ElasticSearch

- It is a powerful, flexible and performance technology developed with Java that performs operations such as analyzing big data, searching among them and suggesting.
- Because it has Apache Lucene infrastructure, it has an excellent full-text search capability.
- It has a document-based structure and therefore keeps data in JSON format.
- All search, CRUD and analysis operations can be done easily through the Rest API it provides.

Search Engine Candidates:

- Elasticsearch
- Solr
- InfluxDB
- TypeSense
- Algolia
- Sphix

For more:

- https://www.educba.com/elasticsearch-alternatives/
- https://www.linuxlinks.com/searchengines/

Solr vs **Elasticsearch**

Both Solr and Elasticsearch write indexes in Lucene. But, since differences exist in sharding and replication (among other features), there are also differences in their files and architectures. Additionally, Elasticsearch has native DSL support while Solr has a robust Standard Query Parser that aligns to Lucene syntax.
For more: https://logz.io/blog/solr-vs-elasticsearch/

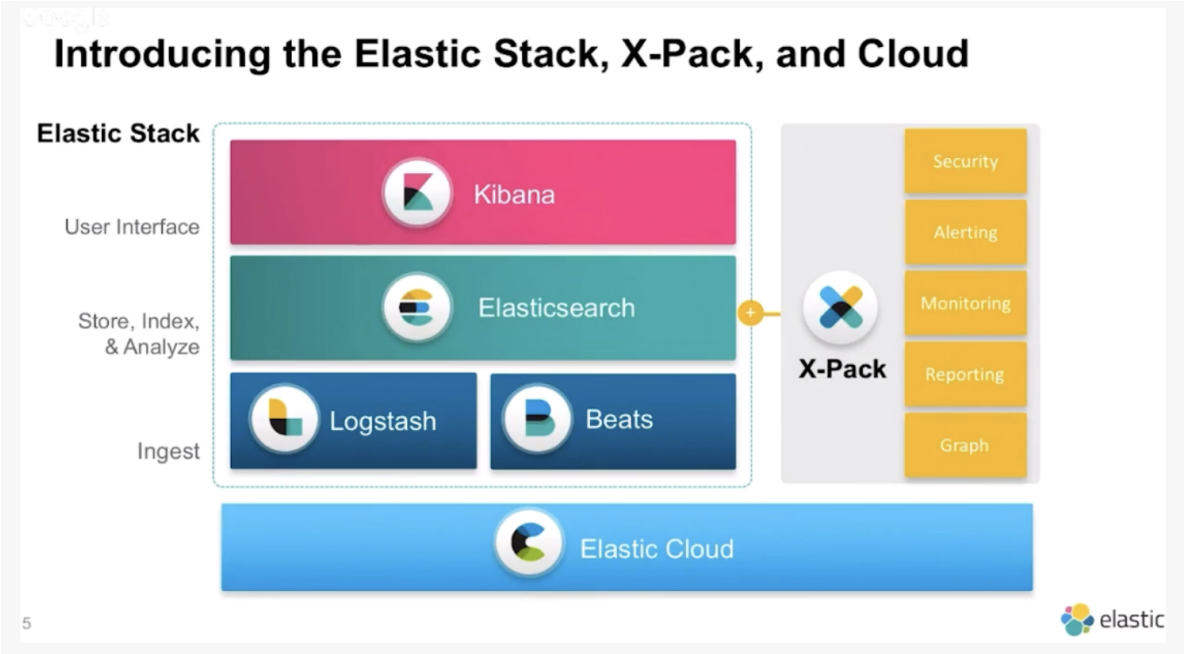
Elastic Search
ElasticSearch is able to achieve fast search responses because instead of searching the text directly, it searches an index instead. This is more or less like searching for a keyword by scanning the index at the back of a book as opposed to searching every word of every page of the book. ElasticSearch can scale up to thousands of servers and accommodate petabytes of data. Its enormous capacity results directly from its elaborate, distributed architecture.

