



Elasticsearch Engineer

An Elastic Training Course

elastic.co/training

Welcome to Elastic Virtual Training



- Perform an audio & video test to ensure you can see & hear instructors
 - app.strigo.io/system-test
- Take these steps to prevent audio/video issues:
 - use supported browser: Chrome or Firefox
 - open pages in "Incognito" or "Private" window
 - disable ad or script blockers, VPNs or proxies
- In the event of issues, take the following steps:
 - refresh web page
 - try another browser
 - restart computer (last resort)

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

Keys to a Great Experience

- Engage and Communicate
- Be Respectful and Considerate
- Introductions

elastic.co/community/codeofconduct



Course Objectives

By the end of this course, you will be able to:

- Describe the **basic operations** and **dataflow** of an Elasticsearch cluster
- Configure how Elasticsearch data is **modeled** and **mapped**
- Write basic **search requests** and work with their responses
- Create **complex queries** using ranges, filters, and aggregations
- Identify various ways to **process** data
- Configure processors to **modify** and **enrich** data
- Specify strategies to **optimize** and **scale** clusters
- Create **data lifecycle** and **backup** policies
- Manage multi-cluster **operations** and **troubleshooting**

Elasticsearch Engineer: Agenda

- **Module 1: Getting Started**
- Module 2: Data Modeling
- Module 3: Search
- Module 4: Aggregations
- Module 5: Data Processing
- Module 6: Distributed Datastore
- Module 7: Data Management
- Module 8: Cluster Management

Module 1

Getting Started

"Let's start with the fundamentals of the Elastic Stack architecture, connecting to your lab infrastructure, and using the tools to extract meaningful insights from the data."



1.1 Stack Introduction

1.2 Index Operations

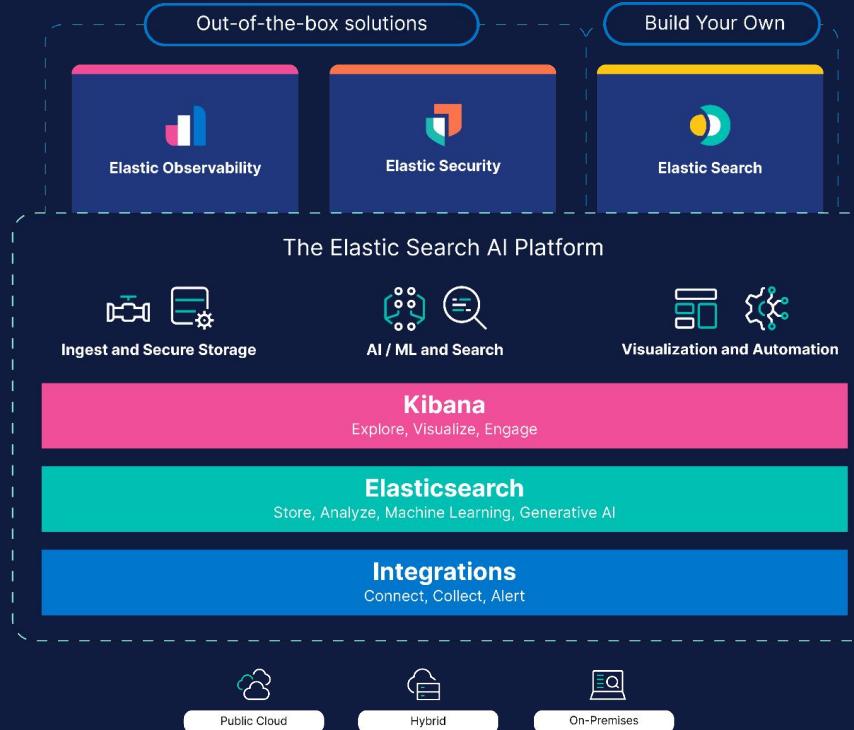
1.3 Search Data

By the end of this lesson, you will be able to:

- Define the core components of the Elastic Stack
- Differentiate between Elasticsearch installation options

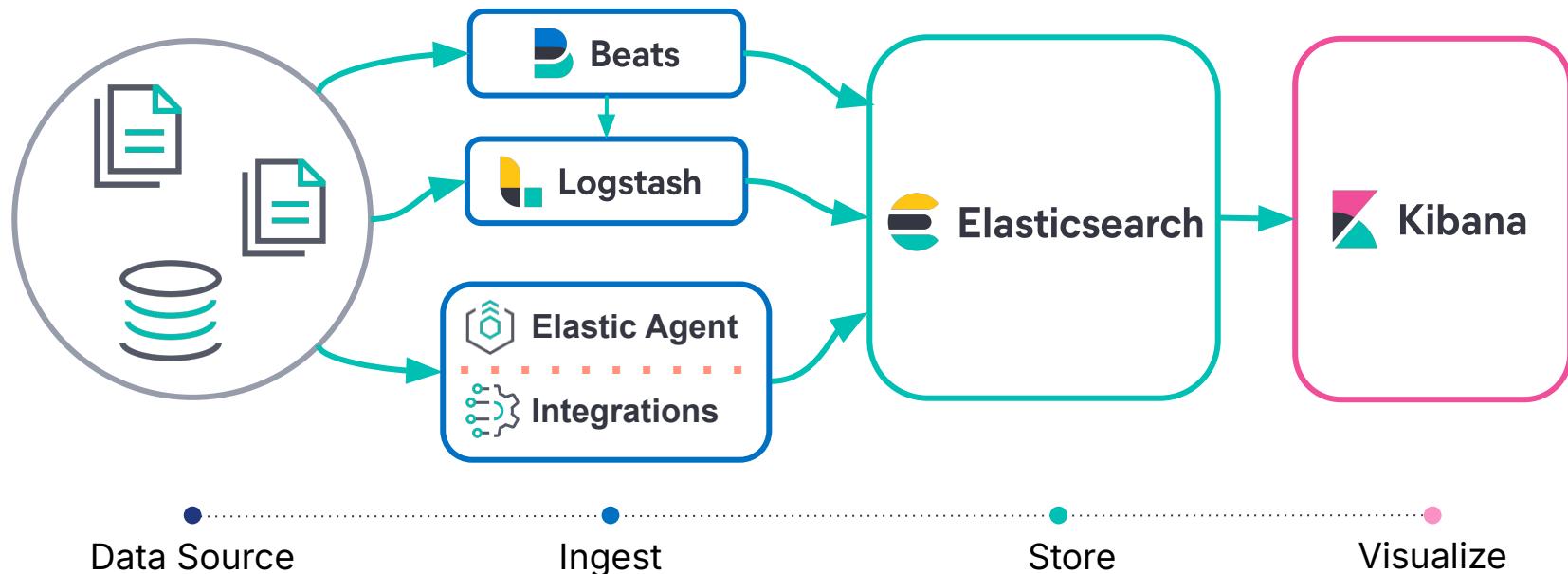


The Elastic Search Platform



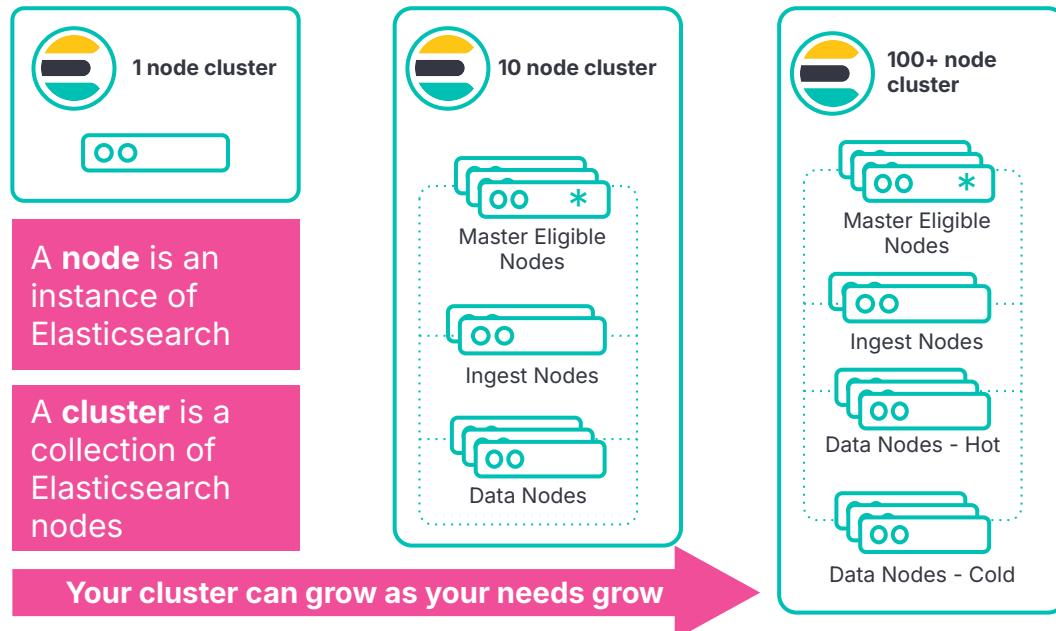
Elasticsearch Data Journey

Collect, connect, and visualize your data from any source



Elastic Stack: Elasticsearch

- Elasticsearch is a **search and analytics** engine for all types of data that is:
 - distributed
 - scalable
 - highly available



Elasticsearch is a Document Store

- Elasticsearch is a **distributed document store**
- **Documents** are serialized JSON objects that are
 - stored in Elasticsearch under a unique document ID
 - distributed across the cluster and can be accessed immediately from any node

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

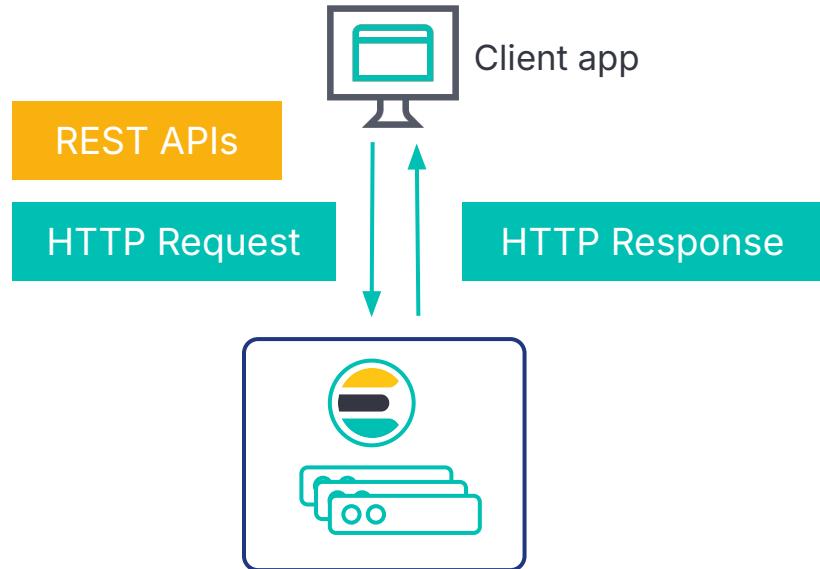
... is stored in
Elasticsearch as
a **document**



Easily Used by Any Language

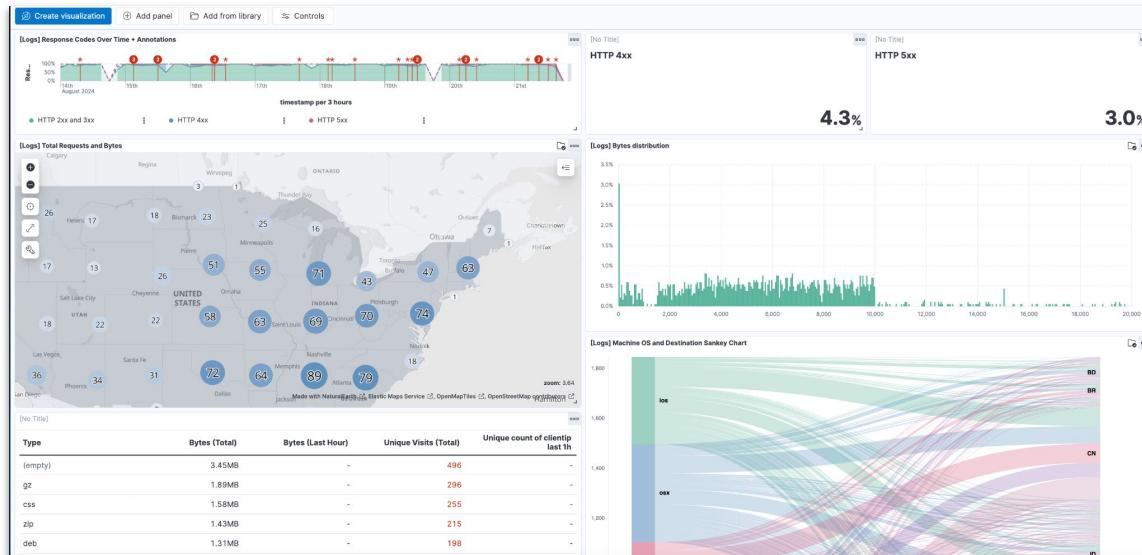
- Elasticsearch provides **REST APIs** to communicate with a cluster over HTTP
 - enables you to write your applications in any language
 - uses a **language client** to interact with Elasticsearch using language-native features

Java, .NET, PHP, Ruby, Python, Perl, or whatever you want!



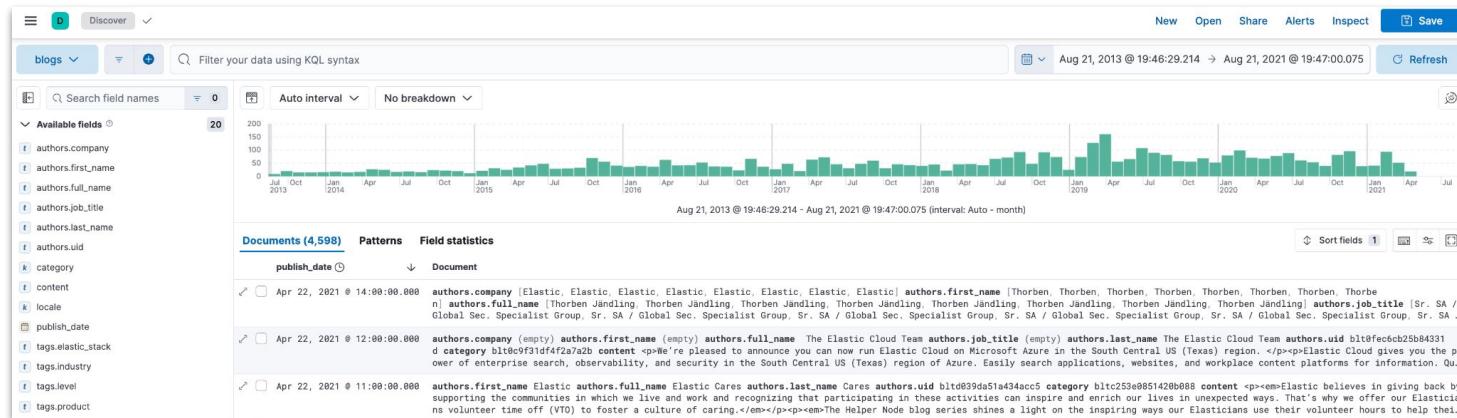
Elastic Stack: Kibana

- Kibana is a front-end application that sits on top of the Elastic Stack
- It provides **search** and **data visualization** capabilities for data 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
 - Examine popular **values**
 - Use the **query bar** and **filters** to see subsets of your data



Installation Options

Elastic Cloud

Elastic Cloud Hosted

- Elastic Stack managed through Elastic Cloud
- elastic.co/cloud



Elastic Cloud Serverless

- Serverless projects managed and automatically scaled by Elastic
 - Elasticsearch Serverless
 - Elastic Observability Serverless
 - Elastic Security Serverless
- elastic.co/cloud/serverless



60+ Cloud Regions Globally

Elastic Self-Managed

Elastic Stack

- Linux and macOS (tar.gz)
 - artifacts.elastic.co
 - linux package managers
 - x86 and aarch64
- Windows (.zip)
 - artifacts.elastic.co
 - self-contained archive
- Docker (and compose)
 - docker.elastic.co
- Locally
 - non-production deployments
 - automatically creates a single-node cluster
 - <https://elastic.co/start-local>

Elastic Cloud on Kubernetes

- elastic.co/elastic-cloud-kubernetes

Elastic Cloud Enterprise

- elastic.co/ece

Summary: Stack Introduction

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
- Elasticsearch has both **hosted** and **self-managed** installation options

Quiz



1. Which component of the Elastic Stack is responsible for storing data?
2. What term is used to refer to a single instance of Elasticsearch?
3. **True or False:** You must develop your own application to query data in Elasticsearch.

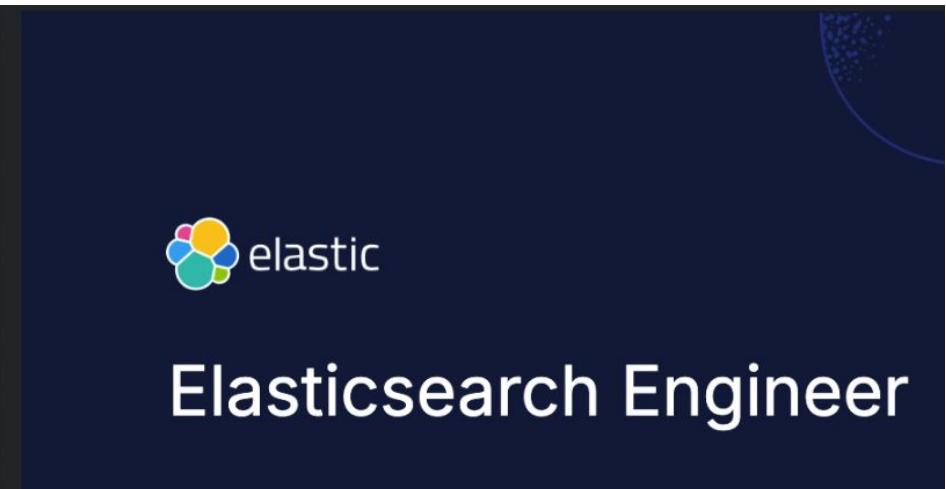
Lab Environment Orientation

Strigo Lab Environment

- Follow your provided link to the Strigo environment
- Perform initial connectivity test (optional)
- Access lab from menu on left



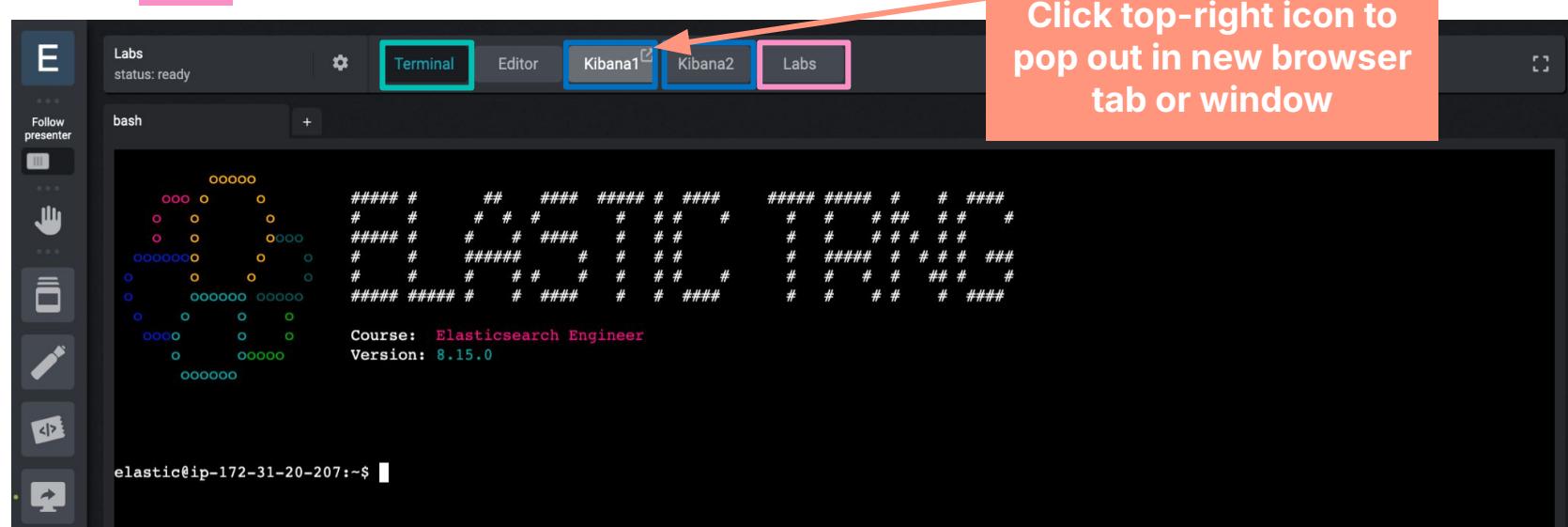
Click "My Lab" to launch your environment



app.strigo.io/system-test

Strigo: Your Elasticsearch Cluster

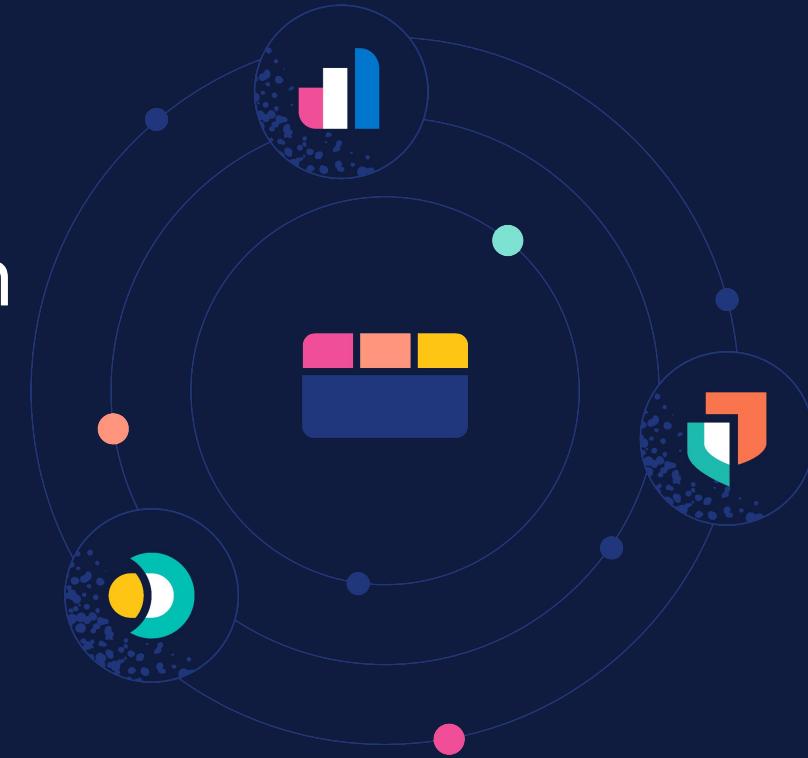
- Your lab environment consists of three key interfaces
 - **Terminal** - interactive shell session
 - **Kibana1** + **Kibana2** - launch Kibana UI on each cluster
 - **Labs** - view the lab instructions site



Lab 1.1

Stack Introduction

View sample data
and get comfortable
navigating Kibana





1.1 Stack Introduction

1.2 Index Operations

1.3 Search Data

By the end of this lesson, you will be able to:

- Index documents into an Elasticsearch index
- Perform basic operations on data using REST APIs
- Identify file formats that are easily uploadable from Kibana

Documents are JSON Objects

- Given the following 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
You know, for Search	User Stories	February 8, 2010	Shay	Banon	Elastic

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

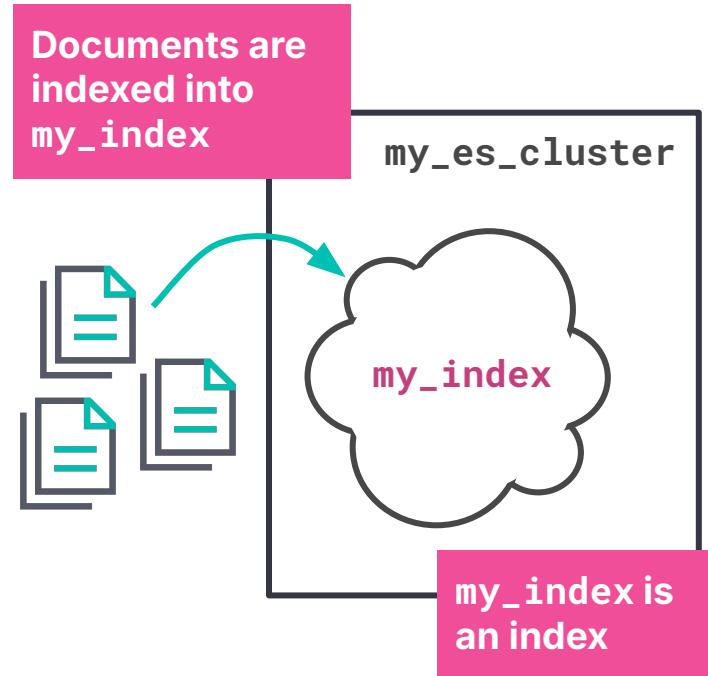
A document consists of **fields**...

```
{  
  "title": "You Know, for Search",  
  "category": "User Stories",  
  "date": "2010-02-08T19...",  
  "author": {  
    "first_name": "Shay",  
    "last_name": "Banon",  
    "company": "Elastic"  
  }  
}
```

... and **values**

Documents are Indexed into an Index

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



Index a Document: curl Example

- To create an index, send a request using **POST** that specifies:
 - index name
 - `_doc` resource
 - document
- By default, Elasticsearch generates the ID for you

index name

`_doc` endpoint

document

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

Index a Document: Dev Tools > Console

- Console providing Elasticsearch & Kibana **REST interaction**
- **User-friendly interface** to create and submit requests
- View **API documentation**

The screenshot shows the Elasticsearch Dev Tools Console interface. At the top, there are tabs: 'Console' (selected), 'Search Profiler', 'Grok Debugger', and 'Painless Lab'. Below the tabs is a menu bar with 'History', 'Settings', 'Variables', and 'Help'. A red box highlights the 'Run query' button, which is located next to a blue play button icon. The main area contains a code editor with a POST request to 'my_blogs/_doc'. The request body is a JSON object with fields: title, category, author (first_name and last_name). A red box highlights the play button icon. To the right of the code editor, a blue box highlights the 'document ID' field, which contains the value '60Cz5pEBqWhDYCLiWpe5'. At the bottom left, a pink box highlights the 'Create a request' button. At the bottom right, another pink box highlights the 'View the response' button. The status bar at the bottom right shows '201 - Created' and '824 ms'.

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

Run query

document ID

Create a request

View the response

Index a Document: PUT versus POST

- When you index a document using:
 - **PUT** - you pass in a document ID with the request if the document ID already exists, the index will be **updated** and the **_version** incremented by 1
 - **POST** - the document ID is automatically generated with a **unique ID** for the document

request

```
PUT my_blogs/_doc/60Cz5pEBqWhDYCLiWpe5
{
  "title" : "Fighting Ebola with Elastic",
  "category": "User Stories",
  "Author" : {
    "first name" : "Emily",
    "last name" : "Mosher"
  }
}
```

change the
“category”
field value

response

```
{
  "_index" : "my_blogs",
  "_type" : "_doc",
  "_id" : "60Cz5pEBqWhDYCLiWpe5",
  "_version" : 2,
  "result" : "updated",
  ...
}
```

_version is
incremented

Retrieve a Document

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

request

document resource

document ID

```
GET my_blogs/_doc/60Cz5pEBqWhDYCLiWpe5
```

response

```
{  
  ...  
  "_id" : "60Cz5pEBqWhDYCLiWpe5",  
  "_source": {  
    "title": "Fighting Ebola with Elastic",  
    "category": "User Stories",  
    "author": {  
      "first_name": "Emily",  
      "last_name": "Mosher"  
    }  
  }
```

Create a Document

- Index a new JSON document with the `_create` resource
 - guarantees that the document is only indexed if it does not already exist
 - can not be used to update an existing document

request

```
POST my_blogs/_create/4
{
  "title" : "Fighting Ebola with Elastic",
  "category": "Engineering",
  "Author" : {
    "first name" : "Emily",
    "last name" : "Mosher"
  }
}
```

response

```
{
  "_index" : "my_blogs",
  "_type" : "_doc",
  "_id" : "4",
  "_version" : 1,
  "result" : "created",
  ...
}
```

Update Specific Fields

- Use the **_update** resource to modify specific fields in a document
 - Add the “**doc**” context
 - **_version** is incremented by 1

request

```
POST my_blogs/_update/4
{
  "doc" : {
    "category": "User Stories"
  }
}
```

update the “**category**” field value

response

```
{
  "_index" : "my_blogs",
  "_type" : "_doc",
  "_id" : "4",
  "_version" : 2,
  "result" : "updated",
  ...
}
```

_version is incremented

Delete a Document

- Use **DELETE** to delete an indexed document

request

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

response

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

Operations Summary

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

Cheaper in Bulk

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

Bulk API Example

- Newline delimited JSON (NDJSON) structure
 - increases the indexing speed
 - **index, create, update** actions expect a newline followed by a JSON object on a single line

```
POST comments/_bulk
{"index" : {}}
{"title": "Tuning Go Apps with Metricbeat", "category": "Engineering"}
{"index" : {"_id":4}}
{"title": "Elasticsearch Released", "category": "Releases"}
{"create" : {"_id":5}}
{"title": "Searching for needle in", "category": "User Stories"}
{"update" : {"_id":2}}
{"doc": {"title": "Searching for needle in haystack"}}
{"delete" : {"_id":1}}
```

Upload a File in Kibana

- Quickly upload a log file or delimited CSV, TSV, or JSON file
 - used for the initial exploration of your data
 - not intended for uploading all production data

Upload data from a file

Upload your file, analyze its data, and optionally import the data into an Elasticsearch index.

The following file formats are supported:

- ❑ Delimited text files, such as CSV and TSV
- ❑ Newline-delimited JSON
- ❑ Log files with a common format for the timestamp

You can upload files up to 100 MB.

Select or drag and drop a file

Default upload size is 100 MB

Maximum file upload size can be set up to 1 GB in Advanced Settings

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

Static dataset: blogs

- Our **static** dataset is a collection of posts from elastic.co/blog
 - index name: **blogs**
 - contains old data that may be accessed as frequently as new data

SEARCH

aNN vs kNN: Understand their differences and roles in vector search

Discover the differences, strengths, and applications of aNN and kNN in vector search. Unravel the accuracy of kNN and the speed of aNN. Explore Elastic's vector search capabilities.

By Elastic Platform Team

OBSERVABILITY

Elastic 8.15: Better semantic search, new OTel distribution, SIEM data import

Elastic 8.15 includes enhancements in semantic search, new OTel collector distribution, AI-driven SIEM data onboarding, LLM integrations, additions to cross-cluster search (CCS), and more functionality within the Elasticsearch Query...

By Brian Bergholm

SECURITY

NEW in Elastic Security 8.15: Automatic Import, Gemini models, and AI Assistant APIs

Elastic Security 8.15 introduces Automatic Import, support for Gemini 1.5 Pro/Flash Models, on-demand scanning for the Elastic Defend integration, a full set of APIs for the Elastic AI Assistant, and a redesigned context pivot in the details flyout.

