Learn Elasticsearch

By

Stanley Stephen

Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/
Email: s.stanley.mca@gmail.com

Topics:

In the new age data application serving up to date information is very critical for operational needs and provides a significant advantage for business in their decision-making process. Data services team serves that important purpose in the Data ART to provide Near real-time (NRT) data through RESTful APIs serving multiple business units across the organization like IBE, EK.COM, CNC, ResConnect, AsConnect, GC+, Amber, etc.

As they serve operational critical systems with high read/write calls - the stability, performance, scalability and the data accuracy needs for the services are always high.

About Elasticsearch

Elasticsearch is a highly scalable open-source search engine and document store powered by Lucene, an open source project from the Apache Foundation. It is suitable for situations that require fast analysis and discovery of information held in big datasets. This guidance looks at some key aspects to consider when designing an Elasticsearch cluster:

- Free, open-source, document (JSON) oriented search and analytics engine built on top of Apache Lucene
- Developed in Java (cross-platform)
- Distributed (cloud-based)
- Scalable and highly available
- Java API + RESTful HTTP/JSON API
- Used for full-text search, structured search, analytics, or all three in combination
- Near Real-Time searching – slight latency (~ 1 sec) from the time you index a document until the time it becomes searchable

What is an Elasticsearch Used For?

- Fulltext search engine – Elastic is primarily used where there is lots of text and we want to search any data for the best match with a specific phrase.
- JSON Document Storage – A JSON object with some data. It's the basic information unit in ES. The document is a basic information unit that can be indexed.
- Data Aggregation – The Elastic aggregation's framework helps provide aggregated data based on a search query. It is based on simple building blocks called aggregations that can be composed in order to build complex summaries of the data. An aggregation can be seen as a unit-of-work that builds analytic information over a set of documents. The context of the execution defines what this document set is (e.g. a top-level aggregation executes within the context of the executed query/filters of the search request).
- Geo-Search – Elasticsearch can be used to geo-localized any product. For example, the search query: 'all the restaurants that serve pizza within 30 minutes' can use Elasticsearch to display information of the relevant pizzerias instantly.
- Auto-Suggest – It allows the user to start typing a few characters and receive a list of suggested queries as they type.
- Auto-Complete – Elasticsearch database helps in autocompleting the search query by completing a search box on partially-typed words, based on the previous searches.
- Metrics & Analytics – Elasticsearch analyzes a ton of dashboards consisting of several emails, logs, databases, and syslogs, to help businesses make sense of their data and provide actionable insights.

- Product Search – Elasticsearch is used to facilitate faster product search using properties and name (textual search and structured data).

Few Things on Elastic

Distributed and highly scalable

Elasticsearch was built to scale and its distributed architecture allows you to scale Elasticsearch to a lot of servers and accommodate petabytes of data. Elasticsearch automatically distributes your data and query load across all of the available nodes in the cluster. It also take care of how to balance multi-node clusters to provide scale and high availability.

Complimentary plugins and tools

Elasticsearch is integrated with Kibana, a well-known visualisation and reporting tool, and also provides integration with tools like Beats and Logstash. You can add functionality to the applications with ES plugins.

Easy, Fast & Direct access

Ingest the de-normalize data as Elastic do not allow joins or sub-queries. Full-text searches are super-fast as the document storage is in close contiguity to the corresponding metadata. This significantly brings down the number of data reads, and Elastic limits the index growth rate by keeping it compressed. Elasticsearch is the best suitable for near real-time use cases like application monitoring and anomaly detection.

Schema free

Elasticsearch does not require some definitions such as index, type, and field type before the indexing process, and when an object is indexed later with a new property, it will automatically be added to the mapping definitions.

Document oriented (JSON)

Elasticsearch uses JSON -JavaScript Object Notation, as the serialisation format for documents. JSON serialisation is supported by various programming languages, and has become the standard format used by the NoSQL movement. It is simple, concise, and easy to read.

What is segment?