Features:

- **Lots of search options**
- **Document-oriented**
- **Speed**
- **Scalability**
- **Data record**
- **Query fine tuning**
- **RESTful API**
- **Distributed approach**
- **Multi-tenancy**

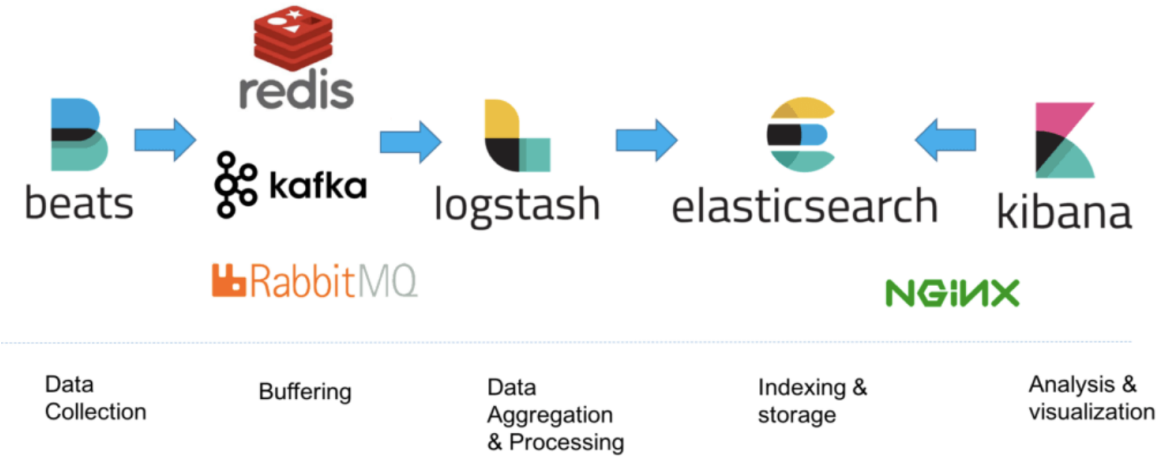
For more:

- https://www.elastic.co/elastic-stack/features
- https://dzone.com/articles/elastic-search-advantages-case-studies-amp-books



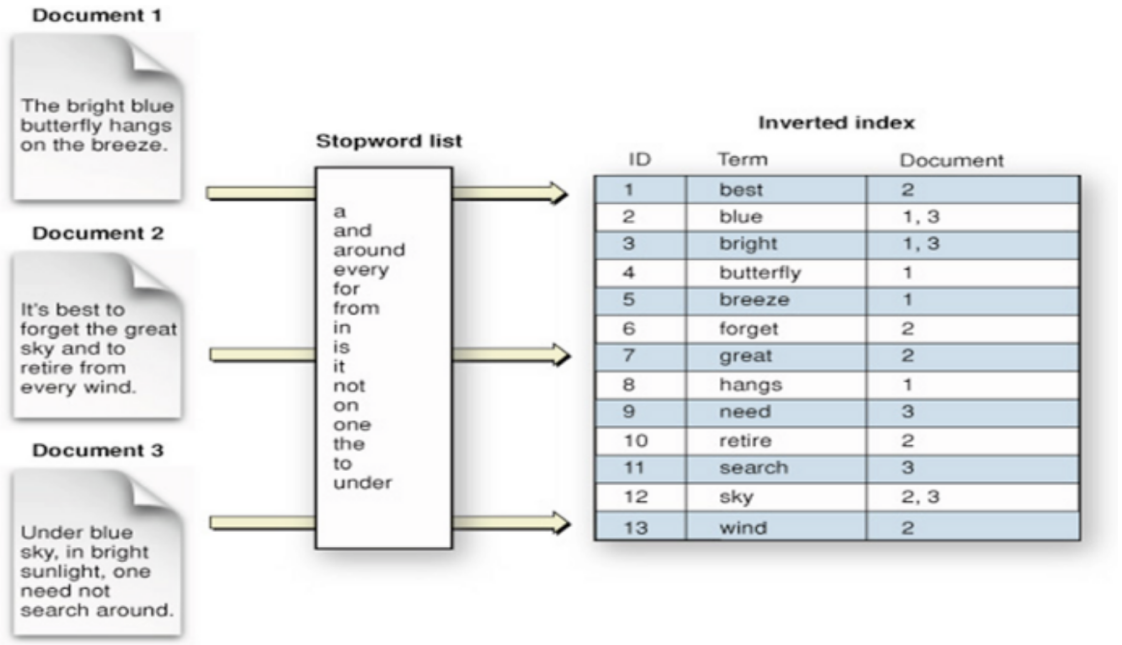
Beats is responsible for sending the logs collected from the clients to the elk stack server. **Logstash** is the part that compiles these collected logs in a server and performs the conversion and meaning-making process that needs to be done on it. **Elasticsearch** is the part that turns these logs into searchable and analyzable by indexing them. The part that visualizes all this detailed work is **Kibana**.