By Jamie Hynds, James Spiteri

Time series dataset: logs

- Our **time series** dataset is web server access log files
 - index name:
web_traffic
 - grows quickly
 - data can be read-only
 - older data may 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  
}
```

Can also be referred to as
“event-based data”

Summary: Index Operations

Module 1 Lesson 2

Summary



- A **document** is a serialized JSON object that is stored in Elasticsearch under a unique ID
- An index in Elasticsearch is a logical way of grouping data
- You can perform operations on documents in an index using REST APIs
- Most of our users' data falls into one of two categories:
 - **static data**
 - **time series data**

Quiz

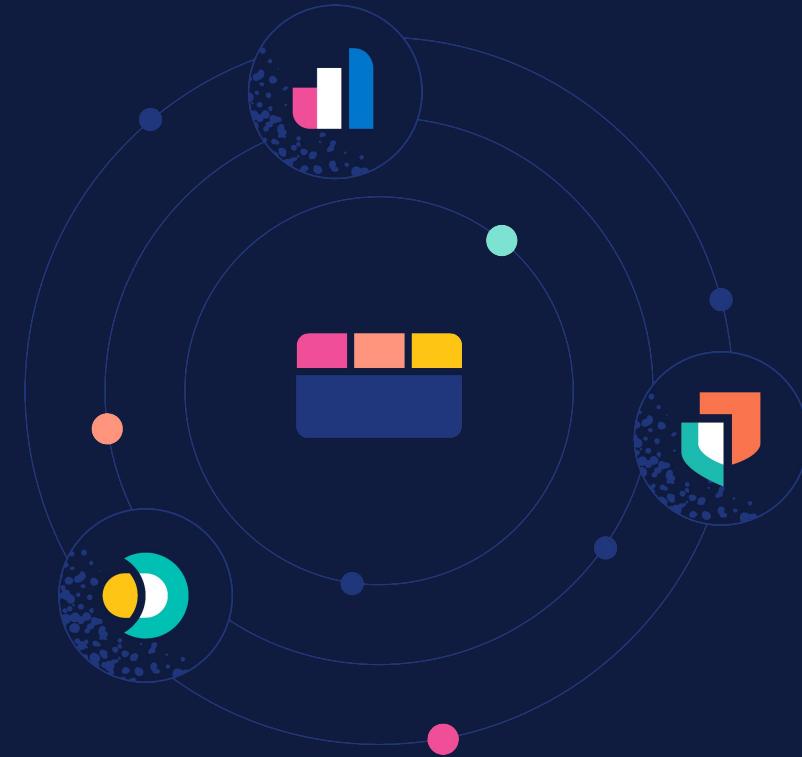


1. **True or False:** A data set of an inventory of items that grows slowly would be categorized as a static data set.
2. **True or False:** The Bulk API provides an efficient way to index multiple documents in a single POST request.
3. What happens if you index a document using a document ID and the `_id` already exists in the index?

Lab 1.2

Index Operations

Index documents using
the REST APIs and
load documents using
the file uploader in
Kibana





1.1 Stack Introduction

1.2 Index Operations

1.3 Search Data

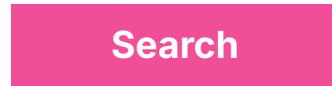
By the end of this lesson, you will be able to:

- Identify how search can be used; provided a relevant use case
- Create a basic search request using Kibana and Query DSL
- Add an aggregation as part of a search request using Query DSL
- Define search requests using ES|QL



Elasticsearch Use Cases

- Search is used to solve different problems



Enabling search-based solutions
on Cisco.com



Detecting anomalies within their
network



Examining security logs and
detecting malicious behavior

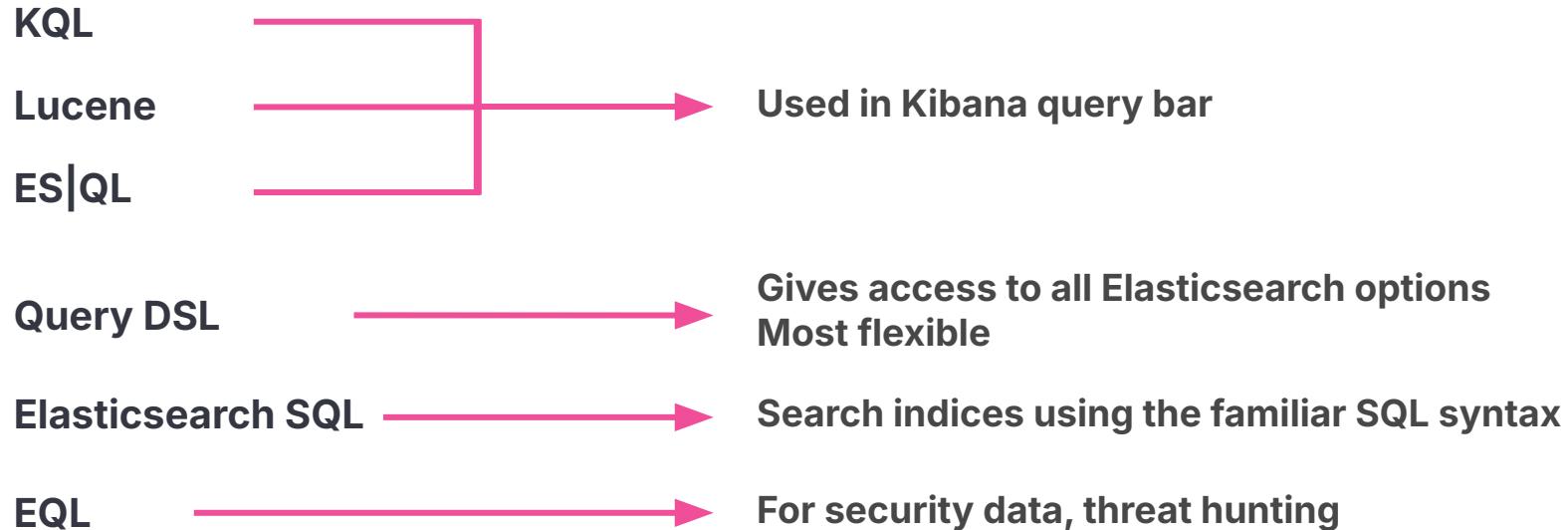
Different use cases, different challenges...

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

Queries

Query Languages

- There are several query languages to choose from:



Using KQL or Lucene in Kibana

The screenshot shows the Kibana interface with three main sections: 'data view' (left), 'query bar' (center), and 'time filter' (right). The 'data view' section contains a bar chart representing the number of documents ('blogs') from 2013 to 2021. The x-axis is labeled with years (2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021) and the y-axis ranges from 0 to 200. The chart shows a steady increase in document count over time, with a notable peak around 2019. Below the chart, there are tabs for 'Documents (4,600)', 'Patterns', and 'Field statistics'. The 'documents' tab is selected. The 'query bar' section at the top has a search input field 'Filter your data using KQL syntax' with the placeholder 'blogs'. The 'time filter' section shows a date range from 'Aug 20, 2013 @ 11:24:49.657' to 'Aug 20, 2021 @ 11:27:30.760'.

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 **group of documents**

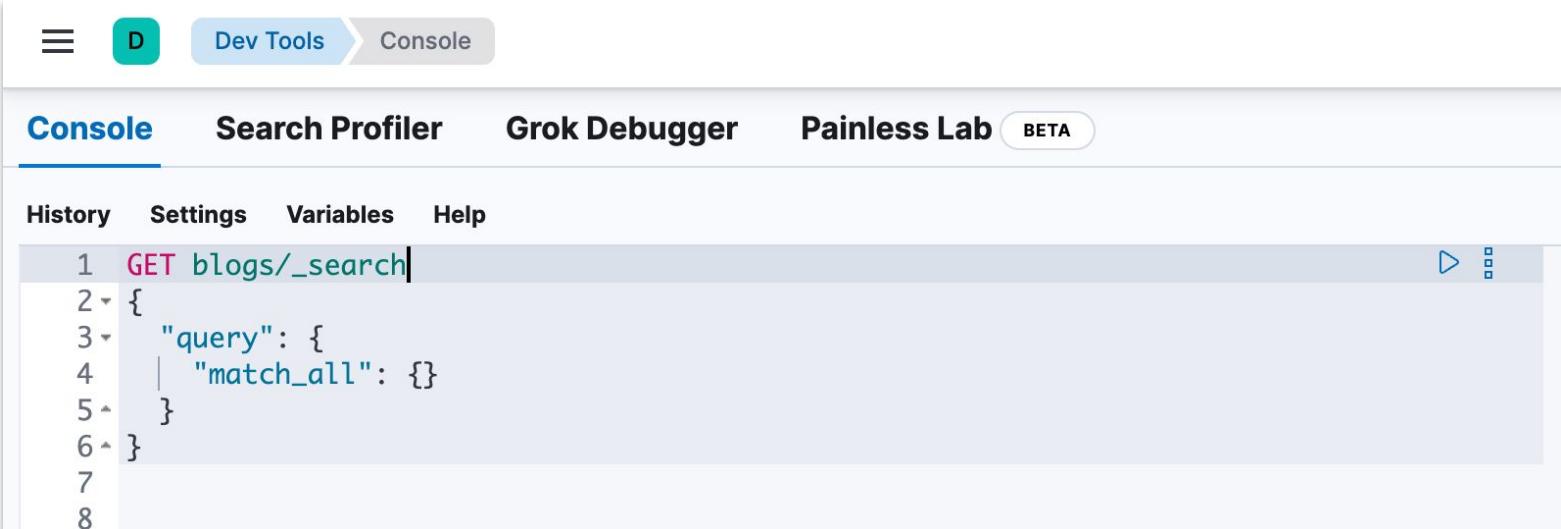
Which blog titles are about "community"?

In which month did our blogs have the most views?

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

Using Query DSL

- Send a request using the search API:
 - GET <index>/_search



The screenshot shows the Elasticsearch Dev Tools interface with the 'Console' tab selected. The top navigation bar includes 'Dev Tools' (with a dropdown menu), 'Console', 'Search Profiler', 'Grok Debugger', and 'Painless Lab (BETA)'. Below the tabs is a toolbar with 'History', 'Settings', 'Variables', and 'Help'. The main area is a code editor containing the following search query:

```
1 GET blogs/_search
2 {
3   "query": {
4     "match_all": {}
5   }
6 }
7
8
```

The **match_all** query

- The **match_all** query is the default request for the search API
 - 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,  
    "hits" : [ ... ]  
  }  
}
```

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

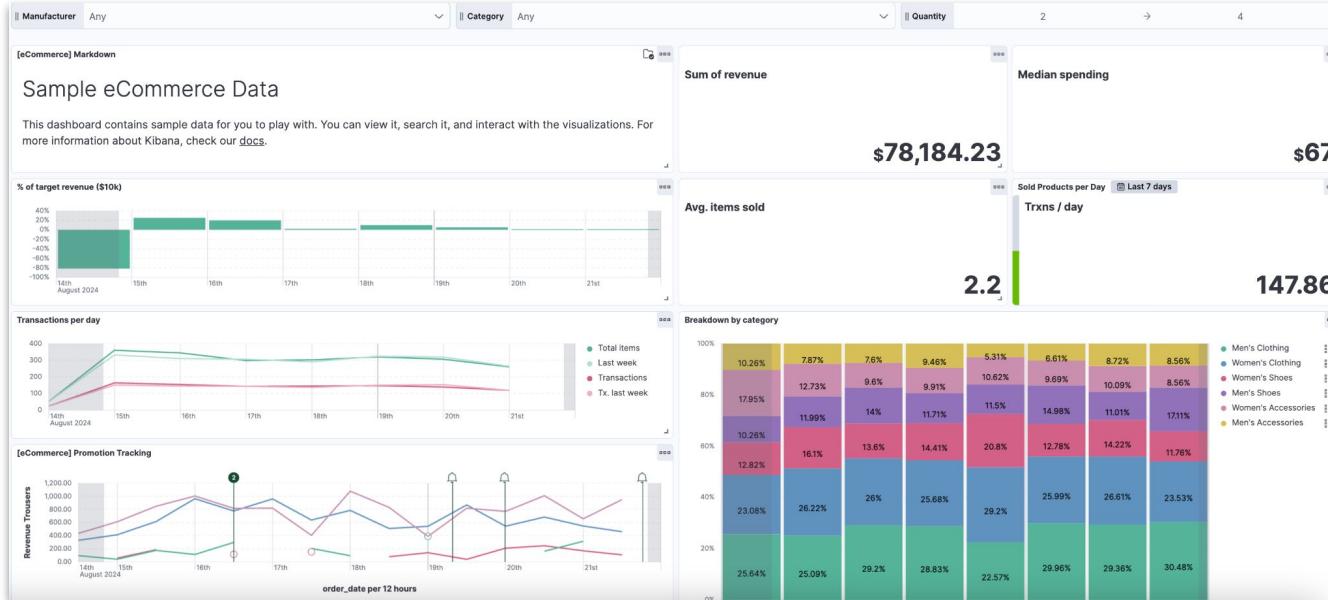
total = the number of documents that matched this query

hits = array containing the documents that hit the search criteria

Aggregations

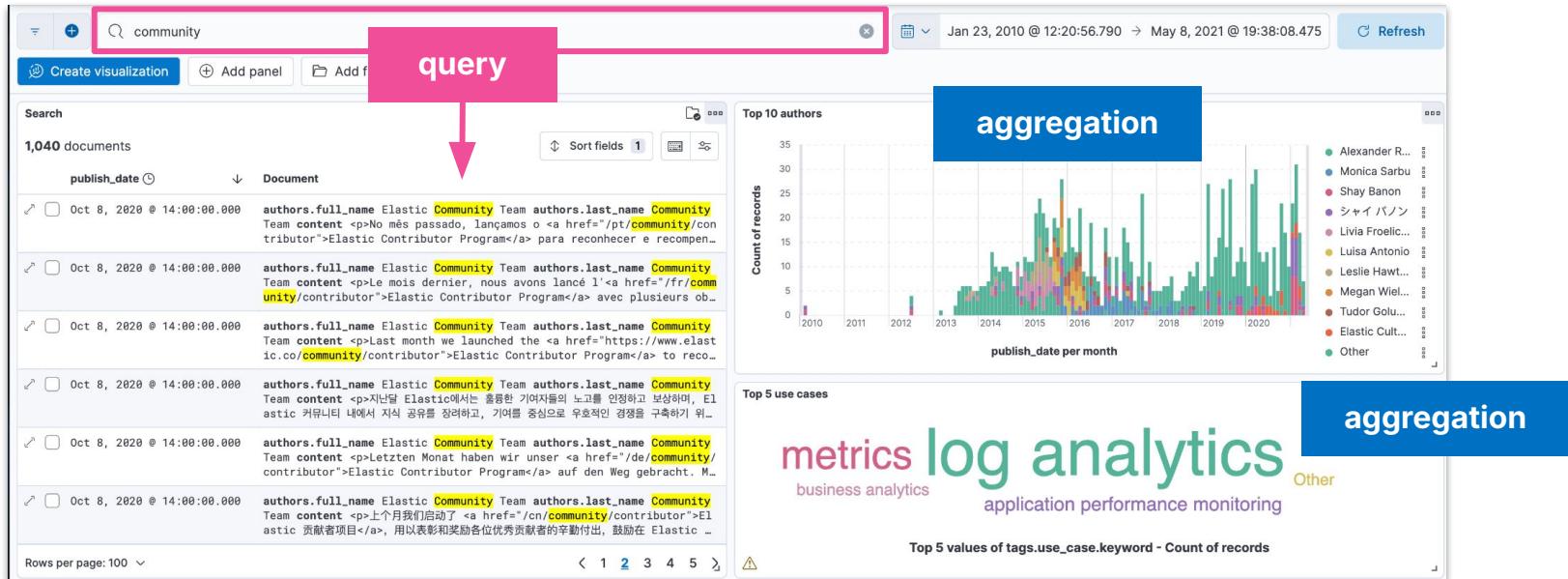
Aggregations

- Visualizations on a Kibana dashboard are powered by aggregations



Queries versus Aggregations

- What is the difference between queries and aggregations?



Let's Aggregate the Data

- Suppose you want to know what date the first blog was published
- You can use an aggregation for that

request

```
GET blogs/_search
{
  "aggs": {
    "first_blog": {
      "min": {
        "field": "publish_date"
      }
    }
  }
}
```

response

```
{
  ...
  "aggregations": {
    "first_blog": {
      "value": 1265658554000,
      "value_as_string": "2010-02-08T19:49:14.000Z"
    }
  }
}
```

ES|QL

Elasticsearch Query Language (ES|QL)

- A piped query language that delivers advanced search capabilities
 - Streamlines searching, aggregating, and visualizing large data sets
 - Brings together the capabilities of multiple languages (Query DSL, KQL, EQL, Lucene, SQL...)
- Powered by a dedicated query engine with concurrent processing
 - Designed for performance
 - Enhances speed and efficiency irrespective of data source and structure

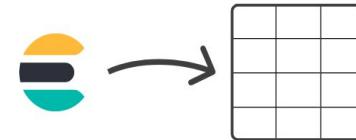
ES|QL Query

- Composed of a series of commands chained together by pipes

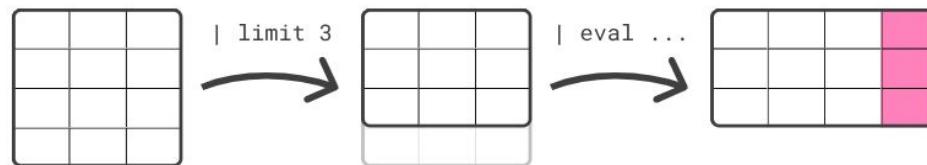
```
source-command | processing-command1 | processing-command2
```

source-command
| processing-command1
| processing-command2

Source commands
retrieve or generate
data in the form of
tables



Processing commands
take a table as input and
produces a table as output



Running an ES|QL Query in Dev Tools

- Wrap the query in a **POST** request to the query API
 - By default, results are returned as a **JSON** object
 - Use the **format** option to retrieve the results in alternative formats

```
POST /_query
{
  "query": "FROM blogs | KEEP publish_date, authors.full_name | SORT (publish_date)"
}
```

POST /_query?**format=csv**

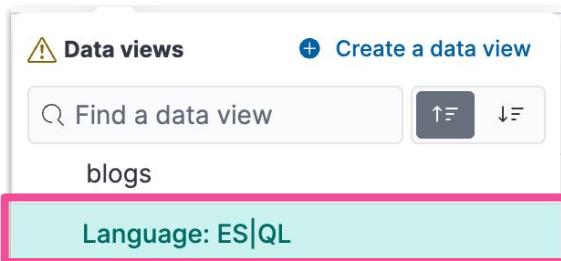
```
{
  "query": """
    FROM blogs
    | KEEP publish_date, authors.first_name, authors.last_name
    | SORT (publish_date)
  """
}
```

Other formats include:
json, tsv, txt, yaml, cbor, and smile

Triple quotes support multi-line requests

Running an ES|QL Query in Discover

1. Select **Language ES|QL** in the **Data views** pull-down
2. Expand the **query editor** to enter multiline commands
3. Click the **Run button** or type **command/alt-Enter** to run the query



A screenshot of the Elasticsearch Discover interface. At the top, a dropdown shows "ES|QL" and a date range from "Aug 20, 2013 @ 11:24:49.657" to "Aug 20, 2021 @ 11:27:30.760". A pink box highlights the "Run" button (a green play icon) and the query editor area which contains the following ES|QL code:

```
1 FROM blogs*
2 | KEEP publish_date, authors.first_name, authors.last_name
3 | SORT publish_date
4 | LIMIT 10
5
```

The main area shows "Documents (100)" and "Field statistics". A blue callout at the bottom states: "A time filter is applied if a @timestamp column is detected".

ES|QL Examples

List of columns to **KEEP**

```
FROM blogs  
| KEEP publish_date, authors.first_name, authors.last_name
```

Filter results **WHERE**
the author's last name is
"Kearns"

```
FROM blogs  
| WHERE authors.last_name.keyword == "Kearns"  
| KEEP publish_date, authors.first_name, authors.last_name
```

STATS...BY groups
rows to calculate
aggregated values

```
FROM blogs  
| STATS count = COUNT(*) BY authors.last_name.keyword  
| SORT count DESC  
| LIMIT 10
```

Summary: Search Data

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 API** to retrieve or find documents in indices
- The **Query DSL** enables you to search and aggregate over your data with a single request
- **ES|QL** is a piped query language that streamlines searching, aggregating, and visualizing data

Quiz

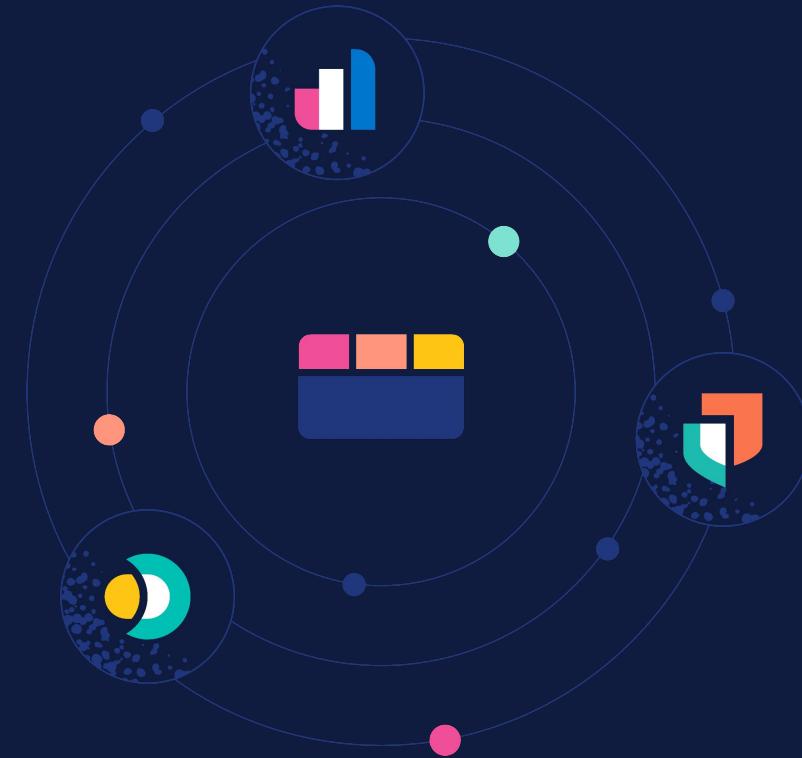


1. Name two of the query languages that are supported by the Kibana query bar.
2. **True or False:** A search request using the search API must include at least one aggregation using the “aggs” parameter.
3. **True or False:** In an ES|QL query, processing commands that are chained together using pipes (|) are processed sequentially.

Lab 1.3

Search Data

Query data using
KQL, Query DSL,
and ES|QL



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
- ES|QL
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/esql.html>

Elasticsearch Engineer: Agenda

- Module 1: Getting Started
- **Module 2: Data Modeling**
- Module 3: Search
- Module 4: Aggregations
- Module 5: Data Processing
- Module 6: Distributed Datastore
- Module 7: Data Management
- Module 8: Cluster Management

Module 2

Data Modeling

In this module, you will learn how text is analyzed to make strings searchable, and the role of mappings in storing and accessing data. You will also learn about analyzers and the differences in how keyword and text fields are handled. The module also covers steps to optimize mappings, define commonly used field data types, and use dynamic templates to automate mapping creation.



2.1 Strings

2.2 Mapping

2.3 Types and Parameters



By the end of this lesson, you will be able to:

- Discuss how text is analyzed to make strings searchable
- Describe the components and functions of an analyzer
- Identify the differences between text and keyword strings

Have You Noticed...

- What do you think these queries return?
 - different results?
 - same results?

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

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

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

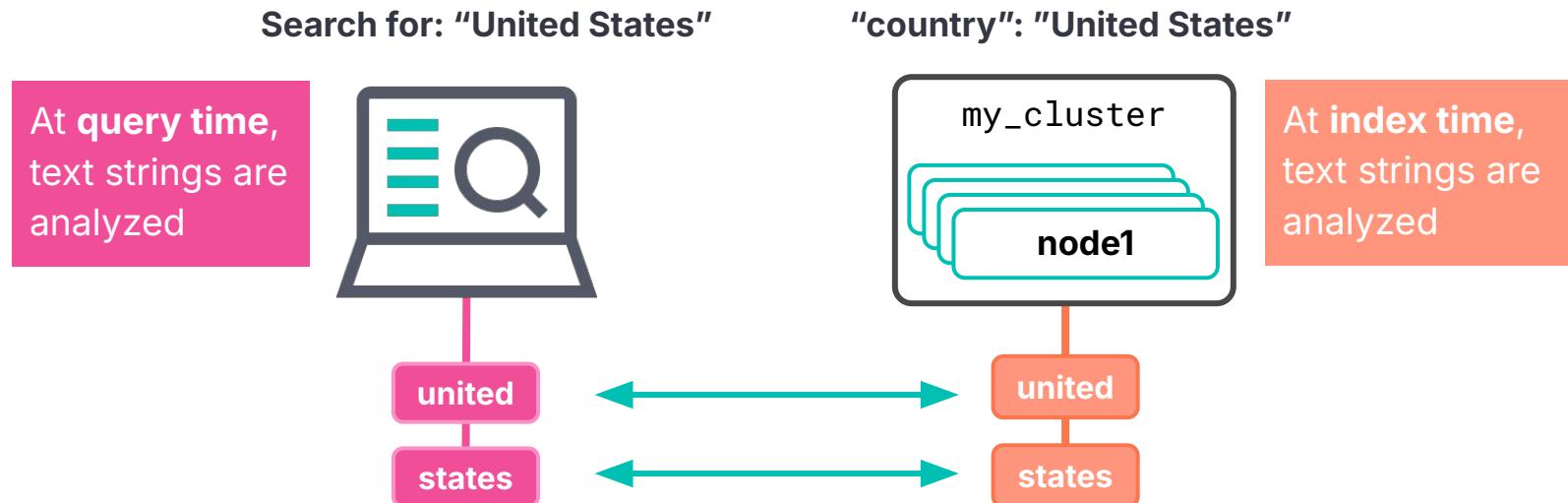
```
GET blogs/_search
{
  "query": {
    "match": {
      "content": "united states"
    }
  }
}
```

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

These two
queries return
the same hits

Analysis Makes Text Searchable

- By default, text analysis breaks up a text string into individual words (tokens) and lowerscases those words



Analyzers

Analyzers

- Text analysis is done by an **analyzer**
- By default, Elasticsearch applies the **standard** analyzer
- There are many other built-in analyzers, including:
 - whitespace, stop, pattern, simple, language-specific analyzers, and more
- The built-in analyzers work great for many use cases
 - you can also define your own custom analyzers

elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html

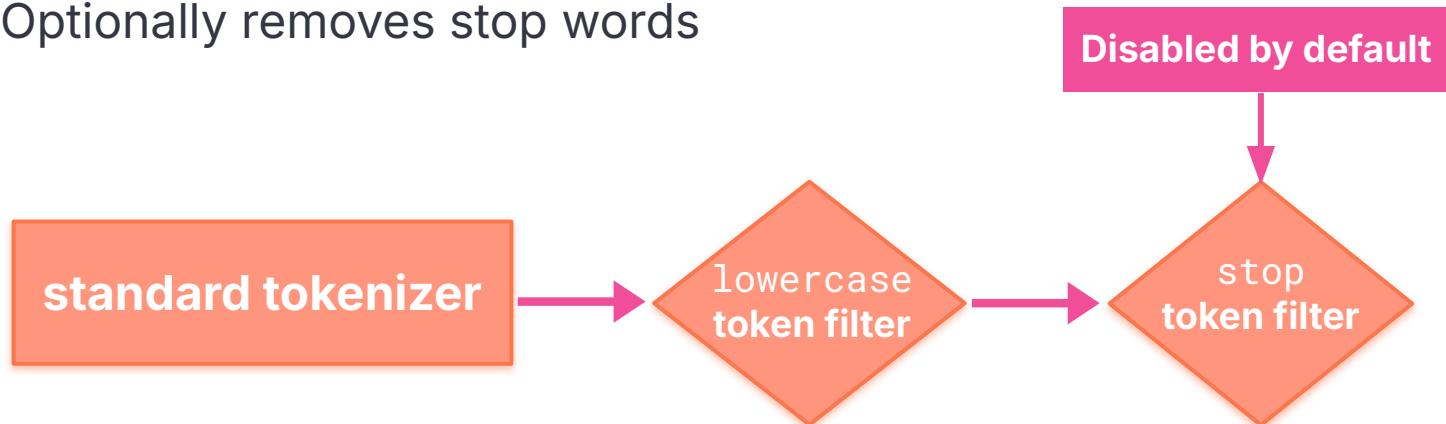
Anatomy of an Analyzer

- 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

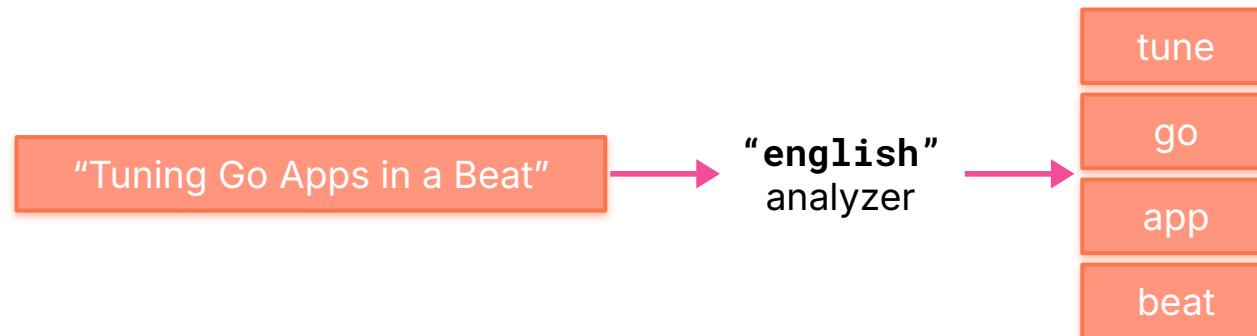


Testing an Analyzer

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

request

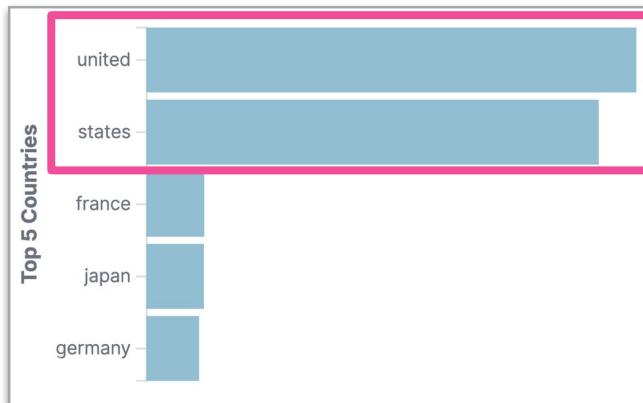
```
GET _analyze
{
  "analyzer": "english",
  "text": "Tuning Go Apps in a Beat"
}
```



Text and Keyword

Some Strings do not Need to be Analyzed

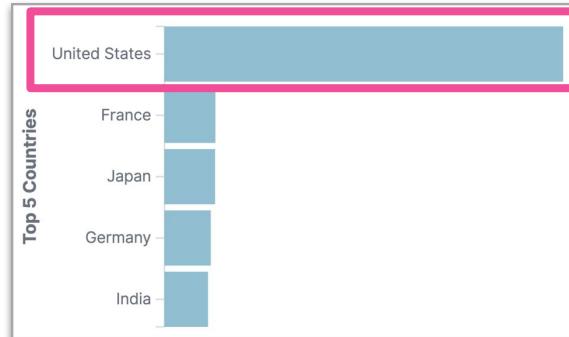
- Text analysis is great for full text search
 - enables you to search for individual words in a case-insensitive manner
- However, it is not great for things like aggregations
 - when you often want to see the original strings



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

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 to display original strings, for example
United States

Summary: Strings

Module 2 Lesson 1

Summary



- Elasticsearch uses two string data types:
 - Text for full-text search
 - Keyword for exact searches, aggregations and sorting
- The standard analyzer is applied, by default, at index time and search time
 - Text strings are analyzed
 - Keyword strings are not analyzed
- Use the **_analyze** API to test an analyzer

Quiz



1. **True or False:** Text strings are analyzed at query time.
2. **True or False:** Keyword strings are lowercased by default.
3. **True or False:** By default, Elasticsearch uses the built-in standard analyzer for indexing and search.

Lab 2.1

Strings

Illustrate the differences between text and keyword types by creating and comparing queries





2.1 Strings

2.2 Mapping

2.3 Types and Parameters



By the end of this lesson, you will be able to:

- Discuss how mappings define fields
- Identify Elasticsearch data types
- Describe the steps to define a mapping
- Determine how to optimize mappings

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**
- 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

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

Define a mapping at index creation

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

Add to a mapping of an existing index

Why have we not defined a mapping yet?

