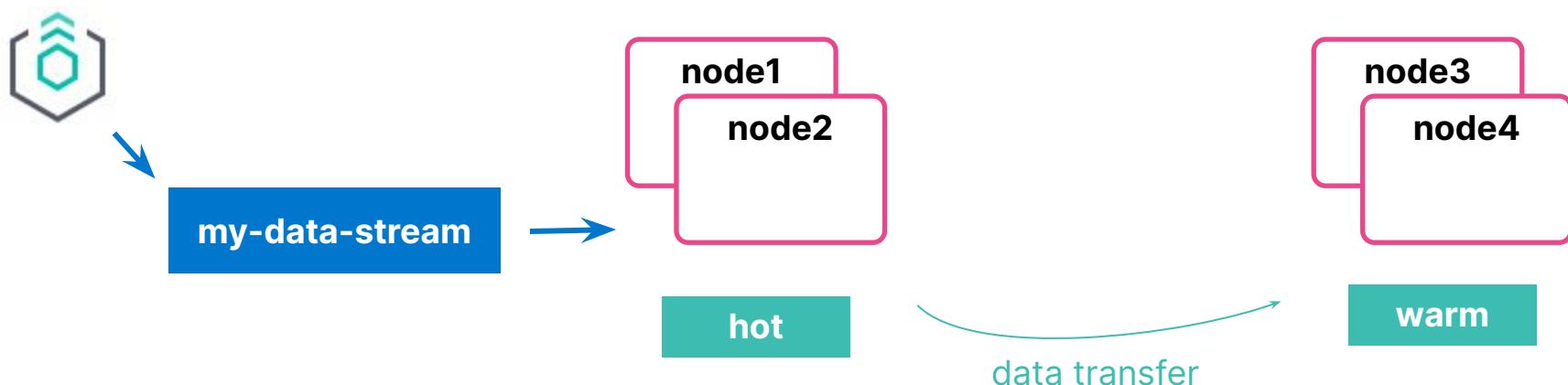
Create an index template

- Use other components relevant to your stream

```
PUT _index_template/my-ilm-index-template
{
  "index_patterns": ["my-data-stream"],
  "data_stream": { },
  "composed_of": [ "my-mappings",   "my-ilm-settings" ],
  "priority": 500
}
```

Start indexing documents

- ILM takes over from here!
- When a rollover happens, the number of indices is incremented
 - the new index is set as the write index of the data stream
 - old indices will automatically move to other tiers



Troubleshooting lifecycle rollovers

- If an index is not green, it will not move to the next phase
- The default poll interval for a cluster is 10 minutes
 - can change with `indices.lifecycle.poll_interval`
- Check the server log for errors
- Make sure you have the appropriate data tiers for migration
- **Reminder:** use a template to apply a policy to new indices
- Get detailed information about ILM status with:

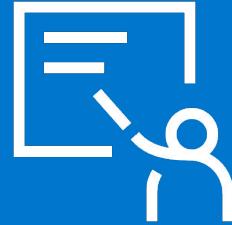
```
GET <data-stream>/_ilm/explain
```

Agent and ILM

- Agent uses ILM policies to manage rollover
- By default, Agent policies:
 - remain in the hot phase forever
 - never delete
 - indices are rolled over after 30 days or 50GB
- The default Agent policies can be edited with Kibana

Summary: Index lifecycle management

Module 7 Lesson 3



Summary

- **Data tiers** allow Elasticsearch to manage where data is stored
- Indices are automatically placed on the **data_content** tier, unless otherwise specified
- **Index lifecycle management (ILM)** enables you to easily configure and automate the rollover pattern
- Lifecycle policies define “what to do” and “when to do it”
 - each policy can be broken up into four phases: hot, warm, cold, and delete
- You can define lifecycle policies using the API or Kibana

Quiz

1. **True or False:** When a hot index rolls over, write requests are automatically sent to the newly created backing index
2. Name the five ILM phases
3. If you want an index in the warm phase for 5 days then have it move to the cold phase, what would you set `min_age` to in the cold phase?

Index lifecycle management

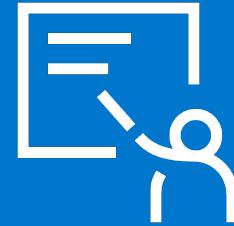
Lab 7.3

Deploy an ILM policy to manage
rollovers.



Searchable snapshots

Module 7 Lesson 4



Dealing with cold/frozen data

- As your data streams and time series data grows, your storage and memory needs increase
 - at the same time, ***the utility of that older data decreases***
- You could delete this older data
 - but if it remains valuable, it is preferable to keep it available
- There is an action available in the cold phase called ***searchable snapshot***
 - but we need to discuss what a snapshot is first...

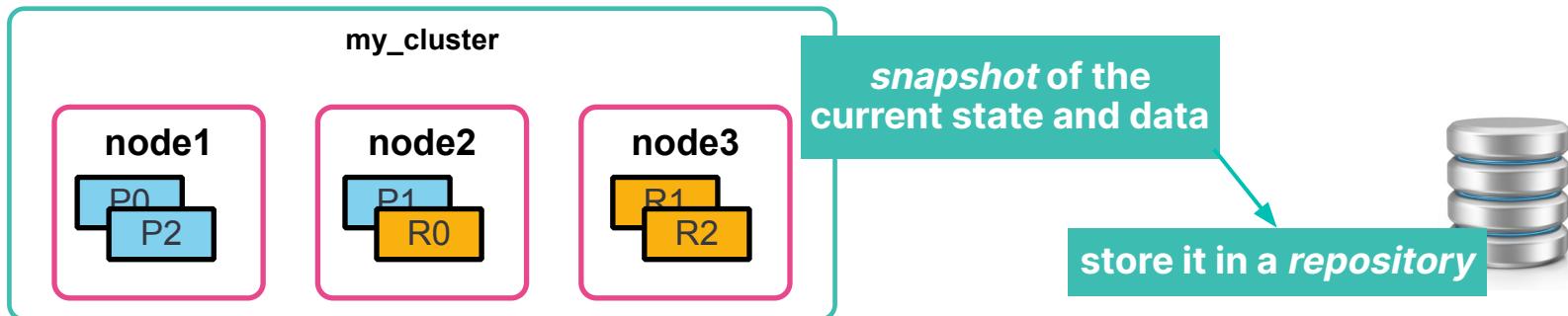
Snapshots

Disaster recovery

- You already know about replica shards:
 - they provide redundant copies of your documents
 - that is ***not the same as a backup!***
- Replicas do not protect you against catastrophic failure
 - you will need to keep a complete backup of your data

Snapshot and restore

- **Snapshot and restore** allows you to **create and manage backups** taken from a running Elasticsearch cluster
 - takes the current state and data in your cluster and saves it to a **repository**
- Repositories can be on a local shared file system or in the cloud
 - the Elasticsearch Service performs snapshots automatically



Types of repositories

- The backup process starts with the creation of a **repository**
 - different types are supported:

Shared file system	define path.repo in every node
Read-only URL	used when multiple clusters share a repository
repository-s3 plugin	for AWS S3 repositories
repository-azure plugin	for Microsoft Azure storage
repository-gcs plugin	for Google Cloud Storage
repository-hdfs plugin	store snapshots in Hadoop

Setting up a repository

- Cloud deployments come with free repositories preconfigured
- Use Kibana to register a repository:

Register repository

Repository name

A unique name for the repository.