- Each node runs 1 ElasticSearch. Each node communicates with each other.
- Kibana keeps configurations on Elastic Search.
- LogStash pipeline: Inputs -> Filters -> Outputs
- Beats collects files, logs and transfers them to LogStash or ElasticSearch.
  FileBeat,MetricBeat, PacketBeat, WinLogBeat, AuditBeat, HeartBeat.



Terms:

- Index: Indexes are collections of Json data. We determine the indexes. We can define it as a word or a sentence. For each word in each added document, there is an indexing system that keeps the information in which document or documents contain that word.
  It includes Apache Lucene's Inverted Indexes that make searching very easy, and mapping that defines the mapping of the data in it.
  While a data is being inserted, it is indexed using Apache Lucene infrastructure.



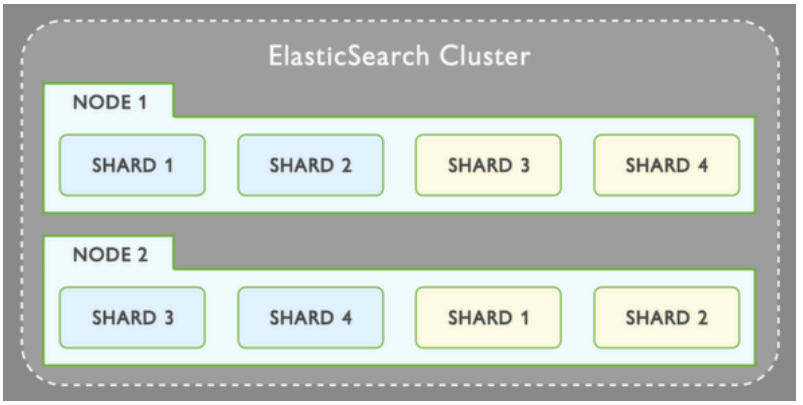
- Mapping: It is the definition of how the relevant document will be transferred to the search engine. This is how we define the type of data when we are indexes.
  Types:
  - Explicit Mapping: Fields and their types are predefined.
  - Dynamic Mapping: Fields and their types are automatically defined by ElasticSearch.

  Field Data Types:
  - Simple Data Types: string, byte, short, integer, long, float, double, boolean, date
  - Complex Data Types: object, nested
  - Special Data Types: geo_point, geo_shape, completion

- Document: Every record in Index like row in RDBMS. Every document has own unique Id.
- Field: A field in document.
- Type: Table in RDBMS.

Relational Database	Elasticsearch
Schema	Mapping
Database	Index
Table	Type
Row	Document
Column	Field

- Cluster: Collection of nodes.