- When you index a document with unmapped fields, Elasticsearch dynamically **creates** the mapping for those fields
 - 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": "2024-08-23T15:48:50",
    "version": 8.15,
    "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

- Elasticsearch will give you both **text** and **keyword** by default

```
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

- The **country_name** field is of type **text**
- **country_name.keyword** is the keyword version of the country_name field

request

```
GET my_index/_mapping
```

response

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

Elasticsearch also created the
country_name.keyword
multi-field

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:

Text	Keyword
Full text searchable	Exact term matches
Case-insensitive	Aggregations & Sorting

- **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"  
}
```

Dynamic mapping uses
"long" for an integer

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

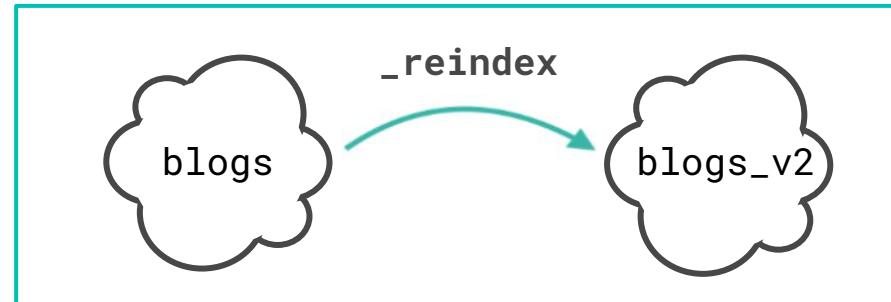
- Create a new index with the updated mapping

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

The Reindex API

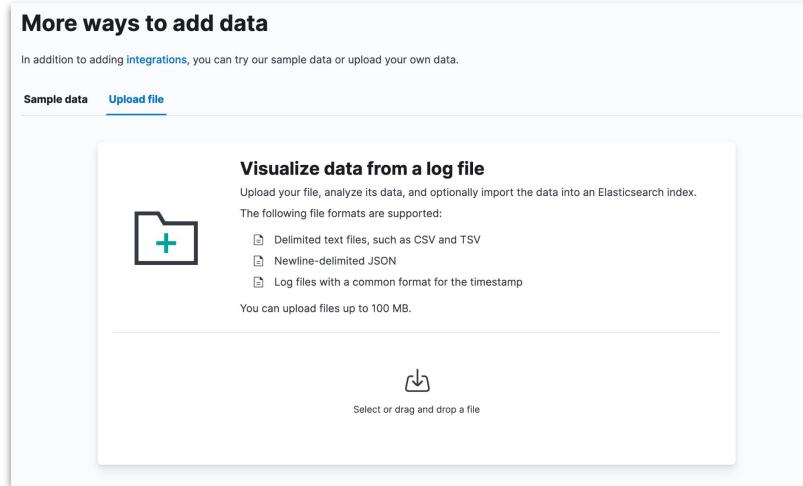
- To populate the new index, use the **reindex API**
 - reads data from one index and indexes them into another
 - use it to modify your mappings

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



Defining your Own Mapping

- Kibana's **file uploader** does an excellent job of guessing data types
 - 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": "November 22, 2024",
  "author": "Firstname Lastname",
  "title": "Elastic is Open Source",
  "seo_title": "A Good SEO Title",
  "url": "/blog/some-url",
  "content": "blog content",
  "locale": "ja-jp",
  "@timestamp": "2024-11-22T07:00:00.000Z",
  "category": "Engineering"
}
```

Fields and values should resemble fields and values you want in the blogs index

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"  
      },  
      "content": {  
        "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"  
        }  
    }  
}
```

Keep **@timestamp** mapping
from the dynamic mapping

Update **content** and
category mapping

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"
      },
      "category": {
        "type": "keyword"
      },
      "content": {
        "type": "text"
      }
    }
  }
}
```

new_blogs index contains
original (dynamic) mappings
and updated mappings

Great Mapping Means Great Performance

- Fast index speed
- Optimized storage space
- Improved searches

Summary: Mapping

Module 2 Lesson 2

Summary



- A **mapping** is Elasticsearch 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

Quiz

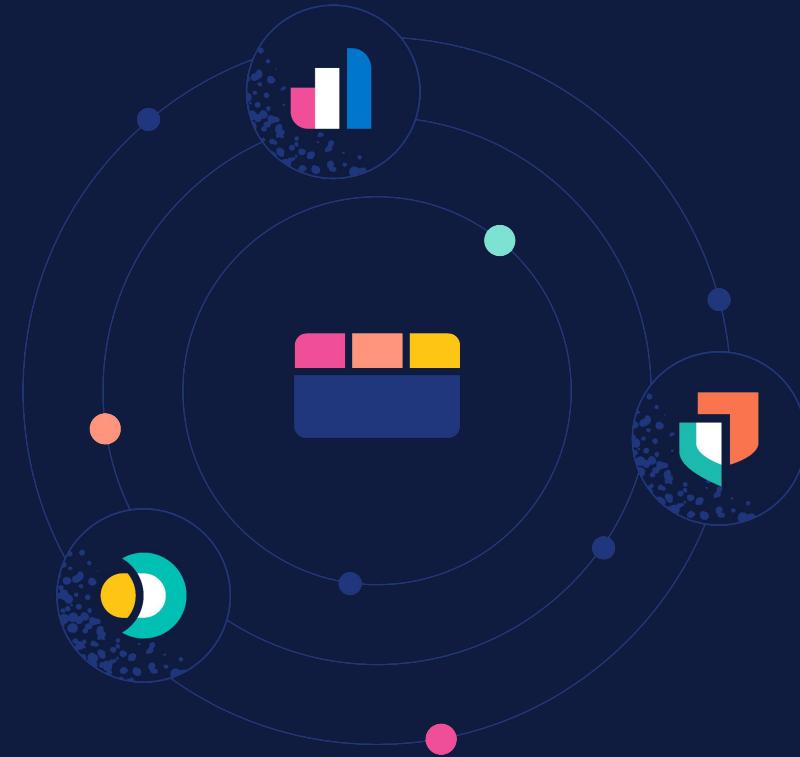


1. **True or False:** A mapping is defined per index.
2. **True or False:** You can change a field's data type from **integer** to **long** because those two types are compatible.
3. Give two reasons why creating your own custom mapping is advisable.

Lab 2.2

Mapping

Update default mappings and apply the changes to a new index





2.1 Strings

2.2 Mapping

2.3 Types and Parameters



By the end of this lesson, you will be able to:

- Specify mapping parameters that are used by field data types
- Describe how to use a dynamic template to define a field's mapping

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 built-in date formats or define your own custom format

```
"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**:

Coerce is **true** by default

All three documents can be indexed.
Rating values will be **coerced** into a long data type

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

Coerces **string** to **long**

Coerces **number** to a **long** and truncates the **value** to 4

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  
}
```

We will never aggregate
or sort by "url"

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 JSON can still be retrieved from the `_source` field but it is not searchable or stored in any other way

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, in **no particular order**

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_address": "157.97.192.70"
}

GET my_index/_mapping
```

response

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

Summary: Types and Parameters

Module 2 Lesson 3

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.

Lab 2.3

Types and Parameters

Enhance your mapping by adding mapping parameters to field data types



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: Search**
- Module 4: Aggregations
- Module 5: Data Processing
- Module 6: Distributed Datastore
- Module 7: Data Management
- Module 8: Cluster Management

Module 3

Search

“ Let’s start with some basic queries to learn the structure of writing queries and then compose more complex search requests. You will also examine the query response in order to discern how Elasticsearch scores the returned documents. ”



By the end of this lesson, you will be able to:

- Define a query that searches for one or more text fields using the Query DSL
- Identify how the query response is affected by the document score

3.1 Full Text Queries

3.2 Term-level Queries

3.3 Combining Queries

Query DSL Overview

- A search language for Elasticsearch
 - query
 - aggregate
 - sort
 - filter
 - manipulate responses

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

Request body used to construct the query

The **match** Query

- Returns documents that **match** a provided text, number, date, or boolean value
- By default, the **match** query
 - uses "**or**" logic if multiple terms appear in the search query
 - is **case-insensitive**

request

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

Search for "**community team**" in the "**title**" field

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"
. . .
```

Any document with the term
"community" or **"team"** in the
"title" field will be a hit

The **match** Query: Using "and" Logic

- The **operator** parameter
 - defines the logic used to interpret text
 - specify "**or**" (default) or "**and**"

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

Only 1 match!

Change the structure of the query to
use optional parameters

The **match** Query: Return More Relevant Results

- The **or** or **and** options might be too wide or too strict
- Use the **minimum_should_match** parameter
 - specifies the minimum number of clauses that must match
 - trims the long tail of less relevant results

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

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

The **match** Query: Searching for Terms

- The **match** query does not consider
 - the order of terms
 - how far apart the terms are
(even if the "and" operator is specified)

request

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

response

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

Terms **community** and **team** can be
in any order and in any proximity

The `match_phrase` Query

- The `match_phrase` searches for the exact sequence of terms specified in the query
 - terms in the phrase must appear in the exact order
 - use the `slop` parameter to specify how far apart terms are allowed for it to be considered a match (default is 0)

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

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

```
GET blogs/_search
{
  "query": {
    "match_phrase": {
      "title": {
        "query": "team community",
        "slop": 3
      }
    }
  }
}
```

Matches the title: "Meet the **team** behind the Elastic **Community** Conference"

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
 - specify the comma-delimited list of fields using square brackets []

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

Specify **title** and **content** fields

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"
      ]
    }
  }
}
```

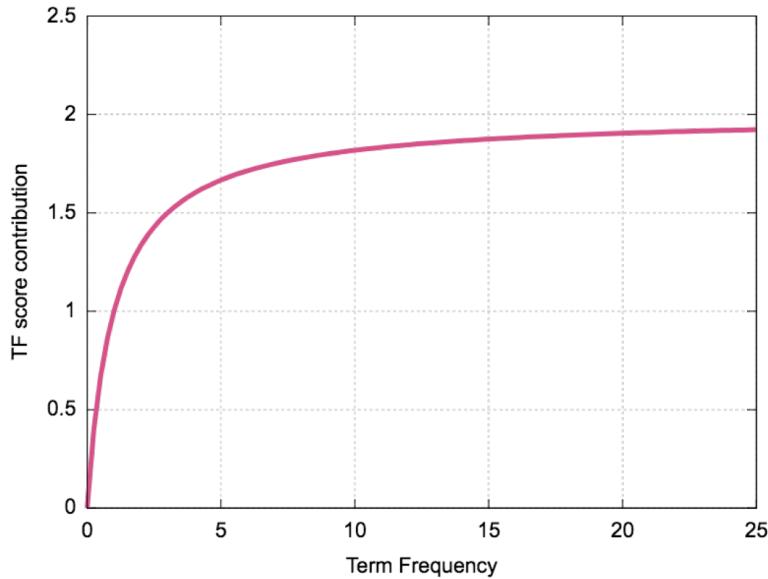
Find blogs that contain the **phrase** "elastic agent" in the title or content fields

The response

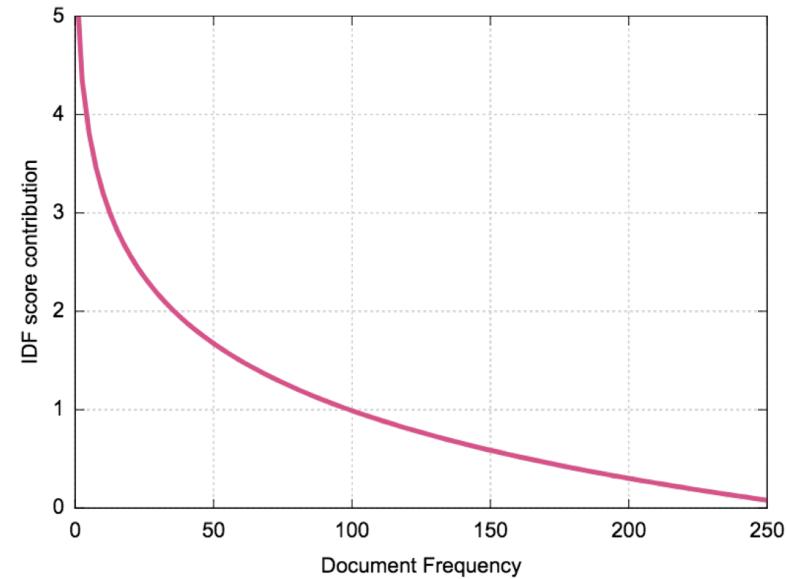
Score!

- Calculate a **score** for each document that is a hit
 - ranks search results based on **relevance**
 - represents how well a document matches a given search query
- **BM25**
 - default scoring algorithm
 - determines a document's score using:
 - **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 the `_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**

Sorting

- Use **keyword fields** to sort on field values
- Results are **not scored**
 - `_score` has no impact on sorting
 - `"_score": null`

```
GET blogs/_search
{
  "query": {
    "match": {
      "title": "Elastic"
    }
  },
  "sort": {
    "title.keyword": {
      "order": "asc"
    }
  }
}
```

Sort results using
title.keyword

Retrieve Selected Fields

- By default, each hit in the response includes the document's **_source**
 - the original data that was passed at index time
- Use **fields** to only retrieve specific fields

The document source is not included in the response when the **_source** parameter is set to **false**

```
GET blogs/_search
{
  "_source": false,
  "fields": [
    "publish_date",
    "title"
  ],
  ...
}
```

The **fields** response returns an array of values for each field

Summary: Full Text Queries

Module 3 Lesson 1



Summary

- The Query DSL enables you to:
 - query
 - aggregate
 - sort, paginate, and manipulate responses
- The match query defaults to the **or** operator, but you can change it into the operator **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

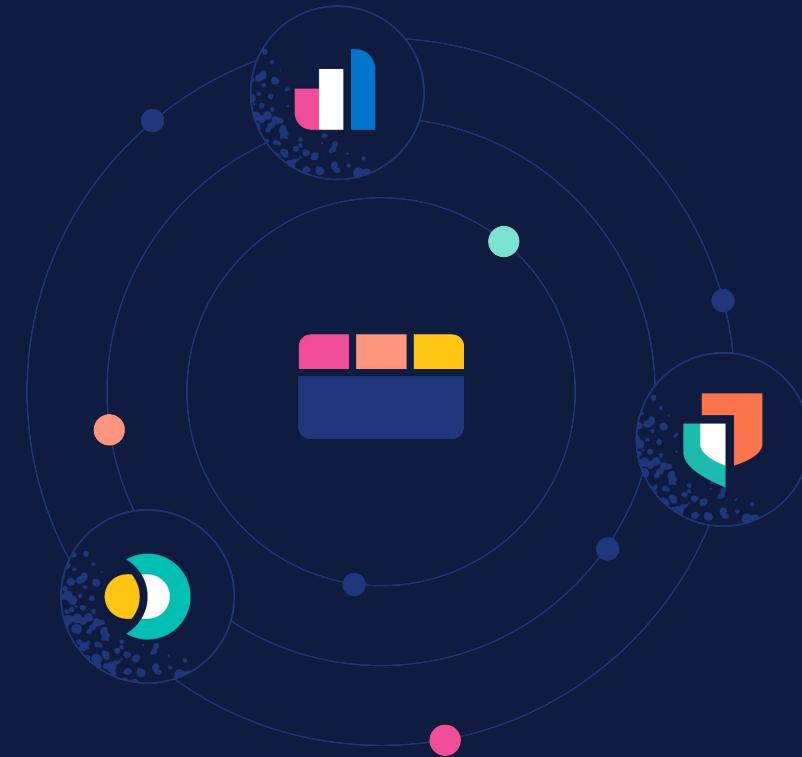


1. **True or False:** The way the **multi_match** query is executed depends on the **type** parameter. For example, setting “type”: “phrase” runs a match_phrase.
2. **True or False:** Search results are always sorted by score.
3. What is the default number of hits returned from a query using the Query DSL?

Lab 3.1

Full Text Queries

Search text fields by creating full text queries and view the corresponding response





3.1 Full Text Queries

3.2 Term-level Queries

3.3 Combining Queries

By the end of this lesson, you will be able to:

- Define term-level queries to find documents based on exact terms
- Perform an async search to handle long-running searches



Matching Exact Terms

- Recall that **full text queries are analyzed** and then searched within the index

Full Text query



Analyzes the query text before the terms are looked up in the index

- Term-level queries** are used for exact searches
 - term-level queries **do not analyze** search terms
 - Returns the **exact match** of the original string as it occurs in the documents

Term-level query



Does not analyze the query text before the terms are looked up in the index

Term-level Queries

- Find documents based on precise values in structured data
 - queries are matched on the exact terms stored in a field

Full text queries

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

Term-level queries

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

Many more

- script
- percolate
- span_ queries
- geo_ queries
- nested
- ...

Matching on a Keyword Field

- Use the **keyword** field to match on an **exact term**
 - the **term** must exactly match the field value, including whitespace and capitalization
- Recall that keyword types are commonly used for:
 - structured content such as IDs, email, hostnames, or zip codes
 - sorting and aggregations

```
GET blogs/_search
{
  "query": {
    "term": {
      "authors.job_title.keyword": "Senior Software Engineer"
    }
  }
}
```

Only matches **Senior Software Engineer**
in the exact order, using the exact
upper and lower case specified

Does not match "senior software engineer", "Software
Engineer", "software engineer", "Engineer", "engineer", ...

Searching for Numbers, Dates, and IPs

- So far you've used the **full-text** queries
 - great for searching words in bodies of text
- How would you query for dates, numbers and IPs?
 - For example:

Find all blogs published in the year 2023

- 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": "2023-01-01",
        "lte": "2023-12-31"
      }
    }
  }
}
```

Find all blogs that were published in 2023

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 = 2023-10-19T11:56:22	
now-1h	2023-10-19T10:56:22
now+1h+30m	2023-10-19T13:26:22
now/d+1d	2023-10-20T00:00:00
2024-01-15 +1M	2024-02-15T00:00:00

The `range` Query: Date Math Example

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

Find all blogs that were published in the past year

The **exists** Query

- Returns documents that contain an indexed value for a field
- Empty strings also indicate that a field exists

How many documents have a category?

```
GET blogs/_count
{
  "query": {
    "exists": {
      "field": "category"
    }
  }
}
```

_count API returns the number of matches

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" : "Fk0tWm1LM1hmVHA2bGNvMHF6a1RhM3ccZWZ0Uk9NcFVUR3VDTzc3OENmYUcyQToyMDYyMQ==",  
  "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 is 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/Fk0tWm1LM1hmVHA2bGNvMHF6a1RhM3ccZWZ0Uk9NcFVUR3VDT...
```

Summary: Term-level Queries

Module 3 Lesson 2



Summary

- Term-level queries:
 - **do not analyze** the query text before the terms are looked up in the index
 - match on **exact terms** using keyword fields
- The **range** query can be used for numbers, dates and IPs
- Use **date math** expressions whenever durations need to be specified
- The **exists** query checks to see if a field exists
- When running a long searches, use **async search**

Quiz



1. **True or False:** The Query DSL can only be used for full text queries.
2. **True or False:** In an asynchronous search, **is_partial** is always set to true while the query is being executed.
3. **True or False:** In a range query, you can set up open-ended ranges such as "prices less than \$50" or "dates greater than or equal to January 1st, 2023."

Lab 3.2

Term-level Queries

Create term-level queries and view the corresponding response



3.1 Full Text Queries

3.2 Term-level Queries

3.3 Combining Queries

By the end of this lesson, you will be able to:

- Combine multiple queries using one or more boolean clauses
- Identify which boolean clauses contribute to a document's score
- Describe the importance of using filters in your queries

Combining Queries using Boolean Logic

- 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 **locale** 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
 - documents that do not match the queries in a **should** clause are returned as hits too
- Use **minimum_should_match** to specify the number or percentage of should clauses returned

```
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"content": "agent"}}
      ],
      "should": [
        {"match": {"locale": "en-us"}},
        {"match": {"locale": "fr-fr"}}
      ],
      "minimum_should_match": 1
    }
  }
}
```

Find all blogs that contain **agent** in the **content** field

Documents that are **written in English or French** will get a higher score

Comparing Query and Filter Contexts

Query Context

**must
should**

calculates a score

slower

no automatic caching

Filter Context

**filter
must_not**

skips score calculation

faster

automatic caching for
frequently used filters

Query vs Filter Context: Ranking

Query context

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

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

The query contributes to ranking

Filter context

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

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

The query **does not** contribute to ranking

The **bool** Query Summary

Clause	Exclude docs	Scoring
must	✓	✓
must_not	✓	✗
should	✗	① ✓
filter	✓	✗

- ① If the *bool* query includes at least one **should** clause, and no **must** or **filter** clauses, it will include documents that match and exclude documents that don't.

Summary: Combining Queries

Module 3 Lesson 3



Summary

- The **bool** query combines one or more boolean clauses, which can be combined
- A **bool** query can query multiple fields at once
- Queries in a **query context** contributes to ranking while queries in a **filter context** do not
- Results from the filter context can be automatically **cached**

Quiz



1. **True or False:** The must clause is the only required clause in a bool query.
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" }
```

Lab 3.3

Combining Queries

Combine queries
using boolean logic



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
- Boolean query
 - www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-bool-query.html
 - www.elastic.co/guide/en/elasticsearch/reference/current/query-filter-context.html

Elasticsearch Engineer: Agenda

- Module 1: Getting Started
- Module 2: Data Modeling
- Module 3: Search
- **Module 4: Aggregations**
- Module 5: Data Processing
- Module 6: Distributed Datastore
- Module 7: Data Management
- Module 8: Cluster Management

Module 4

Aggregations

"In this module, you will learn how aggregations can be utilized to summarize your data in multiple ways. We will use metrics and bucket aggregations to compute values from documents, and combine multiple aggregations to create complex data summaries."



4.1 Metrics and Bucket Aggregations

4.2 Combining Aggregations

4.3 Transforming Data

By the end of this lesson, you will be able to:

- Discuss the functionality of aggregations
- Identify the basic structure of aggregations
- Use aggregations to compute values from documents



Aggregations

- A flexible and powerful capability for analyzing data
 - summarizes your data as metrics, statistics, or other analytics
 - results are typically computed values that can be grouped

Aggregation	Capability
Metric	calculate metrics, such as a sum or average, from field values
Bucket	group documents into buckets based on field values, ranges, or other criteria
Pipeline	take input from other aggregations instead of documents or fields

Basic Structure of Aggregations

- Run aggregations as part of a search request
 - specify using the search API's **aggs** parameter

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

The "aggregations" 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": {
    "first_blog": {
      "min": {
        "field": "publish_date"
      }
    }
  }
}
```

response

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {...},
  "hits" : {
    ...
    "hits" : [ ... ],
    ...
  },
  "aggregations" : {
    "first_blog" : {
      ...
    }
  }
}
```

Searches containing an aggregation, return both **search hits** and **aggregation results**

Top 10 hits

Return Only Aggregation Results

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

```
GET blogs/_search
{
  "size": 0,
  "aggs": {
    "first_blog": {
      "min": {
        "field": "publish_date"
      }
    }
  }
}
```

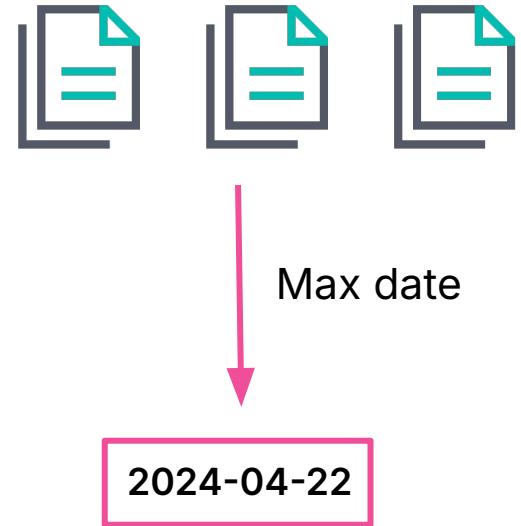
```
GET blogs/_search?size=0
{
  "aggs": {
    "first_blog": {
      "min": {
        "field": "publish_date"
      }
    }
  }
}
```

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

Metrics Aggregations

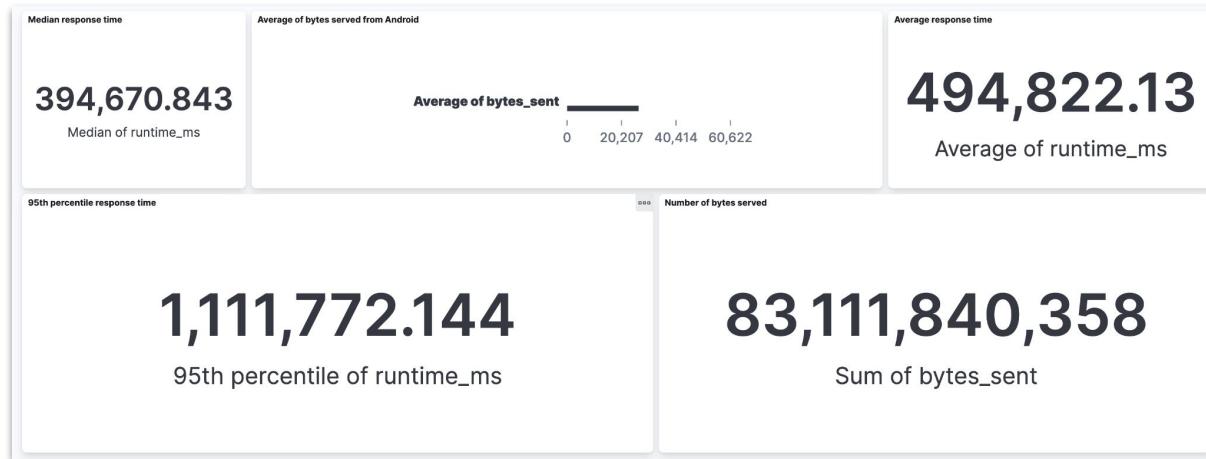
Metrics Aggregations

- Metrics compute numeric values based on your dataset
 - field values
 - values generated by custom script
- Most metrics output a single value:
 - **count, avg, sum, min, max, median, cardinality**
- Some metrics output multiple values:
 - **stats, percentiles, percentile_ranks**



Metrics Aggregations: 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?



The **min** Aggregation

- Returns the minimum value among numeric values extracted from the aggregated documents

request

```
GET blogs/_search?size=0
{
  "aggs": {
    "first_blog": {
      "min": {
        "field": "publish_date"
      }
    }
  }
}
```

Use **max** to find the date of the last blog published

What date was the first blog published?

response

```
"aggregations" : {
  "first_blog" : {
    "value": 1265658554000,
    "value_as_string": "2010-02-08T19:49:14.000Z"
  }
}
```

The **value_count** Aggregation

- Counts the number of values that are extracted from the aggregated documents
 - if a field has duplicates, each value will be counted individually

request

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

Counts all authors that wrote a blog

response

```
"aggregations" : {
  "no_of_authors" : {
    "value" : 4967
  }
}
```

If an author wrote two blogs, then the author will be counted two times

The **cardinality** Aggregation

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

request

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

response

```
"aggregations" : {
  "no_of_authors" : {
    "value" : 956
  }
}
```

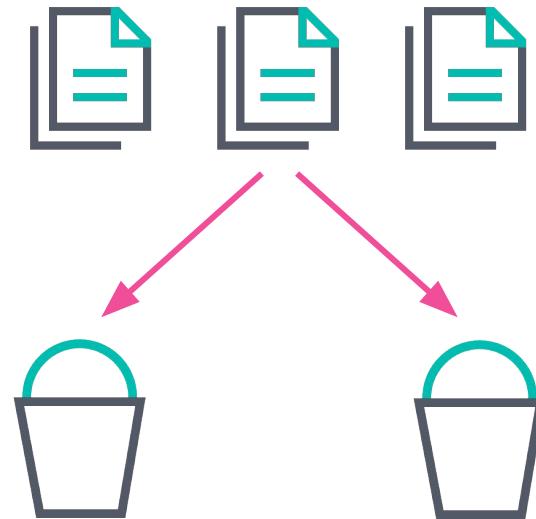
Counts the number of unique
authors that wrote blogs

Bucket Aggregations

Bucket Aggregations

- Group documents according to certain criterion

Bucket by	Aggregation
Time Period	Date Range
	Date Histogram
Numerics	Range
	Histogram
Keyword	Terms
	Significant Terms
IP Address	IPV4 Range



Bucket Aggregations: Questions

- How many requests reach our system daily?
- 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?



The `date_histogram` Aggregation

- Bucket aggregation used with time-based data
- Interval is specified using one of two ways:
 - **calendar_interval:**
 - calendar unit name such as day, month, or year (1d, 1M, or 1y)
 - **fixed_interval:**
 - SI unit name such as seconds, minutes, hours, or days (s, m, h, or d)

`request`

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

month is equivalent to **1M**

2M is not supported

The `date_histogram` Aggregation: Response

response



```
"aggregations" : {  
  "blogs_by_month" : {  
    "buckets" : [  
      {  
        "key_as_string" : "2010-02-01T00... ",  
        "key" : 1264982400000,  
        "doc_count" : 4  
      },  
      {  
        "key_as_string" : "2010-03-01T00... ",  
        "key" : 1267401600000,  
        "doc_count" : 1  
      },  
      ...  
    ]  
  }  
}
```

Bucket aggregation results return in a `buckets` array

Each bucket has a `doc_count` and a `key/key_as_string`

By default, `date_histogram` buckets are sorted by `key` in **ascending** order

The **histogram** Aggregation

- Bucket aggregation that builds a histogram
 - on a given "**field**"
 - using a specified "**interval**"
- Similar to date histogram

```
GET sample_data_logs/_search
{
  "size": 0,
  "aggs": {
    "logs_histogram": {
      "histogram": {
        "field": "runtime_ms",
        "interval": "100"
      }
    }
  }
}
```

A bucket is created for every
100 milliseconds

Bucket Sorting

- Some aggregations enable you to specify the sorting order

Aggregation	Default Sort Order
<code>terms</code>	<code>_count</code> in descending order
<code>histogram</code> <code>date_histogram</code>	<code>_key</code> in ascending order

request

```
GET blogs/_search
{
  "size": 0,
  "aggs": {
    "blogs_by_month": {
      "date_histogram": {
        "field": "publish_date",
        "calendar_interval": "month",
        "order": {
          "_key": "desc"
        }
      }
    }
  }
}
```

date_histogram order is asc by default

The **terms** Aggregation

- Dynamically create a new bucket for every unique term of a specified **field**

"field" =
"authors.job_title.keyword"

Each distinct value of the
field will have its own
bucket of documents



The **terms** Aggregation: Request

- Search for the types of people creating our blogs by their job title:

request

```
GET blogs/_search
{
  "size": 0,
  "aggs": {
    "author_buckets": {
      "terms": {
        "field": "authors.job_title.keyword",
        "size": 5
      }
    }
  }
}
```

Specify the number of buckets using "**size**" (default is 10)

Specify a "**field**" to bucket the terms by

The **terms** Aggregation: Response

- **key** represents the distinct value of field
- **doc_count** is the number of documents in the bucket
- **sum_other_doc_count** is the number of documents not in any of the top buckets

Returned order is descending by doc _count (default)

response

```
"aggregations": {  
  "author_buckets": {  
    "doc_count_error_upper_bound": 0,  
    "sum_other_doc_count": 2316,  
    "buckets": [  
      {  
        "key": "",  
        "doc_count": 1554  
      },  
      {  
        "key": "Software Engineer",  
        "doc_count": 231  
      },  
      {  
        "key": "Stack Team Lead",  
        "doc_count": 181  
      },  
      ...  
    ]  
  }  
}
```

Summary: Metrics and Bucket Aggregations

Module 4 Lesson 1

Summary



- Aggregations are summaries of data
- Aggregations are part of a search request
- Set "size" to 0 for faster results
- Metrics aggregations compute values from your data
- Bucket aggregations group your data together based on certain criterion

Quiz



1. Is date_histogram a bucket, metric, or pipeline aggregation?
2. **True or False:** Buckets are always sorted by document count.
3. Which aggregation would you use to organize logging events into buckets by log level ("error", "warn", "info", etc.)?

Lab 4.1

Metrics and Bucket Aggregations

Write metric and
bucket
aggregations using
the Query DSL





4.1 Metrics and Bucket Aggregations

4.2 Combining Aggregations

4.3 Transforming Data



By the end of this lesson, you will be able to:

- Create a search request that includes a query to reduce the scope of the aggregation
- Specify multiple aggregations in the same search request
- Define sub-aggregations in the search request
- Sort results by a metric value in a sub-aggregation

Working with Aggregations

- **Combine aggregations**
 - Specify different aggregations in a single request
 - Extract multiple insights from your data
- **Change the aggregation's scope**
 - Use queries to limit the documents on which an aggregation runs
 - Focus on specific, or relevant data
- **Nest aggregations**
 - Create a hierarchy of aggregation levels, or sub-aggregations, by nesting bucket aggregations within bucket aggregations
 - Use metric aggregations to calculate values over fields at any sub-aggregation level in the hierarchy

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 unique number of authors for French blogs

```
GET blogs/_search?size=0
{
  "query": {
    "match": {
      "locale": "fr-fr"
    }
  },
  "aggs": {
    "no_of_authors": {
      "cardinality": {
        "field": "authors.last_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.last_name.keyword"
      },
      "first_name_stats": {
        "string_stats": {
          "field": "authors.first_name.keyword"
        }
      }
    }
  }
}
```

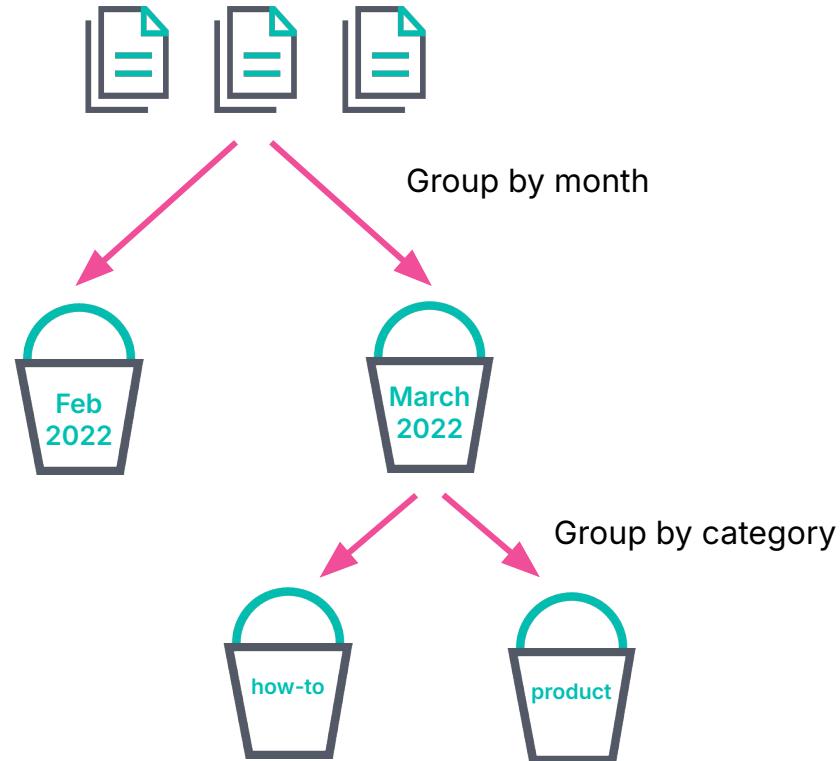
response

```
"aggregations": {
  "no_of_authors": {
    "value": 956
  },
  "first_name_stats": {
    "count": 4961,
    "min_length": 2,
    "max_length": 41,
    "avg_length": 5.66539...,
    "entropy": 4.752609555991666
  }
}
```

Aggregations listed
in order of request

Sub-Aggregations

- Embed aggregations (bucket and metric) inside other aggregations
 - **separate groups** based on criteria
 - **apply metrics** at various levels in the aggregation hierarchy
- No depth limit for nesting sub-aggregations



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.last_name.keyword" }
      } } } } }
```

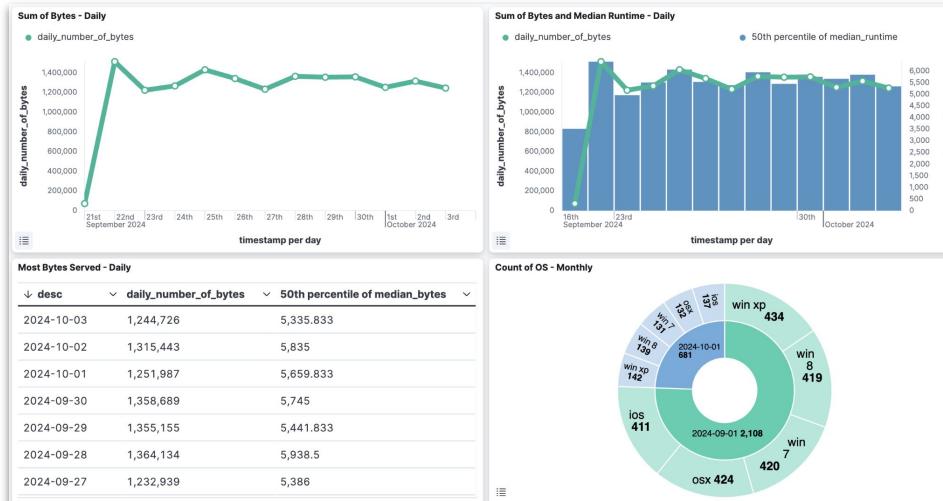
no_of_authors is a
sub-aggregation of
blogs_by_month

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 Aggregations: Questions

- What is the sum of bytes per day?
- What is the sum of bytes per day and the median bytes size?
- On what day were the most bytes served?
- What are the most popular operating systems per month?



What is the sum of bytes per day?

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" :
"2024-02-18T00:00:00.000Z",
        "key" : 1708214400000,
        "doc_count" : 249,
        "daily_number_of_bytes" : {
          "value" : 1531493
        }
      }
    ]
  }
}
```

Compute a metric per bucket
(besides doc_count)

What is the sum of bytes per day and the median bytes size?

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" :
"2024-02-18T00:00:00.000Z",
        "key" : 1708214400000,
        "doc_count" : 249,
        "daily_number_of_bytes" : {
          "value" : 1531493
        },
        "median_bytes" : {
          "values" : {
            "50.0" : 6193
          }
        }
      }
    ]
  }
}
```

Calculate multiple metrics on bucket aggregations

On what day were the most bytes were served?

request

```
"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]  
                }  
            }  
        }  
    }  
}
```

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

response

```
"aggregations": {  
    "requests_per_day": {  
        "buckets": [  
            {  
                "key_as_string": "2024-03-30T00:00:00.000Z",  
                "key": 1711756800000,  
                "doc_count": 329,  
                "daily_number_of_bytes": {  
                    "value": 1558542  
                },  
                "median_bytes": {  
                    "values": {  
                        "50.0": 6193  
                    }  
                }  
            }  
        ]  
    }  
}
```

First bucket is the day that most bytes were served!

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

What are the most popular operating systems per month?

request

```
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”

What are the most popular operating systems per month?

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

response

```
"aggregations" : {
  "logs_by_month" : {
    "buckets" : [
      {
        "key_as_string" :
"2024-09-01T00:00:00.000Z",
        "key" : 1706745600000,
        "doc_count" : 2786,
        "machine.os" : {
          "doc_count_error_upper_bound" : 0,
          "sum_other_doc_count" : 0,
          "buckets" : [
            {
              "key" : "win xp",
              "doc_count" : 576
            },
            {
              "key" : "osx",
              "doc_count" : 556
            },
            {
              "key" : "win 8",
              "doc_count" : 556
            }
          ]
        }
      }
    ]
  }
}
```

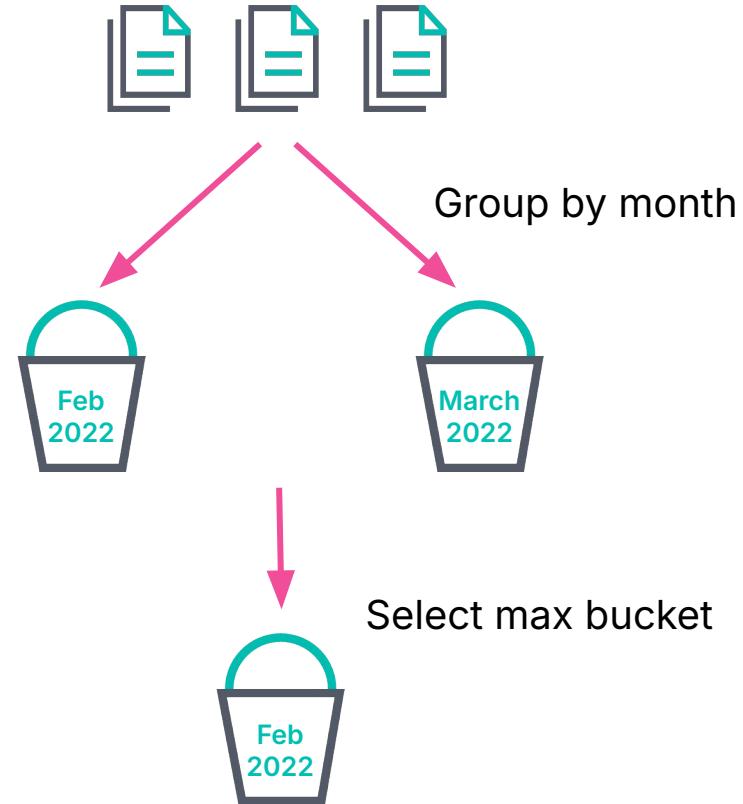
Buckets by month

Sub-buckets by OS

Pipeline Aggregations

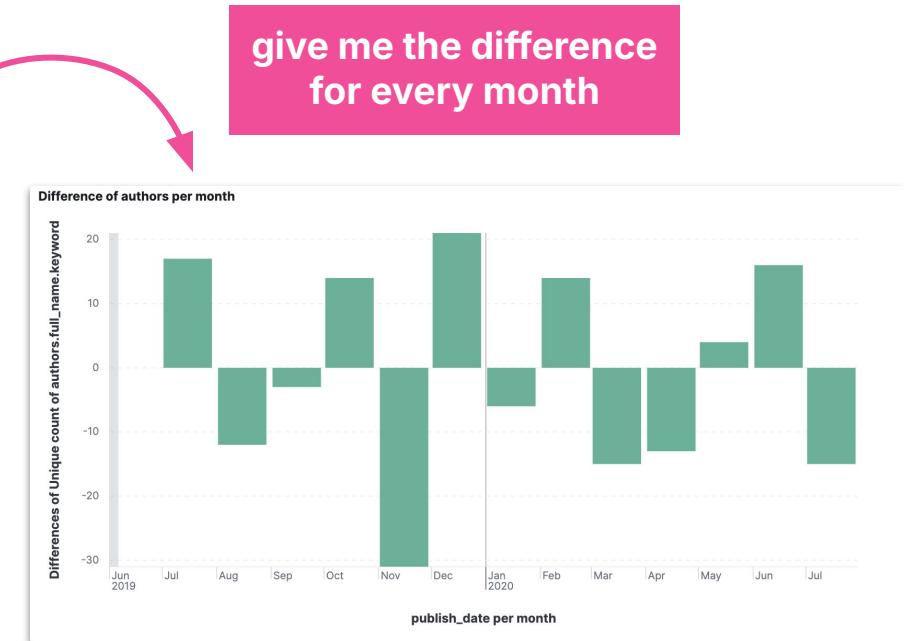
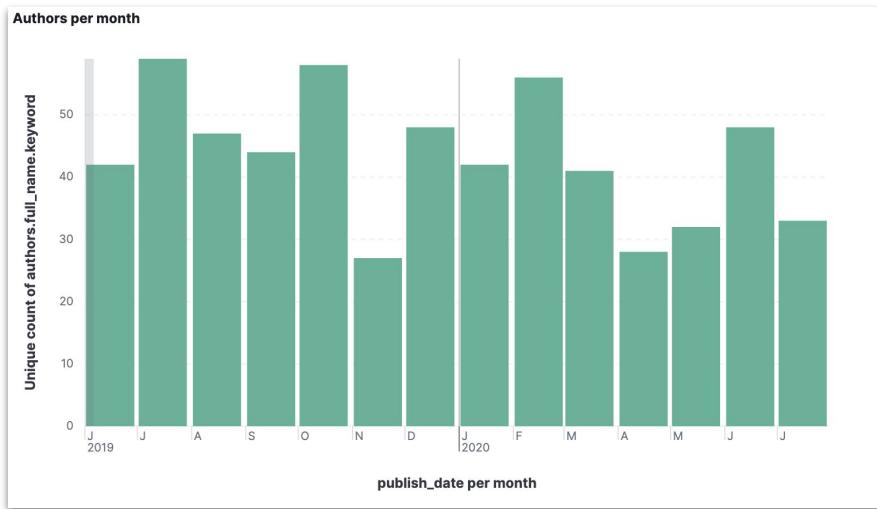
Pipeline Aggregations

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



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" },  
    "ags": {  
      "no_of_authors": {  
        "cardinality": {  
          "field": "authors.last_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" : 26,  
       "no_of_authors" : {"value" : 22},  
       "diff_author_ct": {"value" : -32},  
     },  
      {"key_as_string" : "2019-12...",  
       "key" : 1575158400000,  
       "doc_count" : 46,  
       "no_of_authors" : {"value" : 44},  
       "diff_author_ct": {"value" : 22},  
     },  
      ...  
    ] } }
```

$$44 - 22 = 22$$

Summary: Combining Aggregations

Module 4 Lesson 2

Summary

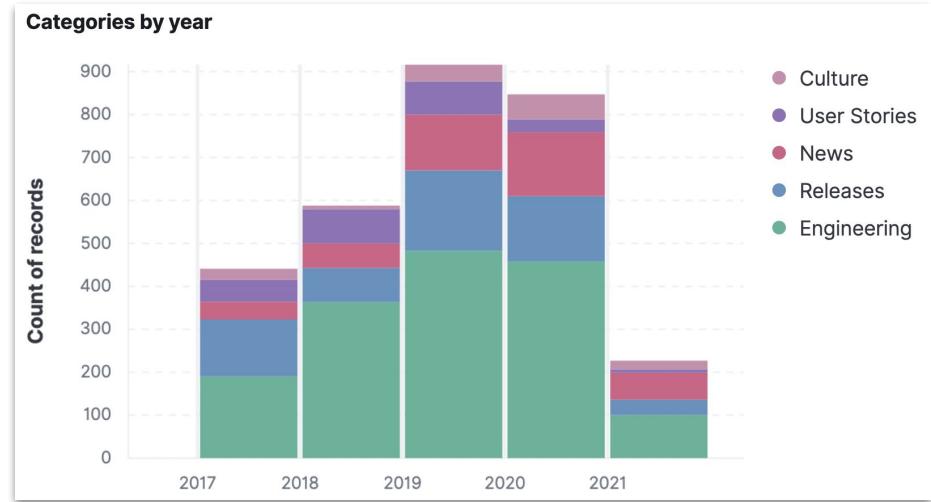


- Reduce the scope of the aggregation by combining an aggregation with a query
- Specify multiple aggregations to return multiple insights about your data in the same request
- Bucket aggregations support bucket or metric sub-aggregations
- Pipeline aggregations take input from other aggregations instead of documents or fields

Quiz



1. You create a date_ histogram to group documents by month and then calculate the moving average of the “price” field over a three-month window. What type of aggregation have you created?
2. What aggregations would you use to build this  Visualization?
3. **True or False:** metrics can be embedded under a bucket as a sub-aggregation



Lab 4.2

Combining Aggregations

Create search queries by combining aggregations





4.1 Metrics and Bucket Aggregations

4.2 Combining Aggregations

4.3 Transforming Data

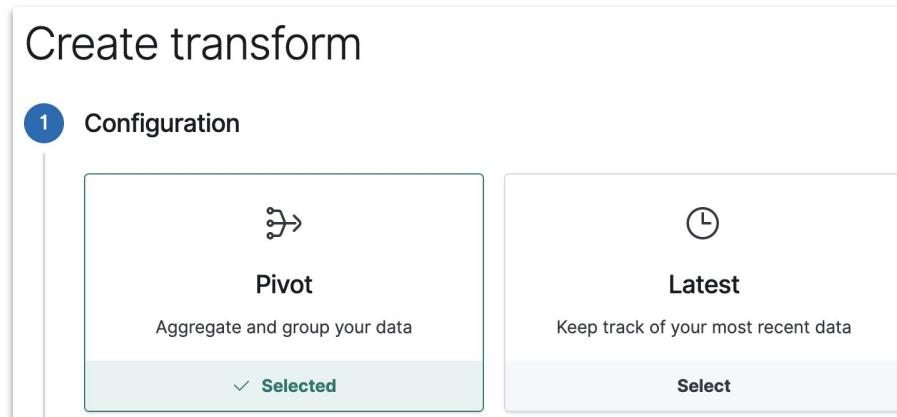
By the end of this lesson, you will be able to:

- Identify when there is a need to transform your data
- Describe the two types of transforms: pivot and latest
- Perform a transform in Kibana using event-centric data



Transform Your Data for Better Insights

- Summarize existing Elasticsearch indices using aggregations to create more efficient datasets
 - **pivot** event-centric data into entity-centric indices for improved analysis
 - retrieve the **latest** document based on a unique key, simplifying time-series data

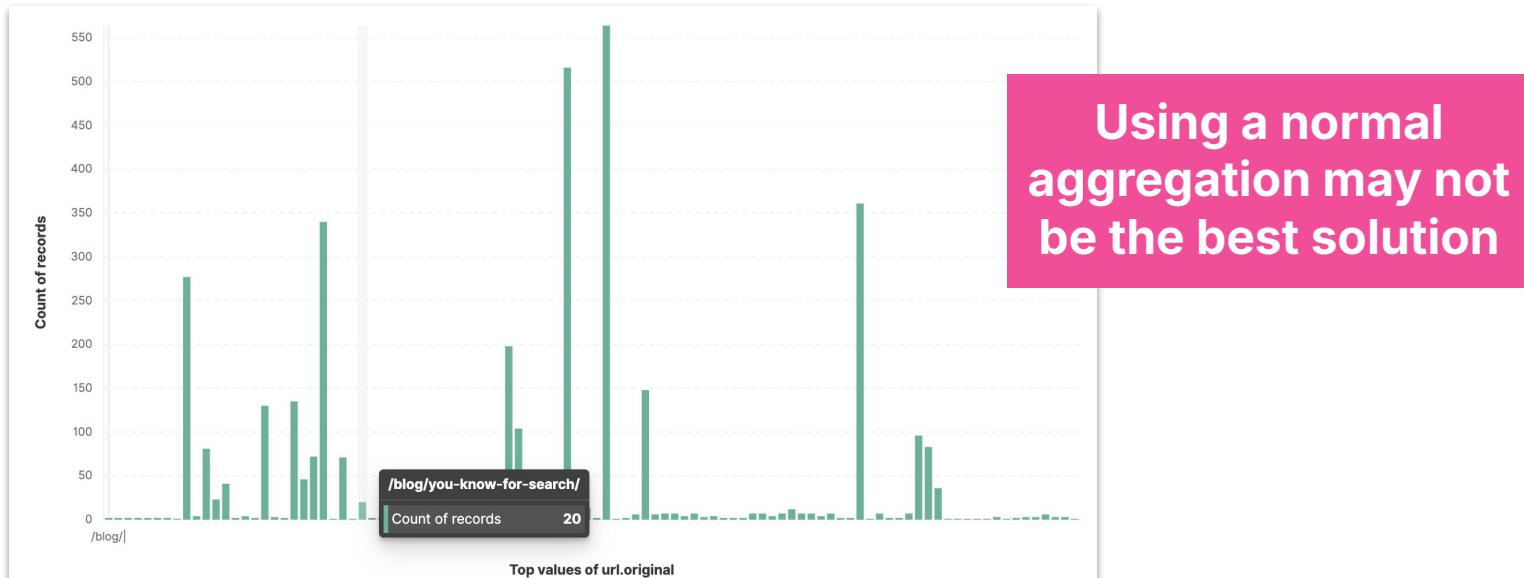


Cluster-efficient Aggregations

- Elasticsearch Aggregations provide powerful insights but can be resource-intensive with large datasets
 - complex aggregations on large volumes of data may lead to memory issues or performance bottlenecks
- Common challenges:
 - need for a complete feature index (vs. just top-N items)
 - need to sort aggregation results using pipeline aggregations
 - want to create summary tables to optimize query performance
- Solution:
 - **transform your data** to create more efficient and scalable summaries for faster, optimized querying

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_sec)** = average time to load page
- You can also edit runtime mappings directly in the UI

 Pivot

Transform configuration

Group by

url

Add a group by field ...

Aggregations

@timestamp.value_count

runtime_sec.avg

Add an aggregation ...

Preview

Columns 3 Sort fields

url	@timestamp.value_count	runtime_sec.avg
https://www.elastic.co/bl...	980	1.5160153044805842
https://www.elastic.co/bl...	957	1.5124555874456307
https://www.elastic.co/bl...	932	1.557218884753439
	953	1.5414281211801242
	901	1.5567047752679783

Check the preview of your transform

Configuring Transform Settings

- **Continuous Mode:** transforms run continuously, processing new data as it arrives
- **Retention Policy:** identify and manage out-of-date documents in the destination index
- **Checkpoints:** created each time new source data is ingested and transformed
- **Frequency:** advanced option to set the interval between checkpoints (max 1 hour)

2 **Transform details**

Transform ID
web_traffic_transform

Transform description
Transforming Web Traffic

Use transform ID as destination index name

Destination index
web_traffic_transform

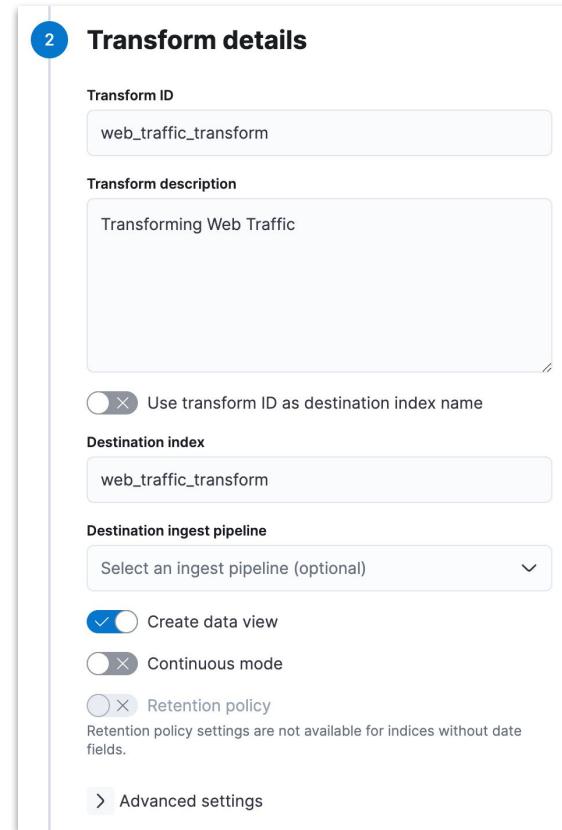
Destination ingest pipeline
Select an ingest pipeline (optional)

Create data view

Continuous mode

Retention policy
Retention policy settings are not available for indices without date fields.

> Advanced settings



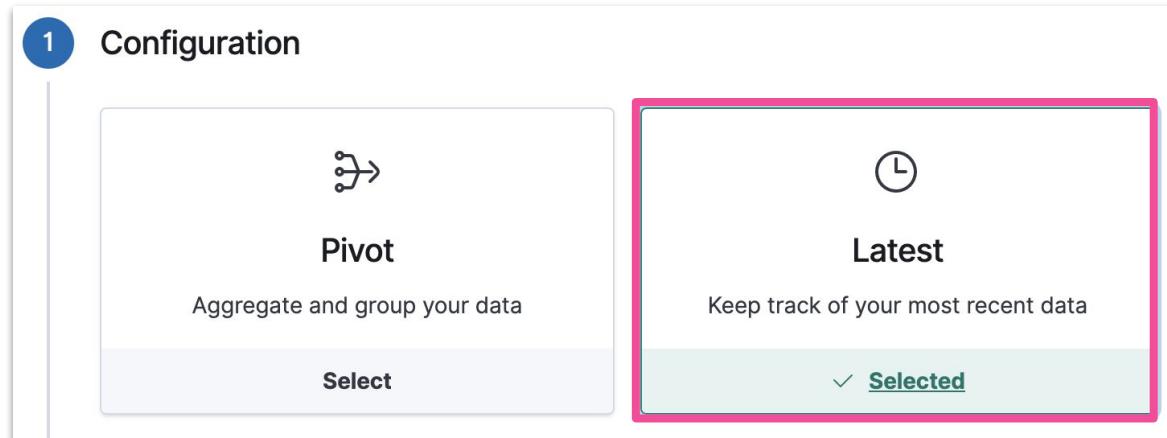
Destination Index

- Pre-create the **destination index** with custom settings for performance
 - use the **Preview transform API** to review **generated_dest_index**
 - optimize index mappings and settings for efficient storage and querying
 - disable **_source** to reduce storage usage
 - use **index sorting** if grouping by multiple fields

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

Latest Transforms

- Use **Latest** transforms to copy the most recent documents into a new index
- Examples:
 - track the latest **purchase** for each customer
 - capture the latest **event** for each host



Summary: Transforming Data

Module 4 Lesson 3

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.

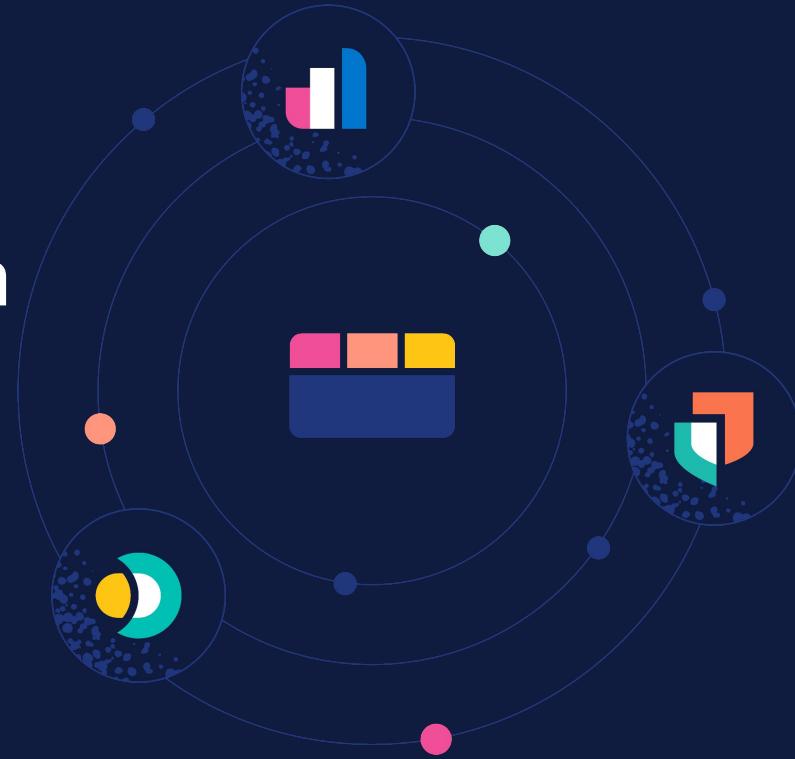
What transforms type can you use to answer the following question?

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

Lab 4.3

Transforming Data

Using transforms to
create more
efficient datasets



More Resources

- Aggregations
 - www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html
- Transforms
 - www.elastic.co/guide/en/elasticsearch/reference/current/transform-overview.html
- Transform APIs
 - www.elastic.co/guide/en/elasticsearch/reference/current/transform-apis.html

Elasticsearch Engineer: Agenda

- Module 1: Getting Started
- Module 2: Data Modeling
- Module 3: Search
- Module 4: Aggregations
- **Module 5: Data Processing**
- Module 6: Distributed Datastore
- Module 7: Data Management
- Module 8: Cluster Management

Module 5

Data Processing

"Learn how to transform your data by updating fields or enriching documents."



5.1 Changing Data

5.2 Enriching Data

5.3 Runtime Fields

By the end of this lesson, you will be able to:

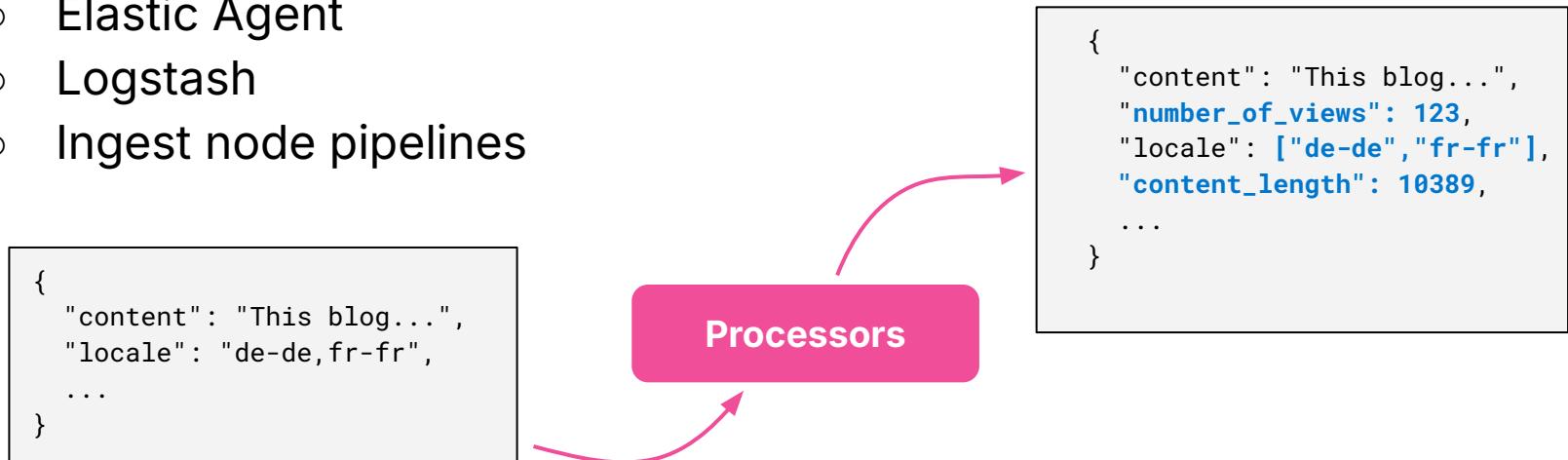
- Create pipelines with a set of actions to update your data
- Use the pipelines to apply changes to incoming documents or the documents of your existing indices



Processors

Processors

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



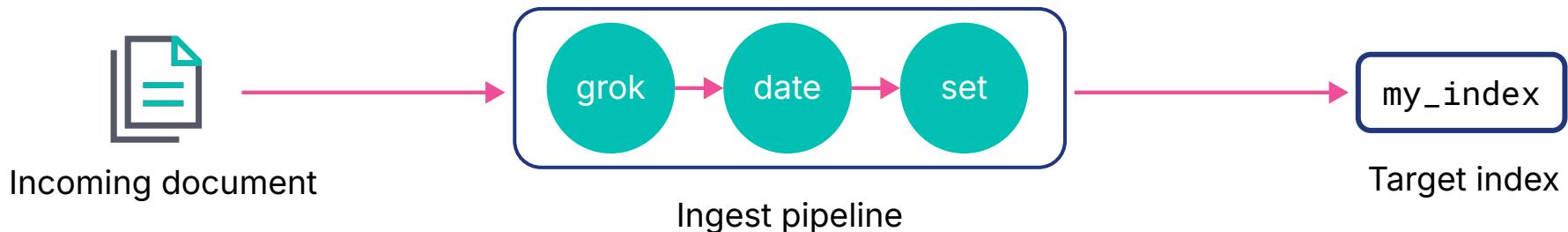
Processors

- Elastic Agent 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">• set• remove• rename• dot_expander• ...	<ul style="list-style-type: none">• split/join• grok• dissect• gsub• ...	<ul style="list-style-type: none">• csv/json• geoip• user_agent• script• pipeline• ...

Ingest Node Pipelines

- **Ingest node pipelines**
 - perform common transformations on your data before indexing
 - consist of a series of **processors** running sequentially
 - 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 Kibana. At the top, there's a breadcrumb navigation: Stack Management > Ingest Pipelines > Create pipeline. To the right is a 'Create pipeline docs' link. The main form has fields for 'Name' (set_views) and 'Description' (optional). Below this is a 'Processors' section with a sub-section 'Add your first processor'. It includes a note about using processors to transform data before indexing, a 'Learn more' link, and buttons for 'Add a processor' and 'Import processors'. To the right of the processor section are two toggle switches: 'Add version number' (disabled) and 'Add metadata' (disabled). Further right is a 'How to use this pipeline during data ingestion' section with a 'Bulk request' button and a code snippet for a PUT request to create a pipeline named 'my-pipeline':

```
PUT books/_bulk?pipeline=my-pipeline
{
  "create": {
    "index": "books",
    "id": "12345",
    "score": 1.0,
    "type": "book",
    "source": {
      "title": "Snow Crash",
      "author": "Neal Stephenson"
    }
  },
  "create": {
    "index": "books",
    "id": "12346",
    "score": 1.0,
    "type": "book",
    "source": {
      "title": "Revelation Space",
      "author": "Alastair Reynolds"
    }
  }
}
```

At the bottom left is a green 'Create pipeline' button with a checkmark icon, and at the bottom right is a 'Cancel' button.