Name

my-repo

Repository type

Elasticsearch supports file system and read-only URL repositories. Additional types require plugins. [Learn more about plugins.](#)



Shared file system

[Learn more](#)

✓ Selected



Read-only URL

[Learn more](#)

Select

Taking a snapshot manually

- Once the repository is configured, you can take a snapshot
 - using the `_snapshot` endpoint or the UI
 - snapshots are a ***“point-in-time” copy*** of the data and ***incremental***
- Can back up only certain indices
- Can include cluster state

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

Automating snapshots

- The `_snapshot` endpoint can be called manually
 - every time you want to take a snapshot
 - at regular intervals using an external tool
- Or, you can automate snapshots with ***Snapshot lifecycle management (SLM)*** policies
 - policies can be created in Kibana
 - or using the `_slm` API

Create policy



Restoring from a snapshot

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

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

- Can also restore using Kibana:

Restore 'daily-snap-jwh-r8drtlob2clxom8ioq'

The image shows the Kibana Restore wizard interface. It consists of three horizontal tabs labeled '1 Logistics', '2 Index settings', and '3 Review'. Below the tabs, there is a section titled 'Restore details' containing the text 'Restore 'daily-snap-jwh-r8drtlob2clxom8ioq''. To the right of this section is a link 'Snapshot and Restore docs'. At the bottom of the interface, there is a yellow bar with the text '⚠ This snapshot contains data streams'.

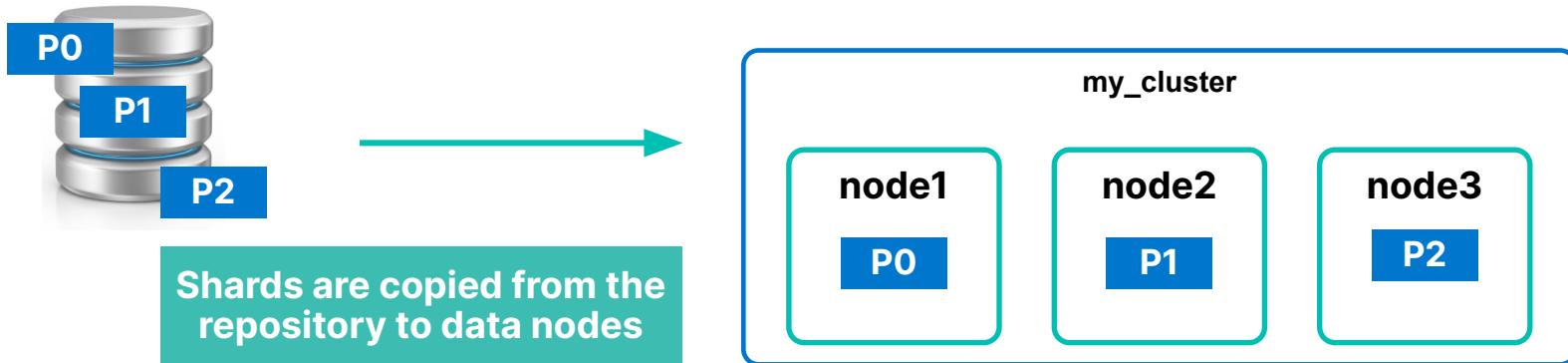
Searchable snapshots

Searchable snapshots

- There is an action available in the cold and frozen phases called ***searchable snapshot***
- Benefits include:
 - search frozen data in a very cost-effective fashion
 - reduce storage costs (no replica shards needed)
 - use the same mechanism you are already using (snapshots)

How searchable snapshots work

- Searching a searchable snapshot index is the same as searching any other index
 - when a snapshot of an index is searched, the index must get **mounted** locally in a **temporary index**
 - the shards of the index are allocated to data nodes in the cluster



Setting up searchable snapshots

- In the cold or frozen phase, you configure a searchable snapshot by ***selecting a registered repository***:

Cold phase

Move data into phase when: 7 days old ⓘ

Move data to the cold tier when you are searching it less often and don't need to update it. The cold tier is optimized for cost savings over search performance.

Searchable snapshot

Convert to a fully-mounted index that contains a complete copy of your data and is backed by a snapshot. You can reduce the number of replicas and rely on the snapshot for resiliency.

[Learn more ↗](#)

Convert to fully-mounted index

Advanced settings

Snapshot repository

my_snapshot

Each phase uses the same snapshot repository.

Delete data after this phase ⚡

Add searchable snapshots to ILM

- Edit your ILM policy to add a searchable snapshot to your ***cold or frozen phase***
 - ILM will automatically handle the index mounting
 - the cold phase uses ***fully mounted*** indices
 - the frozen phase uses ***partially mounted*** indices
- If the ***delete phase*** is active, it will delete the searchable snapshot by default:
 - turn off with "**`delete_searchable_snapshot": false`**"
- If your policy applies to a data stream, the searchable snapshot will be included in searches by default

Benefits of searchable snapshots

- Perfect for managing a large amount of historical data
 - functions as a normal index for searches and shard allocation
- No replicas needed, the snapshot itself acts as the replica
- Dramatically reduced storage space and cost
- Search speed is slower, but still functional

Summary: Searchable snapshots

Module 7 Lesson 4



Summary

- The ***Snapshot and Restore API*** enables you to create and manage backups taken from a running Elasticsearch cluster
- Snapshots are a “point-in-time” copy of the data
- You can automate snapshots with ***Snapshot lifecycle management (SLM)***
- ***Searchable snapshots*** allow you to keep older data on the cluster without using a lot of resources
- Searchable snapshots can be automated as part of an Index Lifecycle Management policy

Quiz

1. What is the only setting needed when configuring a searchable snapshot in ILM?
2. **True or False:** Searchable snapshots need replica shards
3. **True or False:** You can take a snapshot of your entire cluster in a single REST request

Searchable snapshots

Lab 7.4

**Set up a repository and add
searchable snapshots to your ILM
policy.**



More Resources

- More details on the various ILM actions:
 - www.elastic.co/guide/en/elasticsearch/reference/current/ilm-actions.html
- Data management:
 - www.elastic.co/blog/elasticsearch-data-lifecycle-management-with-data-tiers
- Frozen tier:
 - www.elastic.co/blog/introducing-elasticsearch-frozen-tier-searchbox-on-s3
- Searchable snapshots:
 - www.elastic.co/blog/introducing-elasticsearch-searchable-snapshots

Elasticsearch Engineer: Agenda

- Module 1: Getting started
- Module 2: Data modeling
- Module 3: You know, for search
- Module 4: Data processing
- Module 5: Aggregations
- Module 6: The one about shards
- Module 7: Data management
- **Module 8: Cluster management**