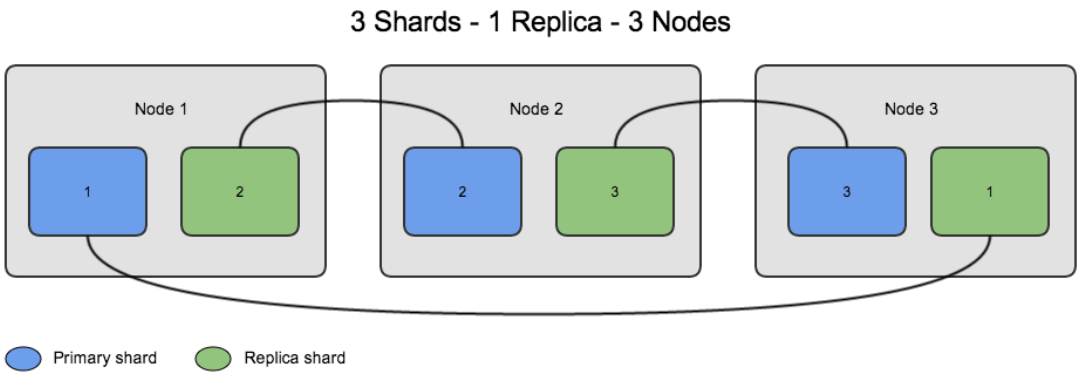


- Node: Every machine that includes ElasticSearch data. Indexing and searching mechanism are in nodes.
- Shard: Shard is Apache Lucene itself. Too much data in indexes can cause performance, scalability, and maintenance issues.
  Therefore, indexes can be divided into shards, which are instances of Lucene. While creating the index, the number of shards is determined and the data is distributed to the shards.
  Shards are divided into nodes. Thus, transactions become both distributed and parallel, and performance increases. It allows to split and scale the content volume horizontally.
  The amount of primary shards cannot be changed later. Instead, more replicas can be created for read. In the worst case, a new index can be created.

```
PUT /testindex
{
 "settings": {
   "number_of_shards": 3
   , "number_of_replicas": 1
 }
}
```

  In above setting, 3 primary shards and 3 replica shards are created. In total 6 shards.
  For 5 primary and 3 replica, in total 20 shards.

- Replica: We create replica of shard in another node. When shard is updated, its replica is synchronized. Write requests go to just shard, read requests go to shards or replicas.

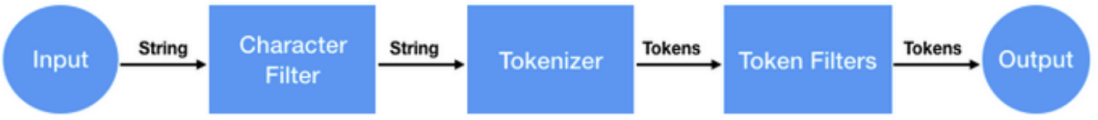


APIs

- Cat API: information such as index, cluster, node, shard can be displayed.
- Cluster API: Cluster related information.
- Index API: It is responsible for works such as create, delete, monitoring, mapping, settings on the index.
- Document API: It allows us to perform CRUD operations on documents.
- Search API: Used to search. URI Query and Query DSL

Analyzer

ElasticSearch'e yolladığımız dökümanlar performans ve istediğimiz şekilde çalışmasını sağlamak için indexlenmeden önce bir dizi işlemden geçer. Bu işlemlerin bütünü analysis'tir. Bu işlemi yapan kısma ise Analyzer denir.



1. Character Filter: It is the process of converting certain characters to different characters in order to obtain more accurate search results.
   you & me -> you and me
2. Tokenizer: After the character filter is applied to the text, the relevant document needs to be parsed into word sets. With the Lucene inverted index structure, it processes the document by parsing it into words called tokens.
   E.g; If we use the standard tokenizer, it will parse it into tokens based on spaces.
   "I like ES" -> tokens: "I", "like", "ES"
3. Token Filter: It is the process of taking tokens and modifying them according to needs. In the lowercase token filter, all tokens are converted to lowercase.