Example of a Pipeline

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

Add processor

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

66f551b0c5f2ef72e2a7125b

[Add document](#) ✓ Document added

Documents [Clear all](#)

```
"locale": "en-us",
"content": "04 May 2016 \"There are two hard problems
in .NET, caching expressions and naming things\" By
Martijn Laarman Share Share on Twitter Share on
Facebook Share on LinkedIn We recently released NEST
2.3.1 that contains an important fix for a memory leak
```

Use the Pipeline

Apply a pipeline to documents in indexing requests

```
POST new_index/_doc?pipeline=set_views  
{"foo": "bar"}
```

Set a default pipeline

```
PUT new_index  
{  
  "settings": {  
    "default_pipeline":  
      "set_views"  
  }  
}
```

Only applied if no other
pipeline is used

Set a final pipeline

```
PUT new_index  
{  
  "settings": {  
    "final_pipeline":  
      "set_views"  
  }  
}
```

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 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**
 - add a **query**

```
POST _reindex
{
  "max_docs": 100,
  "source": {
    "index": "blogs",
    "query": {
      "match": {
        "versions": "6"
      }
    }
  },
  "dest": {
    "index": "blogs_fixed"
  }
}
```

Apply a pipeline

- All the documents from **old_index** will go through the pipeline before being indexed to **new_index**

```
POST _reindex
{
  "source": {
    "index": "old_index"
  },
  "dest": {
    "index": "new_index",
    "pipeline": "set_views"
  }
}
```

Reindex from a Remote Cluster

- Connect to the remote Elasticsearch node using basic auth or API key
- 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
 - use a pipeline to update the **_source**

```
POST blogs/_update_by_query?pipeline=set_views
{
  "query": {
    "match": { "category" : "customers" }
  }
}
```

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.title.keyword": "David Kravets"
    }
  }
}
```

Summary: Changing Data

Module 5 Lesson 1

Summary



- **Ingest pipelines** use Elasticsearch Ingest APIs to perform common transforms to your data before indexing
- Pipelines can also be defined using Elastic Agent or Logstash
- Use Kibana **Ingest Pipelines UI** to manage your pipelines
- 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

Quiz



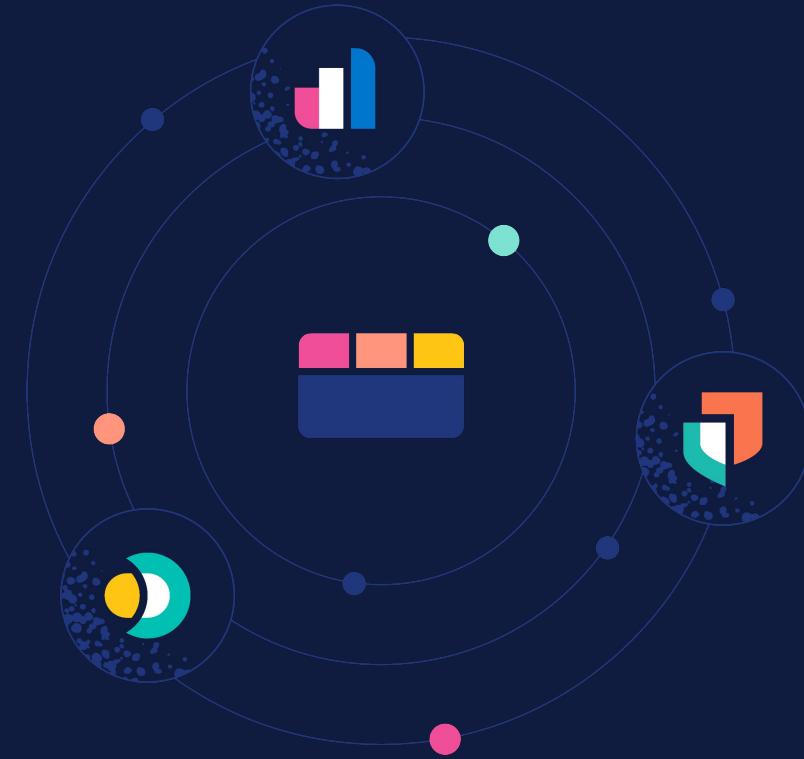
1. **True or False:** In a **Reindex** request, you can add a query clause to only reindex a subset of the documents.
2. **True or False:** The processors of an ingest pipeline are executed in parallel.
3. What does the following configuration do?

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

Lab 5.1

Changing Data

Define ingest pipelines
and use it to apply
updates to your
indices





5.1 Changing Data

5.2 Enriching Data

5.3 Runtime Fields

By the end of this lesson, you will be able to:

- Identify scenarios where denormalization is necessary for performance optimization.
- Enhance your data by using the enrich processor

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**

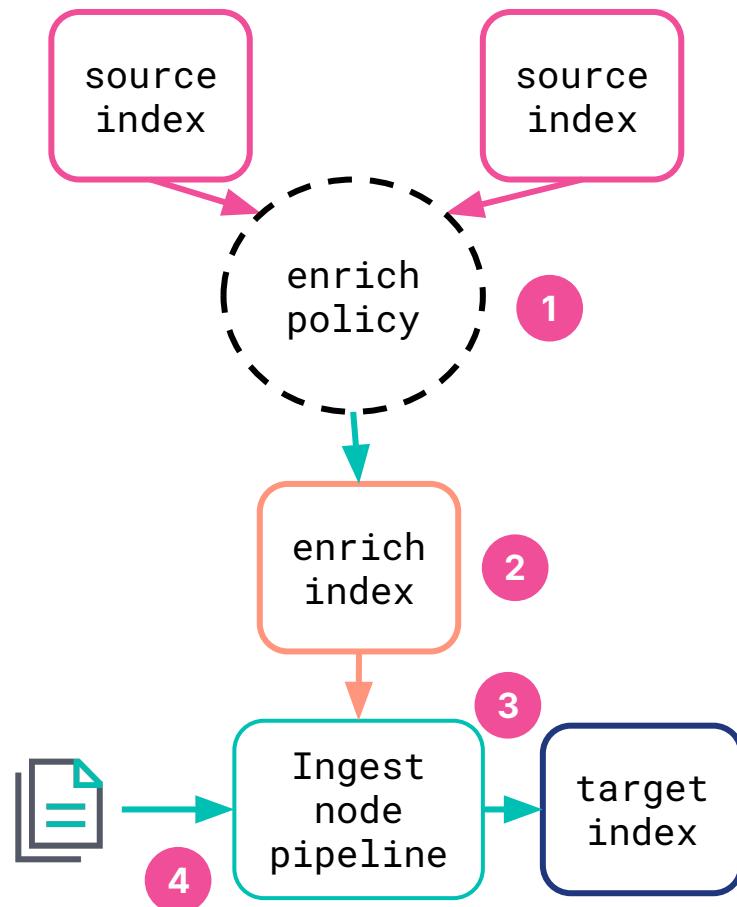
- In the **blogs** index, there is a **category** field with ID values
- There is an associated **categories** index with **uid** and **title** fields

blogs		categories	
title	category	uid	title
How finding time for cooking and culture helps forge better connections at work	bltc253e0851420b088	blt26ff0a1ade01f60d	customers
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	company-news
		blt1d90b8e0edce3ea9	how-to

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
 1. Set up an enrich policy
 2. Create an enrich index for the policy
 3. Create an ingest pipeline with an enrich processor
 4. 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
{
  "match": {
    "indices": "categories",
    "match_field": "uid",
    "enrich_fields": ["title"]
  }
}
```

policy name

match / geo_match / range

one or more source indices

match field from the source indices

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
```

- When executed, the enrich policy creates a system index called the enrich index
 - the processor uses this index to match and enrich incoming documents
 - it is read-only, meaning you can't directly change it
 - more efficient than directly matching incoming document to the source indices

Step 3: Create Ingest Pipeline with Enrich Processor

```
PUT /_ingest/pipeline/categories_pipeline
{
  "processors" : [
    {
      "enrich" : {
        "policy_name": "cat_policy",
        "field" : "category", _____
        "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
```

Updating the policy

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

1. create and execute a new enrich policy
2. replace the previous enrich policy with the new enrich policy in any in-use enrich processors
3. use the delete enrich policy API or Index Management in Kibana to delete the previous enrich policy

Updating an enrich index

- Once created, you cannot update or index documents to an enrich index. Instead,
 - update your source indices and execute the enrich policy again
 - this creates a new enrich index from your updated source indices
 - previous enrich index will deleted with a delayed maintenance job, by default this is done every 15 minutes
- you can reindex or update any already ingested documents using your ingest pipeline.

Performance considerations

- The enrich processor performs several operations and may impact the speed of your ingest pipeline.
- We strongly recommend testing and benchmarking your enrich processors before deploying them in production.
- We do not recommend using the enrich processor to append real-time data. The enrich processor works best with reference data that doesn't change frequently

Summary: Enriching Data

Module 5 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 an enrich policy first
 - Create an enrich index for the policy
 - Create an ingest pipeline with an enrich processor
 - Use the pipeline to enrich incoming documents

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 manually update the enrich policy.

Lab 5.2

Enriching Data

Add a new field
using based on
another index





5.1 Changing Data

5.2 Enriching Data

5.3 Runtime Fields



By the end of this lesson, you will be able to:

- Write Painless scripts
- Use scripts to create runtime fields

Painless Scripting

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" | "id" : "...",  
    "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
 - 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** or **ctx**

```
PUT _ingest/pipeline/url_parser
{
  "processors": [
    {
      "script": {
        "source": "ctx['new_field'] = ctx['url'].splitOnToken('/')[2]"
      }
    }
  ]
}
```

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 description (optional)
 - a value (optional)
 - a format (optional)

Create field
Data view: 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, Lens, Visualize, and TSVB. Useful for shortening a long field name. Queries and filters use the original field name.

Set custom description
Add a description to the field. It's displayed next to the field on the Discover, Lens, and Data View Management pages.

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.

[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

Create field

Data view: Kibana Sample Data eCommerce

Name: my_field, Type: Keyword

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

Set custom description
Add a description to the field. It's displayed next to the field on the Discover, Lens, and Data View Management pages.

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: wJexHY4BV-4XF4AiBUa

Filter fields:

my_field	Hello world
category	Men's Clothing
category.keyword	Men's Clothing
currency	EUR
customer_first_name	Eddie
customer_first_name.keyword	Eddie
customer_full_name	Eddie Underwood
customer_full_name.keyword	Eddie 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.getDayOfWeekEnum().toString())"
      }
    }
  }
}
```

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['publish_date'].value.getDayOfWeekEnum().toString())"
      }
    }
  },
  "aggs": {
    "my_agg": {
      "terms": {
        "field": "day_of_week"
      }
    }
  }
}
```

Summary: Runtime Fields

Module 5 Lesson 3

Summary



- Elasticsearch compiles new scripts and stores the compiled version in a cache
- **Painless** is the scripting language used in Elasticsearch
- Painless has a Java-like syntax (and can contain actual Java code)
- **Runtime fields** allow for creating arbitrary non-indexed data fields
- Runtime fields are evaluated at query time
- How about adding few more points like

Quiz

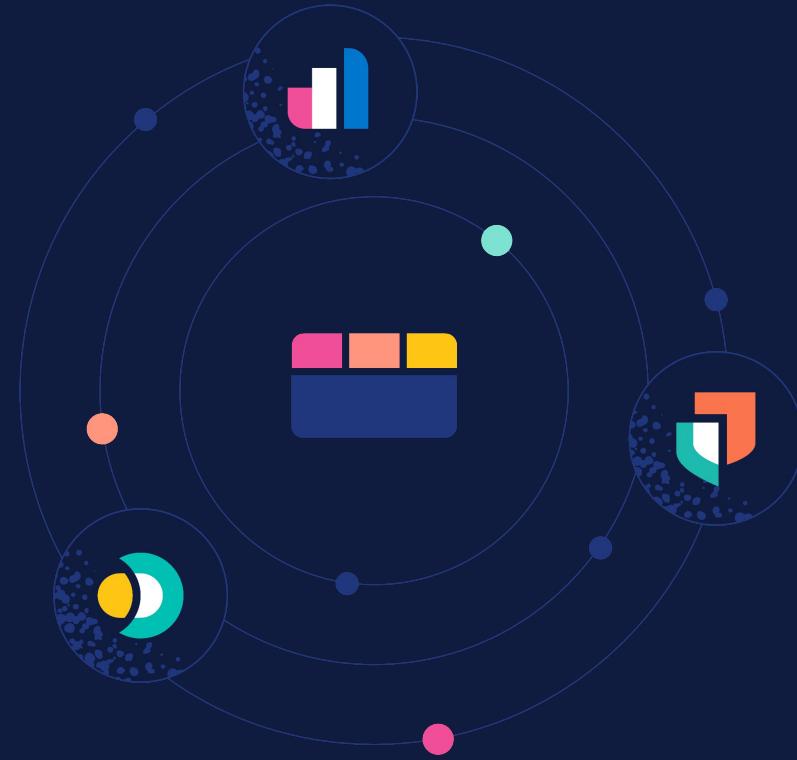


- How to avoid recompiling a script every time a parameter changes?
- **True or False:** A runtime field can be added to a mapping dynamically.
- **True or False:** You need to first define runtime fields in the mappings before using it in queries.

Lab 5.3

Runtime Fields

Write Painless
scripts and define
runtime fields



Elasticsearch Engineer: Agenda

- Module 1: Getting Started
- Module 2: Data Modeling
- Module 3: Search
- Module 4: Aggregations
- Module 5: Data Processing
- **Module 6: Distributed Datastore**
- Module 7: Data Management
- Module 8: Cluster Management

Module 6

Distributed Datastore

"Explore how Elasticsearch distributes data across nodes. This module will also dive into how distributed search and indexing operations work behind the scenes."



6.1 Understanding Shards

6.2 Scaling Elasticsearch

6.3 Distributed Operations

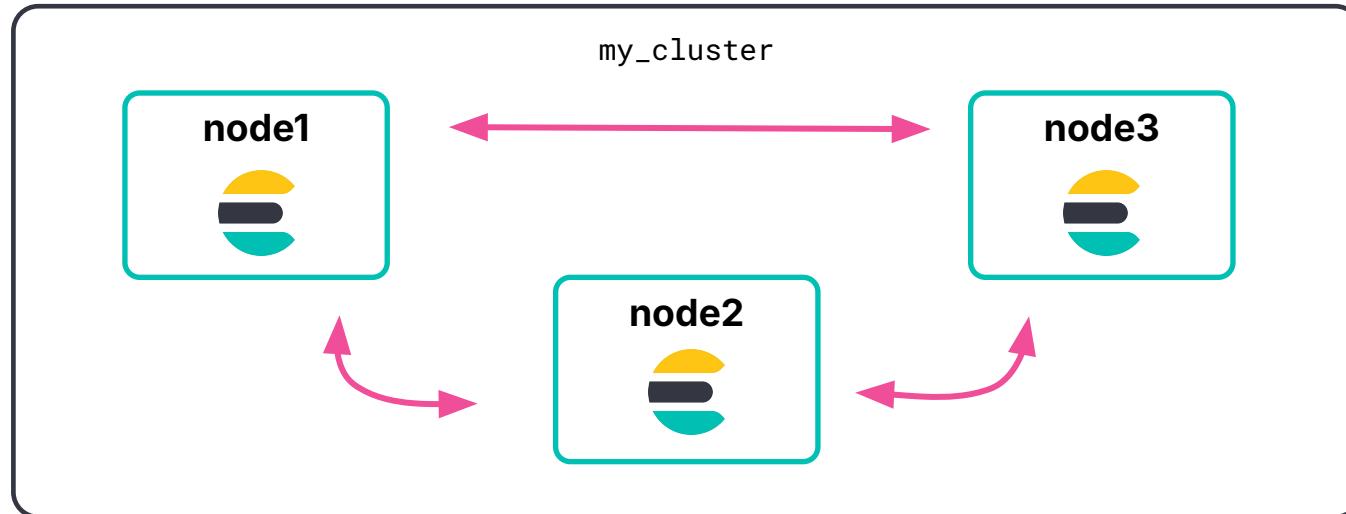
By the end of this lesson, you will be able to:

- Define the function of a shard
- Determine how primary and replica shards are allocated



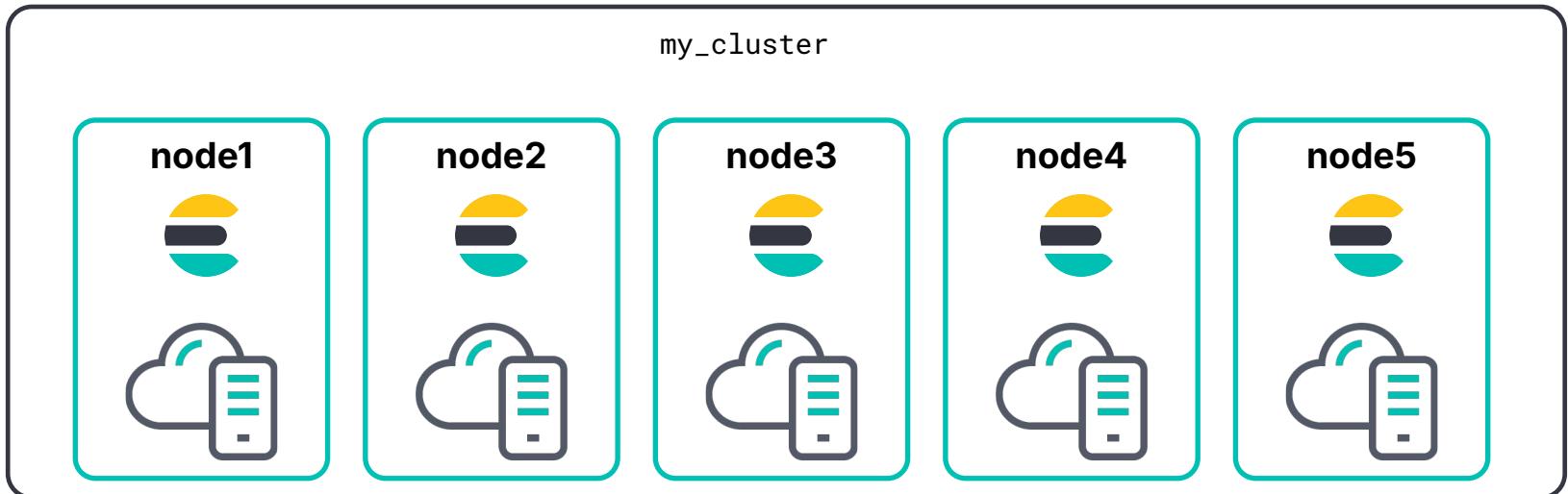
The Cluster

- 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



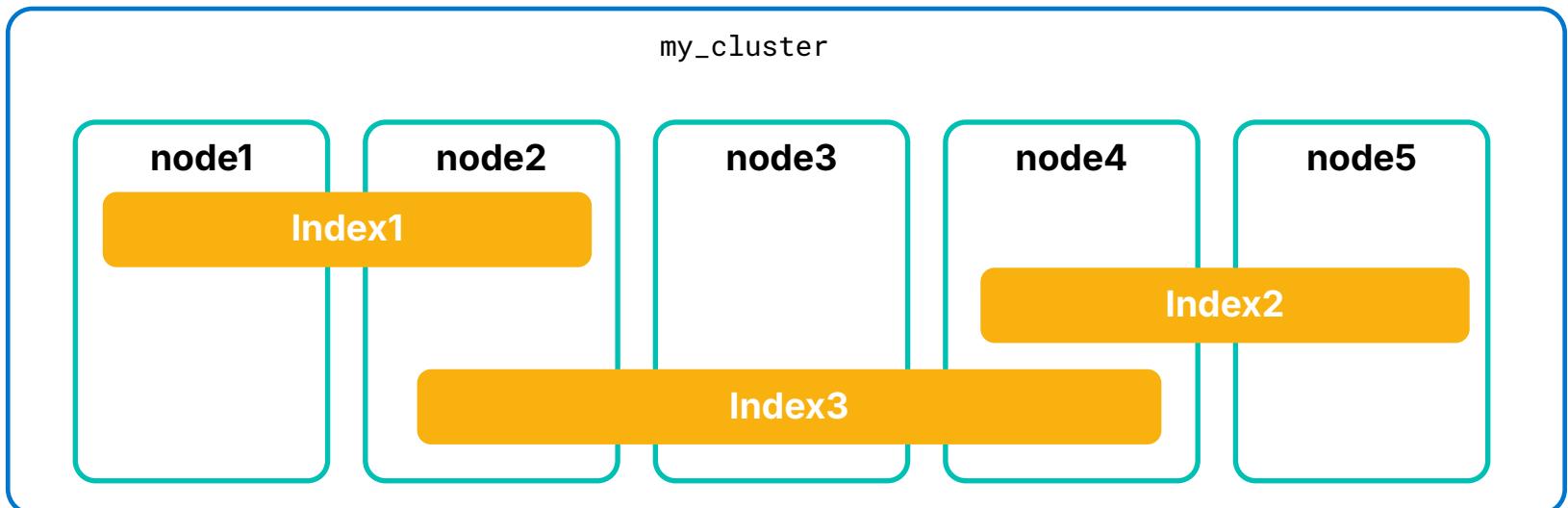
The Node

- 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



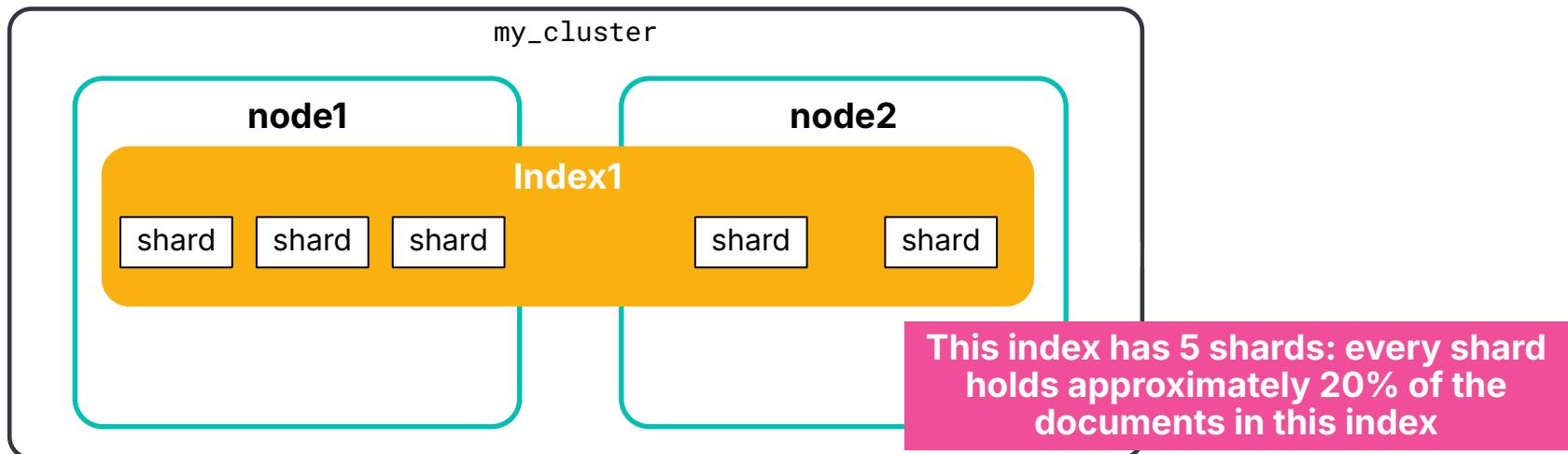
The Index

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



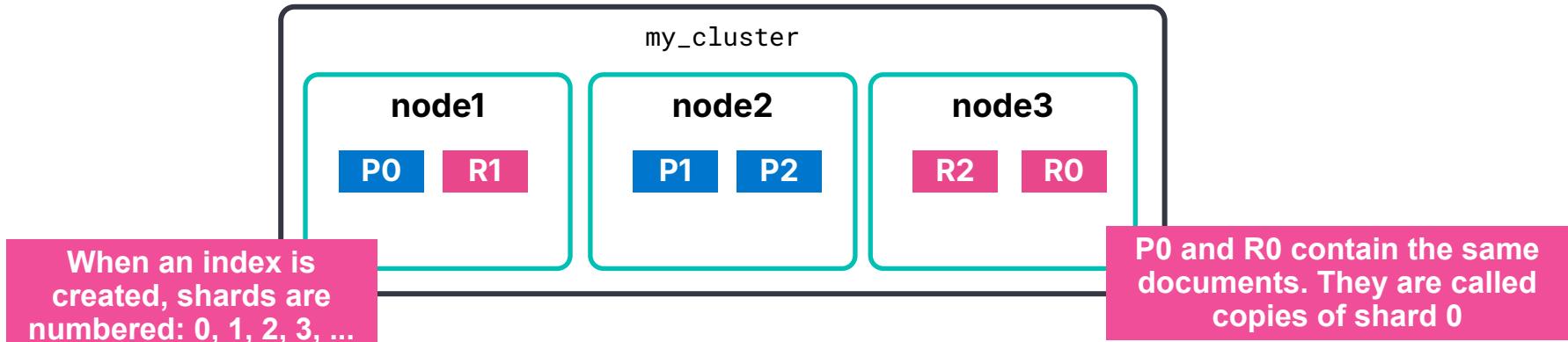
The Shard

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



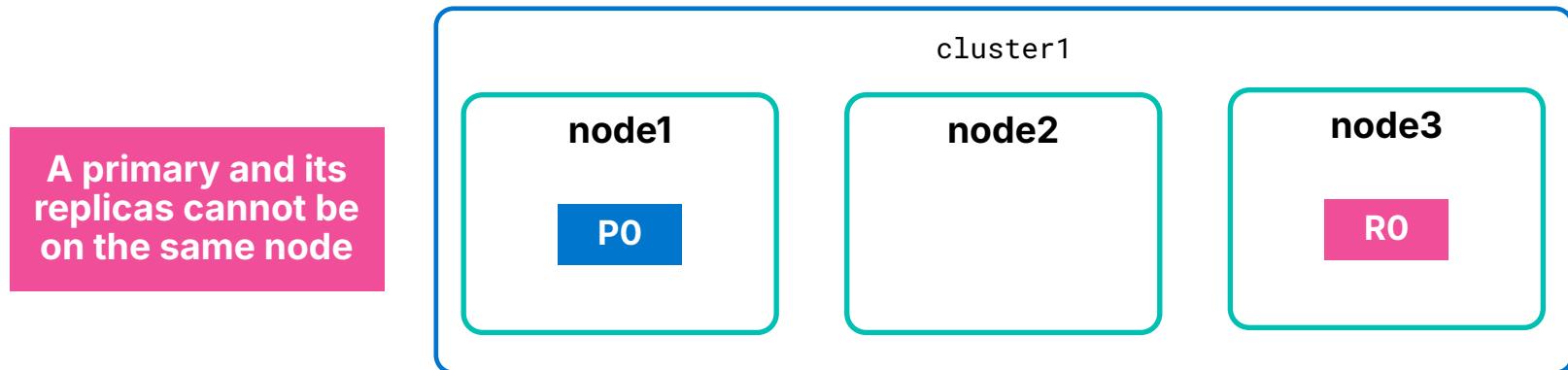
Primary vs. replica

- 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

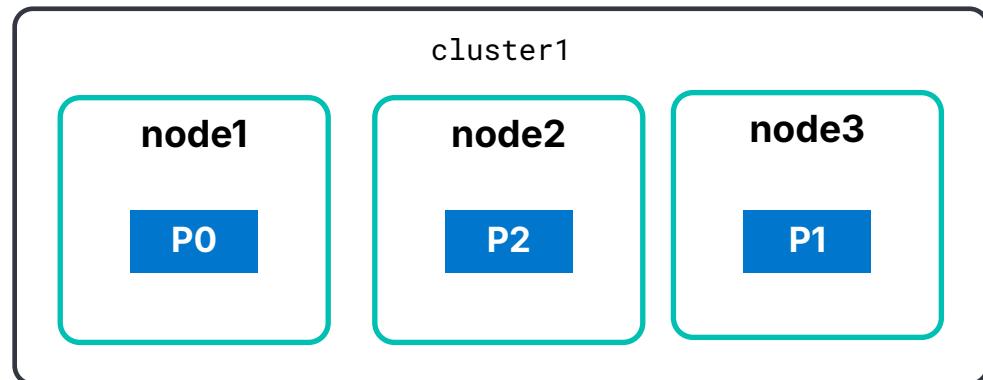


Configuring the Number of Primaries

- 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

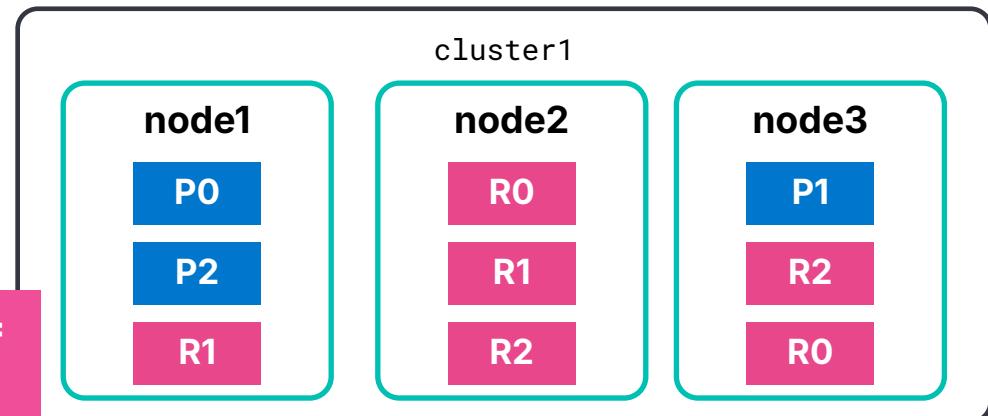
Configuring the Number of Replicas

- 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

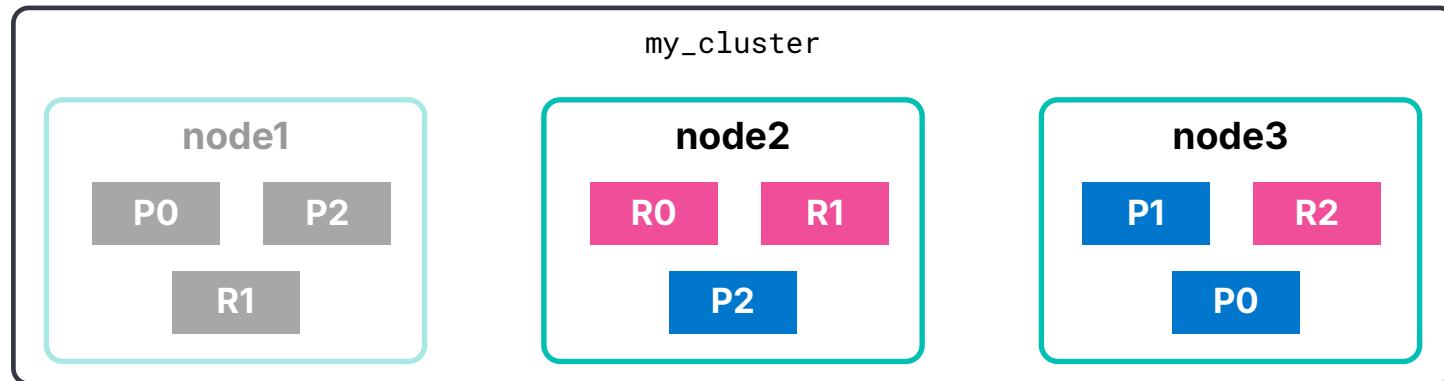
```
PUT my_new_index/_settings
{
  "number_of_replicas": 2
}
```

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

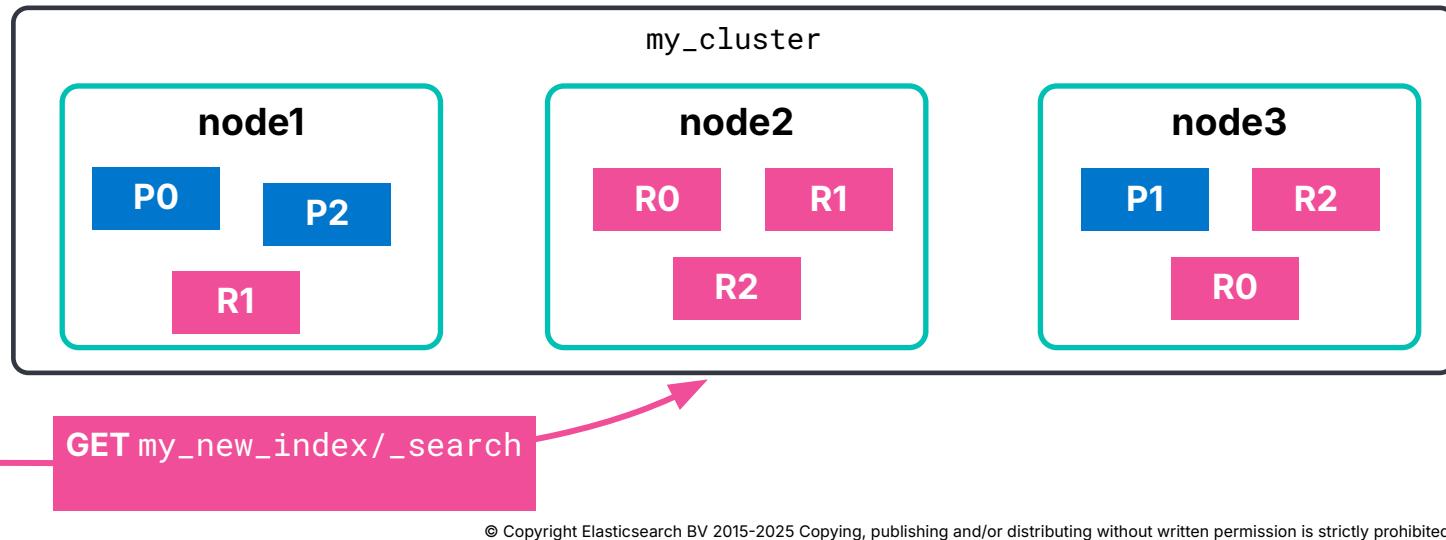


If node1 fails, its data is still available on other nodes

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

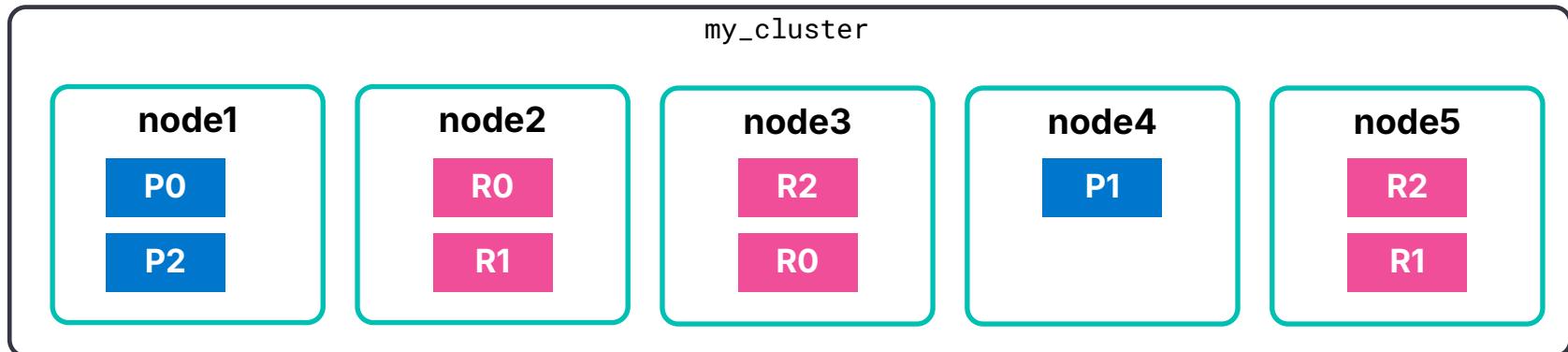
Why create replicas?

- 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.

Lab 6.1

Understanding Shards

Create new indices
with different
number of shards
and replicas



6.1 Understanding Shards

6.2 Scaling Elasticsearch

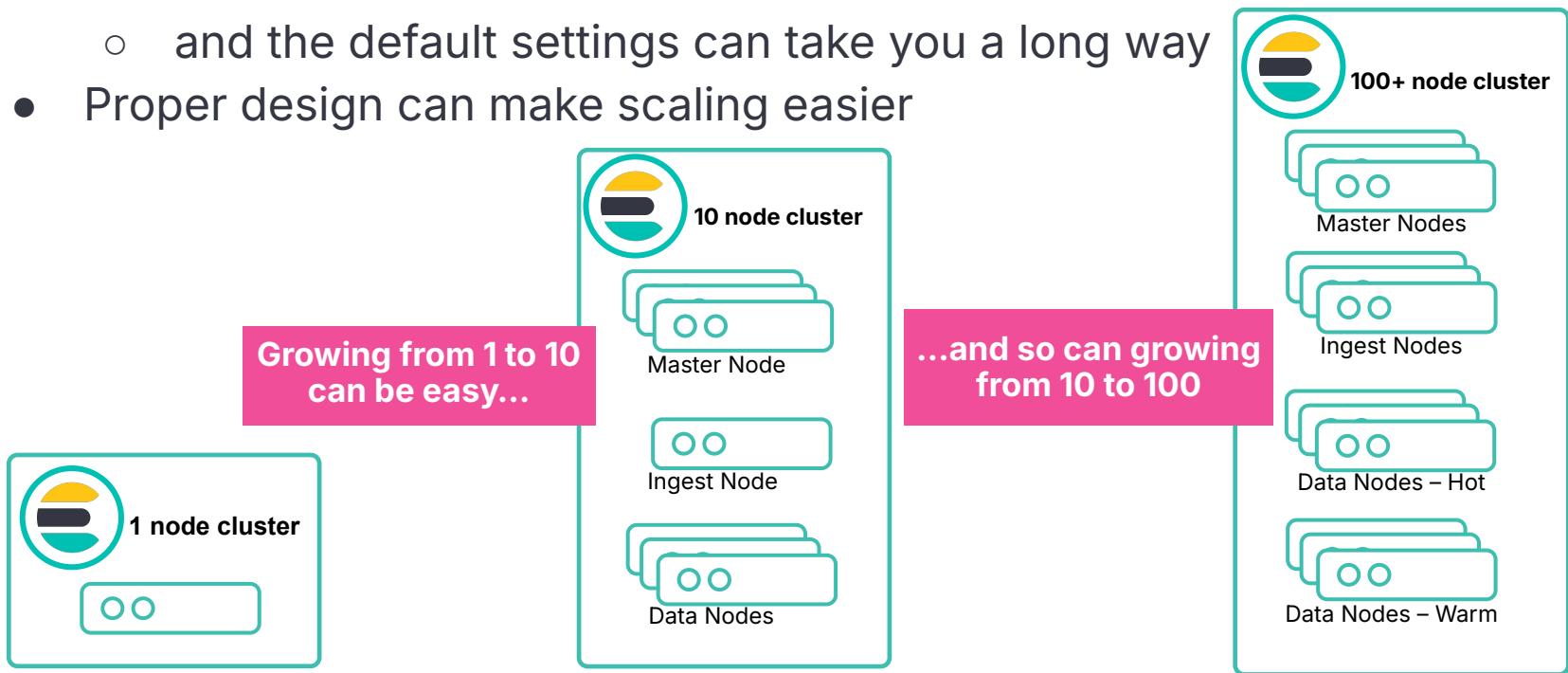
6.3 Distributed Operations

By the end of this lesson, you will be able to:

- Discuss the importance of proper design for scaling a cluster
- Choose the right number of shards for your cluster
- Optimize the performance of your cluster for both write and read operations

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
  }
}
```

node1

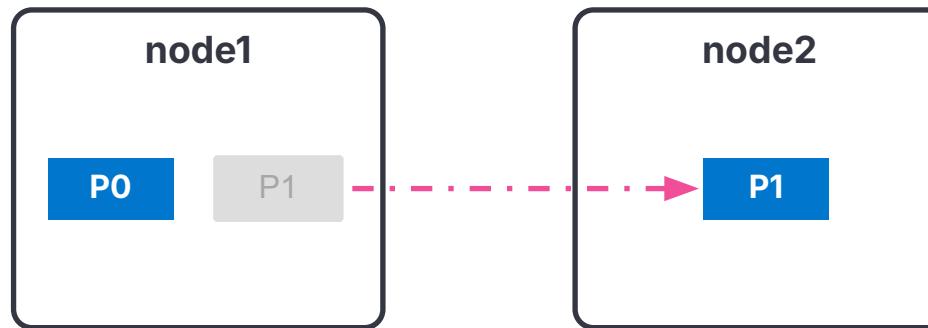
P0

P1

**Plan ahead by
overallocating my_index**

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

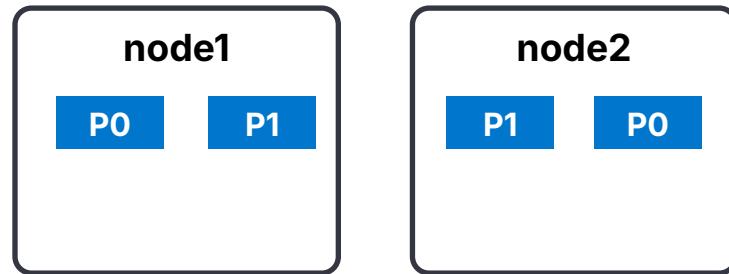
```
PUT my_index
{
  "settings": {
    "number_of_shards": 4,
    "number_of_replicas": 0
  }
}
```



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

We could easily have
this data on 10 shards

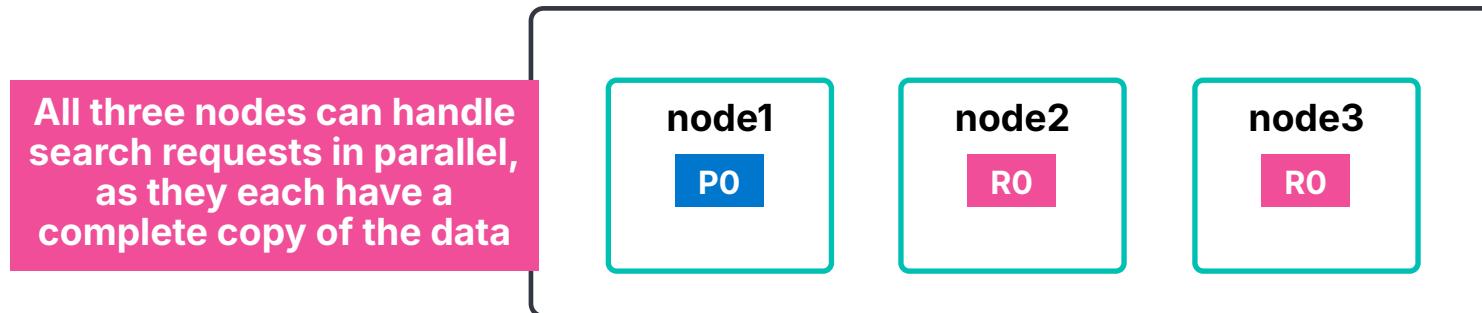
- **Common scenario**
 - 3 different logs
 - 1 index per day each
 - 5 shards
 - 6 months retention
 - ~2700 shards

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



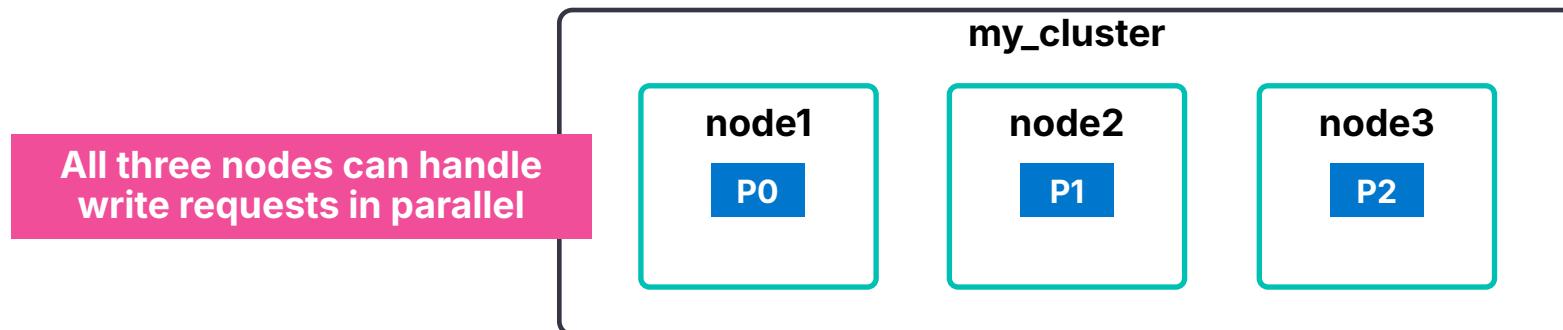
Optimizing for Read Throughput

- 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
- Use filters, as they are cacheable

Scaling for Writes

Scaling for Writes

- Write throughput scales by **increasing number of primaries**
 - having many primary shards on different nodes allows Elasticsearch to "fan out" the writes, so each node 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 three node cluster, under what circumstances would you create an index with more than three 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.

Lab 6.2

Scaling Elasticsearch

Configure shards to optimize throughput.
Scale up the cluster by adding a node





6.1 Understanding Shards

6.2 Scaling Elasticsearch

6.3 Distributed Operations

By the end of this lesson, you will be able to:

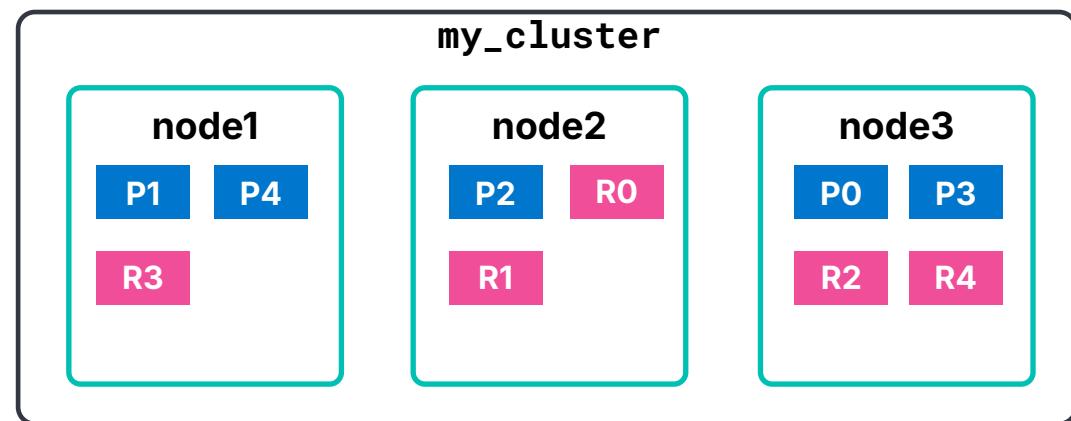
- Describe how distributed write operations are executed across multiple shards
- Illustrate how Elasticsearch routes search queries to the appropriate shards and merges results from different nodes to deliver the final response

Write Operations

How Data Goes In

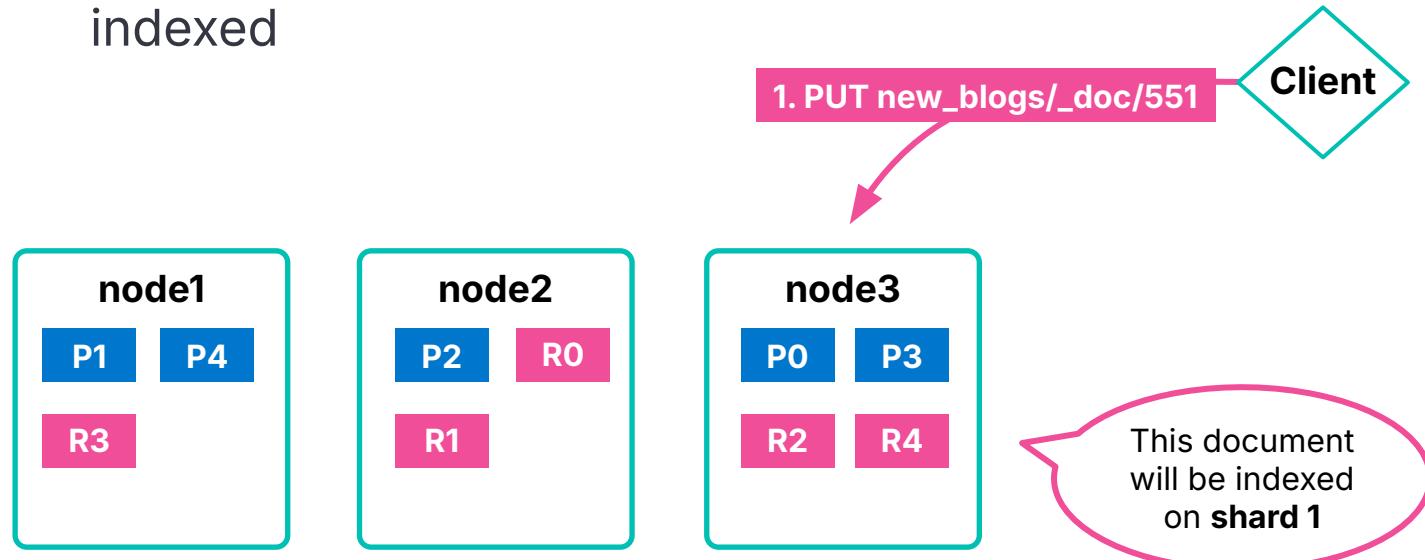
- Let's take a look at the details of how a document is indexed into a cluster
 - suppose we index the following document into an index which has **5 primary shards with 1 replica**

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



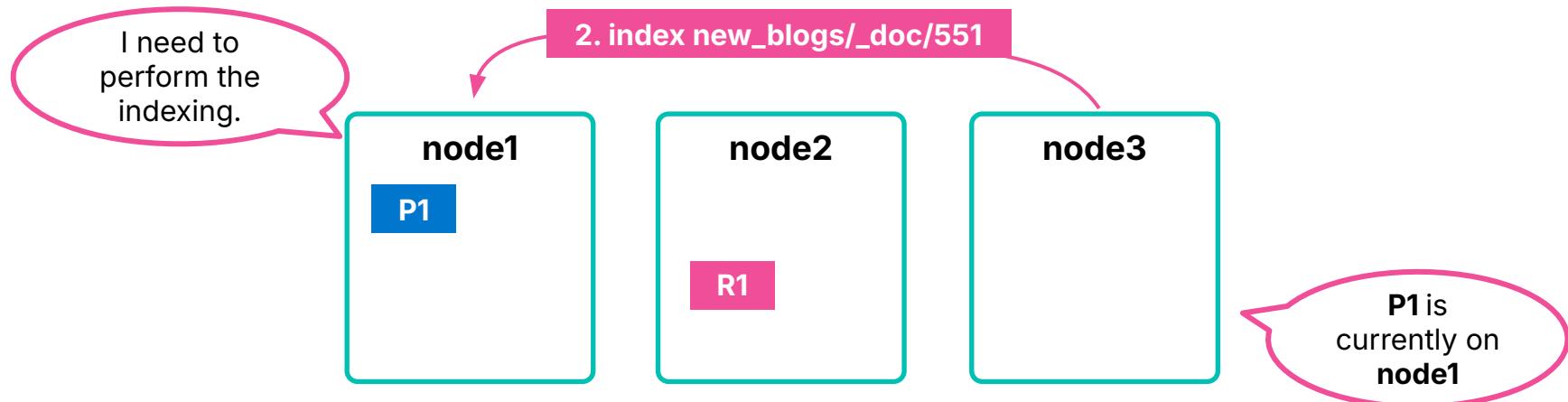
Document Routing

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



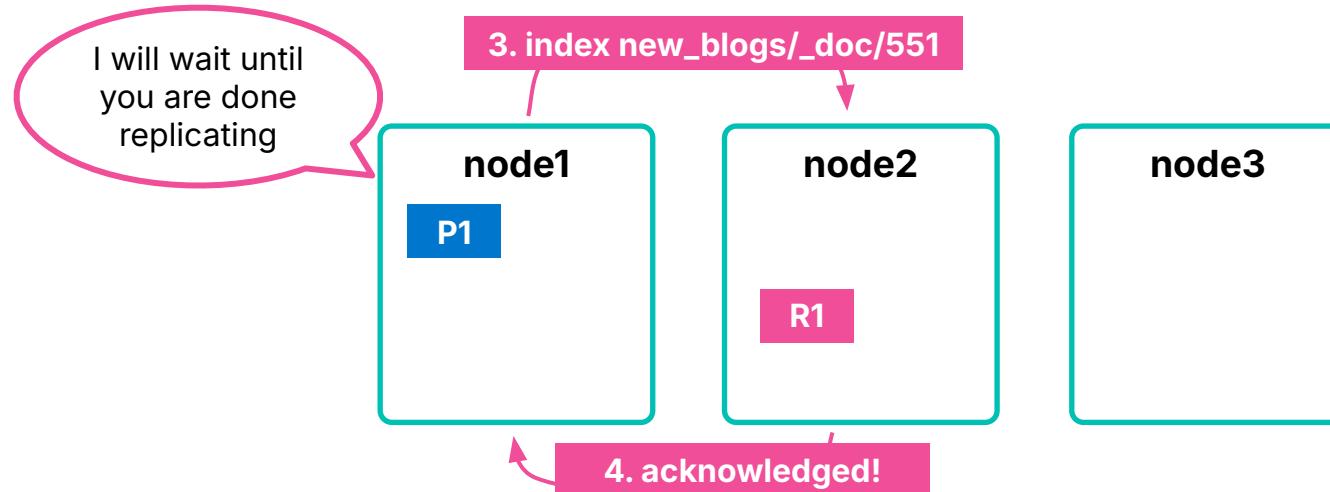
Write Operations on the Primary Shard

- 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



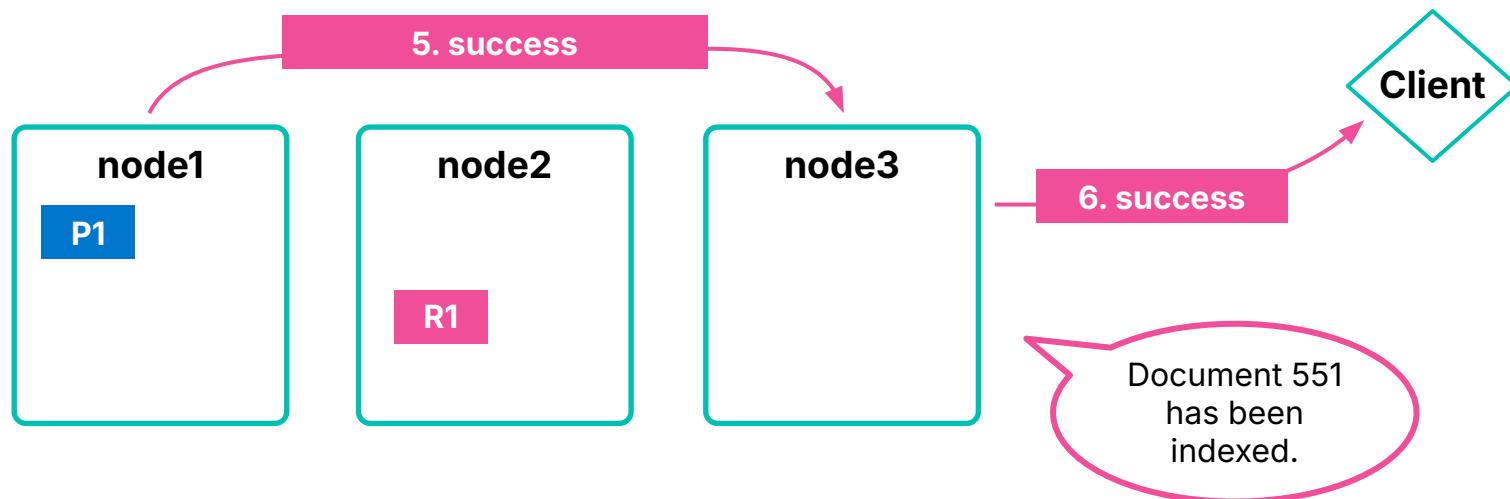
Replicas are Synced

- **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



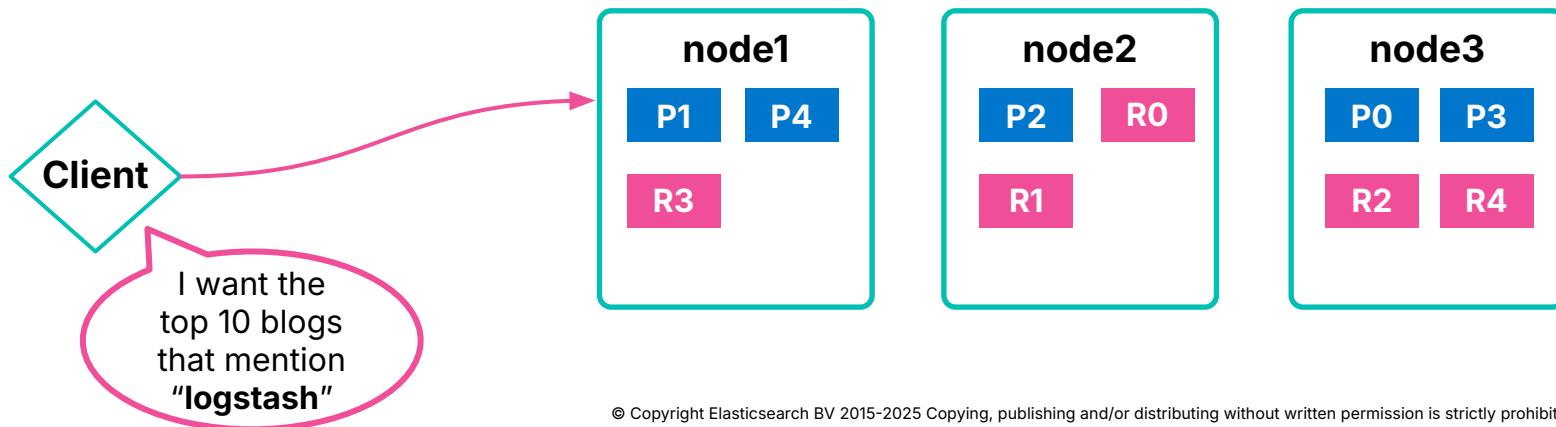
Updates and Deletes

- **Updating or deleting** is similar to indexing a document
- An **update** to a document is actually three steps:
 1. the source of the current document is retrieved
 2. the current version of the document is deleted
 3. a merged new version of the entire document is indexed

Search Operations

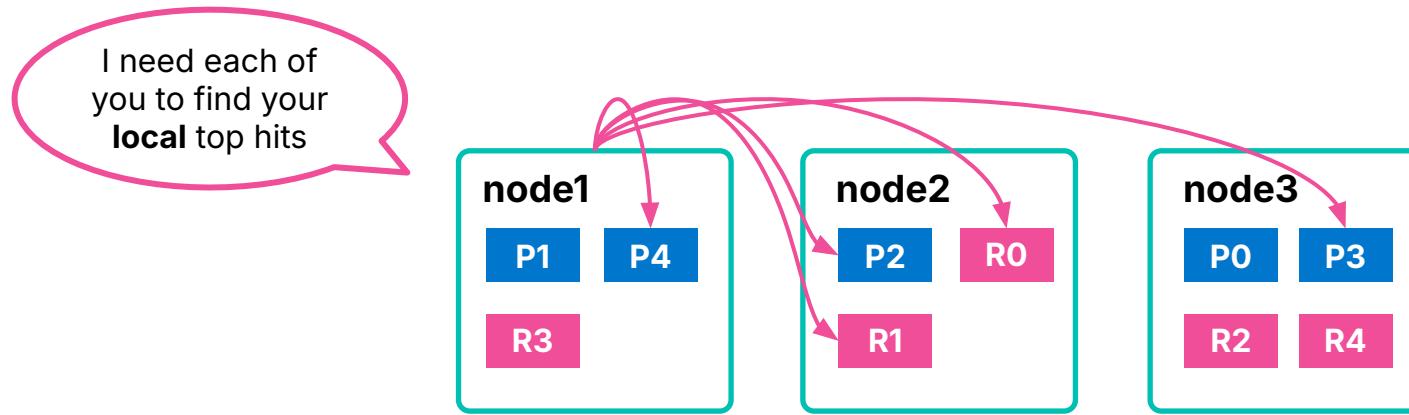
Anatomy of a Search

- 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

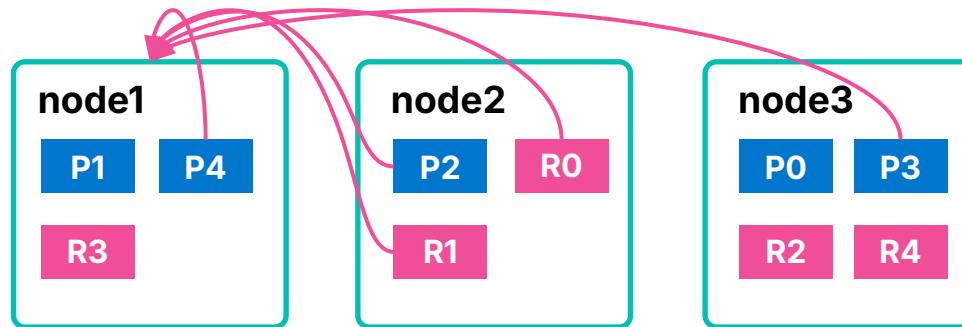
- The initial part of a search is referred to as the **scatter phase**
 - the query is broadcast to a **shard copy** of every shard in the index
 - each shard executes the query **locally**



The Scatter Phase

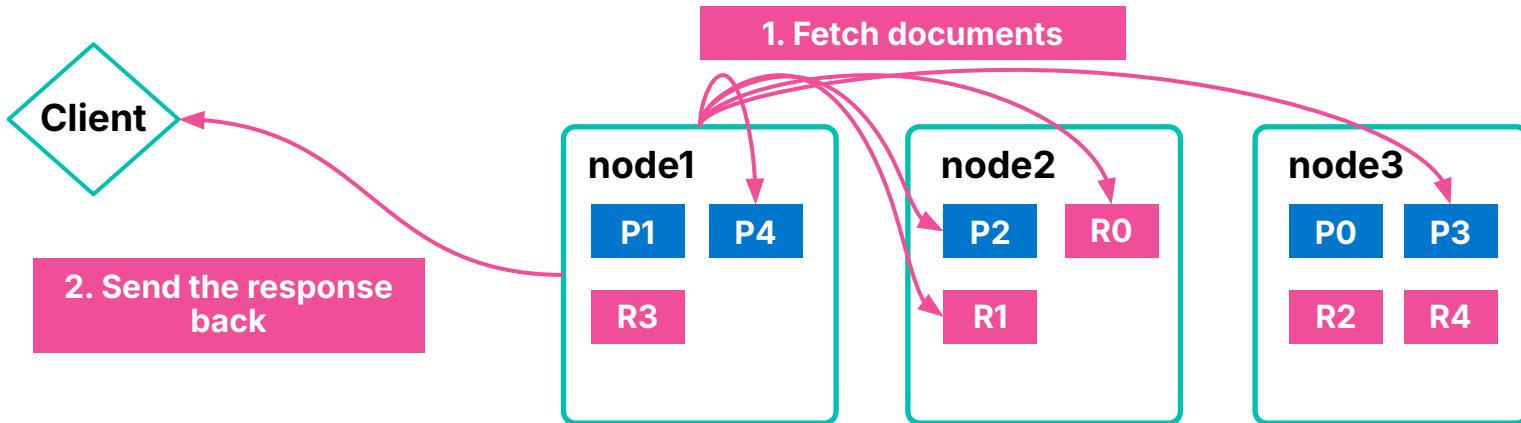
- Each shard returns the **doc IDs** and **sort values** of its top hits to the coordinating node
- The coordinating node **merges these values** to create a **globally sorted list of results**

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 scatter phase and a gather 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:** Search operation can be redirected to a replica shard.

Lab 6.3

Distributed Operations

Perform search requests on different shards.



Elasticsearch Engineer: Agenda

- Module 1: Getting Started
- Module 2: Data Modeling
- Module 3: Search
- Module 4: Aggregations
- Module 5: Data Processing
- Module 6: Distributed Datastore
- **Module 7: Data Management**
- Module 8: Cluster Management

Module 7

Data Management

"In this module, you will learn how to organize data and optimize performance across large datasets."



7.1 Data Management Concepts

7.2 Data Streams

7.3 Index Lifecycle Management

7.4 Searchable Snapshots



By the end of this lesson, you will be able to:

- Make the distinction between static and time series data
- Use aliases to handle multiple indices
- Define index templates to apply the same mappings and settings to multiple indices

Managing Data

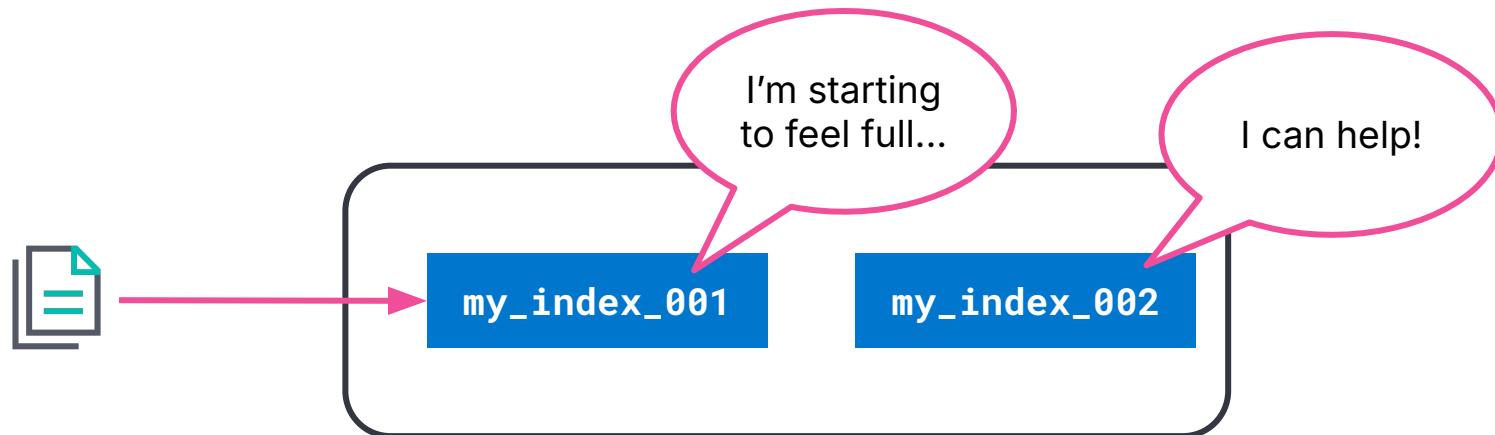
- Data management needs differ depending on the type of data you are collecting:

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

Index Aliases

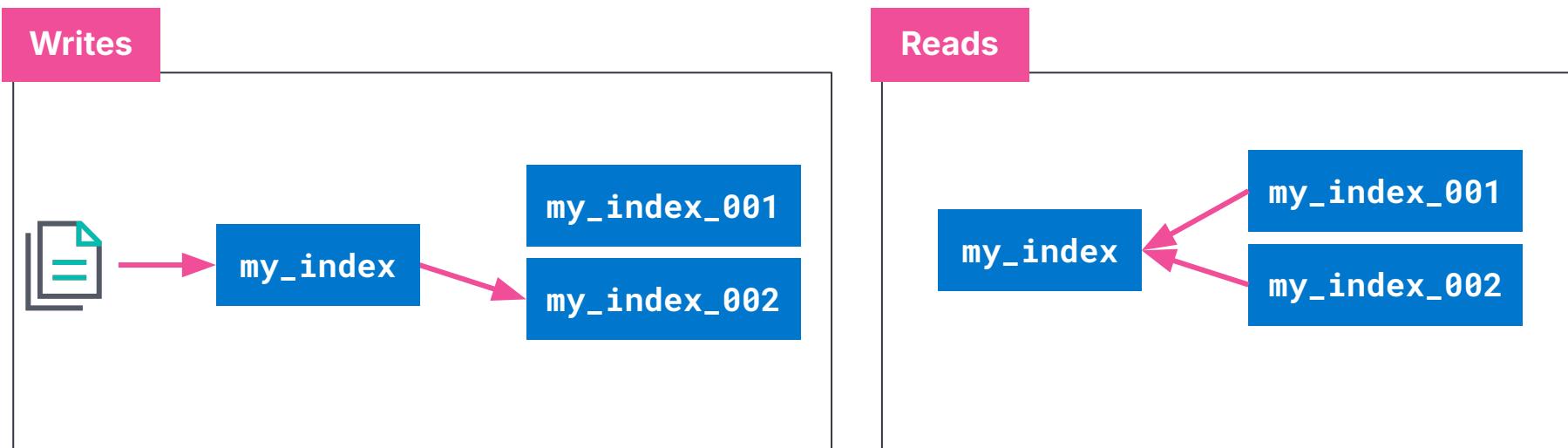
Scaling Indices

- 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`
- Define an alias at index creation

```
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

logs-template

Index patterns

The index patterns to apply to the template.

Index patterns

logs* ×



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**:

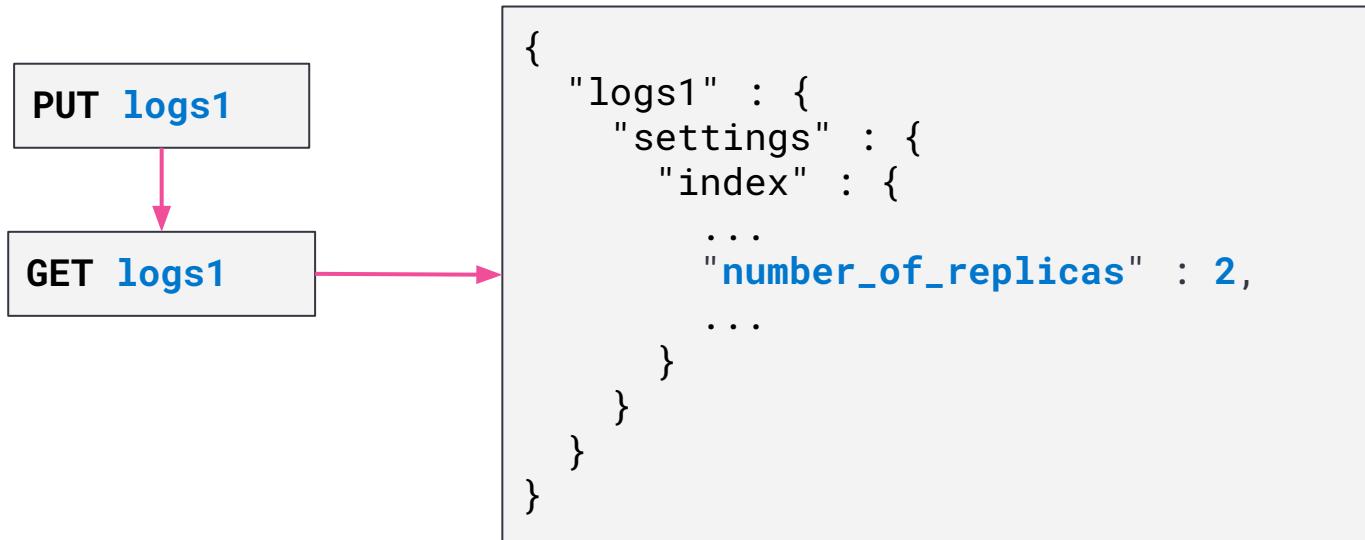
```
PUT _index_template/logs-template
{
  "index_patterns": [ "logs*" ],
  "template": {
    "settings": {
      "number_of_replicas": 2
    }
  }
}
```

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:

The screenshot shows a horizontal navigation bar with five steps: Logistics (done), Index settings (active), Mappings, Aliases, and Review. Below the bar, the 'Index settings' section is expanded, containing the heading 'Index settings (optional)', a description 'Define the behavior of your indices.', and a code editor with the following JSON configuration:

```
Index settings
{
  "index": {
    "auto_expand_replicas": "0-4"
  }
}
```

On the right side of the 'Index settings' section, there is a link 'Index settings docs' with a magnifying glass icon.

Component Template Example

- Use the component in an index template:

Edit template 'logs-template'

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: 1

= auto-relicas M S A

auto

.kibana-data-quality-dashboard-ecs-mappings M S A

.kibana-data-quality-dashboard-results-mappings M S A

M: mappings
S: index settings
A: aliases

© Copyright Elasticsearch BV 2015-2025 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/logs2
```

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
- If a new index matches more than one index template, the index template with the highest **priority** is used.
- **Component templates** are building blocks for constructing index templates

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.
3. What priority should you assign to custom index templates to ensure they do override built-in templates?

Lab 7.1

Data Management Concepts

Define component template and apply it to the indices of an alias





7.1 Data Management Concepts

7.2 Data Streams

7.3 Index Lifecycle Management

7.4 Searchable Snapshots

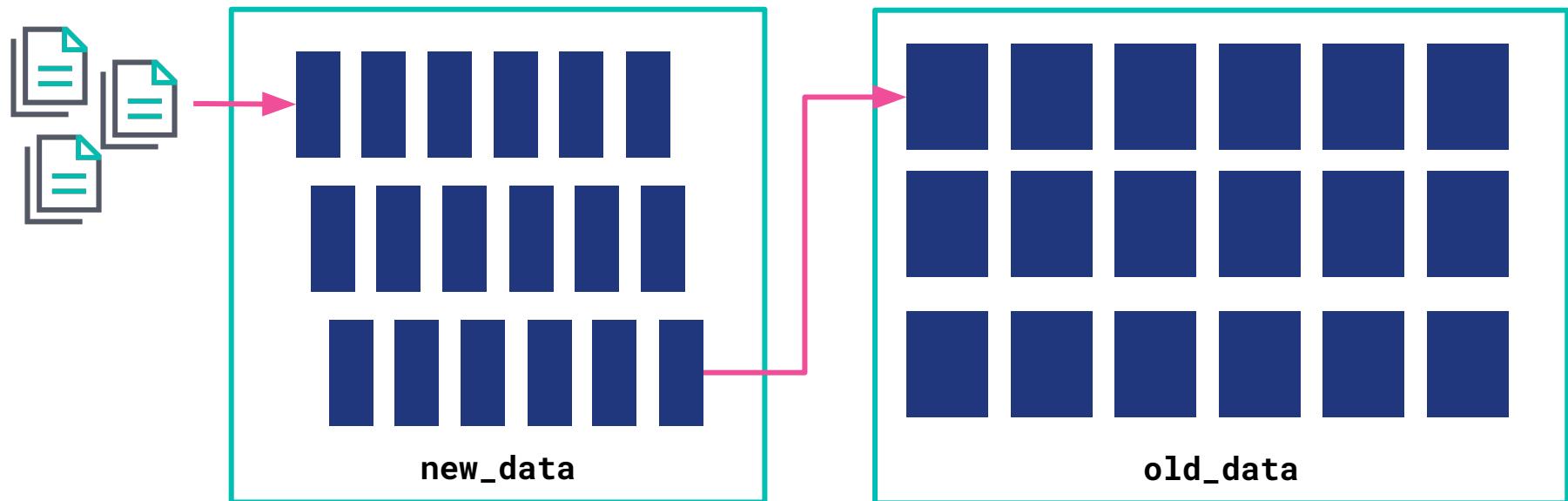
By the end of this lesson, you will be able to:

- Store time-series data across multiple indices with data streams
- Correctly name your data streams based on default conventions
- Configure an index template to use data streams



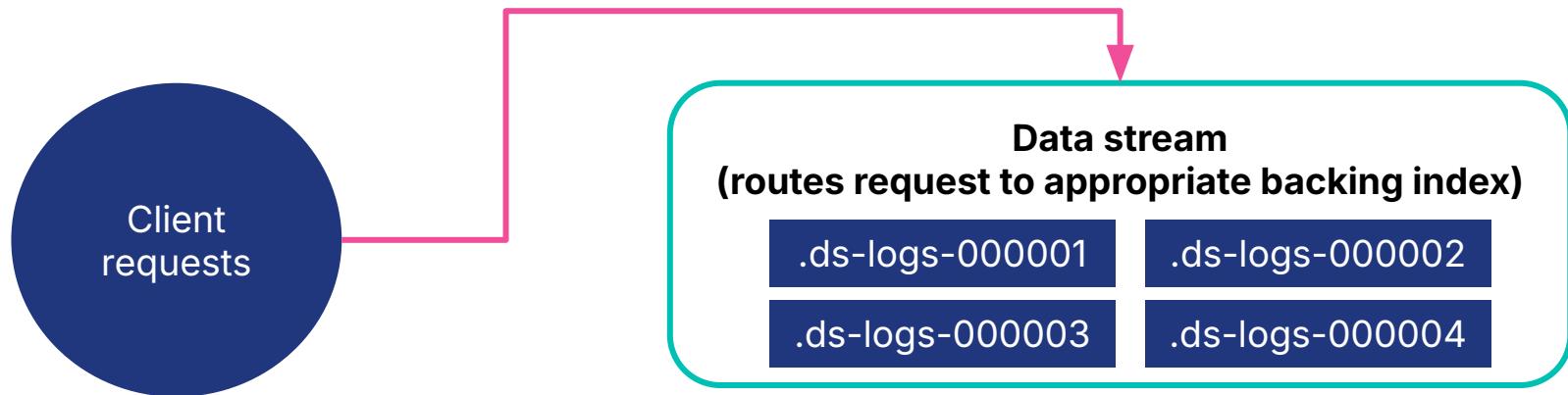
Time Series Data Management

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



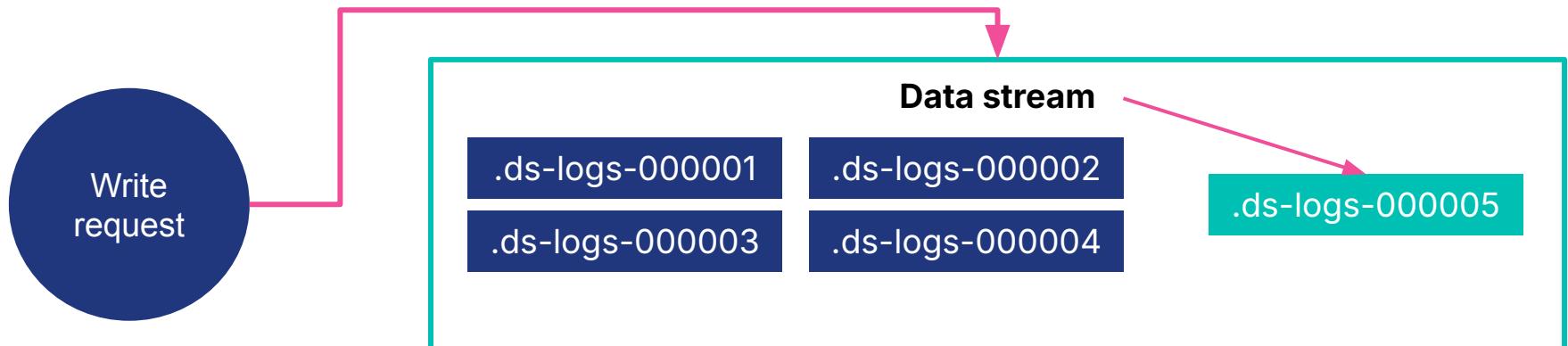
Data Streams

- 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
 - which becomes the stream's new **write index**



Choosing the right Data Stream

- Use the **index.mode** setting to control how your time series data will be ingested.
 - Optimize the storage of your documents

index.mode	Use case	_source	Storage saving
standard	For default settings	Persisted	-
time_series	For storing metrics	Synthetic*	Up to 70%
logsdb	For storing of logs	Synthetic*	~ 2.5 times

* Elasticsearch reconstructs source content on the fly upon retrieval

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



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 a `@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": ["logs-myapp-default"],
  "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

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



```
{
  "_index": ".ds-logs-myapp-default-2024.10.22-000001",
  "_id": "XZPRtZIBS7arFsx0_FAp",
  ...
}
```

Rollover a Data Stream

- The rollover API creates a new index for a data stream
 - Every new document will be indexed into the new index
 - You cannot add new documents to other backing indices

```
POST logs-myapp-default/_rollover
```



```
{  
  ...  
  "old_index": ".ds-logs-myapp-default-2024.10.22-000001",  
  "new_index": ".ds-logs-myapp-default-2024.10.22-000002",  
  ...  
}
```

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/logs-myapp-new*
```

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/logs-myapp-new
```

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

```
POST /_reindex
{
  "source": {
    "index": "logs-myapp-default"
  },
  "dest": {
    "index": "logs-myapp-new",
    "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

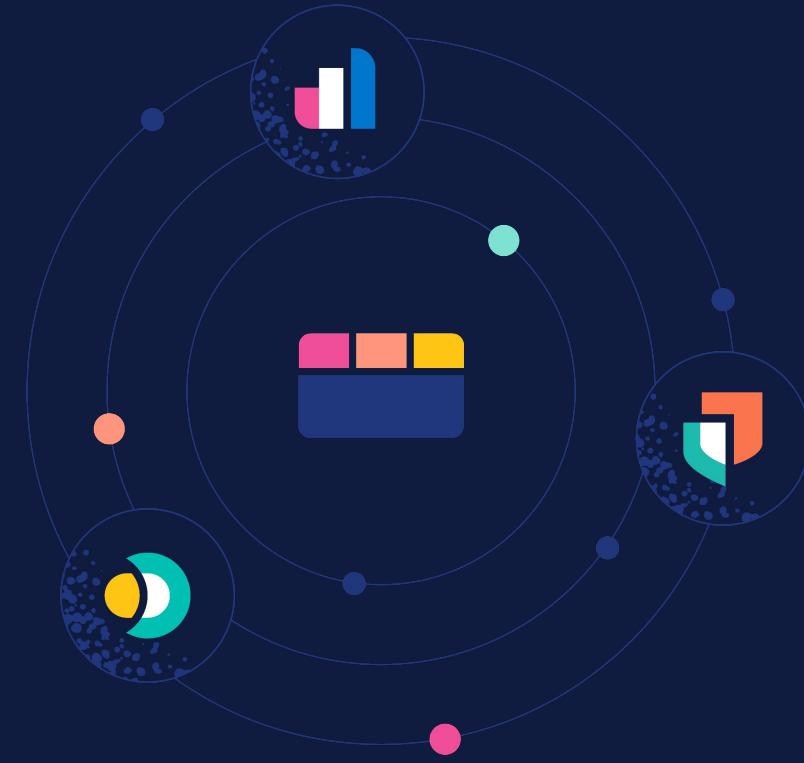


1. **True or False:** When `_bulk` writing to a data stream, you should use the `create` action.
2. How do you configure an index template to use data streams?
3. Where should you perform changes to a data stream's configuration?

Lab 7.2

Data Streams

Create a new data stream and set up a new integration





7.1 Data Management Concepts

7.2 Data Streams

7.3 Index Lifecycle Management

7.4 Searchable Snapshots



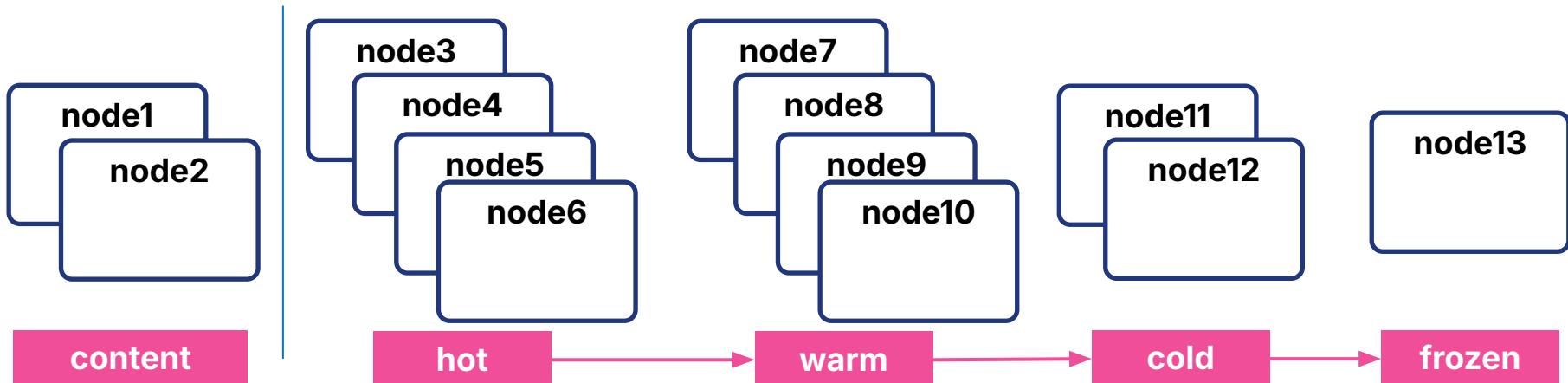
By the end of this lesson, you will be able to:

- Configure nodes for specific tiers and monitor data as it moves through these tiers
- Design ILM policies that automate the management of index lifecycles based on data age and usage patterns
- Monitor the progress of ILM actions and troubleshoot common issues in policy application

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
- Move indices to colder tiers as the data gets older
 - define an **index lifecycle management** policy to manage this

elasticsearch.yml

```
node.roles: [ "data_hot", "data_content" ]
```

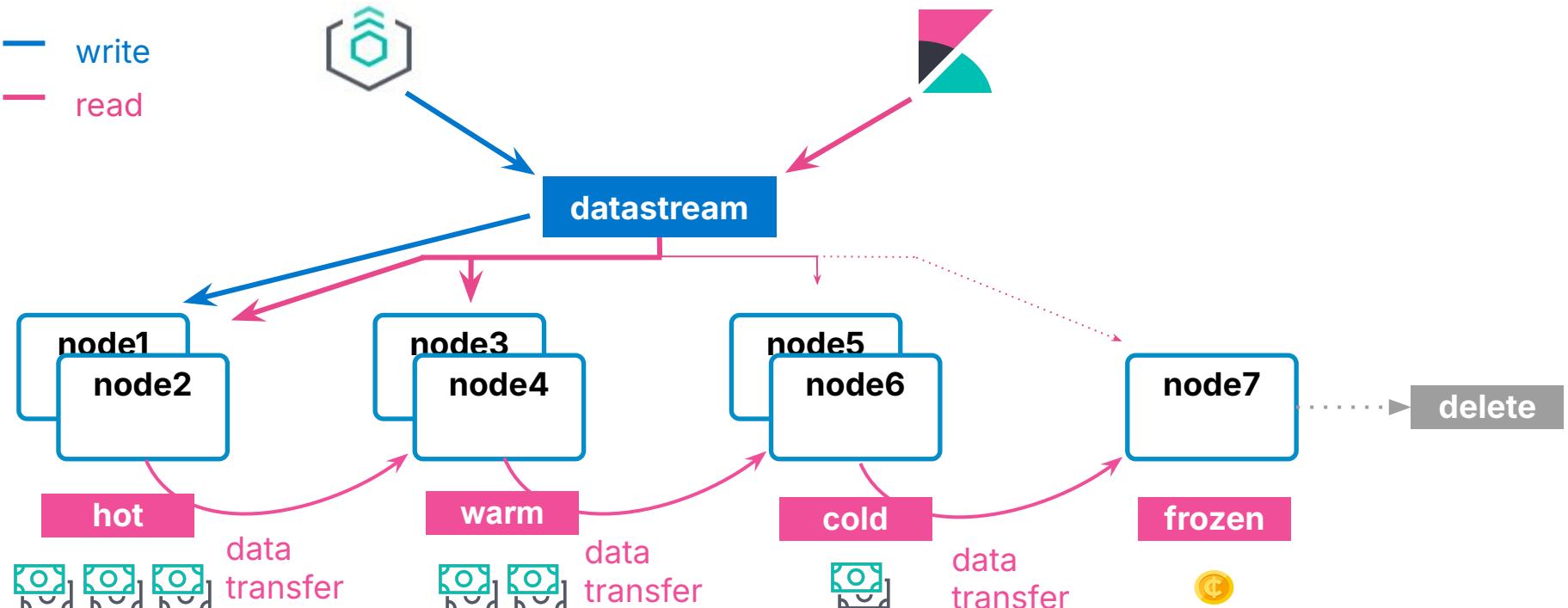
Configuring an Index to Prefer a Data Tier

- Set the **data tier preference** of an index using the **routing.allocation.include._tier_preference** property
 - **data_content** is the default for all indices
 - **data_hot** is the default for all data streams
 - 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"
          }
        }
      }
    },
  }
}
```

Rollover to a new index after 14 days

The screenshot shows the 'Hot phase' configuration with a checkmark icon. Below it, a tooltip explains: 'Store your most recent, most frequently-searched data in the hot phase'. A 'Rollover' section is shown with a note: '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.' It includes a 'Enable rollover' toggle (on), 'Maximum primary shard size' input (50), 'Maximum docs in the primary' input (empty), and a 'Maximum age' input (14) which is highlighted with a pink border. A note below says: 'Note: How long it takes to reach the rollover criteria in the hot phase can vary. [Learn more](#)'.

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 Elasticsearch Settings interface for defining the warm phase. A pink box highlights the 'Warm phase' section. It includes a toggle switch labeled 'Warm phase', a field to 'Move data into phase when' (set to 0 days), and a link to 'Advanced settings'. Below this, another pink box highlights the 'Read only' section, which contains a toggle switch to 'Make index read only'.

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:

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

Snapshot repository

my_snapshot

✖

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 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"
    }
  }
}
```

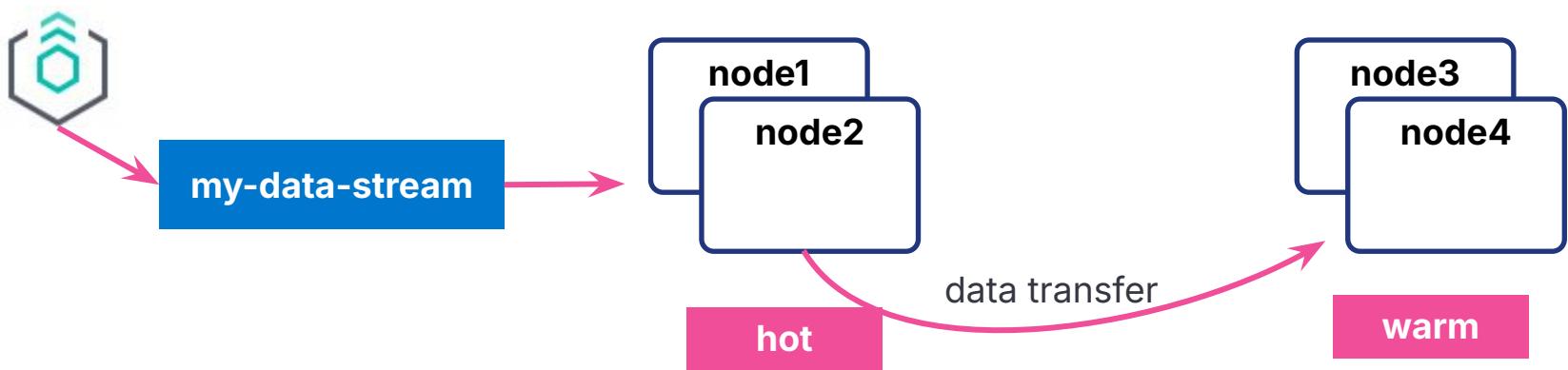
Create an Index Template

- 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 healthy, 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 five phases: hot, warm, cold, frozen 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. Suppose that your index move directly to the warm phase after the rollover. If you want this 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?

Lab 7.3

Index Lifecycle Management

Create a
hot/warm/cold ILM
policy and apply it to a
data stream





7.1 Data Management Concepts

7.2 Data Streams

7.3 Index Lifecycle Management

7.4 Searchable Snapshots

By the end of this lesson, you will be able to:

- Take snapshots of indices and restore them when needed
- Automate snapshots through scheduling policies for regular backups
- Use searchable snapshots with the cold and frozen tiers, to reduce storage costs while maintaining query access to older data



Cost Effective Storage

- As your data streams and time series data grow, 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 called **searchable snapshot**
 - but we need to discuss what a snapshot is first...

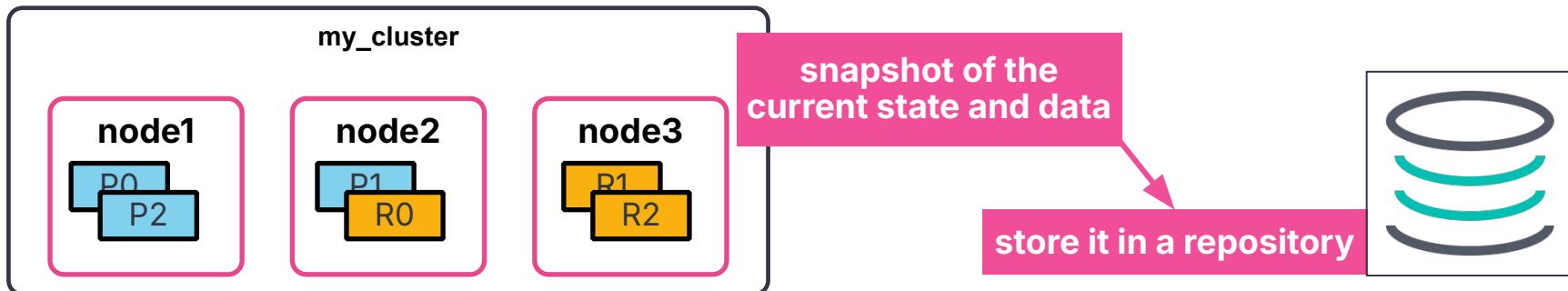
Snapshots

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



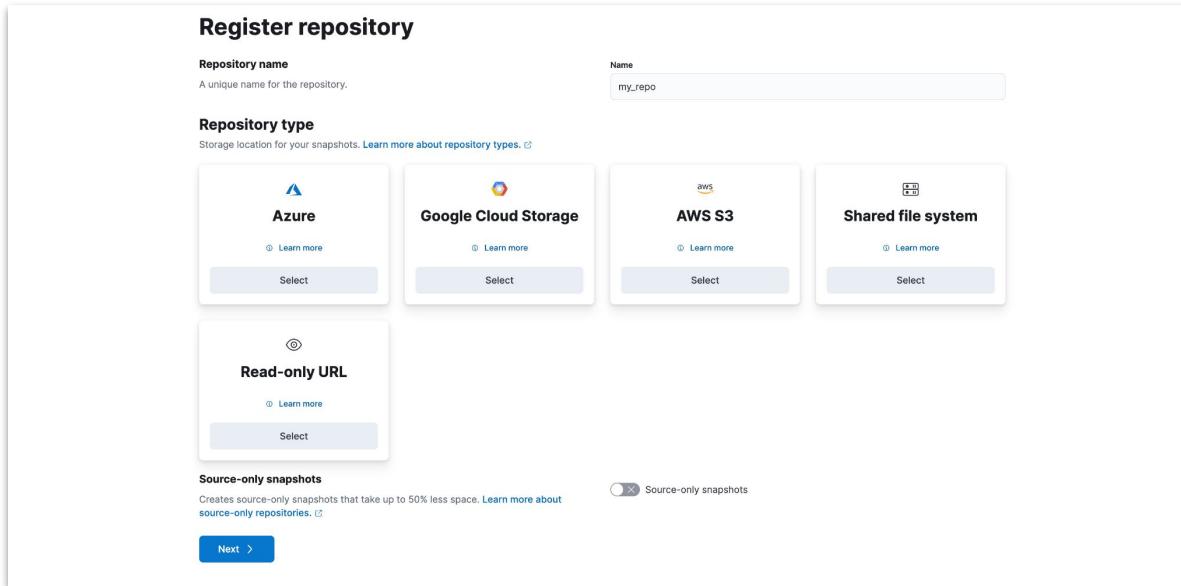
Types of Repositories

- 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
AWS S3	for AWS S3 repositories
Azure	for Microsoft Azure Blob storage
GCS	for Google Cloud Storage
repository-hdfs plugin	store snapshots in Hadoop
Source-only repository	take minimal snapshots

Setting Up a Repository

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



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
}
```

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

1

Logistics

2

Snapshot settings

3

Snapshot retention

4

Review

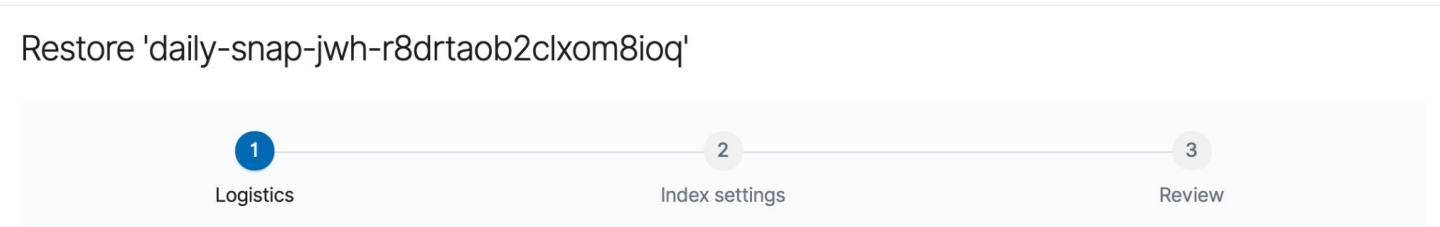
Restoring from a Snapshot

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

```
POST _snapshot/my_repo/my_logs_snapshot_1/_restore
```

- Can also restore using Kibana:

Restore 'daily-snap-jwh-r8drtlob2clxom8ioq'



The screenshot shows the first step of a three-step restore process in Kibana. The title is 'Restore "daily-snap-jwh-r8drtlob2clxom8ioq"'. Step 1, 'Logistics', is highlighted with a blue circle containing the number 1. Step 2, 'Index settings', and Step 3, 'Review', are shown with grey circles containing the numbers 2 and 3 respectively. Below the steps, there's a section titled 'Restore details' with a note: '⚠ This snapshot contains data streams'. At the bottom right, there's a link to 'Snapshot and Restore docs'.

1

Logistics

2

Index settings

3

Review

⚠ This snapshot contains data streams

Snapshot and Restore docs

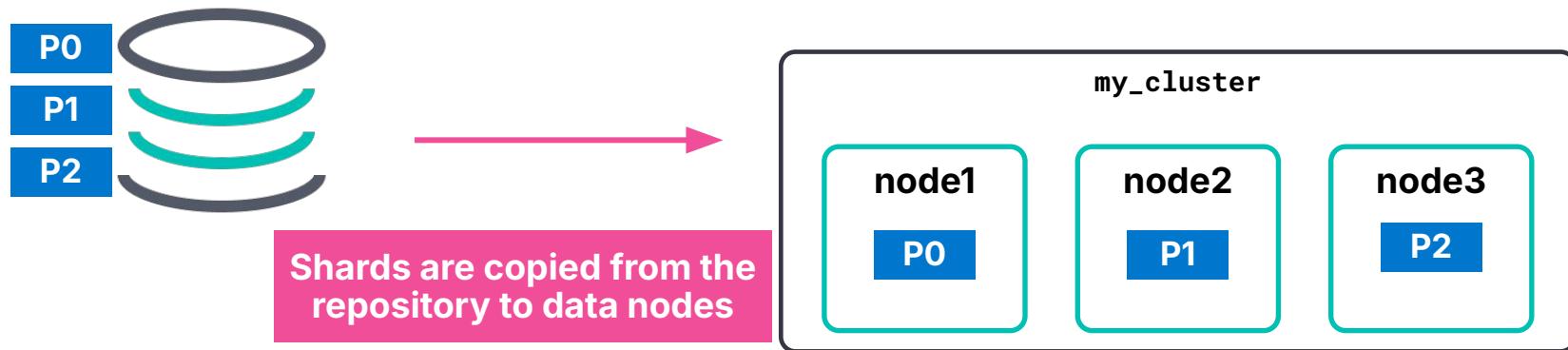
Searchable Snapshots

Searchable Snapshots

- There is an action available called **searchable snapshot**
- Benefits include:
 - search old 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**:

The screenshot shows the 'Cold phase' configuration screen. At the top, there is a toggle switch labeled 'Cold phase' with a blue circle indicating it is selected. To its right are fields for 'Move data into phase when:' (set to 7 days), a dropdown for 'old' (with an info icon), and a trash bin icon. Below this, a descriptive text states: '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.' Under the heading 'Searchable snapshot', there is a description: '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.' A 'Learn more' link is provided. Below this, another toggle switch is shown with the label 'Convert to fully-mounted index'. At the bottom left, a 'Advanced settings' link is visible. On the right side, there is a 'Delete data after this phase' button with a trash bin icon. A pink rectangular box highlights the 'Snapshot repository' section, which contains a dropdown menu set to 'my_snapshot' and a note: 'Each phase uses the same snapshot repository.'

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 hot and 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

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.

Lab 7.4

Searchable Snapshots

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
- Source-only repository:
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/snapshots-source-only-repository.html>

Elasticsearch Engineer: Agenda

- Module 1: Getting Started
- Module 2: Data Modeling
- Module 3: Search
- Module 4: Aggregations
- Module 5: Data Processing
- Module 6: Distributed Datastore
- Module 7: Data Management
- **Module 8: Cluster Management**

Cluster Management

Module 8

Module 8

Cluster Management

In this module, you will learn to manage Cross-Cluster operations for distributed Elasticsearch clusters. You'll also explore how to monitor cluster performance and troubleshoot issues



8.1 Multi-Cluster Operations

8.2 Troubleshooting

By the end of this lesson, you will be able to:

- Connect to remote clusters
- Replicate documents to remote clusters
- Perform search across multiple clusters

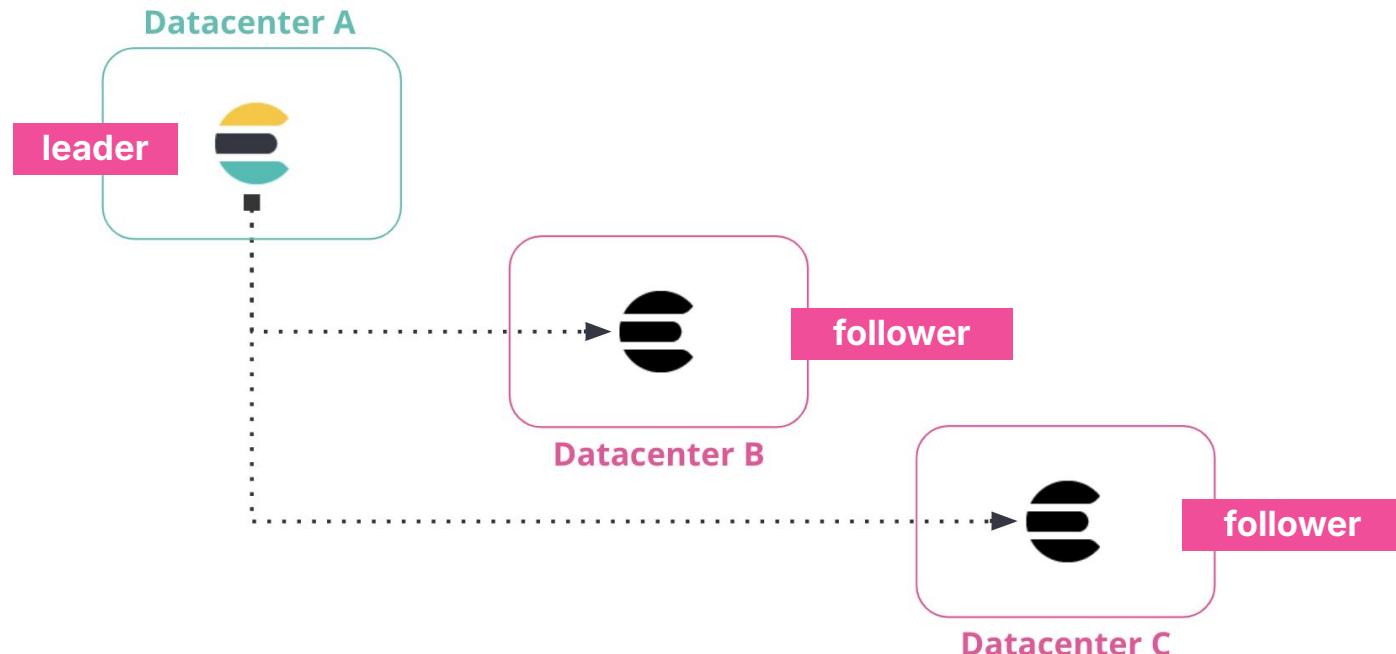
Cross-Cluster Replication

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...

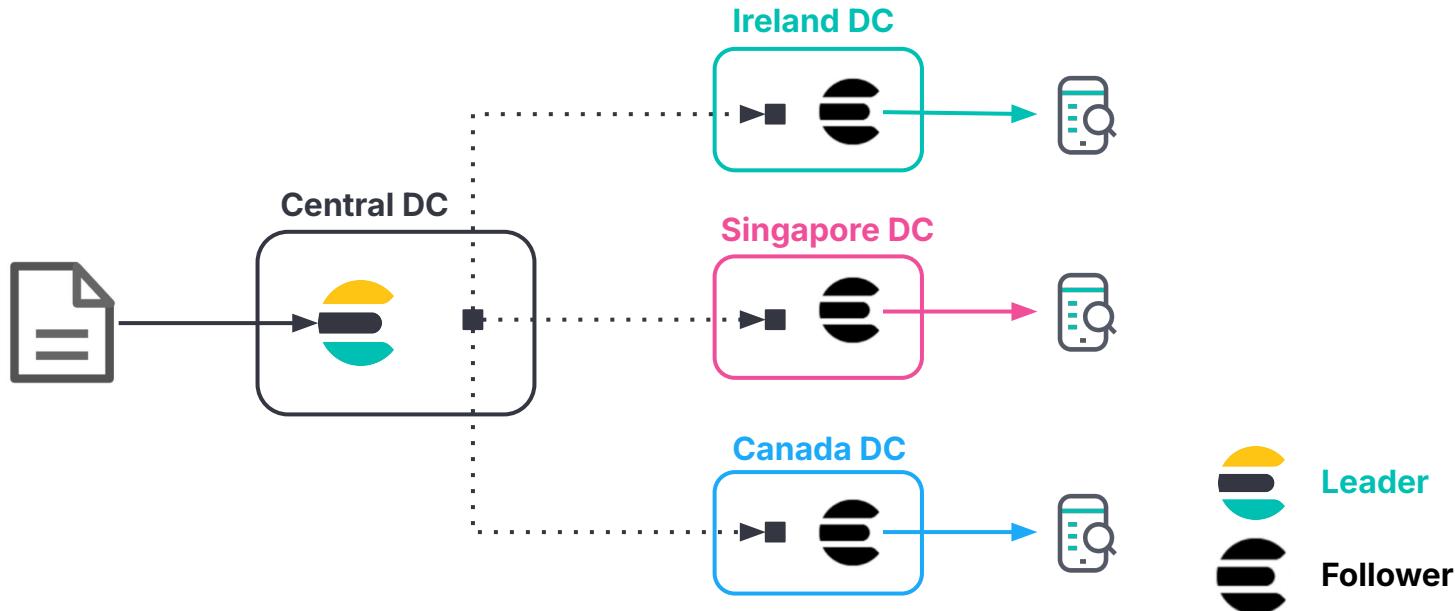
Disaster Recovery and High Availability

- Replicate data from one data center to one or more other data centers:



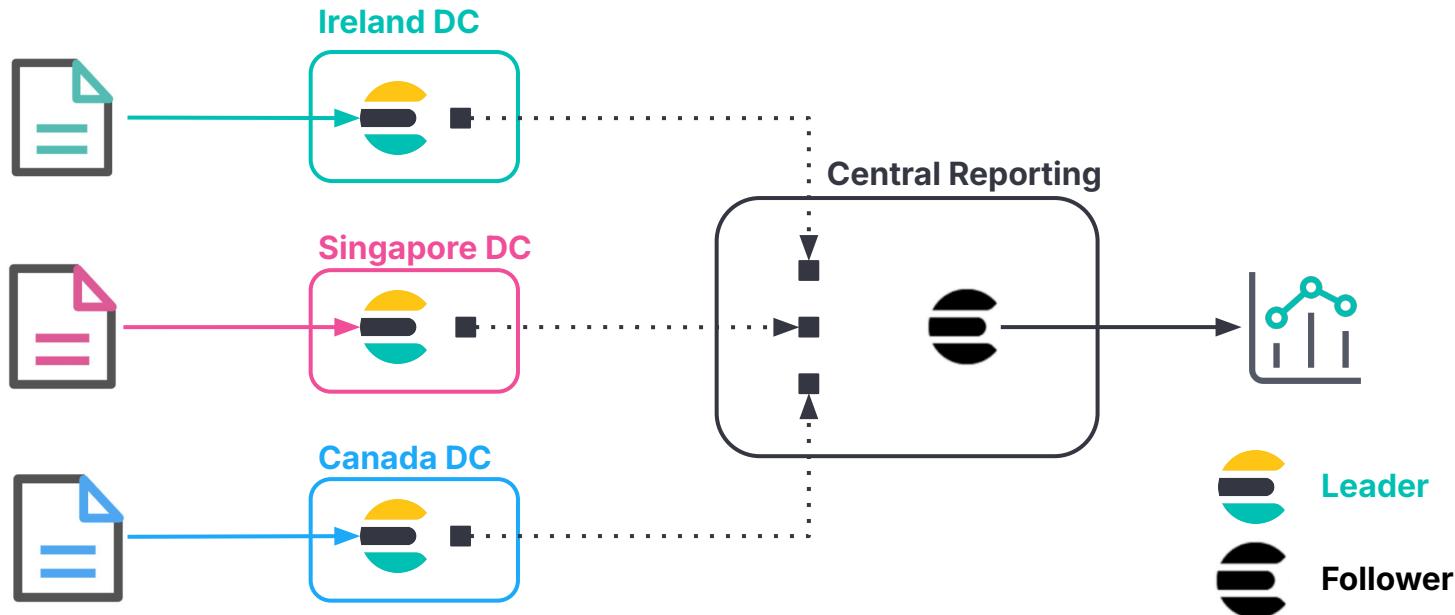
Data Locality

- 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

The screenshot shows the Elasticsearch CCR UI interface. It consists of three main sections: 'Add follower index', 'Remote cluster', and 'Leader index'.

Add follower index: This section contains fields for 'Remote cluster' (set to 'cluster2') and 'Follower index docs' (set to 'copy_of_the_leader_index'). A note states: 'A unique name for your index.'

Remote cluster: Shows the selected cluster 'cluster2' and a link to '+ Add remote cluster'.

Leader index: Shows the index 'index_to_be_replicated' and a note: 'The index on the remote cluster to replicate to the follower index.' Below it, a note says: 'Note: The leader index must already exist.'

Follower index: Shows the index 'copy_of_the_leader_index' and a note: 'Spaces and the characters \ / ? , " < > | * are not allowed.'

request: A pink box highlights the API request being sent to the '_CCR/follow' endpoint:

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

Note: The note 'Spaces and the characters \ / ? , " < > | * are not allowed.' applies to the 'Follower index' field.

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"
}
```

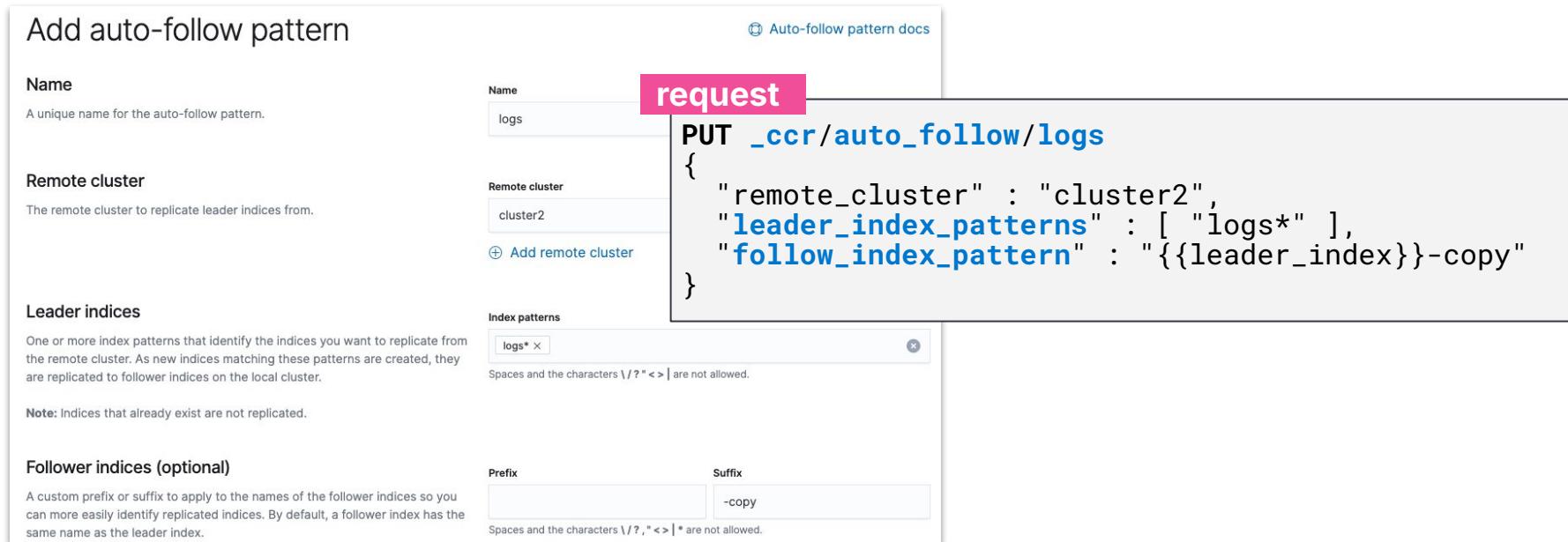
Index patterns

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

Prefix

Suffix

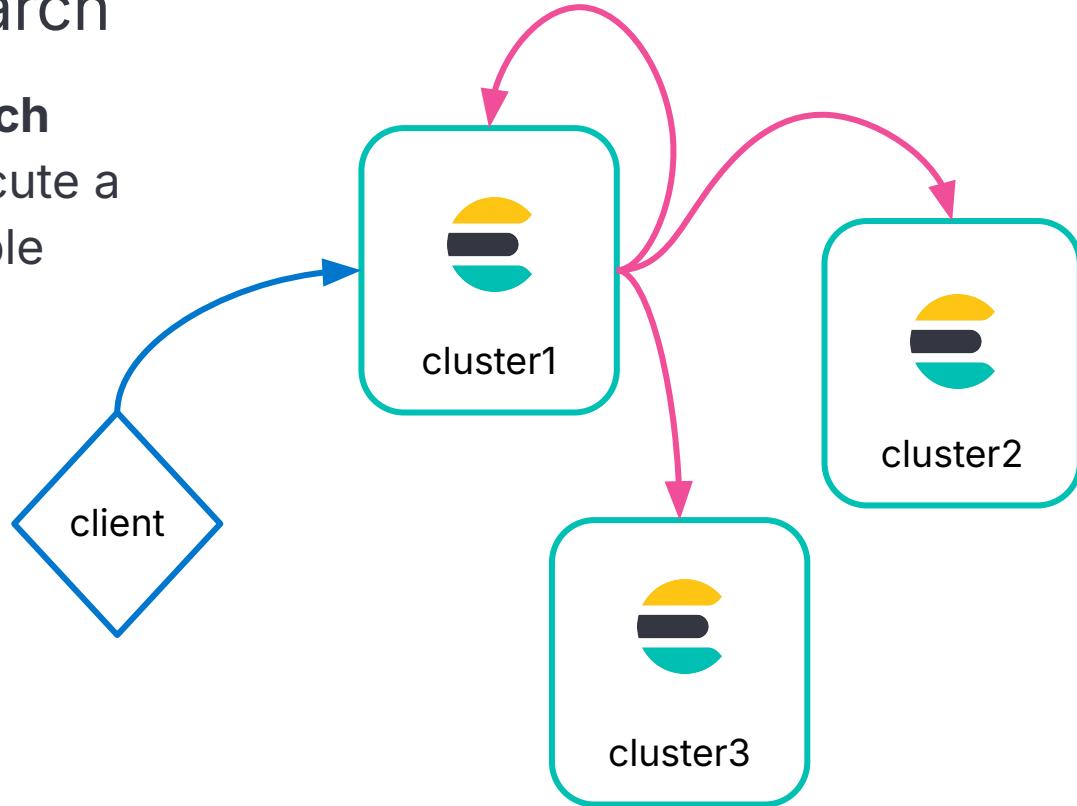
Spaces and the characters \ / ? , < > | * are not allowed.



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 eu-west-1: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

```
GET blogs,eu-west-1:blogs,us-*:blogs/_search
{
  "query": {
    "match": {
      "title": "network"
    }
  }
}
```

Search Response

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

```
"hits": [  
  {  
    "_index": "eu-west-1: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.

Lab 8.1

Multi Cluster Operations

Setup cross-cluster
replication and search



8.1 Multi-Cluster Operations

8.2 Troubleshooting

By the end of this lesson, you will be able to:

- Assess the health of your cluster, index, and shards
- Find the root cause of issues that might impact your cluster
- Monitor your clusters using the Elastic stack

The Health API

Health API

- The Health API provide an overview of the health of a cluster
 - Diagnose issues across different components like shards, ingestion, and search
 - Health reports include specific recommendations to fix the issues

```
GET /_health_report
```

Use `?verbose=false`
for automatic pulling

Health Status Levels

- Each indicators has a health status
- The cluster's status is controlled by the worst indicator status

Green

The indicator is healthy

Unknown

Could not be determined

Yellow

Degraded states

Red

Outage or feature unavailable

Health Indicators Breakdown

master_is_stable	Checks if the master is changing too frequently
shards_availability	Check if the cluster has all shards available
disk	Reports health issues caused by lack of disk space
ilm	Reports health issues related to ILM
repository_integrity	Checks if any snapshot repositories becomes corrupted, unknown or invalid
slm	Reports health issues related to SLM
shards_capacity	Checks if the cluster has enough room to add new shards

Health Indicator Symptoms and Impacts

```
{"status": "red",
"indicators": { ...
  "shards_availability": {
    "status": "red",
    "symptom": "This cluster has 1 unavailable primary shard, 1
unavailable replica shard.",
    "details": {},
    "impacts": [{ ...
      "description": "Cannot add data to 1 index [blogs_elser].
Searches might return incomplete results.",
      "impact_areas": ["ingest", "search"]
    }],
    ...
  }
}
```

Diagnosis comes
next ...

Health Indicator Diagnosis

```
"diagnosis": [ {  
    "cause": "Elasticsearch isn't allowed to allocate some shards from  
            these indices to any of the nodes in the cluster.",  
    "action": "Diagnose the issue by calling the allocation explain API  
            for an index [GET _cluster/allocation/explain]..."  
    "help_url": "https://ela.st/diagnose-shards",  
    "affected_resources": {"indices": ["blogs_elser"]}  
}]
```

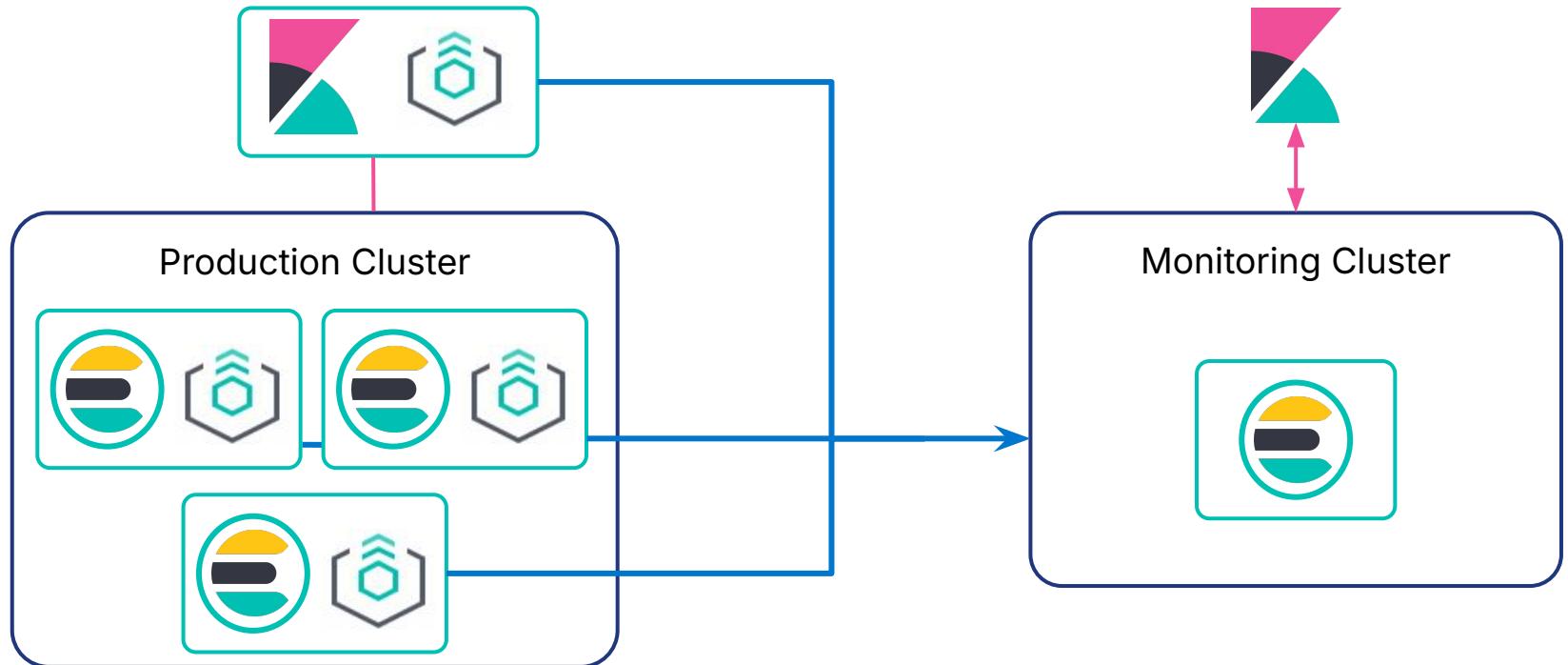
Monitoring Your Clusters

Monitoring the Elastic Stack

- To monitor the Elastic Stack, you can use...**the Elastic Stack!**
 - **Metricbeat** to collect metrics
 - **Filebeat** to collect logs
 - Or use **Elastic Agent**
- 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 Elastic Agent

- Use **Elastic agent** to collect both metrics and logs



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 **Health API** reports the health status of an Elasticsearch cluster.
- The overall cluster health depends on multiple indicators
- 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

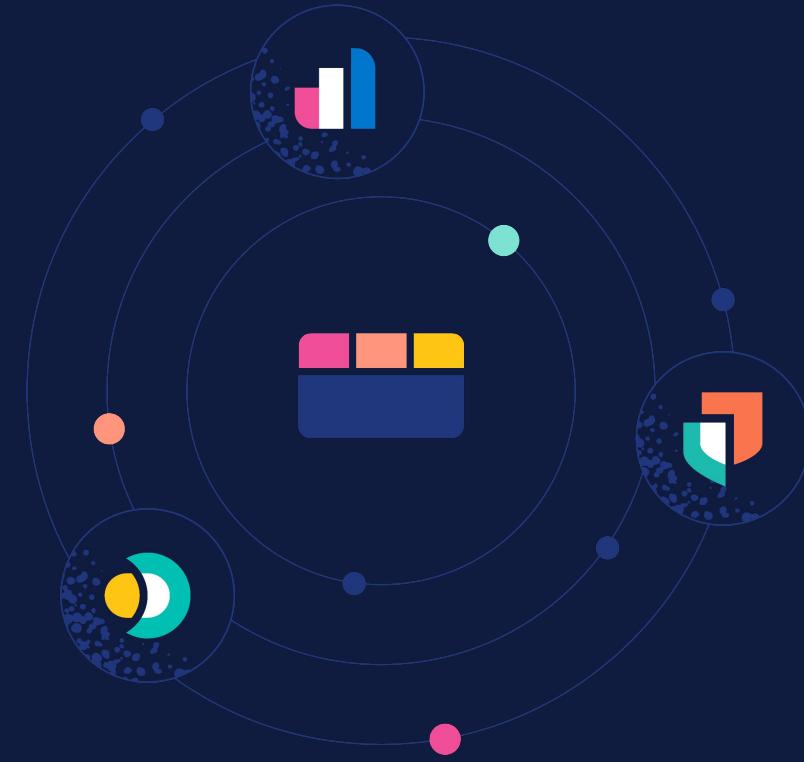
Quiz

1. **True or False:** You can use Elastic Agent to monitor all components of the Elastic Stack.
2. Which status indicates a critical issue in the Elasticsearch Health API?
3. **True or False:** The Health API can automatically fix issues it detects in your cluster.

Lab 8.2

Troubleshooting

Monitor your cluster to look for issues



More Resources

- Health API blog
 - <https://www.elastic.co/blog/cluster-health-diagnosis-elasticsearch-health-api>
- Monitoring Elasticsearch
 - <https://www.elastic.co/guide/en/elasticsearch/reference/current/monitor-elasticsearch-cluster.html>

Conclusion

Cloud
Search

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 Support Hub

- <https://support.elastic.co/home>
 - access a wealth of technical resources
 - maximize your experience using Elastic solutions

The screenshot shows the Elastic Support Hub homepage. At the top, there's a navigation bar with tabs for "Recommended For You" (Beta), "Newest Articles", and "Recently Viewed". On the right side of the header is a blue button labeled "Open case".

The main content area is divided into two sections. The left section, titled "Recommended For You", lists four knowledge articles:

- "How to fix a broken Kibana Monitoring UI for an ESS cluster after upgrading to v8" - Last updated 7 months ago.
- "Where is kibana audit log stored when it is configured in Elastic Cloud" - Last updated 5 months ago.
- "Enabling Metrics collection on Elasticsearch Service may create a lot of data" - Last updated 4 months ago.
- "How to schedule hit Cluster Reroute API with retry_failed to retry failed shards" - Last updated 4 months ago.

At the bottom of this section is a "Search all articles >" link.

The right section contains two promotional boxes:

- A box titled "Start your Elastic Cloud trial" featuring a "Create deployment" button. It describes how users can create an Elastic Cloud deployment including Elasticsearch, Kibana, and other features.
- A box titled "Talk to Elastic Support" with a "Get set up >" button. It encourages users to get in touch with an expert by setting up their account.

Elastic Certification

Validate Skills

Apply practical knowledge with performance-based testing

Boost Productivity

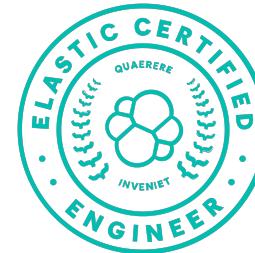
Overcome obstacles with confidence and ease as you move from dev to production

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



Elasticsearch
Engineer

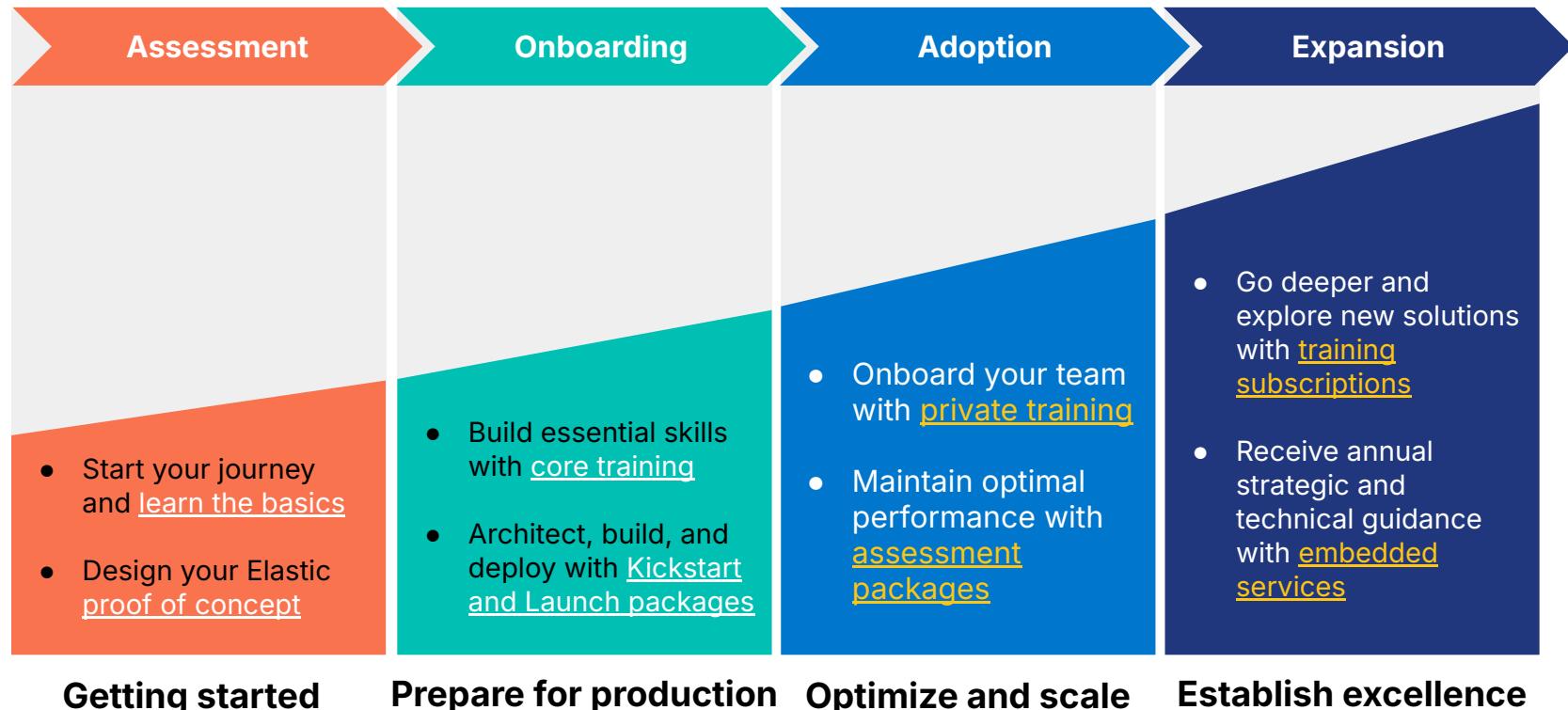


Data Analyst



Observability
Engineer

Elastic Enablement Journey



Thank you.

Please complete
the survey



Quiz Solutions

Quiz Solutions

1.1 Stack Introduction

1. Elasticsearch is the core of the Elastic Stack and is used to store, search, and analyze data
2. A node is a single instance of Elasticsearch
3. False. You can use Kibana's Discover to query and filter your data

1.2 Index Operations

1. True. Static data is characterized by slow growth, some updates to existing records, and generally being used for backing search-based applications
2. True. The Bulk API allows a large number of documents to be ingested, updated, and even deleted in a single POST
3. The document will be updated and the version will be incremented by 1

1.3 Search Data

1. KQL, Lucene, and ES|QL can be used in Kibana query bar
2. False. You do not need to have an aggregation included within a search request
3. True. An ES|QL query is composed of a series of commands that are processed sequentially; where the output of one operation becomes the input for the next

2.1 Strings

1. True. At query time, text strings are analyzed
2. False. Keyword strings are exact and the case is not changed
3. True. The standard analyzer is the default analyzer which is used if no other analyzer is specified

2.2 Mapping

1. True. There is one mapping per index
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 Types and Parameters

1. Use the format parameter to change a field's date format
2. False. The copy_to parameter does not add a new field to the _source that is returned with the hits
3. False. If you set enabled to false, you cannot query or aggregate on it

3.1 Full Text Queries

1. True. The type parameter specifies what type of query is executed for multi_match
2. False. You can change the sort order with the sort parameter
3. 10

3.2 Term-level Queries

1. False. The Query DSL enables you to do any kind of query
2. True. While the query is being executed, `is_partial` is set to true
3. True. Range queries allow you to create exact, relative, or open-ended ranges

3.3 Combining Queries

1. False. All clauses are optional when using a bool query
2. False. A filter clause has no effect on a document's score
3. Filter context. You are looking for an exact match, which is a yes or no type query

4.1 Metrics and Bucket Aggregations

1. Buckets. The date_histogram groups documents by their date in fixed date range intervals.
2. False. Some buckets are sorted alphabetically
(You can sort buckets by a metric value in a sub-aggregation)
3. Terms aggregation. The terms aggregation creates buckets for each unique value of the field.

4.2 Combining aggs

1. A pipeline aggregation, where the output from the histogram is used for the input to the moving average.
2. You need to combine a date_histogram (buckets by year) and terms (buckets by category)
3. True. When metrics are embedded under a bucket as a sub-aggregation, they calculate their value based on the documents residing inside the bucket.

4.3 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

5.1 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

5.2 Enriching Data

1. True
2. To create and enrich index (1) create an enrich policy then (2) execute it
3. False. Once created, you can't update or change an enrich policy. Create and execute a new enrich policy.

5.3 Runtime Fields

1. Use parameters in your script
2. True
3. False, you can directly define a runtime mapping at query time

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
3. Higher than 200

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.

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. RED
3. False (It provides recommendations, but administrators need to take action.)

Elasticsearch Engineer

© 2015-2025 Elasticsearch BV. All rights reserved. Decompiling, copying, publishing and/or distribution without written consent of Elasticsearch BV is strictly prohibited.