Cluster management

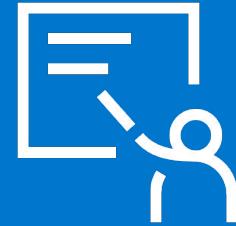
Module 8

Topics

- Multi cluster operations
- Troubleshooting
- Optimizing search performance

Multi cluster operations

Module 8 Lesson 1



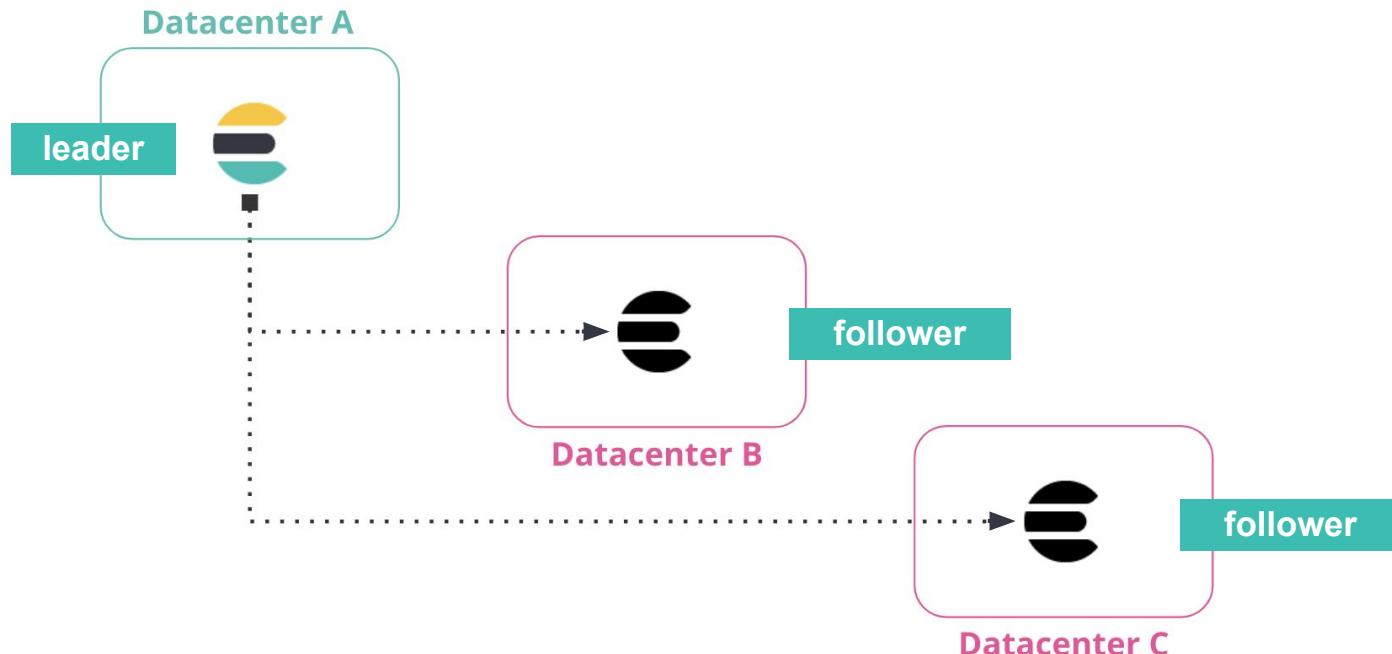
Cross-cluster replication

Cross-cluster replication

- ***Cross-cluster replication*** (CCR) enables replication of indices across clusters
- Uses an ***active-passive model:***
 - you index to a ***leader index***,
 - the data is replicated to one or more read-only ***follower indices***
- Let's take a look at some use cases for CCR...

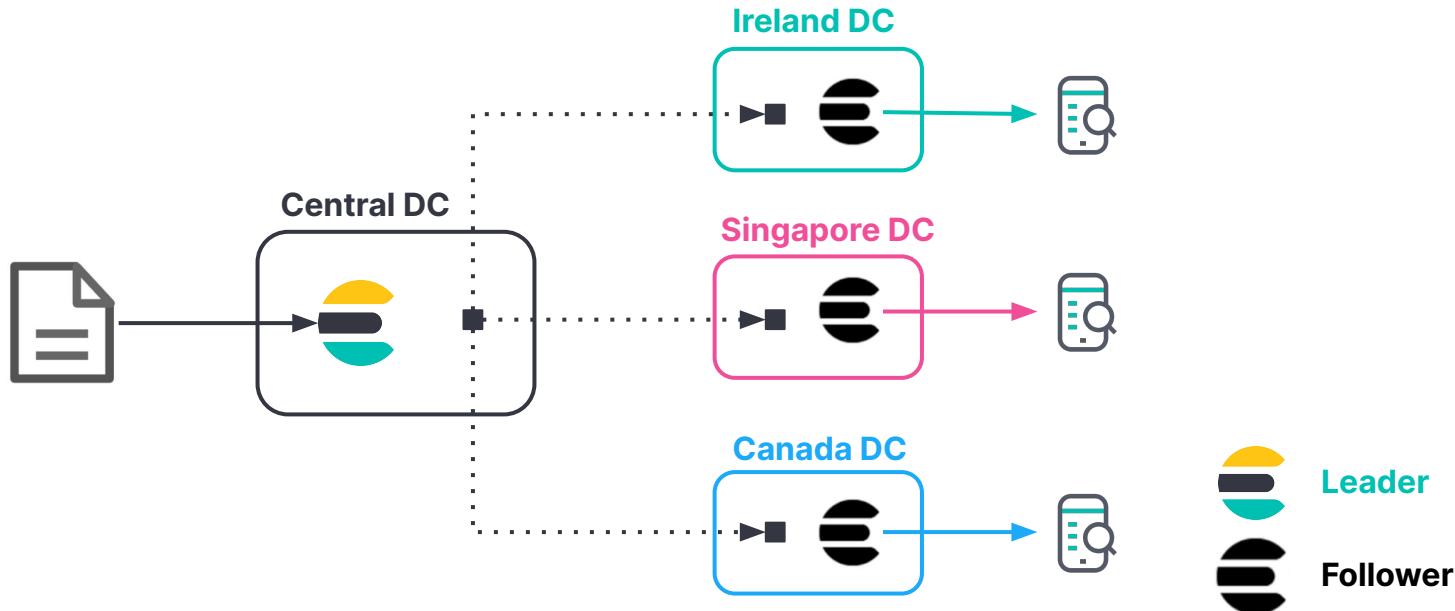
Disaster recovery and high availability

- Replicate data from one datacenter to one or more other datacenters:



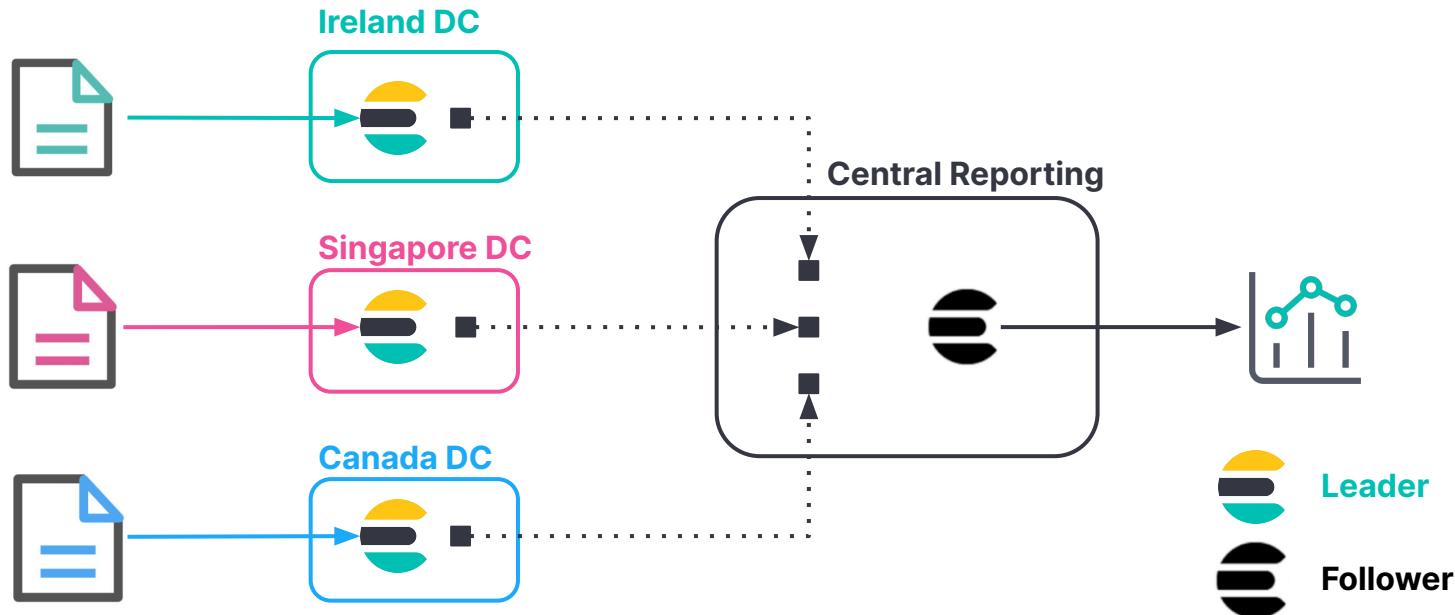
Data locality

- Bring data closer to your users or application servers to reduce latency and response time:



Centralized reporting

- Replicate data from many smaller clusters to a centralized reporting cluster:



Replication is pull-based

- The replication is ***driven by the follower index***
 - the follower watches for changes in the leader index
 - operations are ***pulled by the follower***
 - causes no additional load on the leader
- Replication is done at the ***shard level***
 - the follower has the same number of shards as the leader
 - all operations on each leader shard are replicated on the corresponding follower shard
- Replication appears in ***near real-time***

Configuring CCR

- Configure a remote cluster using Kibana
 - the follower configures the leader as a remote cluster***
- You need a user that has the appropriate roles, and configure the appropriate TLS/SSL certificates:

www.elastic.co/guide/en/elasticsearch/reference/current/CCR-getting-started.html

Remote Clusters

Add a remote cluster

Search...