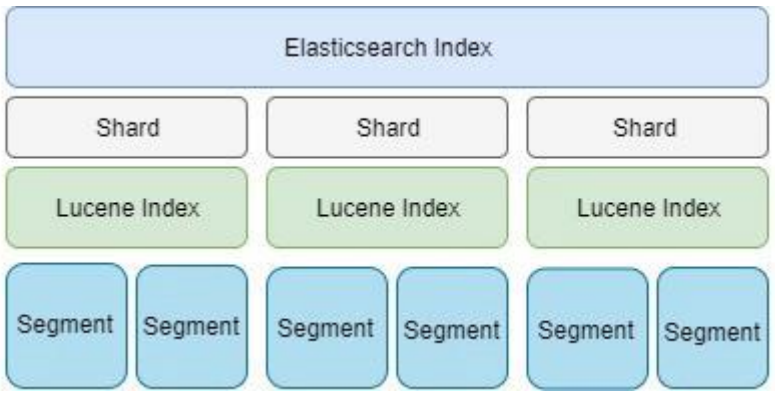
Where data in lucene is flused, merged and stored on disc.

Lucene segments

Each Elasticsearch index is divided into multiple shards. Shards are both logical and physical division of an index. Each shard is a Lucene index. The Lucene index is divided into smaller files called segments. A segment is a small Lucene index which is immutable. Lucene does a search in all segments sequentially.

Merge segments

When a document is being indexed, Lucene creates a new segment and writes it. Note that, the Lucene can also create more segments when the indexing throughput is important. From time to time, Lucene merges smaller segments into a larger one. The merge can also be triggered manually from the Elasticsearch API as well. During a merge process, Lucene takes 2 segments, and moves the content into a third, new one. Then the old segments are deleted from the disk. This means that Lucene needs enough free space on the disk to create a segment the size of both segments it needs to merge.



Meta Fields

Each document has associated by certain metadata fields by an elastic such as the _index, mapping _type, and _id meta-fields. The behaviour of some of these meta-fields could be custom when a mapping type was created.

Identity meta-fields

_index: The index to which the document belongs.
_uid: A composite field consisting of the _type and the _id.
_type: The document’s mapping type.
_id: The document’s ID.

Document source meta-fields

_source: The original JSON representing the body of the document.
_size:The size of the _source field in bytes, provided by the mapper-size plugin.

Indexing meta-fields

_all: A catch-all field that indexes the values of all other fields.
_field_names: All fields in the document that contain non-null values.
_timestamp: A timestamp associated with the document, either specified manually or auto-generated.
_ttl: How long a document should live before it is automatically deleted?

Routing meta-fields

_parent: Used to create a parent-child relationship between two mapping types.
_routing: A custom routing value that routes a document to a particular shard.

Naming Standards

Cluster name

Elastic cluster name across environments should be unique and it should contain alphanumeric characters or hyphens, should start with a letter, and cannot end with a hyphen or contain two consecutive hyphens

Elastic cluster name should follow standards like <business>-<sub/domain name>-<environment> e.g, airline-edh-dev, airline-edh-stg, airline-edh-prod and should not keep default name.

Indices name

Ensure that, below Elastic search indices have the following naming restrictions:

All letters must be lowercase.

Index names cannot begin with _ or -.

Index names can't contain spaces, commas, :, ", *, +, /, \, |, ?, #, >, or <.

Don't include sensitive information in index, type, or document ID names.

Keep the index name in the date format value like year-month-day/week (yyyy-mm-dd) based on indexed values like time bound documents (e.g., keeping all production servers log files, like log-{apptype}-{appname}-2020-05-15 e.g., log-appserver-edhservice-2020-05-01. Some case advised to keep only year or year and month alone in the index name based on the value .

The indices name should follow on source feed or way of getting ingested. e.g., idx_order_speed_pnr-(yyyy-mm-dd) here idx refer the type index, order refer the feed or data getting ingested and speed_pnr refer the data being ingested in a near real-time. In case of batch, it will be idx_order_batch-<yyyy-mm-dd>

Aliases name

Alias name follows similar to index naming standards and suggest to use with prefix als_ instead of idx_ to . e.g., als_flightmanifest_speed which refer an alias name to all selected flight manifest indices.

Type name

Type name follows similar to index naming standards and suggest to use with prefix typ_ instead of idx_ to . e.g., typ-flight" which refer an alias name to all selected flight index.

