Community Experience Distilled

# Elasticsearch Blueprints

A practical project-based guide to generating compelling search solutions using the dynamic and powerful features of Elasticsearch

Vineeth Mohan

[PACKT] open source*
community experience distilled
PUBLISHING

# Elasticsearch Blueprints

A practical project-based guide to generating compelling search solutions using the dynamic and powerful features of Elasticsearch

**Vineeth Mohan**

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

# Elasticsearch Blueprints

First published: July 2015

Production reference: 1200715

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78398-492-3

`www.packtpub.com`

# Credits

**Author**
Vineeth Mohan

**Reviewers**
Kartik Bhatnagar
Tomislav Poljak

**Acquisition Editor**
Harsha Bharwani

**Content Development Editor**
Ajinkya Paranjape

**Technical Editor**
Mrunmayee Patil

**Copy Editor**
Neha Vyas

**Project Coordinator**
Harshal Ved

**Proofreader**
Safis Editing

**Indexer**
Mariammal Chettiyar

**Production Coordinator**
Nilesh R. Mohite

**Cover Work**
Nilesh R. Mohite

# About the Author

**Vineeth Mohan** is an architect and developer. He currently works as the CTO at Factweavers Technologies and is also an Elasticsearch-certified trainer.

He loves to spend time studying emerging technologies and applications related to data analytics, data visualizations, machine learning, natural language processing, and developments in search analytics. He began coding during his high school days, which later ignited his interest in computer science, and he pursued engineering at Model Engineering College, Cochin. He was recruited by the search giant Yahoo! during his college days. After 2 years of work at Yahoo! on various big data projects, he joined a start-up that dealt with search and analytics. Finally, he started his own big data consulting company, Factweavers.

Under his leadership and technical expertise, Factweavers is one of the early adopters of Elasticsearch and has been engaged with projects related to end-to-end big data solutions and analytics for the last few years.

There, he got the opportunity to learn various big-data-based technologies, such as Hadoop, and high-performance data ingress systems and storage. Later, he moved to a start-up in his hometown, where he chose Elasticsearch as the primary search and analytic engine for the project assigned to him.

Later in 2014, he founded Factweavers Technologies along with Jalaluddeen; it is consultancy that aims at providing Elasticsearch-based solutions. He is also an Elasticsearch-certified corporate trainer who conducts trainings in India. Till date, he has worked on numerous projects that are mostly based on Elasticsearch and has trained numerous multinationals on Elasticsearch.

I would like to thank Arun Mohan and my other friends who supported and helped me in completing this book. My unending gratitude goes to the light that guides me.

# About the Reviewer

**Kartik Bhatnagar** is a technical architect at the big data analytics unit of Infosys, Pune. He is passionate about new technologies and the leading development work on Apache Storm and MarkLogic NoSQL. He has 9.5 years of development experience with many fortune clients across countries. He has implemented Elasticsearch engine for a major publishing company in UK. His expertise also includes full-stack Amazon Web Services (AWS). Kartik is also active on the stackoverflow platform and is always eager to help young developers with new technologies.

Kartik is presently working on book based on Storm/Python programming, which is yet to be published.

I would like to dedicate this book to my niece, Pranika, who will be 6 months old by the time this book gets published. Sincere thanks to my parents; wife, Aditi; and son, Prayrit, for their constant support and love to make the review of this book possible.

# Table of Contents

# Preface

Elasticsearch is a distributed search server similar to Apache Solr with a focus on large datasets, schemaless setup, and high availability. Utilizing the Apache Lucene library (also used in Apache Solr), Elasticsearch enables powerful full-text searches, autocomplete, the "morelikethis" search, multilingual functionality, as well as an extensive search query DSL.

Elasticsearch's schemafree architecture provides developers with built-in flexibility as well as ease of setup. This architecture allows Elasticsearch to index and search unstructured content, making it perfectly suited for both small projects and large big data warehouses—even with petabytes of unstructured data.

This book will enable you to utilize the amazing features of Elasticsearch and build projects to simplify operations on even large datasets. This book starts with the creation of a Google-like web search service, enabling you to generate your own search results. You will then learn how an e-commerce website can be built using Elasticsearch, which will help users search and narrow down the set of products they are interested in. You will explore the most important part of a search—relevancy—based on the various parameters, such as relevance, document collection relevance, user usage pattern, geographic nearness, and document relevance to select the top results.

Next, you will discover how Elasticsearch manages relational content for even complex real-world data. You will then learn the capabilities of Elasticsearch as a strong analytic search platform, which coupled with some visualization techniques can produce real-time data visualization. You will also discover how to improve your search quality and widen the scope of matches using various analyzer techniques. Finally, this book will cover the various geo capabilities of Elasticsearch to make your searches similar to real-world scenarios.

## What this book covers

*Chapter 1, Google-like Web Search*, takes you along the course of building a simple scalable search server. You will learn how to create an index and add some documents to it and you will try out some essential features, such as highlighting and pagination of results. Also, it will cover topics such as setting an analyzer for our text; applying filters to eliminate unwanted characters, such as HTML tags; and so on.

*Chapter 2, Building Your Own E-Commerce Solution*, covers how to design a scalable e-commerce search solution to generate accurate search results using various filters, such as date-range based and prize-range based filters.

*Chapter 3, Relevancy and Scoring*, unleashes the power and flexibility of Elasticsearch that will help you implement your own scoring logic.

*Chapter 4, Managing Relational Content*, covers how to use the document linking or relational features of Elasticsearch.

*Chapter 5, Analytics Using Elasticsearch*, covers the capability and usage of Elasticsearch in the analytics area with a few use case scenarios.

*Chapter 6, Improving the Search Experience*, helps you learn how to improve the search quality of a text search. This includes the description of various analyzers and a detailed description of how to mix and match them.

*Chapter 7, Spicing Up a Search Using Geo*, explores how to use geo information to get the best out of search and scoring.

*Chapter 8, Handling Time-based Data*, explains the difficulties we face when we use normal indexing in Elasticsearch.

## What you need for this book

You will need the following tools to build the projects and execute the queries in this book:

- **cURL**: cURL is an open source command-line tool available in both Windows and Unix. It is widely used to communicate with web interfaces. Since all communication to Elasticsearch can be done through standard REST protocols, we will use cURL throughout the book to communicate with Elasticsearch. The official site for cURL is `http://curl.haxx.se/download.html`.

- **Elasticsearch**: You need to install Elasticsearch from its official site, `http://www.elasticsearch.org/`. When this book was written, the latest Elasticsearch version available was 1.0.0, so I would recommend that you use this one. The only dependency of Elasticsearch is Java 1.6 or its higher versions. Once you make sure you have installed Java, download the Elasticsearch ZIP file, the installation instructions for which are mentioned in *Chapter 1*, *Google-like Web Search*.

## Who this book is for

If you are a developer who has good practical experience in Elasticsearch, Lucene, or Solr and want to know how to implement Elasticsearch in real-world scenarios, then this book is for you.

## Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Note that for most of the fields that have a string value, such as `sex`, `purposeOfVisit`, and so on, we add the `not_analyzed` field type definition."

A block of code is set as follows:

```
curl -X PUT "http://$hostname:9200/planeticketing" -d '{
        "index": {
            "number_of_shards": 2,
            "number_of_replicas": 1
        }
    }'
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
          "query": "nausea fever"
        }
      },
      "negative": {
        "multi_match": {
          "fields": [
            "title",
            "content"
          ],
```