<input type="checkbox"/> Name ↑	Status	Mode	Addresses	Connections	Actions
<input type="checkbox"/> cluster2	✓ Connected	default	server4:9304	1	

Rows per page: 20 ▾

< 1 >

You can also use the `_cluster settings API`

Configuring CCR

- Use the ***Cross-Cluster Replication UI***, or the **`_CCR endpoint`**
 - create a follower index that references both the remote cluster and the leader index

Add follower index

Remote cluster
The cluster that contains the index to replicate.

Leader index
The index on the remote cluster to replicate to the follower index.
Note: The leader index must already exist.

Follower index
A unique name for your index.

Request

```
PUT copy_of_the_leader_index/_CCR/follow
{
  "remote_cluster" : "cluster2",
  "leader_index" : "index_to_be_replicated"
}
```

Auto-following functionality

- Useful when your leader indices automatically rollover to new indices
 - you **follow a pattern** (instead of a static index name)

Add auto-follow pattern

Name
A unique name for the auto-follow pattern.

Remote cluster
The remote cluster to replicate leader indices from.

Leader indices
One or more index patterns that identify the indices you want to replicate from the remote cluster. As new indices matching these patterns are created, they are replicated to follower indices on the local cluster.

Note: Indices that already exist are not replicated.

Follower indices (optional)
A custom prefix or suffix to apply to the names of the follower indices so you can more easily identify replicated indices. By default, a follower index has the same name as the leader index.

request

```
PUT _ccr/auto_follow/logs
{
  "remote_cluster" : "cluster2",
  "leader_index_patterns" : [ "logs*" ],
  "follow_index_pattern" : "{leader_index}-copy"
}
```

Index patterns
Logs*

Spaces and the characters \ / ? < > | are not allowed.

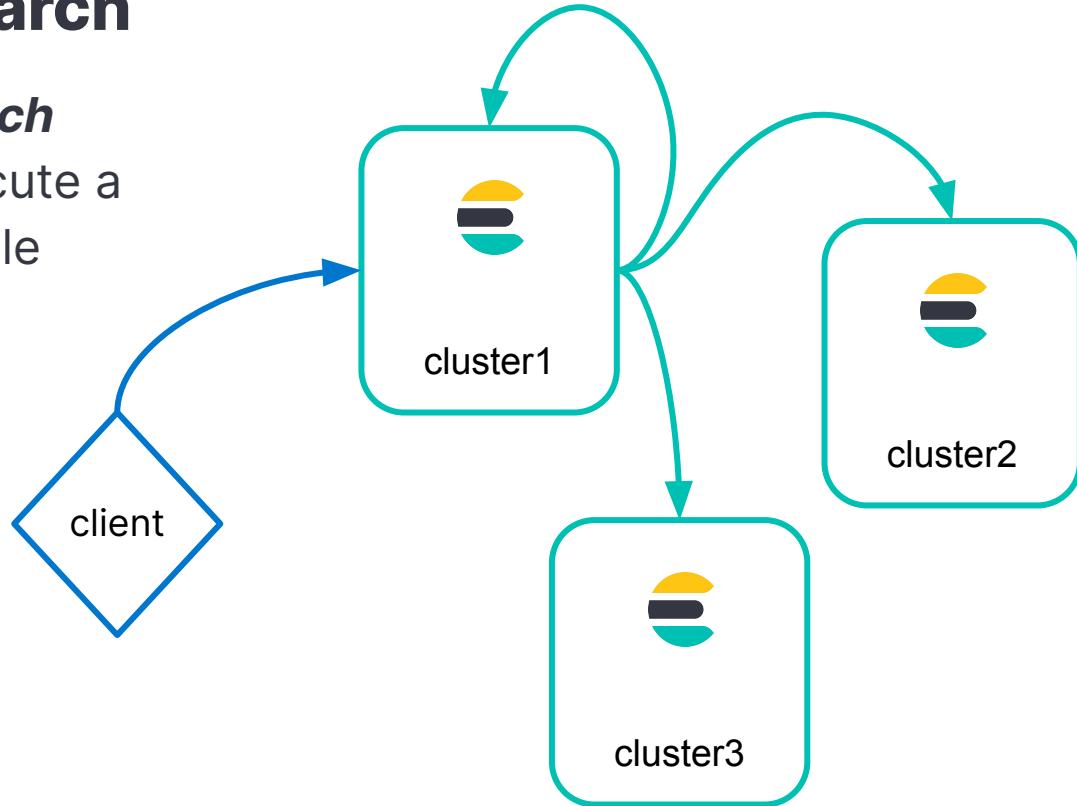
Prefix
Suffix
-copy

Spaces and the characters \ / ? , < > | * are not allowed.

Cross-cluster search

Cross-cluster search

- ***Cross-cluster search*** enables you to execute a query across multiple clusters



Searching remotely

- To search an index on a remote cluster, prefix the index name with the remote cluster name

remote cluster name

index name

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

Searching multiple clusters

- To perform a search across multiple clusters, list the cluster names and indices
 - you can use wildcards for the names of the remote clusters

request

```
GET blogs,EMEA_DE_cluster:blogs,APAC_*:blogs/_search
{
  "query": {
    "match": {
      "title": "network"
    }
  }
}
```

Search response

- All results retrieved from a remote index will be prefixed with the remote cluster's name

response

```
"hits": [  
  {  
    "_index": "EMEA_DE_cluster:blogs",  
    "_id": "3s1CKmIBCLh5xF6i7Y2g",  
    "_score": 4.8329377,  
    "_source": {  
      "title": "Using Logstash to ...",  
      ...  
    }  },  
  {  
    "_index": "blogs",  
    "_id": "Mc1CKmIBCLh5xF6i7Y",  
    "_score": 4.561167,  
    "_source": {  
      "title": "Brewing in Beats: New ...",  
      ...  
    }  },
```

Summary: Multi cluster operations

Module 8 Lesson 1



Summary

- Cross-cluster **replication** enables you to replicate indices between clusters
- Replication is pull-based, where the follower pulls all the operations that occurred on the leader index
- You can configure CCR to replicate a pattern of index names using the auto-follow functionality
- Search across multiple clusters using cross-cluster **search**

Quiz

1. What are some popular use cases for cross-cluster replication?
2. **True or False:** You can write data into a follower index
3. **True or False:** Cross-cluster search can only be configured between Elastic Cloud deployments

Multi cluster operations

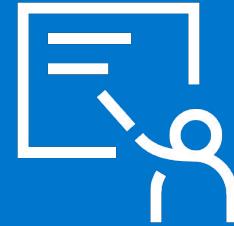
Lab 8.1

Setup cross-cluster replication and search



Troubleshooting

Module 8 Lesson 2



Using the CAT APIs

Where is the trouble?

- The ***compact and aligned text (CAT) API*** can help:
 - `_cat/thread_pool`
 - `_cat/shards`
 - `_cat/health`
- CAT APIs are only for human consumption
 - don't use CAT APIs in your code
 - use the JSON API instead

Threads

- Use `GET _cat/thread_pool` to view thread pool stats for each node in the cluster
- A full queue may be good or bad (“It depends!”)
 - OK if bulk indexing is faster than ES can handle
 - bad if search queue is full

node_name	name	active	queue	rejected
node1	analyze	0	0	0
node1	ccr	0	0	0
node1	fetch_shard_started	0	0	0
node1	fetch_shard_store	0	0	0
node1	flush	0	0	0
...				

Hot threads and tasks

- If you do have thread pools that seem too busy, try looking at the running tasks and hot threads:

Command	Response
<code>GET _tasks</code>	the running tasks on the cluster
<code>GET _cluster/pending_tasks</code>	any cluster-level changes that have not yet executed
<code>GET _nodes/hot_threads</code>	threads using high CPU volume and executing for a long time

Where are the shards?

- A simple list is available with `GET _cat/shards`
 - details are per index
- What if the `state` is `UNASSIGNED`?

index	shard	prirep	state	docs	store	ip	node
blogs	0	p	STARTED	1023	17.1mb	10.1.0.1	node1
blogs	0	r	STARTED	1023	17.1mb	10.1.0.2	node2
blogs	1	p	STARTED	1044	17.2mb	10.1.0.2	node2
blogs	1	r	STARTED	1044	17.2mb	10.1.0.1	node1

Cluster Health

- `GET _cat/health` returns the ***health status*** of the cluster
- Either green, yellow, or red and exists at three levels: shard, index, and cluster
 - **green**: all shards are allocated
 - **yellow**: all primaries are allocated, but at least one replica is not
 - **red**: at least one primary shard is not allocated in the cluster

cluster	status	node.data	shards	unassigned
my_cluster	yellow	1	107	2

Getting to green

- **Red** means *missing primaries*
 - look for allocation issues . . .
 - `GET _cluster/allocation/explain` can help
- **Yellow** means *missing replicas*
 - issue is likely that we asked for more replicas than the number of nodes in the cluster can accommodate
 - `auto_expand_replicas` setting can be helpful

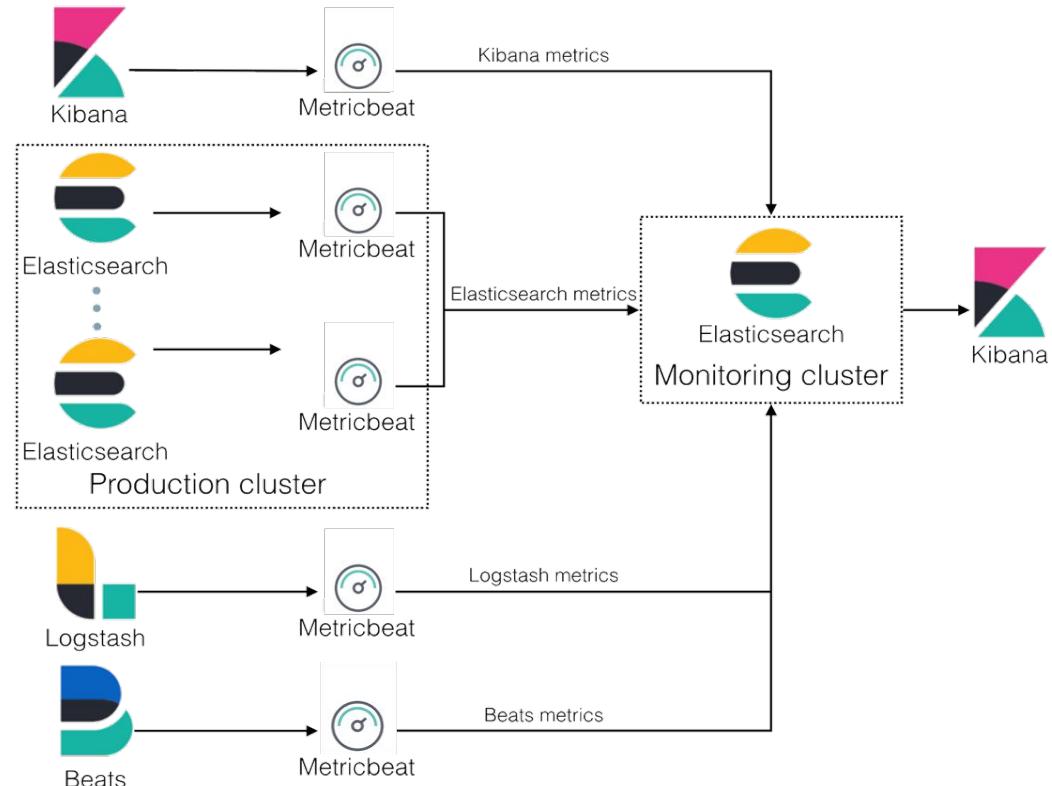
Monitoring your clusters

Monitoring the Elastic Stack

- To monitor the Elastic Stack, you can use... ***the Elastic Stack!***
 - **Metricbeat** to collect metrics
 - **Filebeat** to collect logs
- We recommend using a ***dedicated cluster*** for monitoring
 - to reduce the load and storage on the monitored clusters
 - to keep access to Monitoring even for unhealthy clusters
 - to support segregation of duties (separate security policies)

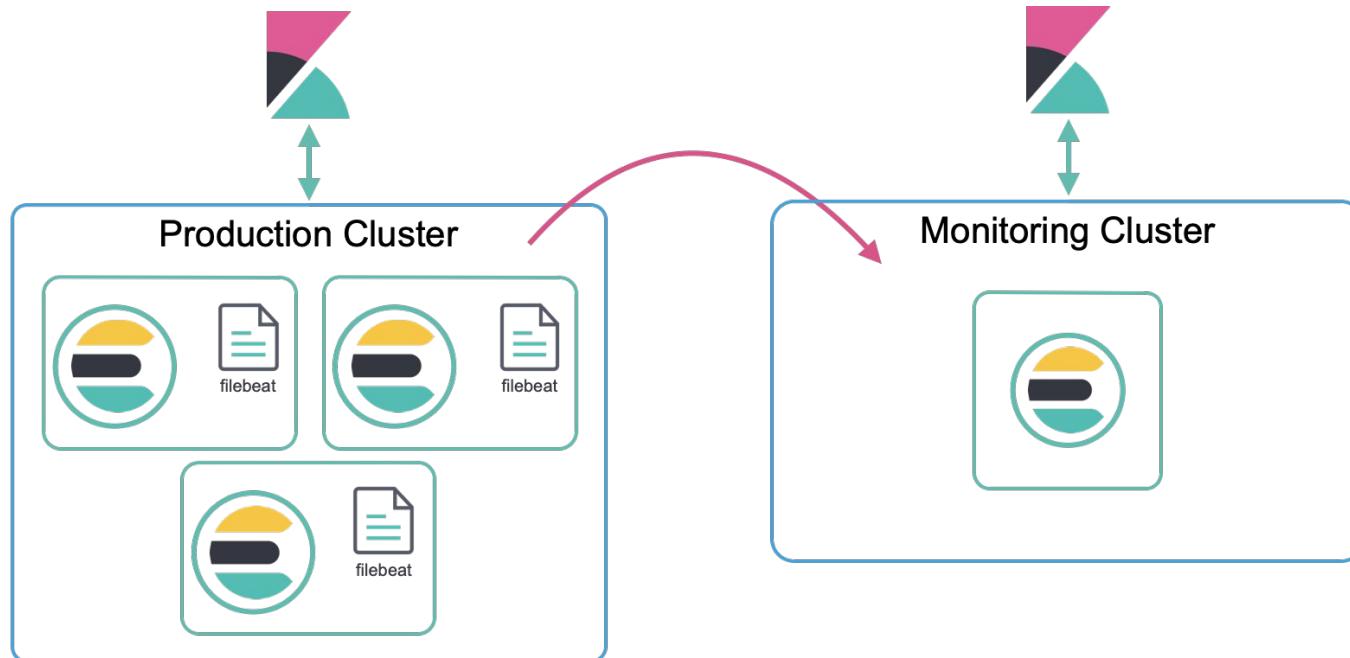
Monitoring with Metricbeat

- Deploy **Metricbeat** on every host you want to monitor:



Monitoring logs with Filebeat

- Use **Filebeat** to ship Elasticsearch logs to your monitoring cluster:



Configuring monitoring on Elastic Cloud

- Enable monitoring via the Cloud console
 - select the deployment used to monitor the Stack

Logs and metrics

Ship to a deployment

Ship logs and monitoring metrics to a deployment where they are stored separately. Then, you can enable additional log types, search the logs, configure retention periods, and use Kibana to view monitoring visualizations. [Learn more](#)

Shipping data to

- 237e6b [My deployment](#)

Data being shipped

Type	Enabled	View data
Logs	● Enabled	View
Metrics	● Enabled	View

[Edit](#) [Stop](#)

Summary: **Troubleshooting**

Module 8 Lesson 2



Summary

- The **CAT API** has many tools to help determine cluster health
- The overall cluster health depends on any missing shards
- Monitor your cluster to help maintain health
- Once you start collecting monitoring data, you can use the built-in Kibana tools to analyze that data

Quiz

1. **True or False:** You can use Metricbeat to monitor all components of the Elastic Stack
2. How would you check if the index thread pool queue is full?
3. **True or False:** Having an **UNASSIGNED** primary shard means your cluster is not in a good state

Troubleshooting

Lab 8.2



Monitor your cluster to look for issues.

Optimizing search performance

Module 8 Lesson 3



Fixing slow searches

Why is my search taking so long?

- Cluster is overloaded
- Too many shards
- Poorly written code
- Too many irrelevant results

Scripting and search performance

- Scripts are incredibly useful
 - but they can not use Elasticsearch's index structures or related optimizations
- Make search faster by ***transforming data during ingest*** instead
 - slower index speeds, but faster query speeds

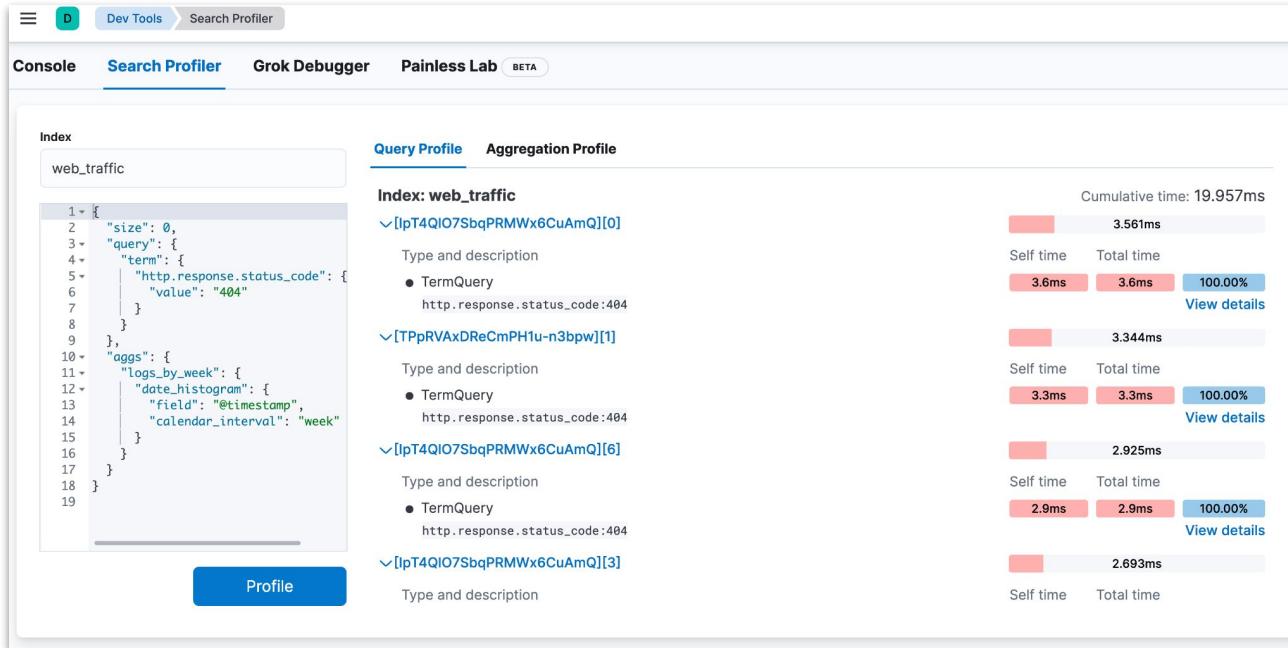
Elasticsearch can tell you!

- The **Search Slow Log** details long-running searches
 - tunable by index to define what “slow” means

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

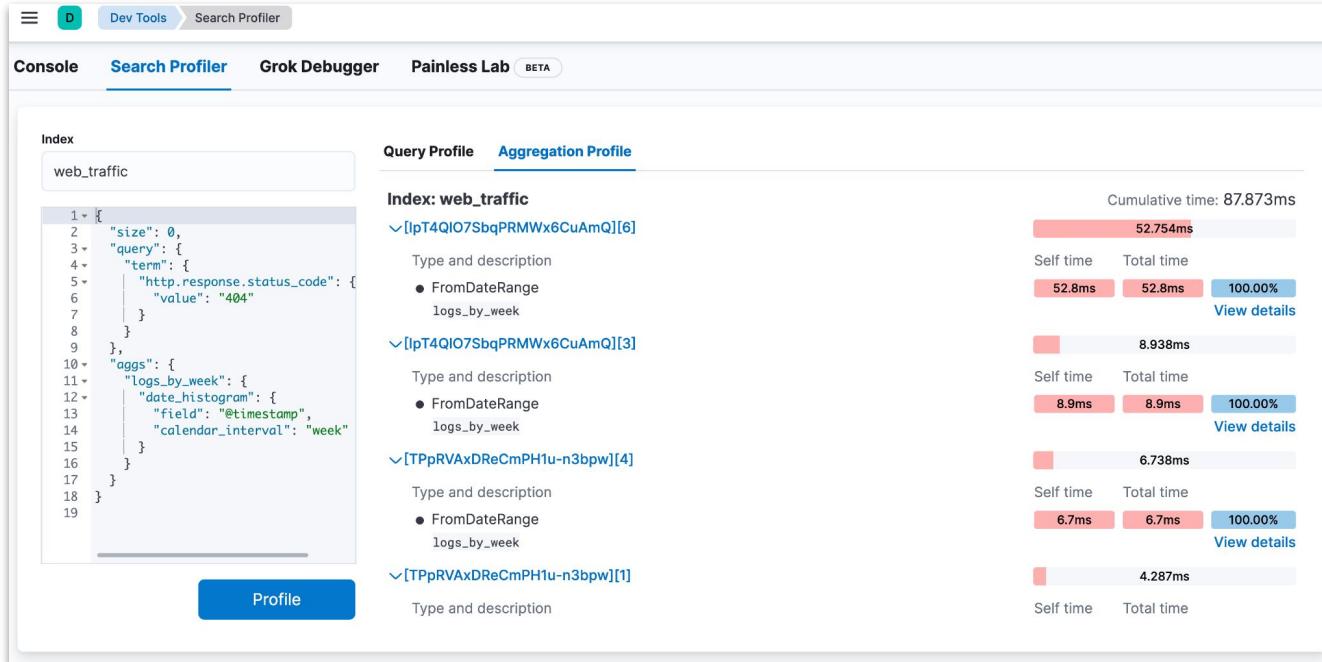
The Search Profiler

- Visualization of search performance:



The Search Profiler

- Both queries and aggregations are profiled per shard:



The profile flag

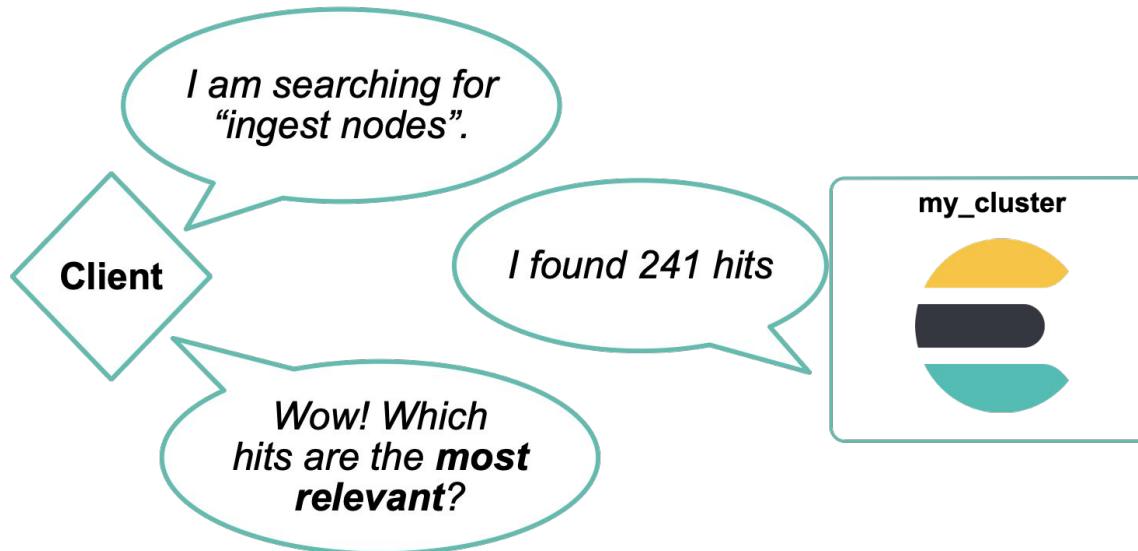
- Set **profile** to **true** to profile your search
 - you can copy-and-paste the response into Search Profiler

```
GET web_traffic/_search
{
    "size": 0,
    "profile": true,
    "aggs": {
        "top_os": {
            "terms": {
                "field": "user_agent.os.full.keyword",
                "size": 20
            }
        }
    ...
}
```

Relevance tuning

What is relevance?

- Calculating the ***quality*** of a matching document
 - filters just include or exclude hits,
 - but queries calculate a ***score***



Relevance tuning

- Get the “best” results at the top of your hit list
 - no need to come back for the second or third page of hits
- It’s all about manipulating the `_score`
- Let’s look at some examples of relevance tuning...

Per-field boosting

- A match in a blog's title might carry more weight than in the content field
 - you can ***boost a field*** using the caret symbol:

```
GET blogs/_search
{
  "query": {
    "multi_match": {
      "query": "shay banon",
      "fields": [
        "title^3",
        "content"
      ]
    }
  }
}
```

The score of the title
field is *multiplied* by 3

Boosting indexes

- If you are searching multiple indices, you can give a ***boost*** to hits from a specific index
 - suppose you want more recent blogs to score higher:

```
GET blogs*/_search
{
  "indices_boost": [
    { "blogs-2022": 2.0 },
    { "blogs-2021": 1.5 }
  ]
}
```

blogs from 2022 will have
their `_score` doubled

blogs from 2021 will have a
50% increase in `_score`

Constant scoring

- The `constant_score` query assigns a constant value to the `_score`
 - all hits have the same `_score`
 - less time is spent calculating weights

```
GET blogs/_search
{
  "query": {
    "constant_score": {
      "filter": {
        "term": { "authors.first_name": "monica" }
      },
      "boost": 1.5
    }
  }
}
```

All blogs by "*monica*" will have a `_score` of 1.5

Scripted scoring

- The `script_score` query uses Painless scripts to calculate the **score**

```
GET my_web_logs/_search
{
  "query": {
    "script_score": {
      "query": {
        "match": { "message": "elasticsearch" }
      },
      "script": {
        "source": "_score / doc['resp_ms'].value"
      }
    }
  }
}
```

Explain your query

- Your query is almost always going to be rewritten before executing
- Use the **explain** flag to see the results in detail

```
GET blogs/_search?explain=true
{
  "query": {
    "multi_match" : {
      "query" : "shay banon",
      "fields" : [ "title^3", "content" ]
    }
  }
}
```

Ways to improve searches

The node query cache

- Results in the filter context are cached in the ***node query cache***
 - allows for very fast lookups of commonly-used filters
 - one more advantage of using filters!
- The cache holds up to 10,000 queries (or up to 10% of the heap)
 - these defaults can be modified with the **`indices.queries.cache.size`** setting

The shard request cache

- Search results may be cached in the ***shard request cache***
 - only requests with **size=0** are cached, which makes it useful for aggregations
- You do not have to do anything else special to use the cache:

```
GET web_traffic/_search
{
  "size": 0, ←
  "aggs": {
    "logs_by_day": {
      "date_histogram": {
        "field": "@timestamp",
        "calendar_interval": "day"
      }
    }
  }
}
```

Because **size=0**, this result can be cached for faster retrieval

Query performance

- Recognize ***expensive queries***
 - certain types of queries will generally execute slowly due to the way they are implemented
- **fuzzy, regex, wildcard** queries are expensive
 - solution: run those queries on fields of type **wildcard**
- Queries using scripts are expensive
 - solution: fix your data when you index it

Aggregation performance

- Improving aggregation performance often comes down to **narrowing the scope** of your search
 - solution 1: use a query
 - solution 2: use a `sampler` or `diversified_sampler` agg to filter out a sample of the top hits
 - solution 3: use a Kibana filter and runtime field with random values to filter out a random sampling of the hits

Summary: Optimizing search performance

Module 8 Lesson 3



Summary

- Search efficiency is key to keeping your cluster healthy
- Various tuning techniques are available to improve relevance
- Caching can speed up repeated searches significantly
- The Search Profiler and Profile API can help identify poorly-crafted searches

Quiz

1. **True or False:** The filter context can cache query responses.
2. What query is commonly used with boosting?
3. **True or False:** The Profile API is the most readable way to locate causes of poor search performance

Optimizing search performance

Lab 8.3



Profiling your searches with various methods.

More Resources

- Tune for search
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-search-speed.html>
- Tune for indexing
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-indexing-speed.html>
- Monitoring Elasticsearch
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/monitor-elasticsearch-cluster.html>

Conclusion

Resources

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

Elastic Certification

Validate Skills

Apply practical knowledge with performance-based testing



Elasticsearch
Engineer

Boost Productivity

Overcome obstacles with confidence and ease as you move from dev to production



Data Analyst

Open New Doors

Enhance professional visibility and expand opportunities

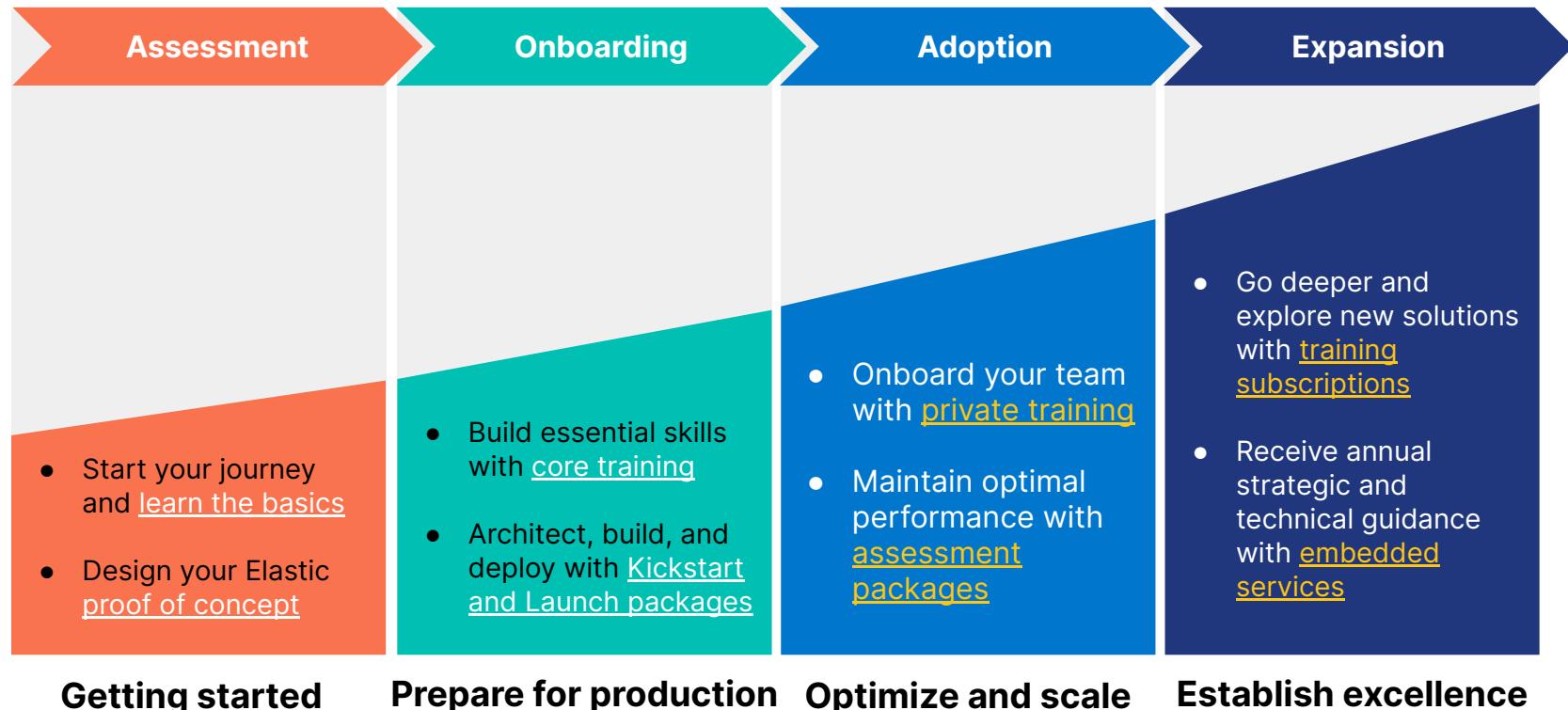
Join the Community

Become part of an exclusive network of over 1,000 certified professionals in nearly 70 countries



Observability
Engineer

Elastic Enablement Journey





Thank You.

Please complete
the survey

Quiz Solutions

1.1 Introduction to the Elastic Stack

1. Elasticsearch
2. A node.
3. False. You can use Kibana's Discover to query and filter your data

1.2 Data in

1. True
2. a. Kibana's Data Visualizer (easiest) or Filebeat or Logstash
b. Fleet (easiest) or Metricbeat
3. True

1.3 Information out

1. KQL and Lucene
2. False. It uses **or**
3. 10

2.1 Strings in Elasticsearch

1. True
2. False

2.2 Mapping

1. True
2. False. A mapping cannot be changed
3. Creating a custom mapping will produce savings in search and index speed, as well as memory and storage requirements

2.3 Text analysis

1. True
2. False. Only one tokenizer allowed

2.4 Types and parameters

1. format
2. False. It's not added to the _source
3. False. You can do neither

3.1 Searching with the Query DSL

1. From and size
2. False. You can change the sort order with the sort parameter
3. 10

3.2 More queries

1. False. The Query DSL enables you to do any kind of query
2. False. A filter clause has no effect on a document's score
3. Filter context. You are looking for an exact match

3.3 Developing search applications

1. False. Use _script
2. Mustache
3. 1 second

4.1 Changing Data

1. True
2. False. They run in order
3. This request configures the default pipeline of the blogs_fixed index so that any document that is indexed into blogs_fixed will pass through blogs_pipeline first

4.2 Enriching Data

1. True
2. To create and enrich index (1) create an enrich policy then (2) execute it
3. False. This is a read only index. If you want to update the enrich index, you will need to re-execute the policy

4.3 Painless Scripting and Runtime Fields

1. Painless and mustache
2. True
3. False, you can directly define a runtime mapping at query time

5.1 Metrics and bucket aggs

1. Buckets. date_histogram groups documents by their date in fixed date range intervals.
2. True
3. Terms aggregation

5.2 Combining aggs

1. Group the documents by author. The first bucket that returns will have the highest doc_count, ie, most blogs written by any author

```
GET blogs/_search
```

```
{  
  "aggs": {  
    "top_authors": {  
      "terms": { "field": "authors.full_name.keyword" }  
    } } }
```

2. date_histogram (buckets by year) and terms (buckets by category)

5.3 More aggregations

1. terms + significant_text
2. Date histogram + cardinality / use max_bucket to find the year with the highest number:

```
GET blogs/_search?size=0
{
  "aggs": {
    "yearly": {
      "date_histogram": {
        "field": "publish_date",
        "calendar_interval": "year"
      },
      "aggs": {
        "nb_authors": {
          "cardinality": {
            "field": "authors.full_name.keyword"
          }
        }
      }
    },
    "max_yearly_authors": {
      "max_bucket": {
        "buckets_path": "yearly>nb_authors"
      }
    }
  }
}
```

5.4 Transforming Data

1. Use transforms when you want (a) to pivot your data from event-centric to entity-centric data or (b) to find the most recent documents of a particular grouping
2. False. Transforms and destination indices can be configured to work optimally with your dataset
3. Use Latest to keep track of the most recent activity for each server. Find the server that hasn't has a new document in the last 10 mins

6.1 Understanding shards

1. 12
2. False. Shards are optimally sized around 30-80GB
3. True

6.2 Scaling Elasticsearch

1. If you're planning to eventually scale up to distribute work, additional primaries are useful.
2. False. Read throughput scales with the number of replicas. Additional primaries may actually slow reads, though it does improve writes.
3. True.

6.3 Distributed operations

1. True
2. True

7.1 Data management concepts

1. False. It's a good idea, but not required
2. False. One and only one template will be applied. Use a "priority" to resolve conflicts

7.2 Data streams

1. True. In fact, data streams are designed to only create, never update
2. Include the data_stream object
3. In the index template

7.3 Index lifecycle management

1. True.
2. Hot/Warm/Cold/Frozen/Delete
3. 5 days. To execute this pattern, you'd set rollover in hot to 1 day, and then set the min_age in the cold phase to 5 days.

7.4 Searchable snapshots

1. The repo name. The repo must first be set up using the Snapshot and Restore API
2. False. The snapshot itself acts as the replica
3. True.

8.1 Multi cluster operations

1. Disaster recovery, high availability, data locality, and centralized reporting are all good use cases.
2. False. The follower only gets its new documents from the leader index.
3. False. Other types of deployments can also enable CCS

8.2 Troubleshooting

1. True
2. Look at the thread pool queues with _cat/thread_pool
3. True

8.3 Optimizing search performance

1. True
2. Looking for multi_match, but the boosting query could also be an answer, or you might also recall the indices_boost clause
3. False. Use the Search Profiler tool, part of the Dev Tools in Kibana

Elasticsearch Engineer

© 2015-2022 Elasticsearch BV. All rights reserved. Decompiling, copying, publishing and/or distribution without written consent of Elasticsearch BV is strictly prohibited.