Mappings

An Elastic mapping definition has the Meta-fields and Fields or properties and this mapping defines how a document is indexed and how its fields are indexed and stored by Lucene.For instance,

- Defines string fields that should be treated as full text fields.
- Defines the fields that contain numbers, dates, or geolocations.
- Date fields format.
- Custom rules to control the mapping for dynamically added fields.

Creating a new index,named my-customer with an explicit mapping

```
curl -X PUT "localhost:9200/my-customer?pretty" -H 'Content-Type: application/json'
-d'
{
  "mappings": {
     "pro perties": {
    "firstname": { "type": "text" },
    "lastname": { "type": "text" },
    "email": { "type": "keyword" },
    "dob": { "type": "date" }
    }
  }
}
'
```

View mapping

Use GET API to get the existing index mapping. For example, 'GET edh-booking/_mapping' used to get the mapping of index name "edh-booking"

Elastic Limitations

Does not support SQL like joins but provides parent-child and nested to handle relations.

Transactions and updates are expensive in a distributed system.

Does not support transaction rollbacks but it offers version-based control to make sure the update is happening on the latest version of the document.

The data you index is searchable only after 1 sec.

1. Elastic Architecture
2. Elastic - Basic Operations
3. Elastic - REST API
4. Elastic Data modelling
5. Elastic Best Practices

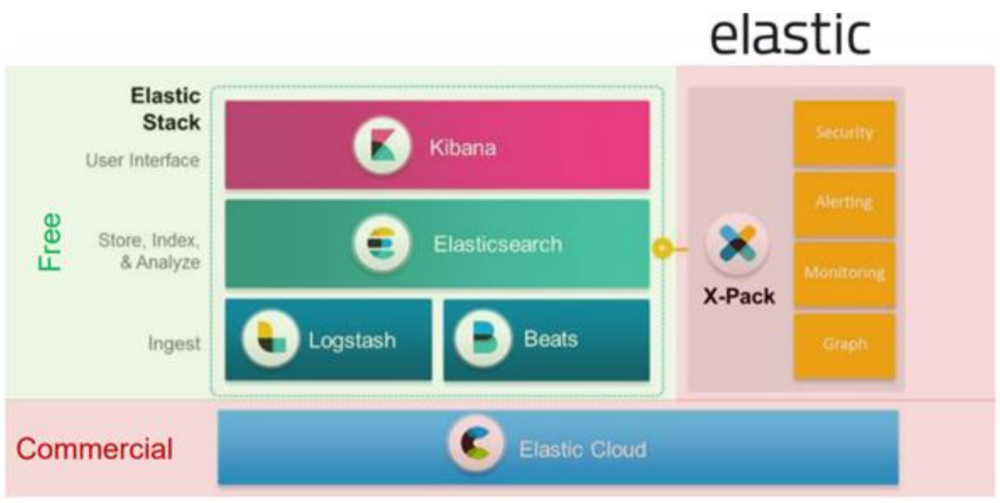
Reference

Topics	**Links**
Elastic Documentation	https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
Elastic learning portal	https://www.tutorialspoint.com/elasticsearch/index.htm

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Elasticsearch_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

1. Elastic Architecture

Elastic Stack



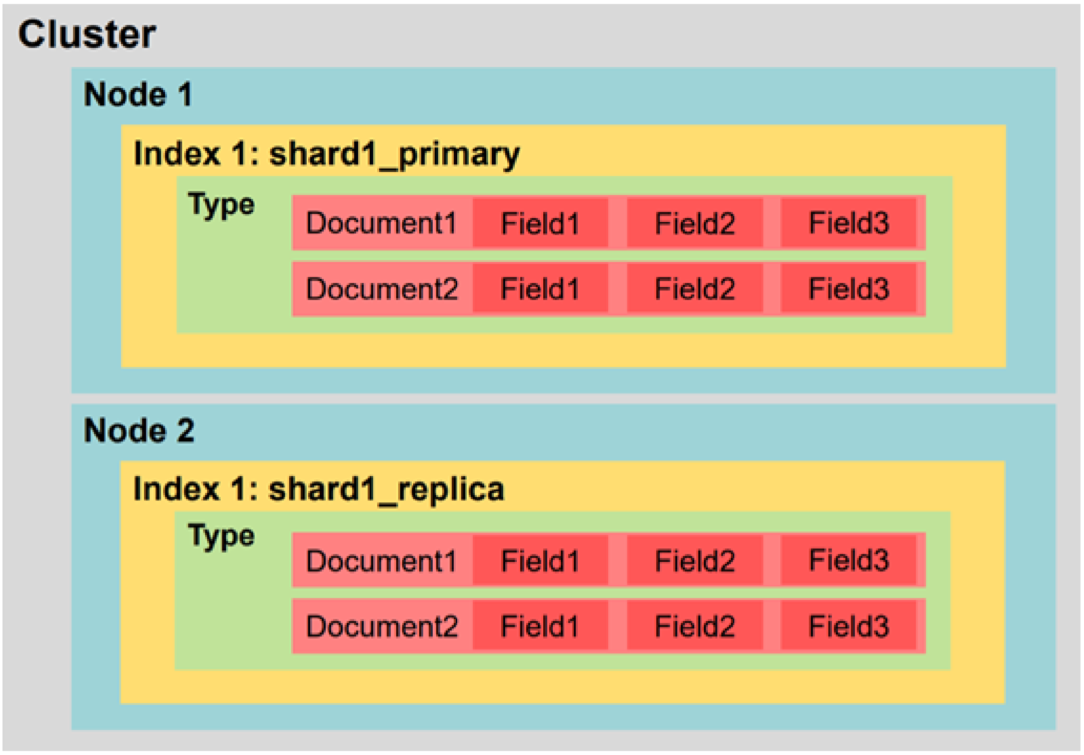
Elasticsearch is a search and analytics engine.

- Kibana – Window into elastic stack. Enables visual exploration and real-time analysis of data in Elasticsearch
- Logstash – Central dataflow engine for gathering, enriching, and unifying all input data from various sources regardless of format or schema and sends it to stash
- Beats – Forward host-based metrics and any data to Elasticsearch or Logstash
- X-Pack – A single extension for Security for Elastic Stack, Alerting and notifications for the Elastic Stack, Monitoring for the Elastic Stack, Real-time Graph Analytics to enable new use cases for Elastic Stack
- Cloud – Hosted Elasticsearch & Kibana on AWS and GCP

Elastic Building Blocks

Term	Description
Cluster	A group of nodes that holds all data
Node	A machine that you start an instance of Elasticsearch, you are starting a *node*. This node that holds some data and participates on the cluster's indexing and querying
Index	A collection of documents that have similar characteristics. An index can be divided into multiple pieces called shards.
Shard &	Primary shard and replica(s) may exist. Each Elasticsearch shard is a

Replica	Lucene index with an upper limit of docs. Sharding is important for 2 reasons: • Horizontal splitting/scaling of content volume • Allows to distribute and parallelize operations across shards (potentially on multiple nodes) thus increasing performance/throughput Shards can also be used by making multiple copies of an index into replicas shards, which could be useful to provide high availability.
Documen t	Basic unit of information that can be indexed. Expressed in JSON. Contains fields in key/value pair(s)



Node in the cluster does the one or more of the following roles in the cluster

- **Data node** which can hold one or more shards that contain index data. A node that has data set to true (default). Data nodes hold data and perform data related operations such as CRUD, search, and aggregations.
- **Master node** that does not hold index data but that performs cluster management operations, such as maintaining and distributing routing information around the cluster (the list of which nodes contain which shards), determining which nodes are available, relocating shards as nodes appear and disappear, and coordinating recovery after node failure. Multiple nodes can be configured as masters, but only one will actually be elected to perform the master functions. If this node fails,

another election takes place and one of the other eligible master nodes will be elected and take over.

- **Ingest node :** A node that has node.ingest set to true (default). Ingest nodes are able to apply an ingest pipeline to a document in order to transform and enrich the document before indexing. With a heavy ingest load, it makes sense to use dedicated ingest nodes and to mark the master and data nodes as node.ingest: false.
- **Coordinate node** : A node that does not hold index data but that handles incoming requests made by client applications to the appropriate data node and this nodes all properties (data, ingest, master) set as false.
- **Tribe node** : A tribe node, configured via the tribe.* settings, is a special type of coordinating only node that can connect to multiple clusters and perform search and other operations across all connected clusters.

By default, Elasticsearch nodes perform all three roles (to enable you to build a complete working cluster on a single machine for development and proof-of-concept purposes), but you can change their operations through the *node.data* and *node.master* settings in the *elasticsearch.yml* file, as follows:

```
# Configuration for a data node

node.data: true

# Configuration for a master node

node.master: true

# Configuration for a ingest node

node.ingest: true
```

2. Elastic - Basic Operations

Basic Operations

This section focuses on performing basic operation using Elastic REST API.

Create an Index

The below script creates an index called "my-customer" manually with all default settings in Elasticsearch.

```
curl -X PUT "http://<hotname>:<port>/my-customer"
```

Delete an Index

Below script delete an index called "my-customer"

```
curl -X DELETE "http://<hotname>:<port>/my-customer"
```

Insert a JSON to an Index

The index "PUT" API adds a typed JSON document in a specific index, making it searchable. The following example inserts the JSON document into the "edh-genome" index, under a type called "customer" with an id of 1:

```
curl -X PUT "<hostname>:<port>/my-customer/customer/1?pretty" -H
'Content-Type: application/json' -d'
{
"user": "Mark",

"organization": "DNATA Tours and Adventures",

"created_date": "2018-09-03T11:41:00"

}
'
```

Query a JSON Document

The get API allows us to query a JSON document from the index based on its id. Below provided example gets a JSON document from an index called my-customer, under a type called customer, with id valued 1:

```
curl -X GET "<hostname>:<port>/my-customer/customer/1?pretty"
```

Update a JSON Document

The JSON document stored in Elasticsearch can be upsert or updated partially. Below script updates the name value to "Steven"

```
curl -X POST "<hostname>:<port>/my-customer/customer/1/_update?pretty" -
H 'Content-Type: application/json' -d'
{
"doc" : {
"name" : "Steven"
}
}
'
```

Delete Document

The delete API allows us to delete a typed JSON document from a specific index based on its id. Below script deletes the JSON document from an index called my-customer, under a type called customer, with id 1:

```
curl -X DELETE "<hostname>:<port>/my-customer/customer/1?pretty"
```

Index Aliases

APIs in Elasticsearch accept an index name when working against a specific index, and several indices when applicable. The index aliases API allows aliasing an index with a name, with all APIs automatically converting the alias name to the actual index name. An alias can also be mapped to more than one index, and when specifying it, the alias will automatically expand to the aliased indices. An alias can also be associated with a filter that will automatically be applied when searching and routing values. An alias cannot have the same name as an index.

Here is a sample of associating the alias name, genome with index my-customer:

```
curl -X PUT "<hostname>:<port>/my-customer/_alias/genome?pretty"
```

Analyzer

What is an analyzer ?

Analyzer is a combination of tokenizer and filters that can be applied to any field for analyzing in Elasticsearch.

The content stored in any index is analyzed by the analysis module. If no analyzer is defined, then by default the built in analyzers, token, filters and tokenizers get registered with analysis module. Elastic supports custom analyser as well based on the user custom requirements.

A standard analyzer which is used when no other analyzer is specified. It will analyze the sentence based on the grammar and produce words used in the sentence.

```
POST _analyze
{
"analyzer": "standard",
"text": "This is an elastic playbook"
}
```

Elastic response as follows

```
{
	"tokens": [
	{
	"token": "this",
	"start_offset": 0,
	"end_offset": 4,
	"position": 0
	},
	{
	"token": "is",
	"start_offset": 5,
	"end_offset": 7,
	"position": 1
	},
	{
	"token": "an",
	"start_offset": 8,
	"end_offset": 10,
	"position": 2
	},
	{
	"token": "elastic",
	"start_offset": 11,
	"end_offset": 18,
	"position": 3
	},
	{
	"token": "playbook",
	"start_offset": 19,
	"end_offset": 27,
	"position": 4
	}
	]
}
```

What is a tokenizer ?

Receives a stream of characters and it breaks into individual tokens or terms based on a certain rule. For example, a whitespace tokenizer breaks text into tokens whenever it sees any whitespace. It would convert the text "Elasticsearch is based on the Lucene library" into the terms [Elasticsearch, is, based,on, the, lucene, library]. Elasticsearch has a number of built in tokenizers which can be used to build custom analyzers.

What is tokenfilters ?

Token filters operate on tokens produced from tokenizers and modify the tokens accordingly. For example, Lowercase filter : Lower case filter takes in any token and converts it to lowercase token.

```
Input => "PlayBook"
Output => "playbook"
```

What is charfilters ?

A character filter receives the original text as a stream of characters and can transform the stream by adding, removing, or changing characters. For example the character filter accepts a map of key and values and then replaces the respective key with value appearing in the original text.

The below filter changes the text "Have a good day :) " into [Have,a, good, day,happy]"

```
char_filter":{
"replace_special_characters": {
   "type":"mapping",
    "mappings":[
     ":) => happy",
     ":( => sad",
     "& => and"
    ]
   }
 }
```

3. Elastic - REST API

The Elasticsearch REST APIs that are exposed using JSON over HTTP and the Interaction with ElasticSearch happens through these REST API and the output is JSON. These APIs are used by the UI components and can be called directly to configure and access Elasticsearch features. Let us look at few of those APIs in this section.

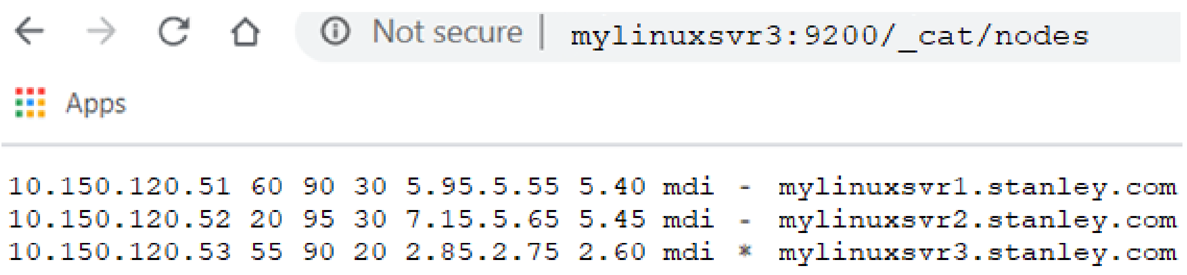
/_cat APIs

The Elasticsearch's cat APIs feature is helps in taking care of giving an easier to read and comprehend printing format of the results. There are various parameters used in cat API which serves different purpose.

Verbose

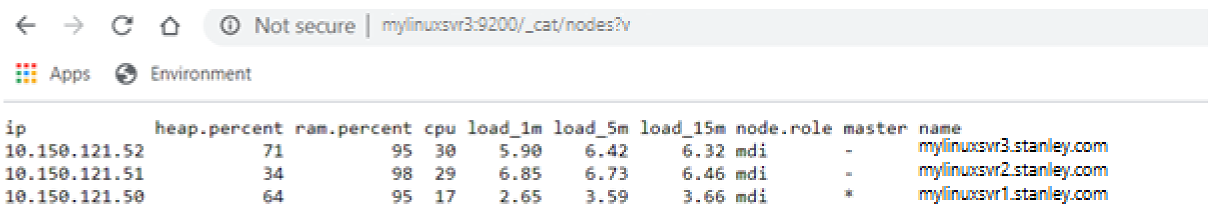
The term V makes the output verbose. Let us look the below example

Without verbose (<clustername:portno>/_cat/nodes)



```
10.150.120.51 60 90 30 5.95.5.55 5.40 mdi - mylinuxsvr1.stanley.com
10.150.120.52 20 95 30 7.15.5.65 5.45 mdi - mylinuxsvr2.stanley.com
10.150.120.53 55 90 20 2.85.2.75 2.60 mdi * mylinuxsvr3.stanley.com
```

With verbose (<clustername:portno>/_cat/nodes?v)

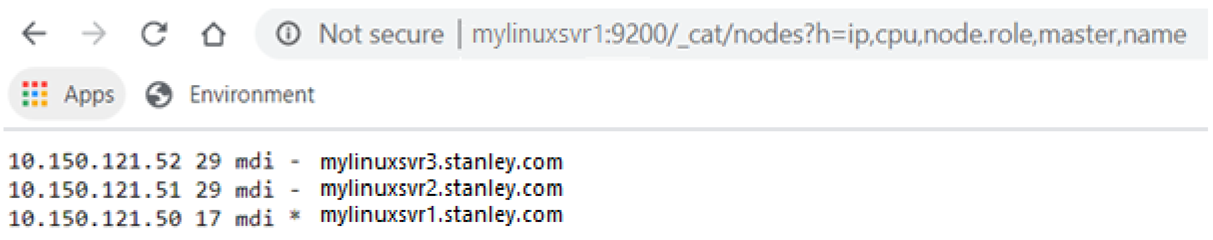


```
ip            heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
10.150.121.52           71          95  30    5.90    6.42     6.32 mdi       -      mylinuxsvr3.stanley.com
10.150.121.51           34          98  29    6.85    6.73     6.46 mdi       -      mylinuxsvr2.stanley.com
10.150.121.50           64          95  17    2.65    3.59     3.66 mdi       *      mylinuxsvr1.stanley.com
```

Headers

This header api (h) provides the values for the given column names.

Column names need to be provided after "h=" and refer the below example "ip,cpu,node.role,master,name"


Not secure | mylinuxsvr1:9200/_cat/nodes?h=ip,cpu,node.role,master,name
Apps
Environment

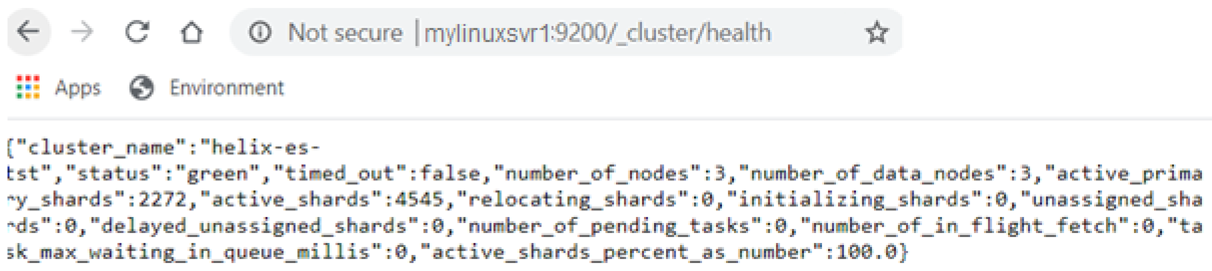
```
10.150.121.52 29 mdi - mylinuxsvr3.stanley.com
10.150.121.51 29 mdi - mylinuxsvr2.stanley.com
10.150.121.50 17 mdi * mylinuxsvr1.stanley.com
```

Refer the list of cat APIs are in cat API.

Cluster APIs

Elasticsearch proivdes a large number of cluster-specific API operations that allow us to manage and monitor the Elasticsearch cluster. These APIs allows to define which Elasticsearch node to call using either the internal node ID, its name or its address.

For example the cluster health API used to provide the details on the cluster and gauge its health. The Elastic provides quite number of clusters API.



```
{"cluster_name":"helix-es-
tst","status":"green","timed_out":false,"number_of_nodes":3,"number_of_data_nodes":3,"active_prima
-y_shards":2272,"active_shards":4545,"relocating_shards":0,"initializing_shards":0,"unassigned_sha
-ds":0,"delayed_unassigned_shards":0,"number_of_pending_tasks":0,"number_of_in_flight_fetch":0,"ta
sk_max_waiting_in_queue_millis":0,"active_shards_percent_as_number":100.0}
```

Document API

There are two of document APIs **Single document APIs & Multi-document APIs** are available in Elastic.

Single Document APIs

The single document APIs such as Index API, Get API, Delete API and Update APIs are used to perform CRUD operations on the index or alias provided.

Multi-Document APIs

This multi-document APIs does the operations on the multiple JSON documents based on the query or filter provided.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/Elasticsearch_by_Stanley.pdf
Email: s.stanley.mca@gmail.com

Bulk API

Performs multiple indexing or delete operations in a single API call. This reduces overhead and can greatly increase indexing speed.

4. Elastic Data modelling

Data is modeled in documents in Elasticsearch which means a single item irrespective of whether it is an entity like a company, country, or product has to be modeled as a single document. A document can contain any number of fields and values associated with it. This information is modeled around the JSON format that helps us to express the behavior of our data using arrays, objects, or simple values.

NOTE : Elasticsearch is schemaless, which means that you can index a document to Elasticsearch before you create its index or mapping. Elasticsearch guesses the best type for each field value.

Modelling Basics:

- Elasticsearch basic principle of data modelling is to reduce the number of shards the elasticsearch looking for the result. shard is nothing but the next bottom level of an index.
- One or more shards forms an index. Allocate the right number of shards will gives you certain amount of performance boost. So avoid allocate many numbers of shards for future scalability because it may affect the current search and indexing time.
- If it is social api data or log, go with time-based index. It will give you lot of flexibility while storing and retrieving.
- If your data are user based, and your search queries will be based on a single user, then it is better to create user based index. But as i said earlier if you need to search on all user data then this modeling will not work. In this case find the next level entity and make it as index point, so that you can search by user id.

5. Elastic Best Practices

Best practices for data aggregation and query performance

The following points summarize tips for maximizing the performance of Elasticsearch queries:

- Avoid wild card queries wherever possible. Wild card query takes more memory which will eventually affect the cluster performance.
- Don't run aggregation on nGram analyzed field. It may bring down the whole cluster.
- If the same field is subject to full-text searching and exact matching, then consider storing the data for the field in analyzed and non-analyzed forms. Perform full-text searches against the analyzed field, and exact matches against the non-analyzed field.
- Only return the data necessary. If you have large documents, but an application only requires information held in a subset of the fields, then return this subset from queries rather than entire documents. This strategy can reduce the network bandwidth requirements of the cluster.
- Wherever possible, use filters instead of queries when searching for data. A filter simply determines whether a document matches a given criterion whereas a query also calculates how close a match a document is (scoring). Internally, the values generated by a filter are stored as a bitmap indicating match/no-match for each document, and they can be cached by Elasticsearch. If the same filter criterion occurs subsequently, the bitmap can be retrieved from cache and used to quickly fetch the matching documents. For more information, see Internal Filter Operation.
- Use *bool*filters for performing static comparisons, and only use *and*, *or*, and *not* filters for dynamically calculated filters, such as those that involve scripting or the *geo-**
- If a query combines *bool*filters with *and*, *or*, or *not* with *geo-** filters, place the *and*/*or*/*not* *geo-** filters last so that they operate on the smallest data set possible.
- Use keyword fields are only searchable by their exact value like IDs, email addresses, hostnames, status codes, zip codes or tags for filtering, for sorting, and for aggregations.
- Use Elastic scroll API s when there is a need to query large data set like over 10K and also it is very necessary to clear the Scroll API Context and optimal the scroll size used in the query.

  Similarly, use a *post_filter* to run expensive filter operations. These filters will be performed last.

- Use aggregations rather than facets. Avoid calculating aggregates that are analyzed or that have many possible values.
- Use the *cardinality* aggregation in preference to the *value_count* aggregation unless your application requires an exact count of matching items. An exact count can become quickly outdated, and many applications only require a reasonable approximation.
- Avoid scripting. Scripts in queries and filters can be expensive and the results are not cached. Long-running scripts can consume search threads indefinitely, causing subsequent requests to be queued. If the queue fills up, further requests will be rejected.

PROS and CONS with Elasticsearch

Pros	**Cons**
Excellent full-text search engine	Not an ACID compliant system
Lucene based open source search library	Data type auto conversion happens if not defined and need to be re-indexed
As the name suggests it stores, index and searching faster	Support only JSON i.e., Not a multi-language support
Optimistic version control available	Number of shards cannot be changed once the index created
Schema-less, Scale, Documentation etc.	As distributed in nature can have negative effects on data consistency