Any command-line input or output is written as follows:

```
curl -XPOST 'http://localhost:9200/wiki/articles/' -d @India.json
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Now, take the **Browser** tab in the head UI."

[ Warnings or important notes appear in a box like this. ]

[ Tips and tricks appear like this. ]

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

### Downloading the example code

You can download the example code files from your account at `http://www.packtpub.com` for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

### Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

### Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

## Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

# 1
# Google-like Web Search

Text search problems are one of the key and common use cases for web-based applications. Developers over the world have been keen to bring an open source solution to this problem. Hence, the Lucene revolution happened. **Lucene** is the heart of most of the search engines that you see today. It basically accepts the text that is to be searched, stores it in an easy searchable form or data structure (inverted index), and then accepts various types of search queries and returns a set of matching results. After the first search revolution, came the second one. Many server-based search solutions, such as Apache SOLR, were built on top of Lucene and marked the second phase of the search revolution. Here, a powerful wrapper was made to interface web users that wanted to index and search text of Lucene. Many powerful tools, notably SOLR, were developed at this stage of revolution. Some of these search frameworks were able to provide document database features too. Then, the next phase of the search revolution came, which is still on-going. The design goal of this phase is provide scaling solutions for the existing stack. **Elasticsearch** is a search and analytic engine that provides a powerful wrapper to Lucene along with an inbuilt document database and provisions various scaling solutions. The document database is also implemented using Lucene. Though competitors of Elasticsearch have some more advanced feature sets, those tools lack the simplicity and the wide range of scalability solutions Elasticsearch offers. Hence, we can see that Elasticsearch is the farthest point to which the search revolution has reached and is the future of text search.

This chapter takes you along the course to build a simple scalable search server. We will see how to create an index and add some documents to it and try out some essential features such as highlighting and pagination of results. Also, we will cover topics such as how to set an analyzer for our text and how to apply filters to eliminate unwanted characters such as HTML tags, and so on.

Here are the important topics that we will cover in this chapter:

- Deploying Elasticsearch
- Concept of the head UI shards and replicas
- Index – type mapping
- Analyzers, filters, and tokenizers
- The head UI

Let's start and explore Elasticsearch in detail.

## Deploying Elasticsearch

First, let's download and install the following tools:

- **cURL**: cURL is an open source command-line tool available in both Windows and Unix. It is widely used to communicate with web interfaces. Since all communication to Elasticsearch can be done through standard REST protocols, we will use cURL throughout the book to communicate with Elasticsearch. The official website of cURL is `http://curl.haxx.se/download.html`.
- **Elasticsearch**: You need to install Elasticsearch from its official site `http://www.elasticsearch.org/`. When this book was written, the latest version of Elasticsearch available was 1.0.0, so I would recommend that you use the same version. The only dependency of Elasticsearch is Java 1.6 or its higher versions. Once you make sure that you have Java installed, download the Elasticsearch ZIP file.

First, let's download Elasticsearch:

1. Unzip and place the files in a folder.
2. Next, let's install the Elasticsearch-head plugin. Head is the standard web frontend of the Elasticsearch server. Most of the Elasticsearch operations can be done via a head plugin. To install head, run the following command from the folder where Elasticsearch is installed:
   ```
   bin/plugin -install mobz/elasticsearch-head # (Linux users)
   bin\plugin -install mobz/elasticsearch-head # (Windows users)
   ```
3. You should see a new folder in the `plugins` directory. Open a console and type the following to start Elasticsearch:
   ```
   bin/elasticsearch   #(Linux users)
   bin\elasticsearch.bat  #(Windows users)
   ```

4. The `-d` command is used to run Elasticsearch in the background rather than the foreground. By running the application in the foreground, we can track the changes taking place in it through the logs spitted in the console. The default behavior is to run in the foreground.

One of the basic design goals of Elasticsearch is its high configurability clubbed with its optimal default configurations that get you started seamlessly. So, all you have to do is start Elasticsearch. You don't have to learn any complex configuration concepts at least to get started. So our search server is up and running now.

To see the frontend of your Elasticsearch server, you can visit `http://localhost:9200/_plugin/head/`.

## Communicating with the Elasticsearch server

cURL will be our tool of choice that we will use to communicate with Elasticsearch. Elasticsearch follows a REST-like protocol for its exposed web API. Some of its features are as follows:

- `PUT`: The HTTP method `PUT` is used to send configurations to Elasticsearch.
- `POST`: The HTTP method `POST` is used to create new documents or to perform a search operation. While successful indexing of documents is done using `POST`, Elasticsearch provides you with a unique ID that points to the index file.
- `GET`: The HTTP method `GET` is used to *retrieve* an already indexed document. Each document has a unique ID called a **doc ID** (short form for document's ID). When we index a document using `POST`, it provides a document ID, which can be used to retrieve the original document.
- `DELETE`: The HTTP method `DELETE` is used to delete documents from the Elasticsearch index. Deletion can be performed based on a search query or directly using the document ID.

To specify the HTTP method in cURL, you can use the `-X` option, for example, `CURL -X POST http://localhost/`. JSON is the data format used to communicate with Elasticsearch. To specify the data in cURL, we can specify it in the following forms:

- **A command line**: You can use the `-d` option to specify the JSON to be sent in the command line itself, for example:

```
curl –X POST 'http://localhost:9200/news/public/'
  –d '{ "time" : "12-10-2010"}
```

- **A file**: If the JSON is too long or inconvenient to be mentioned in a command line, you can specify it in a file or ask cURL to pick the JSON up from the file. You need to use the same `-d` option with a `@` symbol just before the filename, for example:

  ```
  curl -X POST 'http://localhost:9200/news/public/' -d @file
  ```

## Shards and replicas

The concept of **sharding** is introduced in Elasticsearch to provide horizontal scaling. Scaling, as you know, is to increase the capacity of the search engine, both the index size and the query rate (query per second) capacity. Let's say an application can store up to 1,000 feeds and gives reasonable performance. Now, we need to increase the performance of this application to 2,000 feeds. This is where we look for scaling solutions. There are two types of scaling solutions:

- **Vertical scaling**: Here, we add hardware resources, such as more main memory, more CPU cores, or **RAID disks** to increase the capacity of the application.
- **Horizontal scaling**: Here, we add more machines to the system. As in our example, we bring in one more machines and give both the machines 1,000 feeds each. The result is computed by merging the results from both the machines. As both the processes take place in parallel, they won't eat up more time or bandwidth.

Guess what! Elasticsearch can be scaled both horizontally and vertically. You can increase its main memory to increase its performance and you can simply add a new machine to increase its capacity. Horizontal scaling is implemented using the concept of sharding in Elasticsearch. Since Elasticsearch is a distributed system, we need to address our data safety/availability concerns. Using replicas we achieve this. When one replica (size 1) is defined for a cluster with more than one machine, two copies of the entire feed become available in the distributed system. This means that even if a single machine goes down, we won't lose data and at the same time. The load would be distributed somewhere else. One important point to mention here is that the default number of shards and replicas are generally sufficient and also, we have the provision to change the replica number later on.

This is how we create an index and pass the number of shards and replicas:

```
curl -X PUT "localhost:9200/news" -d '{
"settings": {
"index": {
"number_of_shards": 2,
```

```
"number_of_replicas": 1
}
}
}'
```

A few things to be noted here are:

- Adding more primary shards will increase the write throughout the index
- Adding more replicas will increase the durability of the index and the read throughout, at the cost of disk space

## Index-type mapping

An index is a grouping logic where feeds of the same type are encapsulated together. A type is a sub grouping logic under index. To create a type under index, you need to decide on a type name. As in our case, we take the index name as `news` and the type name as `public`. We created the index in the previous step and now we need to define the data types of the fields that our data hold in the type mapping section.

Check out the sample given next. Here, the `date` data type takes the time format to be `yyyy/MM/dd HH:mm:ss` by default:

```
curl -X PUT "localhost:9200/news/public/_mapping" -d '{
"public" :{
"properties" :{
"Title" : {"type" : "string" },
"Content": {"type" : "string" },
"DOP": {"type" : "date" }
}
}
}'
```

Once we apply mapping, certain aspects of it such as new field definitions can be updated. However, we can't update certain other aspects such as changing the type of a field or changing the assigned analyzer. So, we now know how to create an index and add necessary mappings to the index we created. There is another important thing that you must take care of while indexing your data, that is, the analysis of our data. I guess you already know the importance of analysis. In simple terms, analysis is the breaking down of text into an elementary form called **tokens**. This tokenization is a must and has to be given serious consideration. Elasticsearch has many built-in analyzers that do this job for you. At the same time, you are free to deploy your own custom analyzers as well if the built-in analyzers do not serve your purpose. Let's see analysis in detail and how we can define analyzers for fields.

## Setting the analyzer

Analyzers constitute an important part of indexing. To understand what analyzers do, let's consider three documents:

- **Document1 (tokens)**: { `This , is , easy` }
- **Document2 (tokens)**: { `This , is , fast` }
- **Document3 (tokens)**: { `This , is , easy , and , fast` }

Here, terms such as `This`, `is`, as well as `and` are not relevant keywords. The chances of someone wanting to search for such words are very less, as these words don't contribute to the facts or context of the document. Hence, it's safe to avoid these words while indexing or rather you should avoid making these words searchable.

So, the tokenization would be as follows:

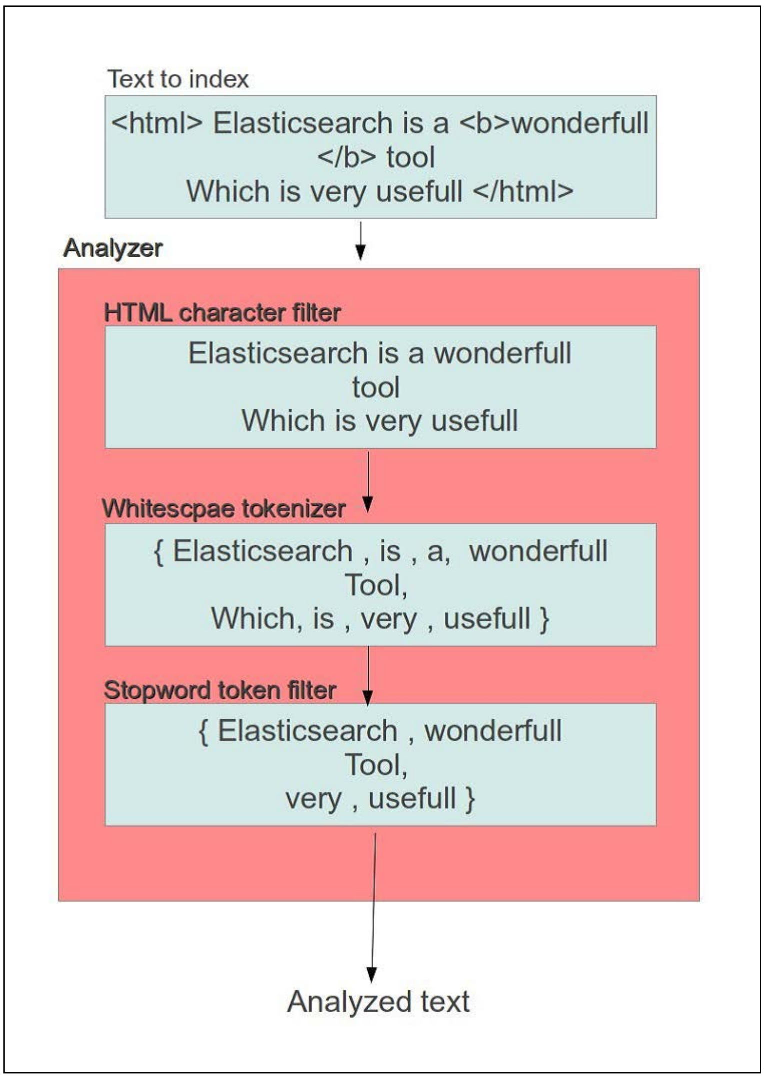
- **Document1 (tokens)**: { `easy` }
- **Document2 (tokens)**: { `fast` }
- **Document3 (tokens)**: { `easy , fast` }

Words such as `the`, `or`, as well as `and` are referred to as **stop words**. In most cases, these are for grammatical support and the chances that someone will search based on these words are slim. Also, the analysis and removal of stop words is very much language dependent. The process of selecting/transforming the searchable tokens from a document while indexing is called **analyzing**. The module that facilitates this is called an analyzer. The analyzer we just discussed is a stop word analyzer. By applying the right analyzer, you can minimize the number of searchable tokens and hence get better performance results.

There are three stages through which you can perform an analysis:

- **Character filters**: Filtering is done at character level before processing for tokens. A typical example of this is an HTML character filter. We might give an HTML to be indexed to Elasticsearch. In such instances, we can provide the HTML `CHAR` filter to do the work.
- **Tokenizers**: The logic to break down text into tokens is depicted in this state. A typical example of this is whitespace tokenizers. Here, text is broken down into tokens by splitting the text based on the white space occurrence.
- **Token filters**: On top of the previous process, we apply a token filter. In this stage, we filter tokens to match our requirement. The length token filter is a typical token filter. A token filter of type `length` removes words which are too long or too short for the stream.

Here is a flowchart that depicts this process:


Text to index
<html> Elasticsearch is a <b>wonderfull </b> tool Which is very usefull </html>
Analyzer
HTML character filter
Elasticsearch is a wonderfull tool Which is very usefull
Whitescpae tokenizer
{ Elasticsearch , is , a, wonderfull Tool, Which, is , very , usefull }
Stopword token filter
{ Elasticsearch , wonderfull Tool, very , usefull }
Analyzed text

It should be noted that any number of such components can be incorporated in each stage. A combination of these components is called an analyzer. To create an analyzer out of the existing components, all we need to do is add the configuration to our Elasticsearch configuration file.