4. Indexing Tokens: Tokens are passed through token filters and then placed in the inverted index.

Built-in Analyzers

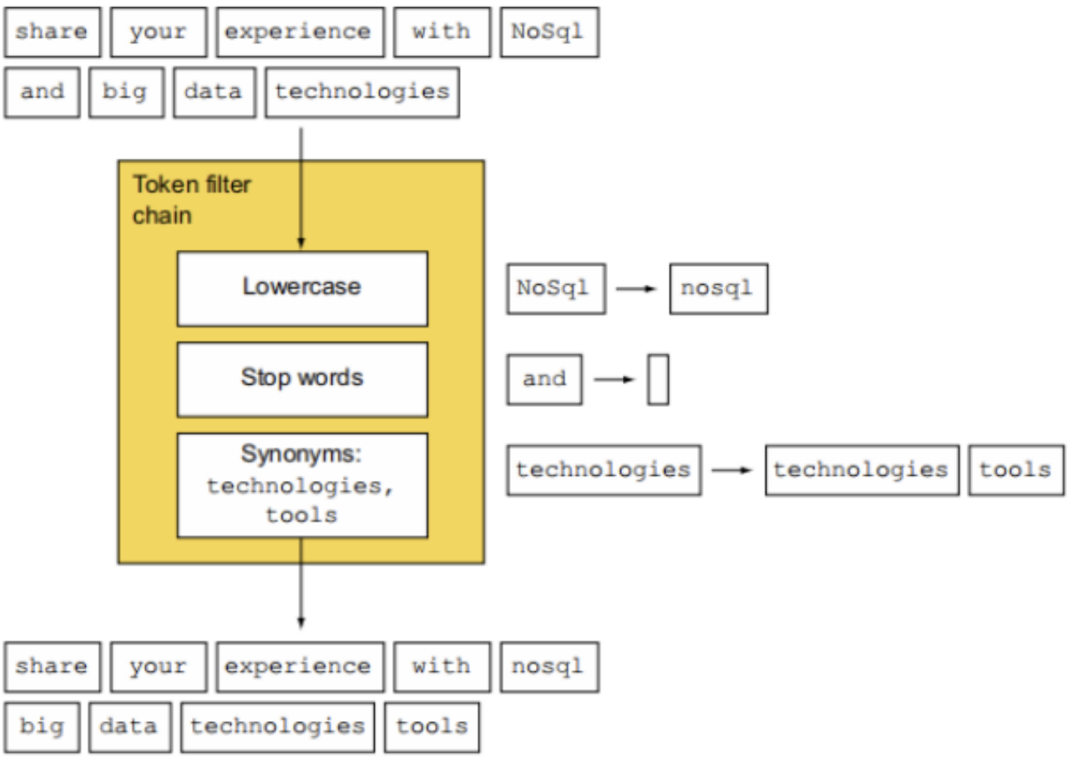
- Standart Analyzer, is default one. It includes standard lowercase token and stock token filter.
- Simple Analyzer,
- Whitespace Analyzer,
- ...

Built-in Tokenizers

- Standart Tokenizer, Gammer based tokenizer'dır. "I like, ES" -> "I", "like", "ES"
- Keyword Tokenizer,
- Letter Tokenizer,
- ...

Built-in Token Filters

- Standart
- Lowercase
- Length
- Stop
- Trim



Fuzzy Searching (i.e. similar words also work):

- link: https://hackernoon.com/how-to-use-fuzzy-query-matches-in-elasticsearch-dh1h3167
- Solution: We add *fuzziness* attribute to field that we need to search while querying.

```
"match": {
  "title.default": {
    "query":"eRçort",
    "fuzziness": "AUTO",
    "fuzzy_transpositions": "true"
  }
}
```

Auto-Complete suggestion

- Defined an example for auto-complete by using *completion suggester* in step 8. Auto-complete works with prefix. As documentation, if we need suggestion, then we should use suggester, not matching approach. Suggester mechanism is very fast. Here is the logic of suggester: