

ELASTICSEARCH BASICS

Understanding elasticsearch architecture:

Elasticsearch is a product in the elastic stack which is a group of four open source products from elastic:

1. Elasticserach
2. Logstash
3. Kibana
4. Beats

What is Elasticsearch?

[Elasticsearch](#) is a distributed search and analytics engine, scalable data store, and vector database built on Apache Lucene. It's optimized for speed and relevance on production-scale workloads. Use Elasticsearch to search, index, store, and analyze data of all shapes and sizes in near real time.

Elasticsearch is the heart of the [Elastic Stack](#). Combined with [Kibana](#), it powers the following Elastic solutions:

- [Observability](#)

- [Search](#)
- [Security](#)

Elasticsearch has a lot of features. Explore the full list on the [product webpage](#).

What is the Elastic Stack?

Elasticsearch is the core component of the Elastic Stack, a suite of products for collecting, storing, searching, and visualizing data. [Learn more about the Elastic Stack](#).

Use cases

[edit](#)

Elasticsearch is used for a wide and growing range of use cases. Here are a few examples:

Observability

- **Logs, metrics, and traces:** Collect, store, and analyze logs, metrics, and traces from applications, systems, and services.
- **Application performance monitoring (APM):** Monitor and analyze the performance of business-critical software applications.
- **Real user monitoring (RUM):** Monitor, quantify, and analyze user interactions with web applications.

- **OpenTelemetry:** Reuse your existing instrumentation to send telemetry data to the Elastic Stack using the OpenTelemetry standard.

Search

- **Full-text search:** Build a fast, relevant full-text search solution using inverted indexes, tokenization, and text analysis.
- **Vector database:** Store and search vectorized data, and create vector embeddings with built-in and third-party natural language processing (NLP) models.
- **Semantic search:** Understand the intent and contextual meaning behind search queries using tools like synonyms, dense vector embeddings, and learned sparse query-document expansion.
- **Hybrid search:** Combine full-text search with vector search using state-of-the-art ranking algorithms.
- **Build search experiences:** Add hybrid search capabilities to apps or websites, or build enterprise search engines over your organization's internal data sources.
- **Retrieval augmented generation (RAG):** Use Elasticsearch as a retrieval engine to supplement

generative AI models with more relevant, up-to-date, or proprietary data for a range of use cases.

- **Geospatial search:** Search for locations and calculate spatial relationships using geospatial queries.

Security

- **Security information and event management (SIEM):** Collect, store, and analyze security data from applications, systems, and services.
- **Endpoint security:** Monitor and analyze endpoint security data.
- **Threat hunting:** Search and analyze data to detect and respond to security threats.

This is just a sample of search, observability, and security use cases enabled by Elasticsearch.

Documents also contain reserved fields that constitute the document metadata such as:

- `_index` – the index where the document resides
- `_type` – the type that the document represents
- `_id` – the unique identifier for the document

```
{  
  "_id": 3,  
  "_type": ["your index type"],  
  "_index": ["your index name"],  
  "_source": {  
    "age": 28,  
    "name": ["daniel"],  
    "year": 1989,  
  }  
}
```

3. Mapping

As far as mapping goes, bear in mind that since Elasticsearch 7.0, *index type* has been deprecated. The following is still relevant to legacy versions of Elasticsearch. *Data type* is still active.

Like a schema in the world of relational databases, mapping defines the different *types* that reside within an index (although for 6.0 until its deprecation in 7.0, only one type can exist within an index). It defines the *fields* for documents of a specific type — the data type (such as keyword and integer) and how the fields should be indexed and stored in Elasticsearch. This includes meta-fields, such as `_index`, `_type`, and `_id`.

```
# Example
```

```
# Example
```

```
curl -XPUT localhost:9200/example -d '{
```

```
  "mappings": {
```

```
    "mytype": {
```

```
      "properties": {
```

```
        "name": {
```

```
          "type": "string"
```

```
        },
```

```
        "age": {
```

```
          "type": "long"
```

```
        }
```

```
      }
```

```
    }
```

```
  }
```

```
}'
```

What is logstash ?

logstash is an **open-source** server-side data processing **pipeline** that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "**stash**" like Elasticsearch. It is highly versatile and can handle various types of data, including **logs, metrics, web applications, and databases**.

Key Features of Logstash

- **Versatile Data Ingestion:** Logstash can ingest data from a wide range of sources, including log files, [databases](#), message queues, and cloud services.
- **Real-time Processing:** It processes data in real-time, allowing you to perform complex transformations and enrichments on the fly.
- **Flexible Data Parsing:** With numerous **plugins**, Logstash can parse, transform, and enrich your data in countless ways.
- **Integration with Elasticsearch and Kibana:** Seamlessly integrates with [Elasticsearch](#) for storage and Kibana for visualization, providing a complete data analysis solution.

What are the requirements for Logstash?

For a basic Logstash node, your setup with **2 cores and 8 GB RAM** is a decent starting point, but the system requirements can vary depending on the volume of data you're processing and the complexity of your pipelines.

What is the structure of Logstash?

The Logstash event processing pipeline has three stages: **inputs** → **filters** → **outputs**. Inputs generate events, filters modify them, and outputs ship them elsewhere.

Kibana

[edit](#)

Kibana enables you to give shape to your data and navigate the Elastic Stack. With Kibana, you can:

- **Search, observe, and protect your data.** From discovering documents to analyzing logs to finding security vulnerabilities, Kibana is your portal for accessing these capabilities and more.

- **Analyze your data.** Search for hidden insights, visualize what you've found in charts, gauges, maps, graphs, and more, and combine them in a dashboard.
- **Manage, monitor, and secure the Elastic Stack.** Manage your data, monitor the health of your Elastic Stack cluster, and control which users have access to which features.

kibana is for administrators, analysts, and business users. As an admin, your role is to manage the Elastic Stack, from creating your deployment to getting Elasticsearch data into Kibana, and then managing the data. As an analyst, you're looking to discover insights in the data, visualize your data on dashboards, and share your findings. As a business user, you want to view existing dashboards and drill down into details.

Kibana works with all types of data. Your data can be structured or unstructured text, numerical data, time series data, geospatial data, logs, metrics, security events, and more. No matter your data, Kibana can help you uncover patterns and relationships and visualize the results.

Kibana concepts

[edit](#)

Learn the shared concepts for analyzing and visualizing your data

As an analyst, you will use a combination of Kibana apps to analyze and visualize your data. Kibana contains both general-purpose apps and apps for the [Enterprise Search](#), [Elastic Observability](#), and [Elastic Security](#) solutions. These apps share a common set of concepts.

Three things to know about Elasticsearch

[edit](#)

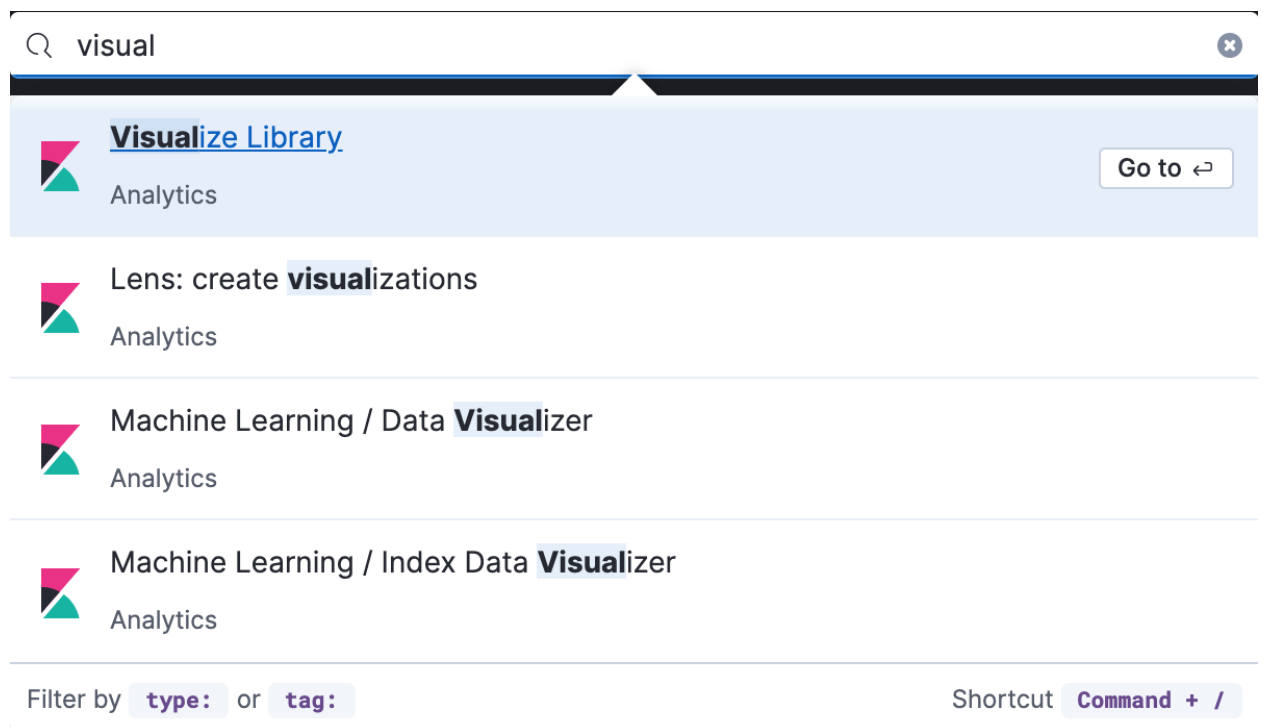
You don't need to know everything about Elasticsearch to use Kibana, but the most important concepts follow:

- **Elasticsearch makes JSON documents searchable and aggregatable.** The documents are stored in an [index](#) or [data stream](#), which represent one type of data.
- **Searchable means that you can filter the documents for conditions.** For example, you can filter for data "within the last 7 days" or data that "contains the word Kibana". Kibana provides many ways for you to construct filters, which are also called queries or search terms.

-

-

Aggregatable means that you can extract summaries from matching documents. The simplest aggregation is **count**, and it is frequently used in combination with the **date histogram**, to see count over time. The **terms** aggregation shows the most frequent values.



-

Run Elasticsearch locally

[edit](#)

DO NOT USE THESE INSTRUCTIONS FOR PRODUCTION DEPLOYMENTS

The instructions on this page are for **local development only**. Do not use this configuration for production deployments, because it is not secure. Refer to [deployment options](#) for a list of production deployment options.

Quickly set up Elasticsearch and Kibana in Docker for local development or testing, using the [start-local script](#).

This setup comes with a one-month trial of the Elastic **Platinum** license. After the trial period, the license reverts to **Free and open - Basic**. Refer to [Elastic subscriptions](#) for more information.

Prerequisites

[edit](#)

- If you don't have Docker installed, [download and install Docker Desktop](#) for your operating system.
- If you're using Microsoft Windows, then install [Windows Subsystem for Linux \(WSL\)](#).

Run start-local

[edit](#)

To set up Elasticsearch and Kibana locally, run the start-local script:

```
curl -fsSL https://elastic.co/start-local | sh
```

This script creates an elastic-start-local folder containing configuration files and starts both Elasticsearch and Kibana using Docker.

After running the script, you can access Elastic services at the following endpoints:

- **Elasticsearch:** <http://localhost:9200>
- **Kibana:** <http://localhost:5601>

The script generates a random password for the elastic user, and an API key, stored in the .env file.

This setup is for local testing only. HTTPS is disabled, and Basic authentication is used for Elasticsearch. For security, Elasticsearch and Kibana are accessible only through localhost.

Installing Elasticsearch

[edit](#)

Hosted Elasticsearch Service

[edit](#)

Elastic Cloud offers all of the features of Elasticsearch, Kibana, and Elastic's Observability, Enterprise Search, and Elastic Security solutions as a hosted service available on AWS, GCP, and Azure.

To set up Elasticsearch in Elastic Cloud, sign up for a [free Elastic Cloud trial](#).

Self-managed Elasticsearch options

[edit](#)

If you want to install and manage Elasticsearch yourself, you can:

- Run Elasticsearch using a [Linux, MacOS, or Windows install package](#).
- Run Elasticsearch in a [Docker container](#).
- Set up and manage Elasticsearch, Kibana, Elastic Agent, and the rest of the Elastic Stack on Kubernetes with [Elastic Cloud on Kubernetes](#).

Configuring Elasticsearch

[edit](#)

Elasticsearch ships with good defaults and requires very little configuration. Most settings can be changed on a running cluster using the [Cluster update settings](#) API.

The configuration files should contain settings which are node-specific (such as node.name and paths), or settings which a node requires in order to be able to join a cluster, such as cluster.name and network.host.

Config files location

[edit](#)

Elasticsearch has three configuration files:

- elasticsearch.yml for configuring Elasticsearch
- jvm.options for configuring Elasticsearch JVM settings
- log4j2.properties for configuring Elasticsearch logging

These files are located in the config directory, whose default location depends on whether or not the installation is from an archive distribution (tar.gz or zip) or a package distribution (Debian or RPM packages).

For the archive distributions, the config directory location defaults to `$ES_HOME/config`. The location of the config directory can be changed via the `ES_PATH_CONF` environment variable as follows:

```
ES_PATH_CONF=/path/to/my/config ./bin/elasticsearch
```

Alternatively, you can export the `ES_PATH_CONF` environment variable via the command line or via your shell profile.

For the package distributions, the config directory location defaults to `/etc/elasticsearch`. The location of the config directory can also be changed via the `ES_PATH_CONF` environment variable, but note that setting this in your shell is not sufficient. Instead, this variable is sourced from `/etc/default/elasticsearch` (for the Debian package) and `/etc/sysconfig/elasticsearch` (for the RPM package). You will need to edit the `ES_PATH_CONF=/etc/elasticsearch` entry in one of these files accordingly to change the config directory location.

Config file format

[edit](#)

The configuration format is [YAML](#). Here is an example of changing the path of the data and logs directories:

path:

data: /var/lib/elasticsearch

logs: /var/log/elasticsearch

The method for starting Elasticsearch varies depending on how you installed it.

Archive packages (.tar.gz)

[edit](#)

If you installed Elasticsearch with a .tar.gz package, you can start Elasticsearch from the command line.

Run Elasticsearch from the command line

[edit](#)

Run the following command to start Elasticsearch from the command line:

```
./bin/elasticsearch
```

When starting Elasticsearch for the first time, security features are enabled and configured by default. The following security configuration occurs automatically:

- Authentication and authorization are enabled, and a password is generated for the elastic built-in superuser.

- Certificates and keys for TLS are generated for the transport and HTTP layer, and TLS is enabled and configured with these keys and certificates.
- An enrollment token is generated for Kibana, which is valid for 30 minutes.

The password for the elastic user and the enrollment token for Kibana are output to your terminal.

We recommend storing the elastic password as an environment variable in your shell. Example:

```
export ELASTIC_PASSWORD="your_password"
```

If you have password-protected the Elasticsearch keystore, you will be prompted to enter the keystore's password. See [Secure settings](#) for more details.

By default Elasticsearch prints its logs to the console (stdout) and to the <cluster name>.log file within the [logs directory](#). Elasticsearch logs some information while it is starting, but after it has finished initializing it will continue to run in the foreground and won't log anything further until something happens that is worth recording. While Elasticsearch is running you can interact with it through its HTTP interface which is on port 9200 by default.

To stop Elasticsearch, press Ctrl-C.

All scripts packaged with Elasticsearch require a version of Bash that supports arrays and assume that Bash is available at `/bin/bash`. As such, Bash should be available at this path either directly or via a symbolic link.

Enroll nodes in an existing cluster

[edit](#)

When Elasticsearch starts for the first time, the security auto-configuration process binds the HTTP layer to `0.0.0.0`, but only binds the transport layer to `localhost`. This intended behavior ensures that you can start a single-node cluster with security enabled by default without any additional configuration.

Before enrolling a new node, additional actions such as binding to an address other than `localhost` or satisfying bootstrap checks are typically necessary in production clusters. During that time, an auto-generated enrollment token could expire, which is why enrollment tokens aren't generated automatically.

Additionally, only nodes on the same host can join the cluster without additional configuration. If you want nodes from another host to join your cluster, you need to set `transport.host` to a [supported value](#) (such as uncommenting the suggested value of `0.0.0.0`), or an IP address that's bound to an interface where other hosts can reach it. Refer to [transport settings](#) for more information.

To enroll new nodes in your cluster, create an enrollment token with the `elasticsearch-create-enrollment-token` tool on any existing node in your cluster. You can then start a new node with the `--enrollment-token` parameter so that it joins an existing cluster.

1. In a separate terminal from where Elasticsearch is running, navigate to the directory where you installed Elasticsearch and run the [elasticsearch-create-enrollment-token](#) tool to generate an enrollment token for your new nodes.

```
bin/elasticsearch-create-enrollment-token -s node
```

Copy the enrollment token, which you'll use to enroll new nodes with your Elasticsearch cluster.

2. From the installation directory of your new node, start Elasticsearch and pass the enrollment token with the `--enrollment-token` parameter.

```
bin/elasticsearch --enrollment-token <enrollment-token>
```

Elasticsearch automatically generates certificates and keys in the following directory:

```
config/certs
```

3. Repeat the previous step for any new nodes that you want to enroll.

Run as a daemon

[edit](#)

To run Elasticsearch as a daemon, specify `-d` on the command line, and record the process ID in a file using the `-p` option:

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

If you have password-protected the Elasticsearch keystore, you will be prompted to enter the keystore's password. See [Secure settings](#) for more details.

Log messages can be found in the `$ES_HOME/logs/` directory.

To shut down Elasticsearch, kill the process ID recorded in the pid file:

```
pkill -F pid
```

Stopping Elasticsearch

[edit](#)

An orderly shutdown of Elasticsearch ensures that Elasticsearch has a chance to cleanup and close outstanding resources. For example, a node that is shutdown in an orderly fashion will remove itself from

the cluster, sync translogs to disk, and perform other related cleanup activities. You can help ensure an orderly shutdown by properly stopping Elasticsearch.

If you're running Elasticsearch as a service, you can stop Elasticsearch via the service management functionality provided by your installation.

If you're running Elasticsearch directly, you can stop Elasticsearch by sending control-C if you're running Elasticsearch in the console, or by sending SIGTERM to the Elasticsearch process on a POSIX system. You can obtain the PID to send the signal to via various tools (e.g., ps or jps):

```
$ jps | grep Elasticsearch
```

```
14542 Elasticsearch
```

what are nodes?

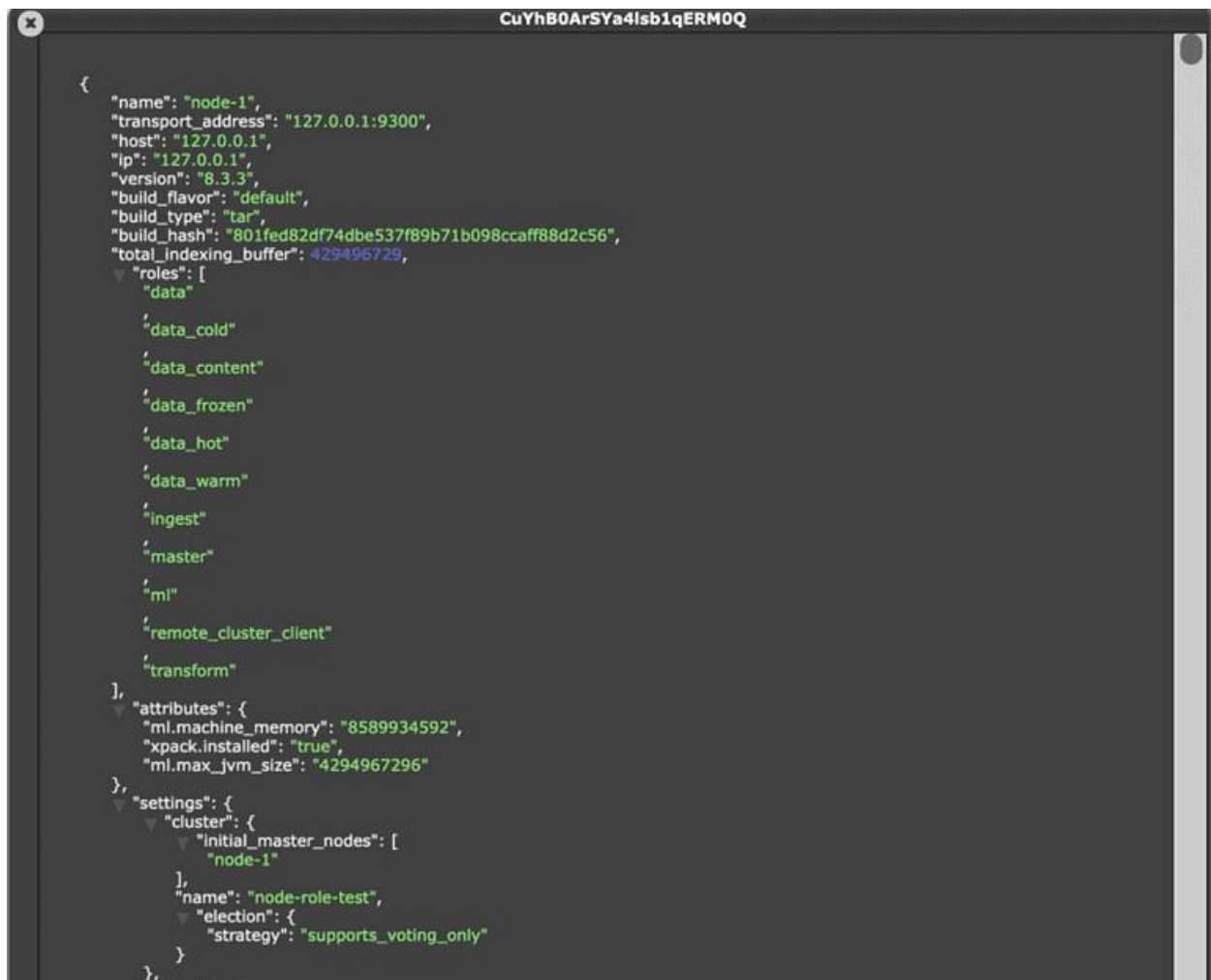
Every Elasticsearch instance we run is called a node, and multiple [nodes](#) comprise a [cluster](#). Each node in a cluster is aware of all other nodes and forwards the requests accordingly. Clusters can consist of only a single node, though this isn't recommended for production. In this article, we will review the different types of node roles and how to configure these roles in Elasticsearch to enable efficient full text search.

What are node roles?

The node role defines the purpose of the node and its responsibilities. We can configure multiple roles for each

node based on the cluster configuration. If we don't explicitly specify the node's role, Elasticsearch automatically configures all roles to that node. This does not differ among the different [versions](#) of Elasticsearch.

The cluster details of such nodes will appear as:

A screenshot of a code editor window with a dark theme. The window title is "CuYhB0ArSYa4Isb1qERM0Q". The code is a JSON object representing an Elasticsearch node configuration. It includes fields for name, transport address, host, ip, version, build flavor, build type, build hash, total indexing buffer, and a list of roles. The roles list includes "data", "data_cold", "data_content", "data_frozen", "data_hot", "data_warm", "ingest", "master", "ml", "remote_cluster_client", and "transform". There are also "attributes" and "settings" sections. The "settings" section includes "cluster" details like "initial_master_nodes" and "name", and "election" strategy.

```
{
  "name": "node-1",
  "transport_address": "127.0.0.1:9300",
  "host": "127.0.0.1",
  "ip": "127.0.0.1",
  "version": "8.3.3",
  "build_flavor": "default",
  "build_type": "tar",
  "build_hash": "801fed82df74dbe537f89b71b098ccaff88d2c56",
  "total_indexing_buffer": 429496729,
  "roles": [
    "data",
    "data_cold",
    "data_content",
    "data_frozen",
    "data_hot",
    "data_warm",
    "ingest",
    "master",
    "ml",
    "remote_cluster_client",
    "transform"
  ],
  "attributes": {
    "ml.machine_memory": "8589934592",
    "xpack.installed": "true",
    "ml.max_jvm_size": "4294967296"
  },
  "settings": {
    "cluster": {
      "initial_master_nodes": [
        "node-1"
      ],
      "name": "node-role-test",
      "election": {
        "strategy": "supports_voting_only"
      }
    }
  }
}
```

Types of node roles

1. Master
2. Data
3. Coordinating
4. Ingest
5. Machine learning

Master node

The node to which we assign a master role is called a “master” node. The master node manages all cluster operations like creating/deleting an index and it keeps track of all available nodes in the cluster.

While creating [shards](#), the master node decides the node upon which each shard should be allocated.

This node will not handle any user requests.

The master node is **responsible for lightweight cluster-wide actions such as creating or deleting an index, tracking which nodes are part of the cluster, and deciding which shards to allocate to which nodes**

What is a data node?

A data node is **an appliance that you can add to your event and flow processors to increase storage capacity and improve search performance.**

INGEST NODE:

What are Ingest Nodes?

Ingest Nodes are a new type of Elasticsearch node you can use to perform common data transformation and enrichment.

By using ingest pipeline you can pre-process your documents, during the Indexing process. They gave pretty much all the Logstash functionality, you can use different types of filters and processors, to match and modify data.

There are also three Ingest plugins available:

1. **Ingest Attachment:** converts binary documents like Powerpoints, PDF to text and metadata
2. **Ingest Geoip:** looks up the geographic locations of IP addresses in an internal database

3. **Ingest user agent**: parses and extracts information from the user agent strings used by browsers and other applications when using HTTP

Enable to run ingest pipeline

Manipulate document

ML NODE :

Machine learning nodes are used to handle Machine learning API requests. The machine learning flag (**xpack.ml.enabled**) is enabled by default and it uses a CPU that supports SSE4.2 instructions.

To configure a machine learning node, add the following configuration to the “elasticsearch.yml” file:

```
node.roles: [“ml”]
```

In the event that you are using the `remote_cluster_client` functionality for machine learning (see below), then you should also configure this role for the ML nodes.

It is also recommended not to use a dedicated master or coordinating node as a machine learning node.

node.roles: ["ml", "remote_cluster_client"]

CO-ORDINATION NODE:

Coordinating node

Coordinating-only nodes act as load-balancers. This type of node routes requests to data nodes and handles [bulk](#) indexing by distributing the requests.

These types of nodes are used in larger clusters. By getting the cluster state from all the nodes, the coordinating-only node will [route](#) requests accordingly.

In small clusters, it is usually not necessary to use a coordinating node, since the same role will be handled by data nodes, and the greater complexity is not justified on a small cluster.

To make a node “coordinating only” node, add the following configuration to the “elasticsearch.yml” file:

```
node.roles: []
```

Index management in Kibana

[edit](#)

Kibana’s **Index Management** features are an easy, convenient way to manage your cluster’s indices, [data streams](#), [index templates](#), and [enrich policies](#). Practicing good index management ensures your data is stored correctly and in the most cost-effective way possible.

To use these features, go to **Stack Management > Index Management**.

Required permissions

[edit](#)

If you use Elasticsearch security features, the following [security privileges](#) are required:

- The monitor cluster privilege to access Kibana's **Index Management** features.
- The view_index_metadata and manage index privileges to view a data stream or index's data.
- The manage_index_templates cluster privilege to manage index templates.

To add these privileges, go to **Stack Management > Security > Roles** or use the [Create or update roles API](#).

Manage indices

[edit](#)

Investigate your indices and perform operations from the **Indices** view.

Index Management

[Index Management docs](#)

[Indices](#)
[Data Streams](#)
[Index Templates](#)
[Component Templates](#)

Update your Elasticsearch indices individually or in bulk. [Learn more.](#)

☒ Include rollout indices
 ☐ Include hidden indices

Lifecycle status
 Lifecycle phase
 Reload indices

<input type="checkbox"/>	Name	Health	Status	Primaries	Replicas	Docs count	Storage size	Data stream
<input type="checkbox"/>	my-frozen-index Frozen	yellow	open	1	1	0	230b	
<input type="checkbox"/>	sensor_rollup Rollup	yellow	open	1	1	2028	246kb	
<input type="checkbox"/>	kibana_sample_data_ecommerce	green	open	1	0	4675	3.8mb	
<input type="checkbox"/>	kibana_sample_data_logs	green	open	1	0	14074	7.6mb	
<input type="checkbox"/>	kibana_sample_data_flights	green	open	1	0	13059	5.4mb	
<input type="checkbox"/>	my-index-000001	yellow	open	1	1	0	208b	

Rows per page: 10
 < 1 >

- To show details and perform operations such as close, forcemerge, and flush, click the index name.

To perform operations on multiple indices, select their checkboxes and then open the **Manage** menu. For more information on managing indices, refer to [Index APIs](#).

- To filter the list of indices, use the search bar or click a badge. Badges indicate if an index is a [follower index](#), a [rollup index](#), or [frozen](#).
- To drill down into the index [mappings](#), [settings](#), and statistics, click an index name. From this view, you can navigate to **Discover** to further explore the documents in the index.

The screenshot shows the Kibana Index Management interface. The breadcrumb navigation at the top reads: Stack Management > Index Management > Indices > Index details > Overview. The left sidebar contains a 'Management' section with links for Ingest, Data, Alerts and Insights, and Security. The main content area is for the index 'kibana_sample_data_flights'. It features a 'Back to all indices' link, a 'Manage index' dropdown, and a 'Discover index' button. Below this is a summary table with the following data:

Status	Documents	Documents deleted	Primaries	Replicas	Aliases
open	13014	0	1	1	none

Below the table is a section titled 'Add data to this index' with a link to 'Learn more'. At the bottom, there is a 'cURL' section with a code block containing the following command:

```
curl -X POST https://your_deployment_url/_bulk?pretty \
-H "Authorization: ApiKey your_api_key" \
-H "Content-Type: application/json" \
-d'
{ "index" : { "_index" : "kibana_sample_data_flights" } }
{ "name": "foo", "title": "bar" }
```

Manage data streams

[edit](#)

Investigate your data streams and address lifecycle management needs in the **Data Streams** view.

The value in the **Indices** column indicates the number of backing indices. Click this number to drill down into details.

A value in the data retention column indicates that the data stream is managed by a [data stream lifecycle policy](#). This value is the time period for which your data is guaranteed to be stored. Data older than this period can be deleted by Elasticsearch at a later time.

Index Management

[Index Management docs](#)

[Indices](#) [Data Streams](#) [Index Templates](#) [Component Templates](#) [Enrich Policies](#)

Data streams store time-series data across multiple indices and can be created from index templates. [Learn more.](#)

<input type="checkbox"/> Name ↑	Health ↕	Indices ↕	Index mode ↕	Data retention ↕	Actions
<input type="checkbox"/> b-data-stream	● green	1	Standard	90 days	
<input type="checkbox"/> logs-enterprise_search.api-default	● green	1	Standard	Disabled	
<input type="checkbox"/> logs-enterprise_search.audit-default	● green	1	Standard	Disabled	
<input type="checkbox"/> metrics-fleet_server.agent_status-default Managed	● green	1	Time series	Disabled	
<input type="checkbox"/> metrics-fleet_server.agent_versions-default Managed	● green	1	Time series	Disabled	

Rows per page: 10 < 1 >

- To view more information about a data stream, such as its generation or its current index lifecycle policy, click the stream's name. From this view, you can navigate to **Discover** to further explore data within the data stream.
- [preview] This functionality is in technical preview and may be changed or removed in a future release. Elastic will work to fix any issues, but features in technical preview are not subject to the support SLA

of official GA features. To edit the data retention value, open the **Manage** menu, and then click **Edit data retention**. This action is only available if your data stream is not managed by an ILM policy.

Manage index templates

[edit](#)

Create, edit, clone, and delete your index templates in the **Index Templates** view. Changes made to an index template do not affect existing indices.

Index Management

[Index Management docs](#)

Indices **Data Streams** **Index Templates** **Component Templates** **Enrich Policies**

Use composable index templates to automatically apply settings, mappings, and aliases to indices. [Learn more.](#)

View 1

Reload

Search...

Create template

<input type="checkbox"/>	Name ↑	Index patterns ↓	Component t... ↓	Data stream	Content	Actions
<input type="checkbox"/>	ilm-history-7 Managed	ilm-history-7*	0	✓	M S A	...
<input type="checkbox"/>	logs Managed	logs-*-*	4	✓	None	...
<input type="checkbox"/>	metrics Managed	metrics-*-*	4	✓	None	...
<input type="checkbox"/>	synthetics Managed	synthetics-*-*	3	✓	None	...

Rows per page: 20

Try it: Create an index template

[edit](#)

In this tutorial, you'll create an index template and use it to configure two new indices.

Step 1. Add a name and index pattern

1. In the **Index Templates** view, open the **Create template wizard**.

Create template

1

2

3

4

5

6

Logistics

Component templates

Index settings

Mappings

Aliases

Review template

Logistics

[Index Templates docs](#)

Name

A unique identifier for this template.

Name

Index patterns

The index patterns to apply to the template.

Index patterns

Type and then hit "ENTER"

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

Data stream

The template creates data streams instead of indices. [Learn more.](#)

☐ ☒ Create data stream

Priority

Only the highest priority template will be applied.

Priority (optional)

Version

A number that identifies the template to external management systems.

Version (optional)

_meta field

Use the _meta field to store any metadata you want.

☐ ☒ Add metadata

Next >

2. In the **Name** field, enter my-index-template.
3. Set **Index pattern** to my-index-* so the template matches any index with that index pattern.
4. Leave **Data Stream**, **Priority**, **Version**, and **_meta field** blank or as-is.

Step 2. Add settings, mappings, and aliases

1. Add [component templates](#) to your index template.

Component templates are pre-configured sets of mappings, index settings, and aliases you can reuse across multiple index templates. Badges indicate whether a component template contains mappings (**M**), index settings (**S**), aliases (**A**), or a combination of the three.

Component templates are optional. For this tutorial, do not add any component templates.

Create template



Component templates (optional)

[Component templates docs](#)

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

Add component template building blocks to this template.
Component templates are applied in the order specified.

Filter 3

.deprecation-indexing-mappings	M S A	+
.deprecation-indexing-settings	M S A	+
data-streams-mappings	M S A	+
logs-mappings	M S A	+
logs-settings	M S A	+
metrics-mappings	M S A	+
metrics-settings	M S A	+
synthetics-mappings	M S A	+

[< Back](#)[Next >](#)[Preview index template](#)

2. Define index settings. These are optional. For this tutorial, leave this section blank.

3. Define a mapping that contains an [object](#) field named geo with a child [geo_point](#) field named coordinates:

Create template

The screenshot shows the 'Create template' wizard in Elasticsearch. At the top, a progress bar has six steps: Logistics (checked), Component templates (checked), Index settings (checked), Mappings (active, step 4), Aliases (step 5), and Review template (step 6). Below the progress bar, the 'Mappings (optional)' section is active. It includes a description 'Define how to store and index documents.' and links for 'Load JSON' and 'Mapping docs'. There are four tabs: 'Mapped fields' (selected), 'Runtime fields', 'Dynamic templates', and 'Advanced options'. A search bar 'Search fields' is present. The field list shows a parent field 'geo' of type 'Object' with a child field 'coordinates' of type 'Geo-point'. At the bottom, there are 'Back' and 'Next' buttons, and a 'Preview index template' button on the right.

Mappings (optional) [Load JSON](#) [Mapping docs](#)

Define how to store and index documents.

[Mapped fields](#) [Runtime fields](#) [Dynamic templates](#) [Advanced options](#)

Define the fields for your indexed documents. [Learn more.](#)

Search fields

geo Object

coordinates Geo-point

+ Add field

< Back Next >

Preview index template

Alternatively, you can click the **Load JSON** link and define the mapping as JSON:

```
{  
  "properties": {  
    "geo": {  
      "properties": {  
        "coordinates": {  
          "type": "geo_point"  
        }  
      }  
    }  
  }  
}
```

```
    }  
  }  
}  
}  
}
```

You can create additional mapping configurations in the **Dynamic templates** and **Advanced options** tabs. For this tutorial, do not create any additional mappings.

4. Define an alias named my-index:

```
5. {  
6.  "my-index": { }  
}
```

7. On the review page, check the summary. If everything looks right, click **Create template**.

Step 3. Create new indices

You're now ready to create new indices using your index template.

1. Index the following documents to create two indices:
my-index-000001 and my-index-000002.
2. POST /my-index-000001/_doc
3. {

```
4.  "@timestamp": "2019-05-18T15:57:27.541Z",
5.  "ip": "225.44.217.191",
6.  "extension": "jpg",
7.  "response": "200",
8.  "geo": {
9.    "coordinates": {
10.      "lat": 38.53146222,
11.      "lon": -121.7864906
12.    }
13.  },
14.  "url": "https://media-for-the-
    masses.theacademyofperformingartsandscience.org/u
   ploads/charles-fullerton.jpg"
15.  }
16.
17.  POST /my-index-000002/_doc
18.  {
19.    "@timestamp": "2019-05-20T03:44:20.844Z",
20.    "ip": "198.247.165.49",
21.    "extension": "php",
22.    "response": "200",
23.    "geo": {
24.      "coordinates": {
25.        "lat": 37.13189556,
26.        "lon": -76.4929875
27.      }
28.    },
29.    "memory": 241720,
```

```
30.      "url":  
        "https://theacademyofperformingartsandscience.org/people/type:astronauts/name:laurel-b-clark/profile"  
    }
```

□ Copy as curl [Try in Elastic](#)

□ Use the [get index API](#) to view the configurations for the new indices. The indices were configured using the index template you created earlier.

GET /my-index-000001,my-index-000002

2. Copy as curl [Try in Elastic](#)

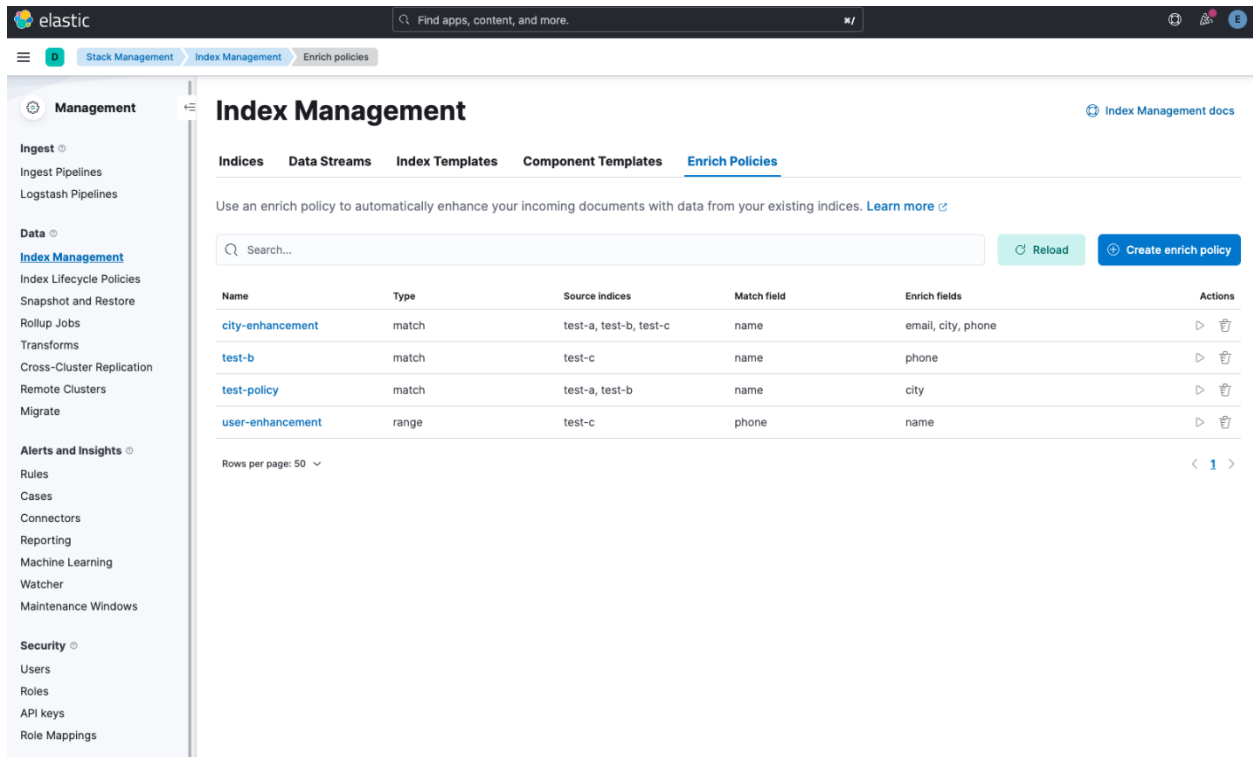
Manage enrich policies

[edit](#)

Use the **Enrich Policies** view to add data from your existing indices to incoming documents during ingest. An enrich policy contains:

- The policy type that determines how the policy matches the enrich data to incoming documents
- The source indices that store enrich data as documents
- The fields from the source indices used to match incoming documents

- The enrich fields containing enrich data from the source indices that you want to add to incoming documents
- An optional [query](#).



When creating an enrich policy, the UI walks you through the configuration setup and selecting the fields. Before you can use the policy with an enrich processor or ES|QL query, you must execute the policy.

When executed, an enrich policy uses enrich data from the policy's source indices to create a streamlined system index called the enrich index. The policy uses this index to match and enrich incoming documents.

Check out these examples:

- [Example: Enrich your data based on geolocation](#)
- [Example: Enrich your data based on exact values](#)
- [Example: Enrich your data by matching a value to a range](#)

BASIC CRUD OPERATION

What is meant by CRUD operations?

CRUD is the acronym for **CREATE, READ, UPDATE and DELETE**. These terms describe the four essential operations for creating and managing persistent data elements, mainly in relational and NoSQL databases.

What are the advantages of CRUD operations?

CRUD operations follow a standard and consistent convention that makes them **easy to understand and implement**. CRUD operations also allow you to abstract the complexity of data storage and manipulation from the user interface, and focus on the functionality and usability of the web application.

What is CRUD?

CRUD is the acronym for **CREATE**, **READ**, **UPDATE** and **DELETE**. These terms describe the four essential operations for creating and managing persistent data elements, mainly in relational and NoSQL databases.

This post will describe how CRUD operations are used for data processing. We will also show the issues that sysadmins or DevOps engineers may find when monitoring a database.

Explaining CRUD

As mentioned, CRUD operations are used in persistent storage applications, meaning these applications will keep their data even after the system powers down. These are different from operations on data stored in volatile storage, like Random Access Memory or cache files.

CRUD is extensively used in database applications. This includes Relational Database Management Systems (RDBMS) like Oracle, MySQL, and PostgreSQL. It also includes NoSQL databases like MongoDB, Apache Cassandra, and AWS DynamoDB.

Operations similar to CRUD can be performed on persistent data structures like files. For example, you can create a Microsoft Word document, update it, read it, and even delete it from the file explorer. However, files are not record-oriented (or document-oriented in the case of MongoDB or Couchbase). The CRUD terminology is specifically related to record-oriented operations instead of flat file operations.

CREATE

The **CREATE** operation adds a new record to a database. In RDBMS, a database table row is referred to as a record, while columns are called attributes or fields. The **CREATE** operation adds one or more new records with distinct field values in a table.

The same principle applies to NoSQL databases. If the NoSQL database is document-oriented, then a new document (for example, a JSON formatted document with its attributes) is added to the collection, which is the equivalent of an RDBMS table. Similarly, in NoSQL databases like DynamoDB, the **CREATE** operation adds an item (which is equivalent to a record) to a table.

READ

READ returns records (or documents or items) from a database table (or collection or bucket) based on some

search criteria. The READ operation can return all records and some or all fields.

UPDATE

UPDATE is used to modify existing records in the database. For example, this can be the change of address in a customer database or price change in a product database. Similar to READ, UPDATES can be applied across all records or only a few, based on criteria.

An UPDATE operation can modify and persist changes to a single field or to multiple fields of the record. If multiple fields are to be updated, the database system ensures they are all updated or not at all. Some big data systems don't implement UPDATE but allow only a timestamped CREATE operation, adding a new version of the row each time.

DELETE

DELETE operations allow the user to remove records from the database. A hard delete removes the record altogether, while a soft delete flags the record but leaves it in place. For example, this is important in payroll where employment records need to be maintained even after an employee has left the company.

How is CRUD performed in a database?

In RDBMS, CRUD operations are performed through Structure Query Language (SQL) commands.

- The INSERT statement is used for CREATE:

```
INSERT INTO <table></table> VALUES (field value 1,  
field value, 2...)
```

- The SELECT statement is used for READ:

```
SELECT field 1, field 2, ...FROM <table></table>  
[WHERE ]
```

- The UPDATE statement is used for UPDATE:

```
UPDATE <table></table> SET field1=value1,  
field2=value2,... [WHERE ]
```

- The DELETE statement is used for DELETE:

```
DELETE FROM <table></table> [WHERE ]
```

CRUD operations in NoSQL databases will depend on the language of the specific database platform. For example, the Cassandra CQL looks very similar to SQL. In MongoDB, on the other hand, the operations are performed with built-in functions:

- CREATE is performed through

```
db.collection.insertOne()
```

or

```
db.collection.insertMany()
```

. The first one adds one document, and the latter adds many documents to a database collection.

- READ is performed through

```
db.collection.find()
```

or

```
db.collection.findOne()
```

.

- UPDATE is performed through

```
db.collection.updateOne()
```

,

```
db.collection.updateMany()
```

, or

```
db.collection.replaceOne()
```

.

- DELETE is performed through

```
db.collection.deleteOne()
```

or

```
db.collection.deleteMany()
```

.

Database developers or DBAs often run CRUD statements manually against the database from a client tool. However, in most production use cases, these statements are embedded within the programming language code. When the program runs, the API for the target database takes the CRUD statement and translates it into the native language of the database.

For example, when an ecommerce site visitor initiates the user registration process, a microservice written in Python or Java may read the input values (such as first name, last name, email, address, and so on), and dynamically build an Oracle PL/SQL command. This statement is then sent to the Oracle driver library, which runs it against the database.

Examples of CRUD Operations

Let's take the ecommerce store example further.

- When the user **CREATEs** a client record, she can **READ** inventory by browsing through the product catalog.

- When she places an order, the backend system **UPDATEs** the inventory temporarily to reflect the reduced number of items available.
- When she purchases the item, the **UPDATE** becomes permanent, and other users **READing** the number of items available can see the change. Meanwhile, another process **CREATEs** a record in the shipping company's database, notifying them of the dispatch request.
- If the user removes an item from the shopping cart, then a temporary record added to the "sales" table is **DELETED**.

In an online travel agency, a user can **CREATE** a booking request, **READ** available flights for the requested route, and make a purchase. This will **UPDATE** a list of available seats for the flight and **CREATE** multiple records in the "itinerary" table. If the user terminates the session halfway, then all rows related to this transaction are **DELETED**.

Testing CRUD Operations

Software operations involving **CRUD** are usually black-box tested. When the testers perform certain operations, they check the backend database rather than analyzing the code to see if the intended changes were made or the correct data returned. Such testing aims to validate each

CRUD operation resulting from various possible user interactions in different scenarios.

CRUD and Database Performance

Efficient database design is the prerequisite for optimal CRUD operations. Without good database design, CRUD operations can adversely affect the performance of a database.

For example, operations like UPDATE or DELETE require exclusive locks on the rows (and their related resources, like data pages or indexes). Locks ensure that when one more row is modified, they are not available to other processes or users for any CRUD operation. This is to ensure the integrity of the data.

You can't read a record when it's being deleted or allow two or more users to update a single record simultaneously. Other types of locks, such as shared locks allow simultaneous READs. Locks can be configured at the database or statement level, and different types of locking will dictate which CRUD operations are allowed and how the CRUD operation will behave.

Needless to say, the type of locking and the number of simultaneous locks due to user sessions will affect the performance of a database. For example, a busy ecommerce site with hundreds or thousands of simultaneous users will have many locks operating at the

same time. The result could be slow responsiveness as the user waits for locks to be released.

This performance challenge is why database administrators work to ensure CRUD operations can complete as quickly as possible. This requires query tuning based on feedback from monitoring solutions. Such monitoring solutions can show current database locks, metrics, and logs to help the administrator identify possible bottlenecks.

Discover the world's leading AI-native platform for next-gen SIEM and log management

Elevate your cybersecurity with the [CrowdStrike Falcon[®] platform](#), the premier AI-native platform for [SIEM](#) and [log management](#). Experience security logging at a petabyte scale, choosing between cloud-native or self-hosted deployment options. Log your data with a powerful, index-free architecture, without bottlenecks, allowing threat hunting with over 1 PB of data ingestion per day. Ensure real-time search capabilities to outpace adversaries, achieving sub-second latency for complex queries. Benefit from 360-degree visibility, consolidating data to break down silos and enabling security, IT, and DevOps teams to hunt threats, monitor performance, and ensure compliance seamlessly across 3 billion events in less than 1 second.

LAB EXERCISE:

CREATING AND MANAGING INDICES PERFORMING BASICS SEARCHES

What are indices in data management?

Indexes are **used to quickly locate data without having to search every row in a database table every time said table is accessed**. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

1. Step 1: Create an index. edit. Create a new index named books : ...
2. Step 2: Add data to your index. edit. ...
3. Step 3: Define mappings and data types. edit. ...
4. Step 4: Search your index. edit. ...
5. Step 5: Delete your indices (optional) edit.

reating Your First Index

Creating an index on Elasticsearch is the first step towards leveraging the awesome power of Elasticsearch.

Last updated

July 7, 2023

Creating an index on Elasticsearch is the first step towards leveraging the awesome power of Elasticsearch. While there is a wealth of resources online for creating an index on Elasticsearch, if you're new to it, make sure to check out the definition of an index in our [Elasticsearch core concepts](#).

If you're already familiar with the basics, we have a blog post / white paper on [The Ideal Elasticsearch Index](#), which has a ton of information and things to think about when creating an index.

Note that many Elasticsearch clients will take care of creating an index for you. You should review your client's documentation for more information on its index usage conventions. If you don't know how many indexes your application needs, we recommend creating one index per model or database table.

Important Note on Index Auto-Creation

By default, Elasticsearch has a feature that will automatically create indices. Simply pushing data into a non-existing index will cause that index to be created with mappings inferred from the data. In accordance with Elasticsearch best practices for production applications, we've disabled this feature on Bonsai.

However, some popular tools such as Kibana and Logstash do not support explicit index creation, and rely on auto-creation being available. To accommodate these tools, we've whitelisted popular time-series index names such as

logstash*

,

requests*

,

events*

,

.kibana*

and

kibana-int*

.

For the purposes of this discussion, we'll assume that you don't have an Elasticsearch client that can create the index for you. This guide will proceed with the manual steps for creating an index, changing settings, populating it with data, and finally destroying your index.

Manually Creating an Index

There are two main ways to manually create an index in your Bonsai cluster. The first is with a command line tool like

curl

or

httpie

. Curl is a standard tool that is bundled with many *nix-like operating systems. OSX and many Linux distributions should have it. It can even be installed on Windows. If you do not have curl, and don't have a package manager capable of installing it, you can [download it here](#).

The second is through the Interactive Console. The Interactive Console is a feature provided by Bonsai and found in your cluster dashboard.

Let's create an example index called

acme-production

from the command line with curl.

```
$ curl -X PUT http://user:password@redwood-12345.us-east-1.bonsai.io/acme-production
```

```
{"acknowledged":true}
```

Authentication required

All Bonsai clusters have a randomly generated username and password. By default, these credentials need to be included with *all* requests in order to be processed. If you're seeing something an [HTTP 401 Error](#) like this:

HTTP 401: Authorization required

then your credentials were not supplied. You can view the fully-qualified URL in your cluster dashboard. It will look like this:

```
http://user:password@redwood-12345.us-east-1.bonsai.io
```

Updating the Index Settings

We can inspect the index settings with a GET call to

```
/_cat/indices
```

like so:

```
$ curl -XGET http://user:password@redwood-12345.us-east-1.bonsai.io/_cat/indices?v
```

```
health status index          pri rep docs.count docs.deleted  
store.size pri.store.size
```

```
green open  acme-production  1  1      0      0  
260b 130b
```

The

?v

at the end of the URL tells Elasticsearch to return the headers of the data it's returning. It's not required, but it helps explain the data.

In the example above, Elasticsearch shows that the

acme-production

index was created with one primary shard and one replica shard. It doesn't have any documents yet, and is only a few bytes in size.

Let's add a replica to the index:


```
$ curl -XPUT http://user:password@redwood-12345.us-east-1.bonsai.io/acme-production/_settings -d '{"index":{"number_of_replicas":2}}' {"acknowledged":true}
```

Now, when we re-query the

/_cat/indices

endpoint, we can see that there are now two replicas, where before there was only one:

```
$ curl -XGET http://user:password@redwood-12345.us-east-1.bonsai.io/_cat/indices?v
```

health	status	index	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
green	open	acme-production	1	2	0	0	390b	130b

Similarly, if we wanted to remove all the replicas, we could simply modify the JSON payload like so:

```
$ curl -XPUT http://user:password@redwood-12345.us-east-1.bonsai.io/acme-production/_settings -d '{"index":{"number_of_replicas":0}}'
```

```
{"acknowledged":true}
```

```
$ curl -XGET http://user:password@redwood-12345.us-east-1.bonsai.io/_cat/indices?v
```

```
health status index          pri rep docs.count docs.deleted  
store.size pri.store.size
```

```
green open  acme-production  1  0      0      0  
318b 159b
```

Adding Data to Your Index

Let's insert a "Hello, world" test document to verify that your new index is available, and to highlight some basic Elasticsearch concepts.

Every document prior to Elasticsearch 7.x should specify a

type

, and preferably an

id

. You may specify these values with the

`_id`

`_id` and the

_type

keys, or Elasticsearch will infer them from the URL. If you don't explicitly provide an id, Elasticsearch will create a random one for you.

In the following example, we use POST to add a simple document to the index which specifies a

_type

_type of

test

and an

_id

_id of 1. You should replace the sample URL in this document with your own index URL to follow along:

```
$ curl -XPOST http://user:password@redwood-12345/acme-production/test/1 -d '{"title":"Hello world"}'
```

```
{
```

```
  "_index" : "acme-production",
```

```
  "_type" : "test",
```

```
"_id" : "1",  
"_version" : 1,  
"_shards" : {  
  "total" : 3,  
  "successful" : 3,  
  "failed" : 0  
},  
"created" : true  
}
```

Because we haven't explicitly defined a mapping (schema) for the acme-production index, Elasticsearch will come up with one for us. It will inspect the contents of each field it receives and attempt to infer a data structure for the content. It will then use that for subsequent documents.

Overview

In Elasticsearch, an index (plural: indices) contains a schema and can have one or more [shards](#) and [replicas](#). An

Elasticsearch index is divided into shards and each shard is an instance of a [Lucene](#) index.

Indices are used to store the [documents](#) in dedicated data structures corresponding to the data type of fields. For example, text fields are stored inside an inverted index whereas numeric and geo fields are stored inside BKD trees.

Mapping Types and Field Data Types in Elasticsearch

Mapping types and **field data types** are fundamental concepts in **Elasticsearch** that define how data is **indexed, stored** and queried within an index.

Understanding these concepts is crucial for effectively modeling our data and optimizing search performance.

In this article, We will learn about the mapping types, field data types, and their significance in Elasticsearch. Also, understand some examples for better understanding.

Introduction to Mapping Types

- **Mapping types** were a way to organize data within an **index** in earlier versions of [Elasticsearch](#) (before 6.x).

- With mapping types, we could define multiple types within a **single index**, each representing a different entity or document structure.
- Mapping types allowed us to define the schema for documents within an index. **For example**, we could have a "book" type and a "movie" type within the same index.
- Starting with Elasticsearch then mapping types are outdated and will be removed in future versions.
- Instead of using mapping types, Elasticsearch now recommends using a single type per index.
- The shift away from mapping types allows for a more **flexible schema design**, reducing **mapping** conflicts and improving **efficiency**.
- In newer versions of Elasticsearch, it is recommended to use different indices to represent different entities or document structures.
- Each index can have its mapping to define the schema for documents within that index.

Field Data Types in Elasticsearch

Field data types define the type of data that can be stored in a field within a document. Elasticsearch provides a wide range of data types to accommodate various **types of**

data, including text, numbers, dates, and more. Let's explore some common field data types in Elasticsearch:

1. Text

- It is used for **full-text searches**. Text fields are analyzed which means that they are broken down into individual terms which are then **indexed**. This allows for efficient full-text search but can result in larger index sizes.

2. Keyword

- It is Used for **exact matching**. Keyword fields are not analyzed and are indexed **as-is**. They are suitable for fields like **IDs**, **email addresses** and **enum** values.

3. Integer

- It is Used for **numeric values**. Elasticsearch supports various numeric types, including **integer**, **long**, **float** and **double**. Numeric fields are indexed in a way that allows for efficient range **queries** and **aggregations**.

4. Long

- The long data type is similar to integer but is suitable for **larger** numbers.

5. Date

- It is Used for **date** and **time** values. Elasticsearch can parse various date formats and supports date **arithmetic** and **formatting** in queries.

6. Boolean

- The boolean data type is used for fields that contain **true** or **false** values.

7. Float and Double

- The **float** and **double** data types are used for fields that contain **floating-point** numbers.

Mapping Types and Field Definitions

Let's create a simple index with mappings for various field **data types** to understand how mappings and field data types work in Elasticsearch.

```
PUT /my_index
{
  "mappings": {
    "properties": {
      "title": {
        "type": "text"
      },
      "category": {
        "type": "keyword"
      },
    },
  },
}
```



```
"quantity": {
  "type": "integer"
},
"price": {
  "type": "float"
},
"is_active": {
  "type": "boolean"
},
"created_at": {
  "type": "date"
}
}
}
```

Explanation:

- We create an index named `my_index` with mappings for fields like `title`, `category`, `quantity`, `price`, `is_active`, and `created_at`.
- Each field is assigned a specific data type (text, keyword, integer, float, boolean, date).

Indexing Documents with Field Data Types

Suppose We need to index a document into our `my_index` index to demonstrate how field data types are applied in Elasticsearch.

```
POST /my_index/_doc/1
{
  "title": "Product A",
  "category": "Electronics",
  "quantity": 100,
  "price": 49.99,
  "is_active": true,
  "created_at": "2022-05-01T12:00:00"
}
```

Explanation: In this example, we use a POST request to index a document with various field data types into the my_index index. Each field in the document corresponds to a specific field data type in **Elasticsearch**, such as **text**, keyword, integer, float, **boolean**, and date.

Retrieving Mapping Information

Suppose We want to retrieve mapping information for the my_index index to understand how field data types are mapped in Elasticsearch.

```
GET /my_index/_mapping
```

Sample Output:

```
{
  "my_index": {
    "mappings": {
      "properties": {
```

```
{
  "title": {
    "type": "text"
  },
  "category": {
    "type": "keyword"
  },
  "quantity": {
    "type": "integer"
  },
  "price": {
    "type": "float"
  },
  "is_active": {
    "type": "boolean"
  },
  "created_at": {
    "type": "date"
  }
}
```

Explanation: The **GET** request to the `_mapping` endpoint retrieves the mapping information for the `my_index` index. This information includes the field data types for each field in the index, allowing us to understand how **Elasticsearch interprets and indexes** our data.

Querying Documents with Field Data Types

Suppose We need to query documents based on their field data types, such as finding products with a price less than 50

```
GET /my_index/_search
```

```
{
  "query": {
    "range": {
      "price": {
        "lt": 50
      }
    }
  }
}
```

Explanation: This GET request to the `_search` endpoint uses a range query to find documents in the **my_index** index where the price field is less than 50. By specifying the field **data type** (float) and the range (lt: less than), we can retrieve relevant documents based on their numeric values.

Conclusion

Mapping types and field data types are foundational concepts in Elasticsearch that play a crucial role in organizing and querying data. By understanding mapping types and selecting appropriate field data types for your

index, you can optimize search performance and ensure accurate representation of your data.

In this article, we explored the basics of mapping types, common field data types, and their significance in Elasticsearch. We also provided practical examples and outputs to illustrate how mappings and field data types are defined, indexed, and queried in Elasticsearch. With this knowledge, you'll be better equipped to design and manage Elasticsearch indices effectively for your applications.

SEARCH DOCUMENTS IN ELASTICSEARCH

indexed documents are available for search in near real-time, using the [_search API](#).

Search all documents

Run the following command to search the books index for all documents

indexed documents are available for search in near real-time, using the [search API](#).

Search all documents

```
{  
  "took": 2,  
  "timed_out": false,  
  "_shards": {  
    "total": 5,  
    "successful": 5,  
    "skipped": 0,  
    "failed": 0  
  },  
  "hits": {  
    "total": {  
      "value": 7,  
      "relation": "eq"  
    },  
  },  
}
```

```
"max_score": 1,
"hits": [
  {
    "_index": "books",
    "_id": "CwICQpIBO6vvGGiC_3Ls",
    "_score": 1,
    "_source": {
      "name": "Brave New World",
      "author": "Aldous Huxley",
      "release_date": "1932-06-01",
      "page_count": 268
    }
  },
  ... (truncated)
]
}
}
```

}

The `took` field indicates the time in milliseconds for Elasticsearch to execute the search

The `timed_out` field indicates whether the search timed out

The `_shards` field contains information about the number of [shards](#) that the search was executed on and the number that succeeded

The `hits` object contains the search results

The `total` object provides information about the total number of matching documents

The `max_score` field indicates the highest relevance score among all matching documents

The `_index` field indicates the index the document belongs to

The `_id` field is the document's unique identifier

The `_score` field indicates the relevance score of the document

The `_source` field contains the original JSON object submitted during indexing

match query

You can use the [match query](#) to search for documents that contain a specific value in a specific field. This is the standard query for full-text searches.

Run the following command to search the books index for documents containing brave in the name field

GET books/_search

```
{  
  "query": {  
    "match": {  
      "name": "brave"  
    }  
  }  
}
```

Delete index API

[edit](#)

New API reference

For the most up-to-date API details, refer to [Index APIs](#).

Deletes one or more indices.

DELETE /my-index-000001

Copy as curl [Try in Elastic](#)

Request

DELETE /<index>

Prerequisites

[edit](#)

- If the Elasticsearch security features are enabled, you must have the delete_index or manage [index privilege](#) for the target index.

Description

[edit](#)

Deleting an index deletes its documents, shards, and metadata. It does not delete related Kibana components, such as data views, visualizations, or dashboards.

You cannot delete the current write index of a data stream. To delete the index, you must [roll over](#) the data stream so a new write index is created. You can then use the delete index API to delete the previous write index.

Path parameters

[edit](#)

<index>

(Required, string) Comma-separated list of indices to delete. You cannot specify [index aliases](#).

By default, this parameter does not support wildcards (*) or _all. To use wildcards or _all, set the [action.destructive_requires_name](#) cluster setting to false.