



Operate an Elasticsearch cluster

David THIBAU – 2022

david.thibau@gmail.com



Agenda

- Introduction

- ELK proposition and its use cases
- Distributions and installation
- ELS API conventions, The DevConsole of Kibana
- Core concepts of clustering

- Indexing documents

- What is a document?
- What is an index ?
- Routing and distributed search
- Alternatives for data ingestion

- Search features

- Search lite
- DSL syntax

- Sizing architecture

- Different Architectures
- Production node
- Specialization of elastic nodes
- Dimensionning indices and sharding
- Index lifecycle policies

- Monitoring and Operations

- Logs
- Low level Cluster API
- Monitoring with beats
- Reindexation, Snapshot and Restore
- Resizing and restarting the cluster

- Security

- Automatic security with ELS 8.x
- Manual configuration
- User authentication and authorization
- Audit logging, IP filtering



Introduction

ELK proposition and use cases

Distributions and installation

ELS API conventions, The DevConsole
of Kibana

Core concepts of clustering



Introduction

The elastic company provides the ***ElasticStack suite*** in 2 editions (OpenSource and Enterprise)

The suite addresses 2 aspects, which can be applied in a BigData environment :

- **Near Real-time search**

Permanently indexed data is available for search, the latency due to indexing is negligible (Near Real Time)

- **Near Real-time data analysis.**

Data is aggregated in real time to provide visualizations and dashboards to extract insights

The core component of this stack is ***ElasticSearch***



ElasticSearch

ElasticSearch is a server offering a REST API which allows :

- To store and index data
(Office documents, tweets, log files, performance metrics, ...)
- To perform search queries (structured, full-text, natural language, geolocation)
- Aggregate data in order to calculate metrics



Features

- ✓ Massively **distributed and scalable** architecture in volume and load
=> Big Data, remarkable performance
- ✓ **High availability** through replication
- ✓ **Muti-tenancy** or multi-index. The document base contains several indexes whose life cycles are completely independent
- ✓ Based on the **Lucene** reference library
=> Full-text search, consideration of languages, natural language, etc.
- ✓ **Flexibility** : Structured storage of documents but without prior schema, possibility of adding fields to an existing schema
- ✓ Very complete **RESTFul API**
- ✓ **Open Source**



Apache Lucene

- Apache project started in 2000, used in many products
- The "low layer" of ELS: a Java library for writing and searching **index files**:
 - An index contains **documents**
 - A document contains **fields**
 - A field may consist of **terms**





ElasticSearch added values

Lucene provides all the needed features for a search engine :

- Sorting by relevance score,
- Natural language, term or phrase queries
- Highlight, facets, auto-suggestion, phonetics, typo

ELS makes using lucene easier

- Via its REST API
- Via its Kibana Client
- Built-in analyzers customized for each language

ELS makes using lucene more efficient and fault-tolerant

- Via clustering

ElasticSearch add some features :

- Mainly aggregations



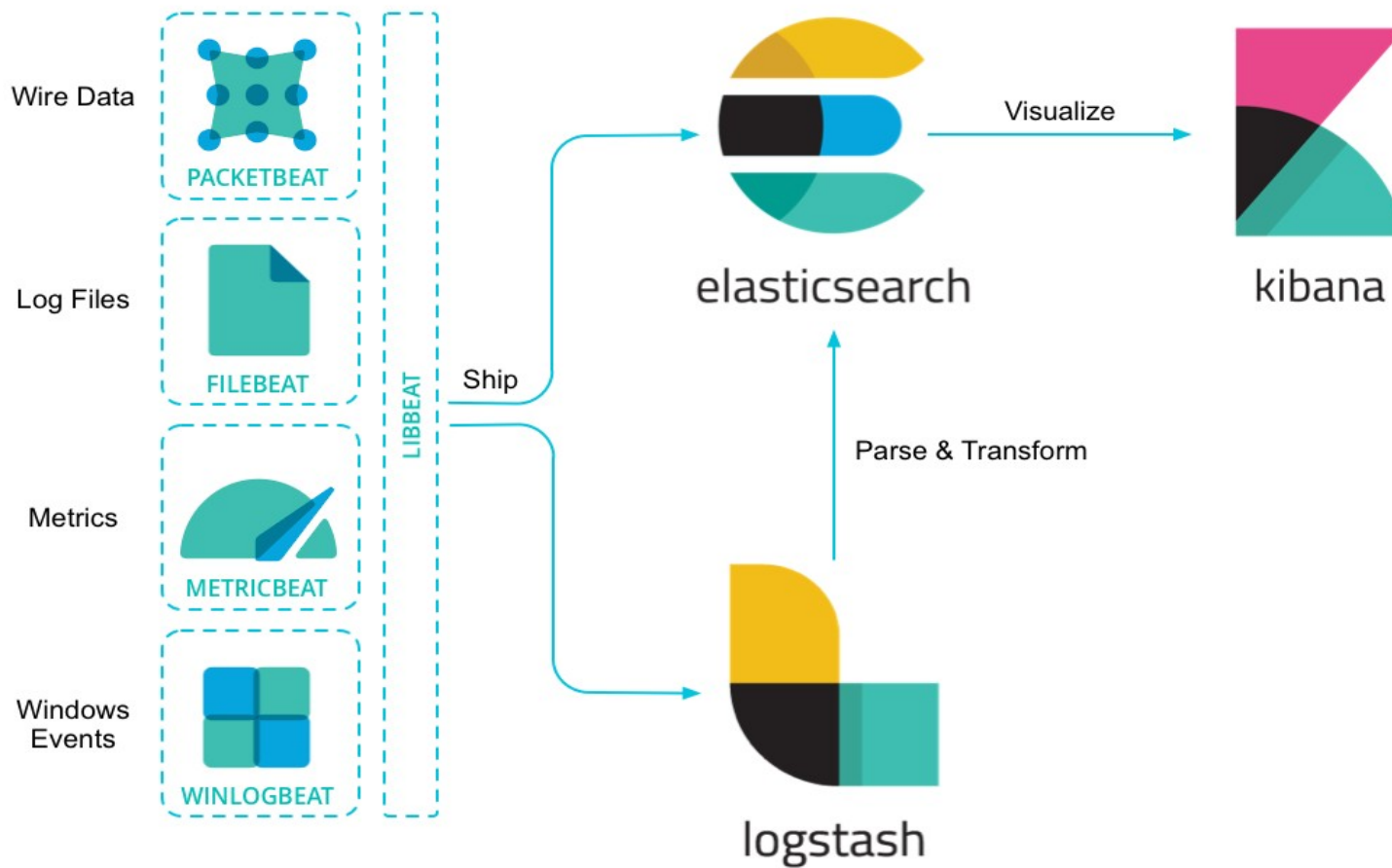
Elastic Stack

ElasticSearch is the core component of a suite of tools dedicated to Real Time Analysis in the context of BigData : ***ElasticStack***

The main components of this suite are :

- ***Logstash / Beats*** : Ingestion of data. Extract and transform before ingestion
- ***Elastic Search*** : Storage, Indexing and Search
- ***Kibana*** : Data visualization, Administration and Developer tool

Architecture



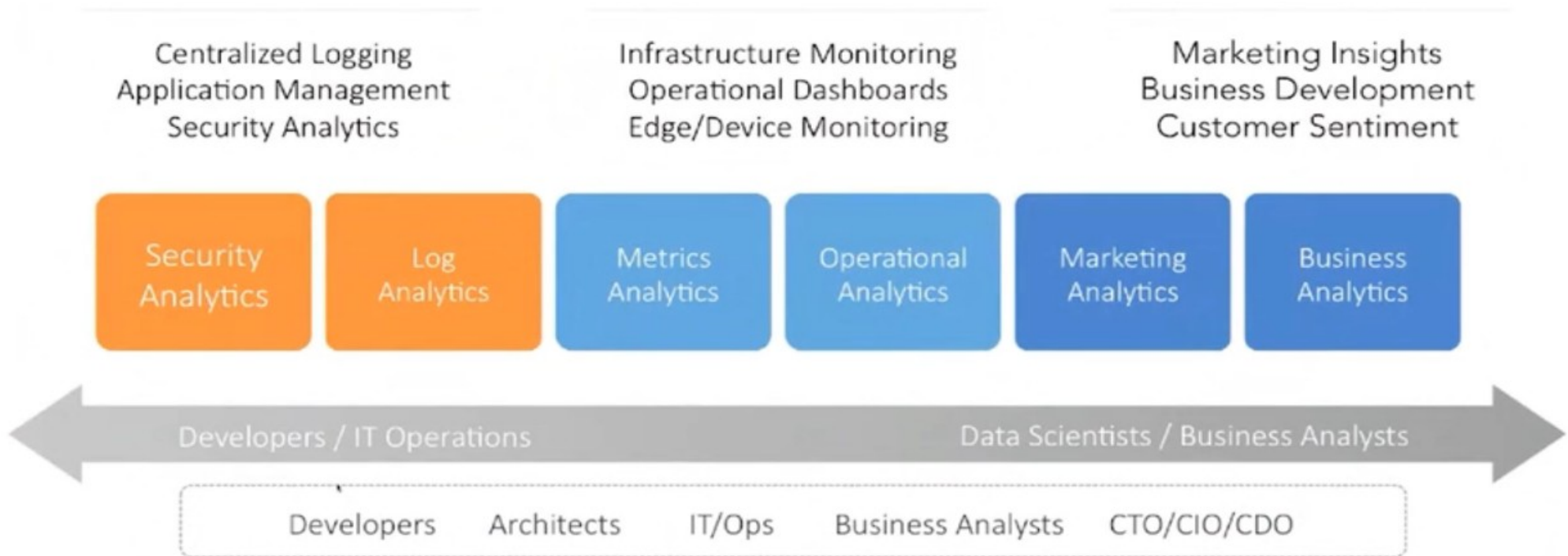


Related Offers

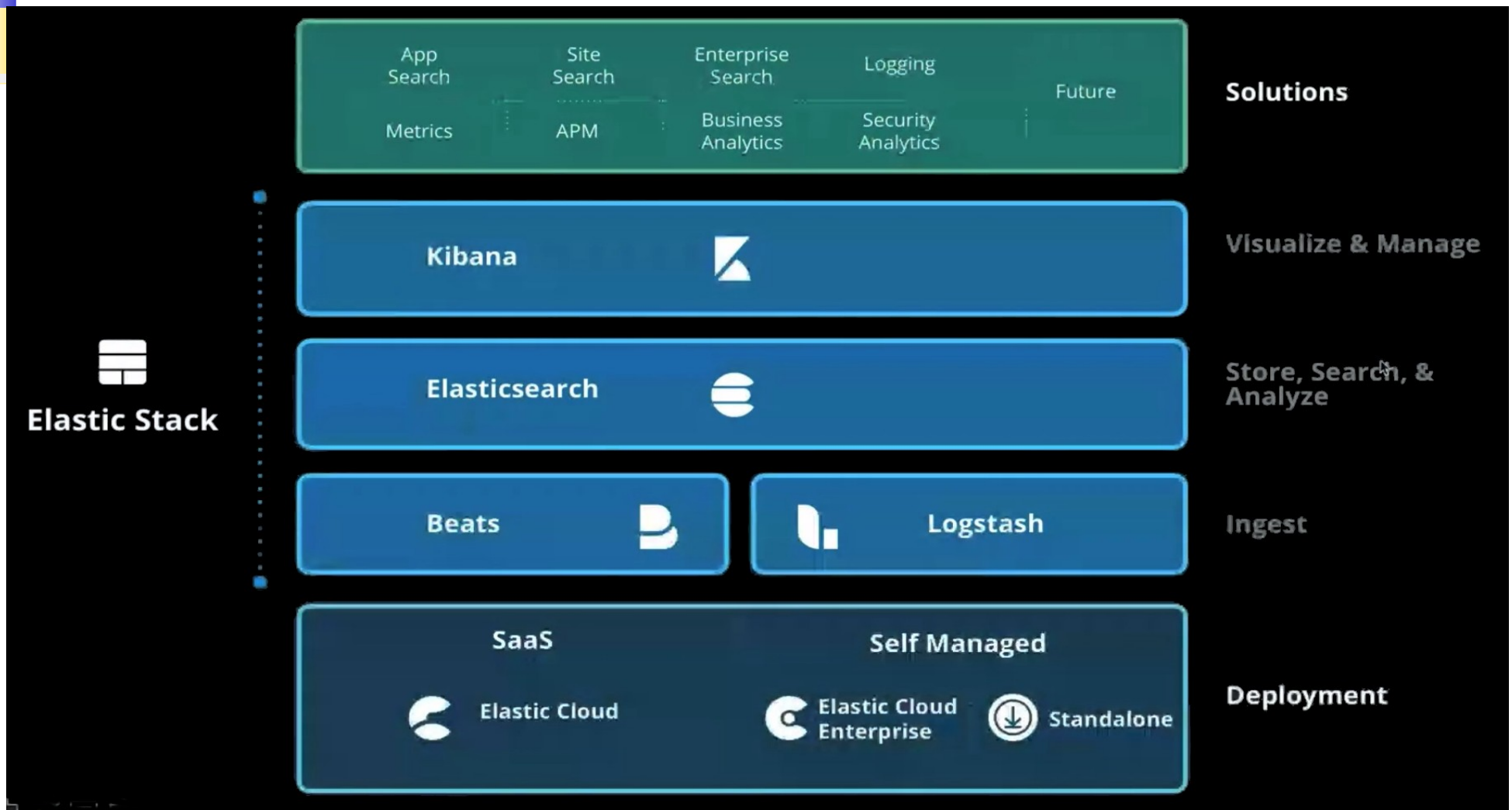
The Elastic company also offers commercial features :

- ***X-Pack*** :
 - Security, Monitoring (open source since release 8.x)
 - Alerts, Machine Learning
 - Application Performance Management
 - Reporting and scheduling
- ***Elastic Cloud*** : Delegate operations of your cluster to the elastic company or Deploy elastic search in your private cloud
- ***Solutions out-of the box*** : Enterprise search, Elastic Observability, Elastic Security

Elastic Use cases



In summary





Introduction

ELK proposition and use cases

Distributions and installation

ELS API conventions, The DevConsole
of Kibana

Core concepts of clustering



Releases

Releases of *ElasticSearch* are available for download in different formats :

- Archive ZIP, TAR,
- Packages DEB or RPM
- Docker Images

Versions

- 8.0.0 : February 2022
- 7.x : Avril 2019 - Continuing
- 6.x : November 2017 to December 2019
- 5.0.0 : October 2016
- 2.4.1 : September 2016
- 2.4.0 : August 2016



Getting started

1. Download and Unzip the archive

2. Start ELS

`bin/elasticsearch`

3. Note the *elastic* user password and access to ELS

`curl https://localhost:9200/`



Installation for production

To install a self-managed Elasticsearch, different options :

- Run as a service on a Linux, MacOS, Windows machine
- Run in a docker container
- Run with Kubernetes with Elastic Cloud on Kubernetes



Configuration

Several configuration files

- ***elasticsearch.yml*** : The property of the ELS node
- ***jvm.options*** : JVM Options
- ***log4j2.properties*** : Logging configuration

The default configuration allows to start quickly but to go to production it is necessary to modify some parameters. (see further)



elasticsearch.yml

ELS add some extra features to the
YAML format :

- Environment variables d'environnement can be used with *`${ENV_VAR}`*
- Some properties may be prompted at startup:
`${prompt.text}` , *`${prompt.secret}`*
- Values can be overridden at startup with the -E option
`./elasticsearch -E node.name=David`



Node settings

Cluster and node settings can be :

- **Static** : Read once in *elasticsearch.yml* at startup
- **Dynamic** : You can update the setting with the *Update Settings API*.
With this API, you can make transient updates (since next restart) or persistent updates



Important configuration for production

Paths : (data and logs). It is better to specify directories outside the distribution

`path.data`, `path.logs`

Cluster and node names : Do not use default values

`cluster.name`, `node.name`

Network host : By default, bound to 127.0.0.1, use a private network address

`network.host`

Discovery and cluster formation settings

See further

JVM settings : Heap size, Heap dump path, GC logging, fatal error log:

You can use *jvm.options* file



Bootstrap checks

On startup ELS performs checks on the environment. If these checks fail :

- In development mode (IP listening to *localhost*), warnings appear in the log
- In production mode (IP with a public adress), ELS refuse to start

These checks include, among other things:

- Heap sizing to avoid resizing and swapping
- Limit on the number of file descriptors very high (65,536)
- Allow 2048 threads
- Virtual memory size and areas (for native Lucene code)
- Filter on system calls
- Checking on behavior during JVM error or OutOfMemory
- ...



Introduction

ELK proposition and use cases
Distributions and installation
The DevConsole of Kibana
Core concepts of clustering



ELS API

ELS provides a typical Restful API

API is divided into categories


- **document API** : CRUD on documents
- **index API** : CRUD on Indexes
- **search API (_search)** : Search for documents
- **Cluster API (_cluster)** : monitoring and update configuration
- **cat API (_cat)** : Tabbed response format for cluster management
- API **X-Pack** : Configuring X-Pack Features, licensing etc
- ...



Clients Alternatives

- Classical REST client:
curl, Postman, SOAPUI, ...
- Libraries provided by Elastic :
Java, Javascript, Python, Groovy, Php, .NET, Perl
- But, the most comfortable client is
Kibana's Dev Console

The Dev Console

 Search Elastic

≡

D

Dev Tools

ConsoleSearch ProfilerGrok DebuggerPainless LabBETA

HistorySettingsHelp

1 GET sessions/_search?size=1

2 {

3 "sort" : [

4 { "datebutsession" : { "order" : "asc" } }

5],

6 "query" : {

7 "bool" : {

8 "filter" : [

9 {

10 "term" : {

11 "référence.keyword" : "GSMC"

12 } ,

13 },

14 {

15 "range" : {

16 "datebutsession" : {

17 "gte" : "2022-08-28"

18 } ,

19 } ,

20 } ,

21]

22 } ,

23 }

24 }

200 - OK286 ms

1 {

2 "took" : 182,

3 "timed_out" : false,

4 "_shards" : {

5 "total" : 1,

6 "successful" : 1,

7 "skipped" : 0,

8 "failed" : 0

9 },

10 "hits" : {

11 "total" : {

12 "value" : 4,

13 "relation" : "eq"

14 },

15 "max_score" : null,

16 "hits" : [

17 {

18 "_index" : "sessions",

19 "_type" : "_doc",

20 "_id" : "sessions-221823A",

21 "_score" : null,

22 "_source" : {

23 "libellé" : "Scrum Master Certifiante - Niveau 1 (PSM1)",

24 "stageintra" : false,

25 "datefinsession" : "2022-09-01T22:00:00.000Z",

26 "lieusession" : "PLB Consultant",

27 "ca inter" : 5294.33,

28 "référence" : "GSMC",

29 "@version" : "1",

30 "nbjours" : 2.0,

31 "ca intra" : 0,



Features

- The console allows translation of CURL commands into its syntax
- It allows auto-completion, auto-indentation
- Passing on a single query line (useful for BULK queries)
- Allows you to run multiple queries
- Can change Elasticsearch server
- Keyboard shortcuts
- Search history
- It can be disabled (Property *console.enabled: false* in *kibana.yml*)



Other usages of Kibana

Kibana can be used by different actors :

- Data managers can inspect data stored in indices, create dashboards, alerts, etc..
- Operators can manage and monitor the cluster
- Developers can test their queries

The UI is highly customisable, you can create « *spaces* » accessed by different user roles, activate or deactivate menus according to users roles

Installation from an archive



```
wget  
https://artifacts.elastic.co/downloads/kibana/kibana-  
<version>-linux-x86_64.tar.gz  
sha1sum kibana-<version>-linux-x86_64.tar.gz  
tar -xzf kibana-<version>-linux-x86_64.tar.gz  
cd kibana/  
./bin/kibana
```



Folders

bin : Executable scripts including startup script and plugin install script

config : Configuration files including *kibana.yml*

data : Default data location, used by Kibana and plugins

optimize : Translated code.

plugins : Location of plugins. Each plugin corresponds to a directory

Configuration



Main configuration properties defined in ***conf/kibana.yml*** :

- ***server.host, server.port*** : default localhost:5601
- ***elasticsearch.url*** : default <https://localhost:9200>
- ***kibana.index*** : Elasticsearch index to store kibana data
- ***kibana.defaultAppId*** : The home page of the UI
- ***logging.dest*** : Log file. Default *stdout*
- ***logging.silent, logging.quiet, logging.verbose*** : Logging level



Introduction

ELK proposition and use cases
Distributions and installation
ELS API conventions, The DevConsole
of Kibana
Core concepts of clustering



Cluster

An *ElasticSearch* **cluster** is a set of servers (nodes) that contains all the data and offers search capabilities on the different nodes

- It has a unique name in the local network (default : "*elasticsearch*").

=> A cluster may consist of only one node

=> A node cannot belong to 2 separate clusters



Node

A **node** is a single server (Java process) that is part of a cluster.

- A node generally stores data, and participates in the indexing and search functionalities of the cluster.
- A node is also identified by a unique name (generated automatically if not specified)
- The number of nodes in a cluster is not limited



Master node

In a cluster, a node is elected as the **master node**, it is in charge of managing the configuration of the cluster and the creation of indexes.

For document operations (indexing, searching), a client can address any of the data nodes

In large architectures, nodes can be specialized

- Eligible master nodes
- Nodes dedicated to data ingestion,
- Nodes dedicated to Machine Learning processes,
- Nodes dedicated to data indexing and search



Discovery and cluster formation

The discovery and cluster formation processes are responsible for

- discovering nodes : Nodes find each other when the master is unknown
- electing a master : Voting mechanism based on quorum of master-eligible nodes. Occurs at startup or when the elected master fails
- forming a cluster : Nodes can leave or join the cluster at any time
- publishing the cluster state each time it changes : The cluster state changes is driven by the master node. It updates the membership view on each node



Discovery Configuration

discovery.type (Static) : *single-node* or *multi-node* (default). Single-node allow to save some process time in case of a single node cluster

cluster.initial_master_nodes (Static) : Sets the initial set of master-eligible nodes in a brand-new cluster. Remove this setting once the cluster has formed. Do not use this setting when restarting nodes or when adding new nodes to an existing cluster.

discovery.seed_hosts : (Static) Provides a list of the addresses of the master-eligible nodes in the cluster.

discovery.seed_providers (Static) : The way to provide seed hosts. (Can be an external file)



Cluster formation

During the first start of the cluster in production, it is necessary to specify *cluster.initial_master_nodes* which lists the list of eligible master nodes. (In general, the names of the nodes are specified)

Then, during a node restart, the search for the master is done with:

- The hosts listed in *discovery.seed_hosts*
- The Last Known Master



Index

An index is a collection of documents that have similar characteristics

- For example an index for customer data, another for the product catalog and yet another for orders

An index is identified by a name (in lower case)

- The name is used for indexing and search operations

In a cluster, you can define as many indexes as you want



Document

A **document** is the basic unit of information that can be indexed.

- document has a set of fields (key/value)
- It can be expressed with JSON format

Inside an index, you can store as many documents as you want



Shard

An index can store a very large amount of documents that may exceed the limits of a single node.

- To overcome this problem, ELS allows you to subdivide an index into several parts called **shards**

When creating the index, it is possible to define the number of shards

Each shard is an independent index that can be hosted on one of the cluster nodes



Benefits of sharding

Sharding allows :

- To **scale the volume** of content
- To **distribute** and parallelize operations => improve performance

The internal mechanism of distribution during indexing and search is completely managed by ELS and therefore transparent to the user



Replica

To tolerate failures, it is recommended to use fail-over mechanisms in the event that a node fails (Hardware failures or upgrade of a system)

ELS allows you to set up copies of shards: **replicas** in order to

- **High-availability** : A replica is hosted on a different node than the primary shard
- To **scale** . Requests can be load balanced on every replica .



Summary

In summary each index can be divided on several shards.

An index has a replication factor

Once replicated, each index has primary shards and replication shards

Number of shards and replicas can be specified at index creation time

After its creation, the number of replicas can be changed dynamically but not the number of shards

Default since ELS 7.x+ : 1 shard and 1 replica



Indexation of Documents

What is a document ?

What is an index ?

Routing and distributed search

Alternatives for ingestion



Introduction

ElasticSearch is a distributed **document** database.

It is able to store and retrieve data structures (in JSON format)

- Documents are made up of **fields**.
- Each field has a **data type**
- Some text fields are **analyzed and indexed** other not



Sample

```
{
  "name": "John Smith",      // String (may be analyzed or not)
  "age": 42,                 // Number
  "confirmed": true,         // Boolean
  "join_date": "2014-06-01", // Date
  "home": {                 // Embedded object
    "lat": 51.5,
    "lon": 0.1
  },
  "accounts": [             // Array
    {
      "type": "facebook",
      "id": "johnsmith"
    }, {
      "type": "twitter",
      "id": "johnsmith"
    }
  ]
}
```



Metadata

Metadata is also associated with each document.

Main metadata are :

- ***_index*** : The location where the document is stored
- ***_id*** : The unique identifier
- ***_version*** : The version number of the document
- ...



Document API

The document API enables CRUD operations on documents :

- Creation: *POST*
- Fetch by ID : *GET*
- Update : *PUT*
- Deletion : *DELETE*



Example Creation with Id

POST `/ {index} /_doc/ {id}`

```
{  
  "field": "value",  
  ...  
}  
...  
{  
  "_index": {index},  
  "_id": {id},  
  "_version": 1,  
  "created": true  
}
```



Retrieving a document

To retrieve a document just provide its metadata (index and id) :

GET `/_{index}/_doc/{id}?pretty`

The response contains the document and its metadata.

Example :

```
{  "_index" : "website",
    "_type" : "blog",
    "_id" : "123",
    "_version" : 1,
    "found" : true,
    "_source" : {
        "title": "My first blog entry",
        "text": "Just trying this out...",
        "date": "2014/01/01"    } }
```



Indexation of Documents

What is a document ?

What is an index ?

Routing and distributed search

Alternatives for ingestion



Introduction

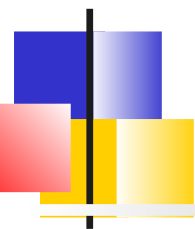
An index is a collection of documents which share the same fields

- Some documents may have all the fields, some others may have just a subset

Even if ELS allows to index documents before creating an index¹, for real use cases some decisions must be taken before starting indexation.

These decisions have impact on future search capabilities

1. *Property `action.auto_create_index` (default `true`)*



Main consideration for indices

At index creation, important decisions must be taken

- The **mapping** : Defines the fields and their datatypes of documents
Consequences on search and agregation capabilities
The mapping API is used to define, visualize, update the mapping of an index
- **Sharding and replica** :
Consequences on performance and fault-tolerance



Supported simples data types

ELS supports the following simple data types :

- 2 kinds of string : **text** (analyzed and full-text searching) or **keyword** (not analyzed, filtering and aggregations)
- Numbers : **byte** , **short** , **integer** , **long** , **float** , **double** , **token_count**
- Booleans : **boolean**
- Dates : **date**
- Bytes : **binary**
- Range : **integer_range** , **float_range** , **long_range** , **double_range** , **date_range**
- IP adress : **IPV4** ou **IPV6**
- Geolocation : **geo_point** , **geo_shape**
- A query (JSON structure) : **percolator**



Complex data types

In addition to simple types, ELS supports

- **arrays** : There is no special mapping for arrays. Each field can contain 0, 1 or n values

```
{ "tag": [ "search", "nosql" ] }
```

 - Array values must be of the same type
 - A null field is treated as an empty array
- **Embeddeed objects** : It is a data structure embedded in a field
 - Each sub-field is accessed with *embeddeddObject.property*
- **Nested object** : It is an array of data structures embedded in a control. Each array element is stored separately and has another *id*
 - It is generally a bad idea to use nested objects



When to define mapping

You can specify the mapping:

- When creating an index
- When adding a new field to an existing index

=> You cannot modify an existing field



Index API: index creation

PUT /gb

```
{
  "mappings": {
    "tweet" : {
      "properties" : {
        "tweet" : {
          "type" : "text",
          "analyzer": "english"
        },
        "date" : {
          "type" : "date"
        },
        "name" : {
          "type" : "keyword"
        },
        "user_id" : {
          "type" : "long"
        }
      }
    }
  }
}
```



Mapping of embedded objects

```
{ "gb": {  
  "properties": {  
    "tweet": { "type": "text" },  
    "user": {  
      "type": "object",  
      "properties": {  
        "id": { "type": "text" },  
        "age": { "type": "long"},  
        "name": {  
          "type": "object",  
          "properties": {  
            "first": { "type": "text" },  
            "last": { "type": "text" }  
          }  
        }  
      }  
    }  
  }  
}
```

Fields are then referenced with the . notation.
Example *user.name.first*



Double mapping

```
"tweet": {  
  "type": "text",  
  "analyzer": "english",  
  "fields": {  
    "fr": {  
      "type": "text",  
      "analyzer": "french"  
    }  
  }  
}
```

2 fields are available for full-text searching : *tweet* and *tweet.fr*



Sharding, replica and other config

At index creation, the ***settings*** block can fix some properties.

Some properties are static, other are dynamic (may change after creation)

- Static properties : *number_of_shards, ...*
- Dynamic properties :
number_of_replicas, ...



Example Index Creation

```
PUT /my-index-000001
{
  "settings": {
    "index": {
      "number_of_shards": 3,
      "number_of_replicas": 2
    }
  }
}
```



Index templates

Index templates allow you to define configuration properties that will be applied when creating indexes if the name of the index respects a pattern (***index-pattern*** property)

They define the 2 aspects:

- *settings*
- *mappings*



API

Index Templates are managed via the ***template*** API

Ex : Creation of an index template

```
PUT _template/template_1
{
  "index_patterns": ["te*", "bar*"],
  "settings": { "number_of_shards": 1 },
  "mappings": {
    "type1": {
      "_source": { "enabled": false },
      "properties": {
        "host_name": {
          "type": "keyword"
        },
        "created_at": {
          "type": "date",
          "format": "EEE MMM dd HH:mm:ss Z YYYY" } } } } }
```




What is a DataStream ?

A ***datastream*** (formerly named *index-pattern*) is a collection of indices which follows a naming pattern.

For example : *logstash-**

Typically, logstash, beats or other systems create new index on a regular basis and postfix the date of creation to the name of the index.

The collection of created indices is then defined as a datastream in ELS and Kibana



Index aliases

An ***alias*** is a secondary name for a group of data streams or indices.

Most Elasticsearch APIs accept an alias in place of a data stream or index name.

Alias are managed by the alias API

```
POST _aliases
{
  "actions": [
    {
      "add": {
        "index": "logs-nginx.access-prod",
        "alias": "logs"
      }
    }
  ]
}
```



Indexation of Documents

What is a document ?

What is an index ?

Routing and distributed search

Alternatives for ingestion



Primary shard resolution

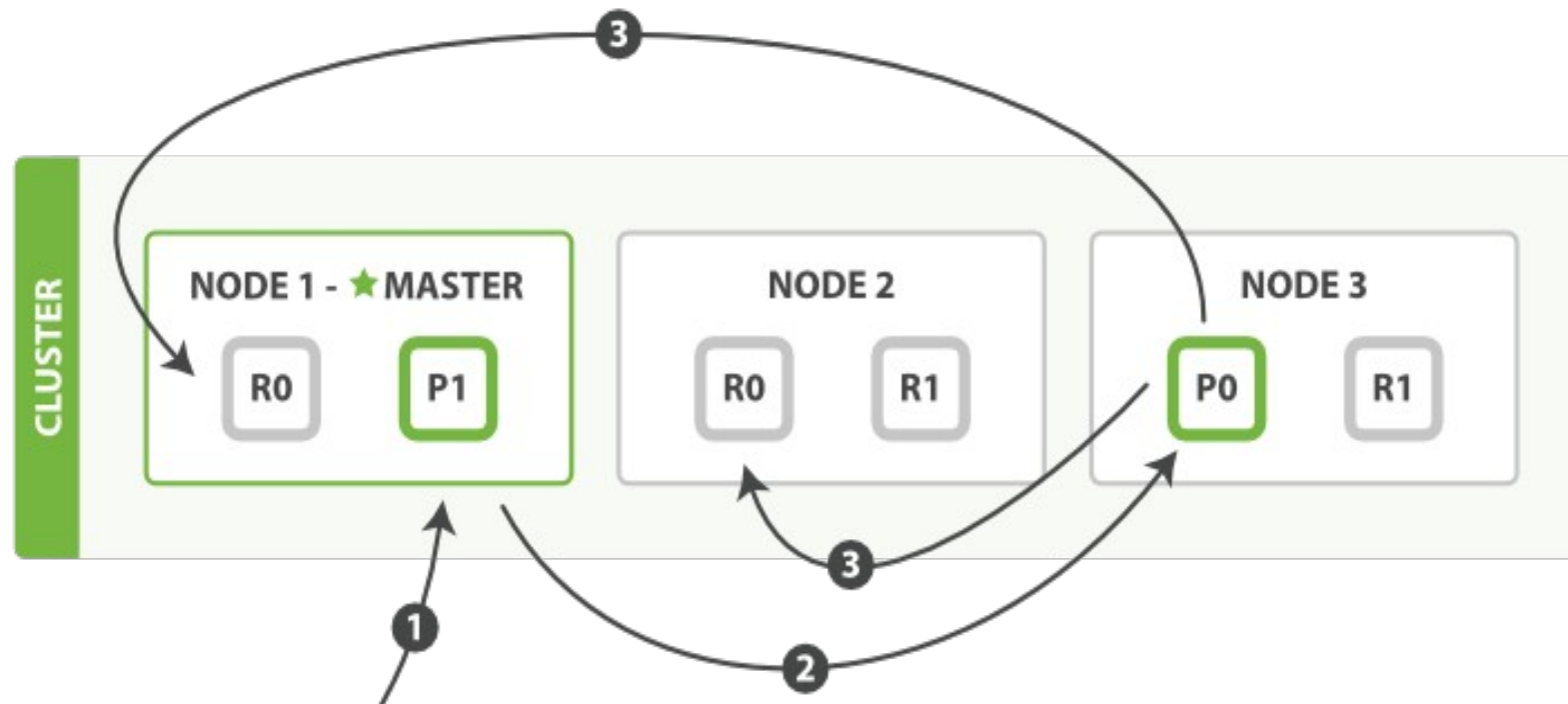
During indexing, the document is first stored in the primary shard.

Resolution of the shard is performed with the following formula:

```
shard = hash(routing) % number_of_primary_shards
```

The value of the ***routing*** parameter is an arbitrary string which by default matches the document ***id*** but which may be explicitly specified

Sequence of updates during indexation

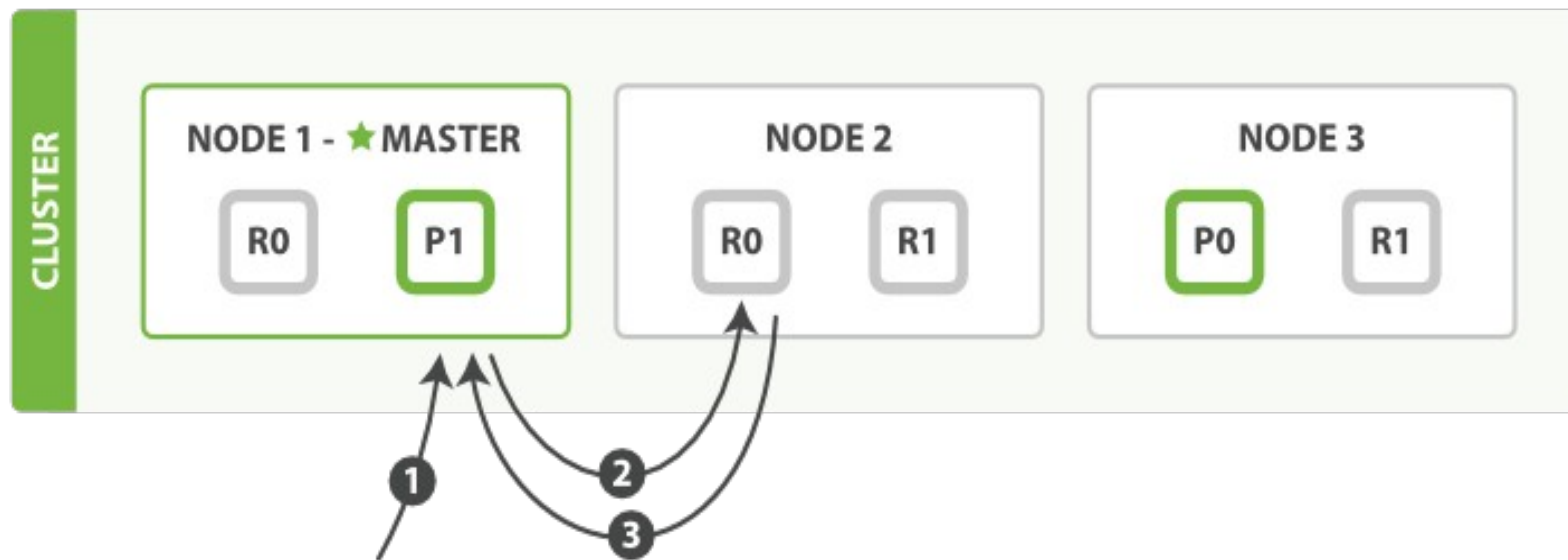




Sequence of updates during indexation (2)

1. The client sends an indexing or deletion request to *node1*
2. The node uses the document *id* to infer that the document belongs to shard 0 .
It forwards the request to *node3* , where the primary copy of shard 0 resides.
3. Node 3 runs the query on the primary shard.
If successful, it forwards the request in parallel to the replicas residing on *node1* and *node2* .
Once the replica update orders succeed, *node3* responds to *node1* which responds to the client

Sequence to retrieve a document





Sequence to retrieve a document (2)

1. The client sends a request to *node1*.
2. The node uses the document *id* to determine that the document belongs to shard 0.
Copies of shard 0 exist on all 3 nodes.
For this time, it forwards the request to *node2* .
3. *node2* returns the document to *node1* which returns it to the client.

For the next read requests, the node will choose a different shard to distribute the load (Round-robin algorithm)



Distributed search

Research requires a complex execution model, with documents scattered across shards

ElasticSearch must consult a copy of each shard of the index or indexes requested

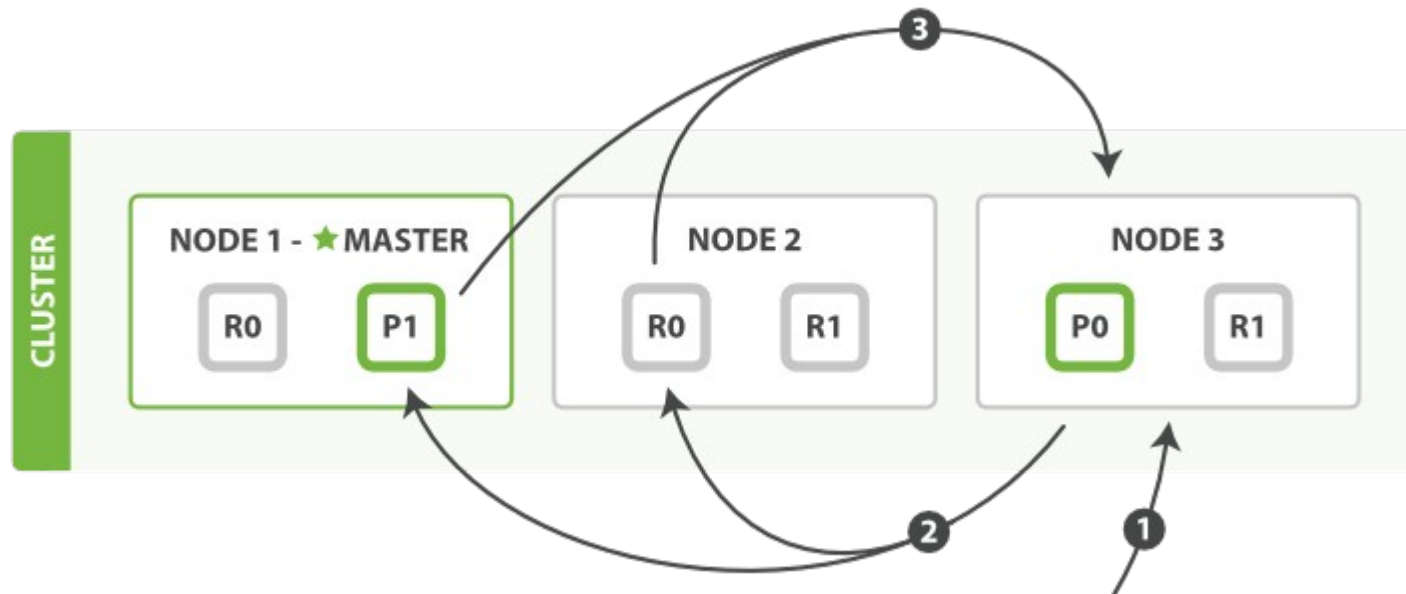
When found; results from different shards must be combined into a single list so that the API can return a page of results

The search is therefore executed in 2 phases:

- ***query***
- ***fetch.***

Query phase

During the 1st phase, the request is broadcast to a copy (primary or replica) of all shards. Each shard performs the search and builds a priority queue of matching documents



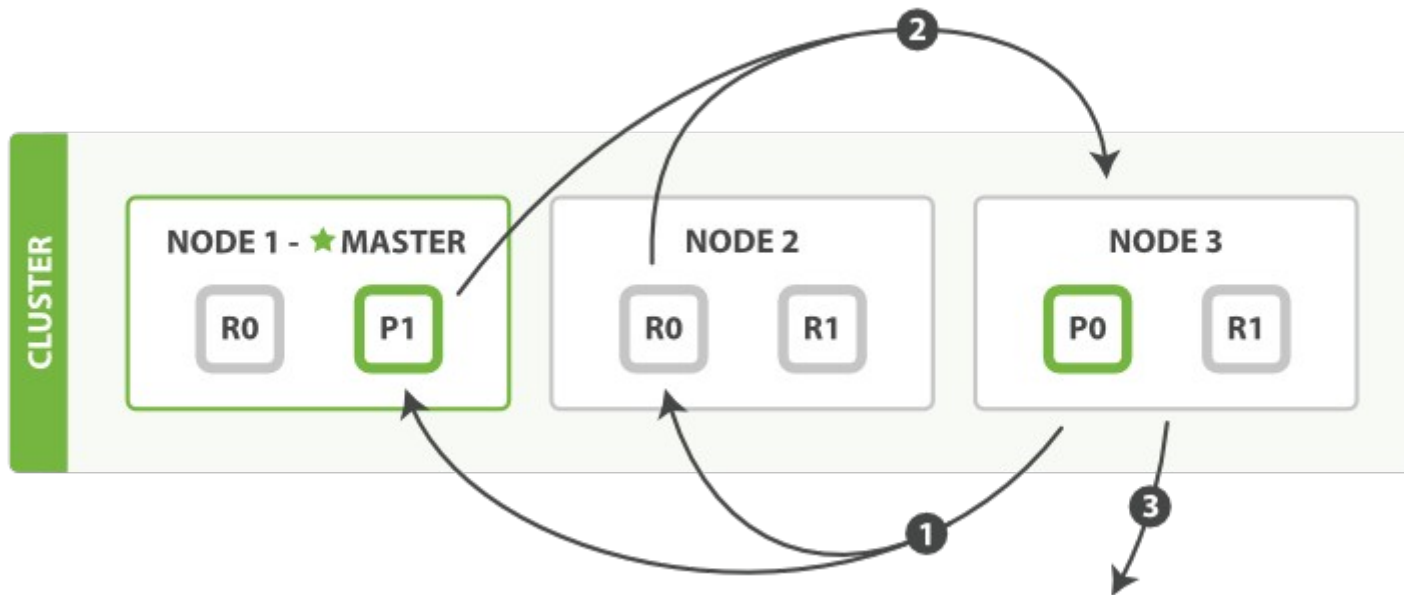


Query phase

1. The client sends a query to *node3* which creates an empty priority queue of $size = from + size$.
2. *node3* forwards the request to a copy of each shard in the index. Each shard executes the query locally and adds the result to a local priority queue of $size = from + size$.
3. Each shard returns the IDs and sort values of all documents in the queue to *node3* which merges these values into its priority queue

Phase fetch

The fetch phase consists of retrieving the documents present in the priority queue.





Fetch phase

1. The coordinator identifies which documents need to be retrieved and issues a multiple GET request to the shards.
2. Each shard loads the documents, enriches them if necessary (highlighting for example) and returns them to the coordinator node
3. When all documents have been retrieved, the coordinator returns the results to the client.



Indexation of Documents

What is a document ? What is an index ?

Main considerations for indices
Routing and distributed search

Alternatives for ingestion



Introduction

Different scenarios exist to ingest data in Elasticsearch

- Writing a **custom program** which uses provided libraries of ELS
- Use of **Beats** which periodically send data to ELS
- Use **logstash** to extract data from different sources, transform them and ingest into ELS
- Use **ElasticSearch pipelines** to transform incoming documents into the right format



A custom Java program

```
try (DirectoryStream<Path> stream = Files.newDirectoryStream(dir, "*.pdf,ppt,doc")) {
    for (Path file : stream) {
        BufferedInputStream bin = new BufferedInputStream(Files.newInputStream(file));
        byte[] data = new byte[bin.available()]; bin.read(data); bin.close();
        String encodedString = Base64.encodeBase64String(data);

        Map<String, Object> jsonMap = new HashMap<>();
        jsonMap.put("name", file.getFileName());
        jsonMap.put("data", encodedString);

        try {
            IndexResponse response = elsClient.index(i -> i
                .index(index)
                .document(jsonMap)
            );
        } catch (Exception x) { System.err.println(x); }
```




Beats

Beats are installed as agents on the different targets (servers) and periodically send their data

- Directly to *ElasticSearch*
- Or, if the data need to be transformed, to *Logstash*

ELK provides :

- ***Packetbeat*** : All the network packets seen by a specific host
- ***Filebeat*** : All the lines written in a log file .
- ***Metricbeat*** : Performance metrics on CPU, Memory, Disk, IO, etc
- ***Winlogbeat*** : Windows log events
- Other beats by third-parties are also available



Logstash

Logstash is a **real-time data collection** tool able to unify and normalize data from different sources

Originally focused on log files, it has evolved to handle very diverse events

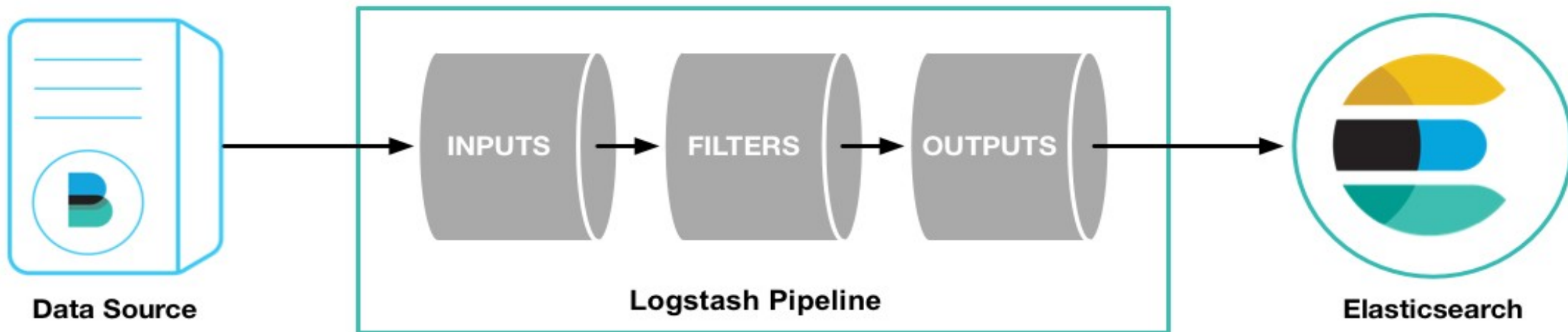
Logstash is based on the notion of processing **pipelines**.

Extensible via plugins, it can read/write from/to very different sources and targets

Pipeline Logstash

A logstash **pipeline** defines :

- An input plugin to read data from a source
- An output plugin to write data to a target (typically ElasticSearch)
- Optional filters to perform transformations



Configuration

Pipelines are defined in files which follow the logstash syntax.
For example :

```
input {  
  file {  
    path => "/var/log/apache2/access*.log"  
  }  
}  
filter {  
  grok {  
    match => { "message" => "%{COMBINEDAPACHELOG}" }  
  }  
  geoip {  
    source => "clientip"  
  }  
}  
output {  
  elasticsearch {  
    hosts => [ "localhost:9200" ]  
  }  
}
```



Elasticsearch Output

Main options of elasticsearch outputs are :

- **hosts** : Data nodes of the cluster
- **index** : The name of the index. Example : logstash-%
{+YYYY.MM.dd}.
- **template** : A path to a template to create the index
- **template_name** : The template's name present in Elasticsearch
- **document_id** : The id of the Elasticsearch document (allow us
updates)
- **pipeline** : The Elasticsearch pipeline to execute before
indexing
- **routing** : To specify the shard to use



Elastic Search Pipelines

Without *logstash*, it is possible to perform common transformations on the data before indexing by using the **ingest pipelines of ElasticSearch**

Features are similar to logstash but less powerful

- ELS pipelines can be used to remove fields, extract values from text, and enrich your data.

It consists of a series of configurable tasks called ***processors***. Each processor runs sequentially, making specific changes to incoming documents



Pipeline

A **pipeline** is defined via :

- A name
- A description
- And a list of *processors*

Processors are predefined

They will be executed in sequence

The ***_ingest*** API allows to manage ELS pipelines



Sample pipeline with the attachment plugin

#Create a pipeline named *attachment*

```
PUT _ingest/pipeline/attachment
{
  "description" : "Extract attachment information",
  "processors" : [
    { "attachment" : { "field" : "data" } }
  ]
}
```

Using a pipeline during indexation of a document.

```
PUT my_index/my_type/my_id?pipeline=attachment
{
  "data":
  "e1xydGYxXGFuc2kNCkxvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbWV0DQpccGFyIH0="
}
```




Result

GET my_index/my_type/my_id

```
{
  "found": true,
  "_index": "my_index",
  "_type": "my_type",
  "_id": "my_id",
  "_version": 1,
  "_source": {
    "data":
    "e1xydGYxXGFuc2kNCkxvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbWV0DQpccGFyIH0=",
    "attachment": {
      "content_type": "application/rtf",
      "language": "ro",
      "content": "Lorem ipsum dolor sit amet",
      "content_length": 28
    }
  }
}
```



Introduction to plugins

The functionalities of ELS can be increased via the notion of plugin

Plugins contain JAR files, but also scripts and configuration files

They can be installed easily with

```
sudo bin/elasticsearch-plugin install [plugin_name]
```

- They must be installed on each node of the cluster
- After an installation, the node must be restarted



Ingest plugins

A particular ingest plugins allow to index office documents (Word, PDF, XLS, ...) :

- ***Attachment Plugin*** : Extracts text data from attachments in different formats (PPT, XLS, PDF, etc.). It uses the Apache Tika library.

```
sudo bin/elasticsearch-plugin install ingest-attachment
```



Search capabilities

Search lite
DSL syntax



Search APIs

2 Search APIs :

- A **lite** version where all parameters are provided with the query string.
- A **full** where search parameters are provided with the request body as a JSON structure.

The query syntax is a DSL



Search response

```
{
  "hits" : {
    "total" : 14,
    "hits" : [ {
      "_index": "us",
      "_type": "tweet",
      "_id": "7",
      "_score": 1,
      "_source": {
        "date": "2014-09-17",
        "name": "John Smith",
        "tweet": "The Query DSL is really powerful and flexible",
        "user_id": 2
      }
    }, ... RESULTS REMOVED ... ],
    "max_score" : 2.5211782
  }, tooks : 4,
  "_shards" : {
    "failed" : 0,
    "successful" : 10,
    "total" : 10
  },
  "timed_out" : false
}
```



Fields of the response

hits : The number of documents that match the query, followed by an array containing the entire top 10 documents.

Each document has a `_score` element that indicates its relevance. By default, documents are sorted by relevance

took : The number of milliseconds taken by the request

shards : The total number of shards that participated in the request. Some may have failed

timeout : Indicates if the request has timed out. The timeout parameter must have been provides like this :

GET `/_search?timeout=10ms`



Search Lite

To perform a search with searchlite, just a GET request with

- Optionally a restriction to an index or some indices
- A **q** parameter with the text searched following the Lucene syntax

Samples :

GET `/_search`

GET `/myIndex/_search?q=elastic`

GET `/myIndices*/_search?q=elastic`

GET `/myIndices*/_search?q=elastic AND search`

...

q parameter



The **q** parameter indicates the query string

The string is parsed and splitted in

- Fields : Fields of the document on which query is performed
- Terms : (Word or a phrase)
- And operators (AND/OR)

The syntax supports :

- Wildcards and regexp
- Grouping (parenthesis)
- Boolean operators (By default OR, + : AND, - : AND NOT)
- Intervals : Ex : [1 TO 5}
- Comparaison operators



Examples search lite

GET /_search?q=tweet:elasticsearch

All documents with the field *tweet* matches (~contains) "*elasticsearch*"

+name:john +tweet:mary

GET /_search?q=%2Bname%3Ajohn+%2Btweet%3Amary

All documents whose the field *name* matches "john" and the field *tweet* matches "mary"

+name:john -tweet:mary

All documents whose the field *name* matches "john" and the field *tweet* does not matches "mary"



Search capabilities

Search lite
DSL syntax



Introduction

Searches with a request body offer more functionalities than search lite.

In particular, they allow you to :

- Combine query clauses more easily
- To influence the score
- Highlight parts of the result
- Aggregate subsets of results
- Return suggestions to the user



Query object

To use DSL, you pass a **query** object in the body of the request:

```
GET /_search
```

```
{ "query": {  
  QUERY_NAME/OPERATOR: {  
    ARGUMENT: VALUE, ARGUMENT: VALUE, ...  
  } } }
```

Example :

```
GET /_search
```

```
{ "query": {  
  "match": { "tweet": "elasticsearch" }  
}
```



Combination operator

bool : Allows you to combine clauses with:

- ***must*** : equivalent to AND with score contribution
- ***filter*** : idem but does not contribute to the score
- ***must_not*** : equivalent to NOT
- ***should*** : equivalent to OR

```
{  
  "bool": {  
    "must": { "match": { "tweet": "elasticsearch" }},  
    "must_not": { "match": { "name": "mary" }},  
    "should": { "match": { "tweet": "full text" }}  
  }  
}
```



Types of operators

Operators can be classified in 2 families :

- Operators that do not perform analysis and operate on a single term (*term*, *fuzzy*).
- The operators that apply the analyzer to the search terms corresponding to the searched field (*match*, *multi_match*) .



Operators without analysis

term : Used to filter exact values :

```
{ "term": { "age": 26 } }
```

terms : Allows multiple values to be specified :

```
{ "terms": { "tag": [ "search", "full_text",  
"nosql" ] } }
```

range : Allows you to specify a date or numeric range:

```
{ "range": { "age": { "gte": 20, "lt":  
30 } } }
```

exists and ***missing*** : Allows you to test whether or not a document contains a field

```
{ "exists": { "field": "title" } }
```




match

The match ***operator*** is the standard operator for performing an exact or full-text search on almost any field.

- If the query is for a full-text field, it parses the search string using the same parser as the field,
- If the search is for an exact value field, it searches for the exact value

```
{ "match": { "tweet": "About Search" } }
```



multi_match

The ***multi_match*** operator allows you to run the same query on multiple fields:

```
{"multi_match": { "query": "full text search", "fields":  
[ "title", "body" ] } }
```

Wildcards can be used for fields

Fields can be boosted with notation ^

```
GET /_search  
{  
  "query": {  
    "multi_match" : {  
      "query" : "this is a test",  
      "fields" : [ "subject^3", "text*" ]  
    }  
  }  
}
```



Other features

ELS provide many other different search operators :

- Partial matching : *prefix, wildcard*
- Search phrase : *match_phrase, match_phrase_prefix*
- Approximate match : *fuzzy queries*
- Geo-queries : *geo_filter, geo_shape*

It also allows to highlight the match in the response



Aggregations

Aggregations are extremely powerful for reporting and dashboards, they can also be used for faceting in traditional search engine

In DSL syntax, an aggregation block uses the keyword ***aggs***

// The max of the price field in all documents

```
POST /sales/_search?size=0
```

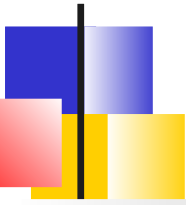
```
{
  "aggs" : {
    "max_price" : { "max" : { "field" : "price" } }
  }
}
```



Aggregations

Several concepts relate to aggregations:

- **Groups or Buckets** : Set of documents that have a field with the same value or share the same criteria.
Groups can be nested.
ELS automatically counts the number of documents in each group
- **Metrics**: Metric calculations on a group of documents (min, max, avg, ..)



Example nested aggregations

```
GET /cars/transactions/_search {  
  "aggs": {  
    "colors": {  
      "terms": { "field": "color" },  
      "aggs": {  
        "avg_price": {  
          "avg": { "field": "price" }  
        }, "make": {  
          "terms": { "field": "make" }  
        }  
      }  
    }  
  }  
}
```



Response

```
{
  ...
  "aggregations": {
    "colors": {
      "buckets": [
        { "key": "red",
          "doc_count": 4,
          "avg_price": {
            "value": 32500
          },
          "make": { "buckets": [
            { "key": "honda", "doc_count": 3 },
            { "key": "bmw", "doc_count": 1 }
          ]
        }
      ],
    },
    ...
  }
}
```



Types of buckets

ELS offers different ways of grouping data :

- By term: Identical values
- Histogram : By range of values
- Date Histogram : By date range
- By IP range
- By absence/presence of a field
- Significant terms : Allow to detect anomalies
- By geo-location (distance)
- ..



Available metrics

ELS offers many metrics:

- ***avg, min, max, sum***
- ***value_count, cardinality*** : Distinct value count
- ***top_hit*** : The top documents
- ***extended_stats*** : Statistical metrics (count, sum, variance, ...)
- ***percentiles*** : percentiles



Architecture

Different architectures

Nodes for production

Specialization of elastic nodes

Cross-cluster replication

Dimensioning indices and sharding

Index lifecycle policies



Introduction

The Elastic solutions can be used in many different scenarios with very different architectures :

- ElasticSearch Engine for a non-critical application :
A **single node** coupled with Kibana can be sufficient
- An application which needs fault-tolerance :
 - **A 3 nodes cluster**, each node has the same features and configuration.
 - More nodes distributed on different data centers
 - Cross-cluster replication : replicate data to a remote follower cluster
- An installation with different usages. High volume of data, Near-real time search requirement : **A n nodes cluster** with nodes specialized for ingestion, data search, master, etc.



Single node as a search engine

Monitoring disk space, performance of queries (If possible use another cluster to monitor)

Have an efficient reindexation process which allows reconfiguration of the index (change of mapping or sharding)

Have a blue-green upgrade process using index aliases

Use of a kibana as administration tool, data inspection, test of queries

Access to the node via a proxy or directly https with authentication



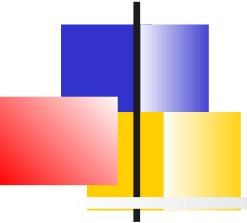
Specialization of nodes

Nodes can be specialized into one or several roles.

- It allows to guarantee the node will be dedicated to certain tasks and will not be disturbed by another computing intensive task.

For example, we can dedicate nodes to be only master.

N-nodes cluster generally specialize nodes and size them regarding to their task



Cross-cluster

For large deployments, it is possible to manage coherently different clusters.

This allows to :

- Perform ***cross-cluser search*** : A query on different indices hosted on different cluster
- Perform ***cross-cluster replication***. Documents are replicated in different data centers



Architecture

Different architectures

Production node

Specialization of nodes

Dimensionning indices and sharding

Index lifecycle policies



Hardware

RAM : A machine with 64 GB is ideal; 32 GB and 16 GB are correct

CPU : Favor the number of cores rather than the speed of the CPU

Disk : If possible fast, SSDs? Avoid NAS (network-attached storage)



JVM

The embeddeed version provided by Elastic

Do not update the JVM configuration provided except dimensionning Heap.

Concerning the Heap, the default sizing is also recommended for most production environments.

The size is determined depending on the rôle of the node



Heap

3 ways to set the heap size.

- The file ***jvm.options***
- The environment variable ***ES_JAVA_OPTS***
export ES_JAVA_OPTS="-Xms2g -Xmx2g"
- Via the CLI
./bin/elasticsearch **-Xmx=10g -Xms=10g**

2 recommendations :

- give ELS 50% of available memory and leave the other half empty.
In fact Lucene will happily occupy the other half
- Do not exceed 32GB



Configuration management

Preferably use configuration management tools like Puppet, Chef, Ansible...

otherwise setting up a cluster with lots of nodes quickly becomes hell



Configure for production

Some properties must be overridden in production

- The **name of the cluster**
`cluster.name: elasticsearch_production`
- Le **name of the nodes**
`node.name: elasticsearch_005_data`
- **Paths**
`path.data: /path/to/data1,/path/to/data2`
`# Path to log files:`
`path.logs: /path/to/logs`
`# Path to where plugins are installed:`
`path.plugins: /path/to/plugins`



Swapping

Avoid swapping. Optionally, disable it at system level

```
sudo swapoff -a
```

Or with Elastic Config

```
bootstrap.memory_lock: true
```

(Formerly `bootstrap.mlockall`)



File descriptors

Lucene uses a lot of files. ELS lots of sockets

Most Linux distributions limit the number of file descriptors to 1024.

This value should be increased to 64,000



Architecture

Different Architectures

Production node

Specialization of elastic nodes

Dimensionning indices and sharding

Index lifecycle policies



Roles of nodes

All nodes in a cluster know each other and can redirect HTTP requests to the correct node. It is possible to specialize the nodes in order to reserved resources for controlling the cluster, data management and ingestion.

Specialization is made thanks to the ***node.roles*** configuration:

- ***master*** : Master nodes
- ***data*** : Store the indices and perform search queries
- ***ingest*** : Execute pipelines (needed for stack monitoring)
- ***<no-role>*** : Coordination nodes which accept request and redirect to a data node containing the indices
- ***remote_cluster_client*** : Cross-cluster search



Master nodes

The master node is responsible for lightweight operations:

- Creating or deleting indexes
- Cluster Node Monitoring
- Shard Allocations

In a production environment, it is important to ensure the stability of the master node.

It is recommended for large clusters not to load the master nodes with ingest, indexing or search jobs.

`node.roles:`

- master



Roles of nodes

Configuration of a **data node**

node.roles:

- data

Configuration of an **ingestion node**

node.roles:

- ingest

Configuration of a **coordination node**

node.roles :

Configuration of **cross-cluster search**

node.roles :

- remote_cluster_client



Fault tolerant architecture

With 3 all-roles nodes and a replication factor of 2

- Status is green with 2 nodes up
- Status is yellow with one node up

Master election can succeed even if one node fails

Optionally a coordination node used as a proxy can be configured in Kibana or the client application

With more than 3-nodes, nodes are typically specialized



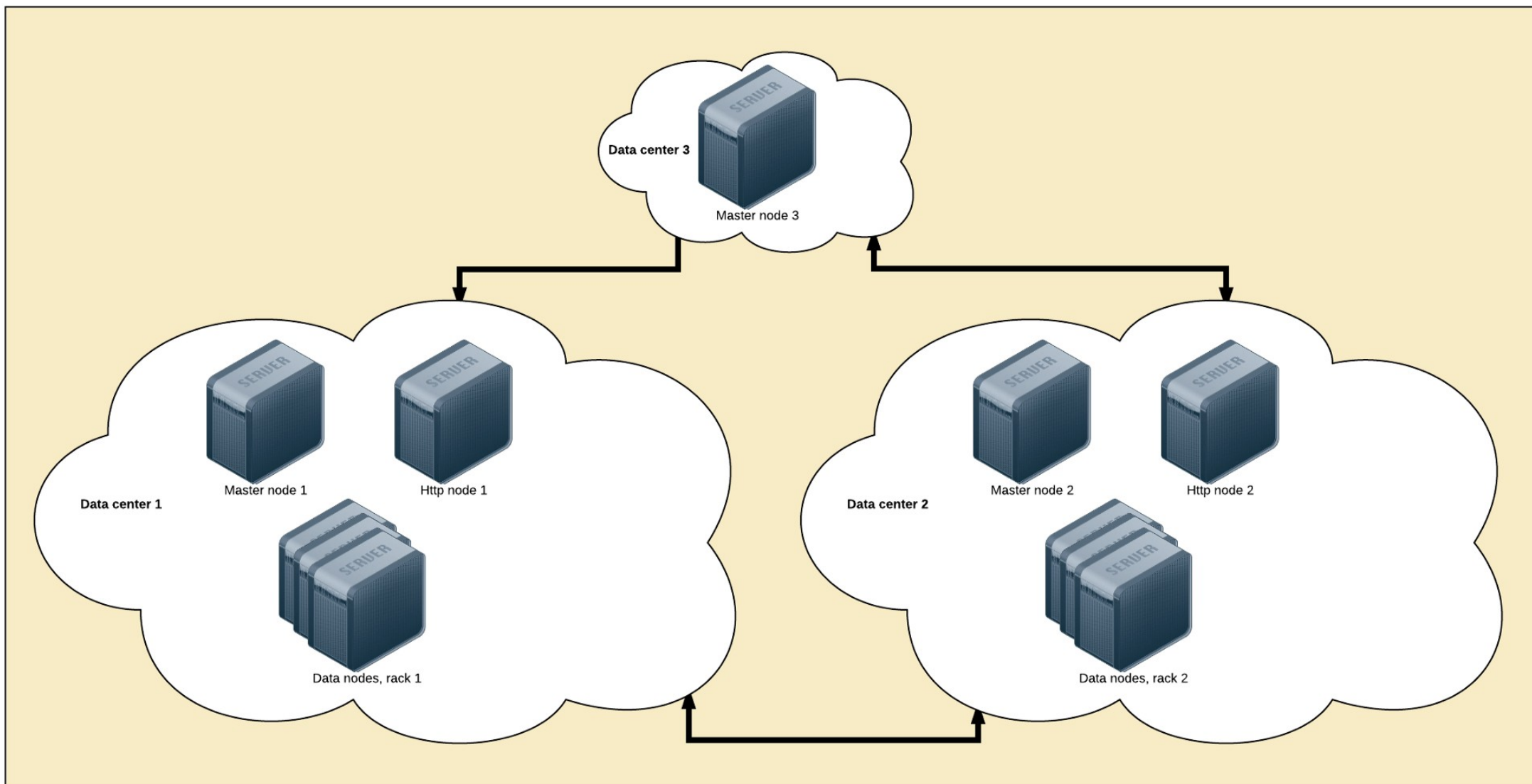
Architecture HA with different data centers

A typical fault-tolerant architecture is to distribute nodes across different data centers

- at least 2 main data centers containing the data nodes and 1 backup containing a possible master node
- 3 master nodes
- 2 nodes to execute http requests (1 per main data center)
- Distributed data nodes on the 2 main data centers



Fault-tolerant architecture

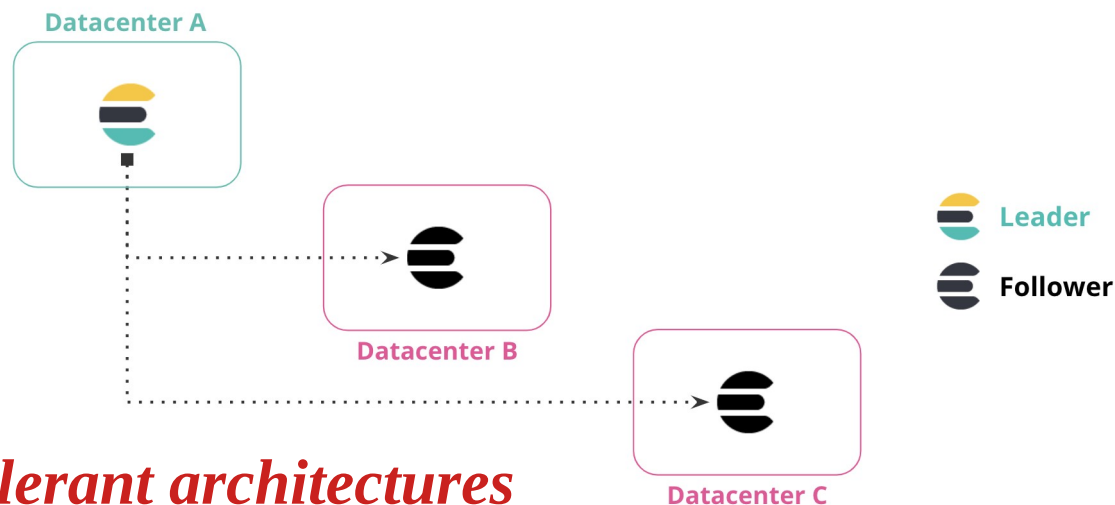




Cross-cluster replication

With cross-cluster replication, you can replicate indices across clusters to:

- Continue handling search requests in the event of a datacenter outage
- Prevent search volume from impacting indexing throughput
- Reduce search latency by processing search requests in geo-proximity to the user





Architecture

Different Architectures

Production node

Specialization of elastic nodes

**Dimensionning indices and
sharding**

Index lifecycle policies



Lucene Segments

An Elasticsearch shard is a Lucene index.
A Lucene index is divided into small files:
segments

| Elasticsearch Index | | | | | | | |
|---------------------|---------|---------------------|---------|---------------------|---------|---------------------|---------|
| Elasticsearch shard | | Elasticsearch shard | | Elasticsearch shard | | Elasticsearch shard | |
| Lucene index | | Lucene index | | Lucene index | | Lucene index | |
| Segment | Segment | Segment | Segment | Segment | Segment | Segment | Segment |



Merging segments

Lucene creates segments during indexing. Segments are immutable

During a search, segments are read sequentially

=> The more segments, lower is the search performance

To optimize the search, small segments are merged into larger ones. For the merge operation to succeed, the disk space must be twice the size of the shard being merged

- This operation is scheduled automatically and is performed by a pool of threads which can be configured by :
index.merge.scheduler.max_thread_count
- Alternativeley, we can trigger a merge via the API :
`curl -XPOST 'localhost:9200/logstash-2017.07*/_forcemerge?max_num_segments=1'`



Sizing the number of shards ?

The number of shards is set at index creation. (Default: 1)

The correct value depends mainly on the estimated data volume of the index.

Resizing an index in production requires reindexing.



Benefits for many shards

Having a lot of shards can brings some advantages :

- Augment the degree of parallelization during search
- Small shards on many nodes makes the recovery process faster when a node fails
- Large shards make Lucene merging processes more difficult.

But having many shards can overload the master and the cluster then becomes very unstable



Recommendations

Experience shows that shards **between 10GB and 50GB** typically work well for logs and time series data

For enterprise search, use smaller shards (i.e. $\leq 10\text{Go}$) .

For example : For an average of 2GB for 1 million documents

- From 0 to 4 million documents per index: 1 shard.
- From 4 to 5 million documents per index: 2 shards
- > 5 million documents: 1 shards per 5 million.



Architecture

Different Architectures

Production node

Specialization of elastic nodes

Dimensionning indices and sharding

Index lifecycle policies



Lifecycle of indices

It is possible to automate index lifecycle management actions such as:

- **Rollover** : Redirect an alias to start writing to a new index when the existing index reaches a certain age, number of documents, or size.
- **Shrink** : Reduce the number of primary shards in an index.
- **Force merge** : Reduce the number of segments in each shards of an index and free up space used by deleted documents.
- **Freeze** : Make an index read-only and minimize its memory footprint
- **Delete** : Delete an index



Mechanism

In practice, a ***lifecycle policy*** must be associated with an *index template*, so that it will be applied to newly created indexes.

The lifecycle policy define conditions of the different ***phases*** of the lifecycle and their related management operations :

- ***Hot*** : The index is updated and queried
- ***Warm*** : The index is no longer updated but still queried.
- ***Cold*** : The index is no longer updated and is rarely queried. Information is still searchable, but queries are slower
- ***Delete*** : The index is no longer used and can be deleted.



Strategies

Then, a lifecycle policy may specify :

- Maximum size or age to perform the rollover.
- The time when the index is no longer updated and the number of primary shards may be reduced.
- Time to Force Merge
- The moment when we can move the index to less efficient hardware.
- The time when availability is not as critical and the number of replicas can be reduced.
- When the index can be safely deleted.



Sample : Creation of a policy

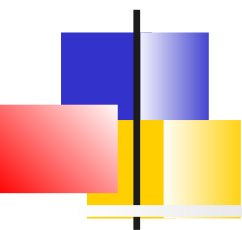
```
PUT _ilm/policy/datastream_policy
{
  "policy": {
    "phases": {
      "hot": {
        "actions": {
          "rollover": {
            "max_size": "50GB",
            "max_age": "30d"
          }
        }
      },
      "delete": {
        "min_age": "90d",
        "actions": {
          "delete": {}
        }
      }
    }
  }
}
```



Template and index creation

```
PUT _template/datastream_template
{
  "index_patterns": ["datastream-*"],
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 1,
    "index.lifecycle.name": "datastream_policy",
    "index.lifecycle.rollover_alias": "datastream"
  }
}
---
PUT datastream-000001
{
  "aliases": {
    "datastream": {
      "is_write_index": true
    }
  }
}
---
GET datastream-*/_ilm/explain
```

Demo : See the index policies in Kibana



Monitoring and Operations

Logs

Low level Cluster API

Monitoring with beats

Reindexation, Snapshot and Restore

Resizing and restarting the cluster



Introduction

You can use Elasticsearch's application logs to monitor your cluster and diagnose issues

ELS use Log4J 2 and can be configured via *log4j2.properties*

The configuration can use system properties specified by ELS :

- *es.logs.base_path*, *es.logs.cluster_name*,
es.logs.node_name



Default configuration

The default configuration generates :

- A rolling file named with the **<cluster.name>_server.json** in JSON format with level INFO
- A legacy text log file named **<cluster.name>.log** with level INFO
- A **deprecation** file which logs WARNING messages about deprecated functionality
- A **slowlog** which may logs slow request or slow indexation
- A **gc** log configured in *jvm.options*



Slowlog

The objective of the *slowlog* is to log queries and indexing requests that exceed a certain time threshold

By default this log file is not enabled. It can be activated by specifying the action (query, fetch, or index), the trace level (WARN, DEBUG, ..) and the time threshold

This is an index level configuration

```
PUT /my_index/_settings
```

```
{ "index.search.slowlog.threshold.query.warn" : "10s",  
  "index.search.slowlog.threshold.fetch.debug": "500ms",  
  "index.indexing.slowlog.threshold.index.info": "5s" }
```

```
PUT /_cluster/settings
```

```
{ "transient" : {  
  "logger.index.search.slowlog" : "DEBUG",  
  "logger.index.indexing.slowlog" : "WARN"  
} }
```

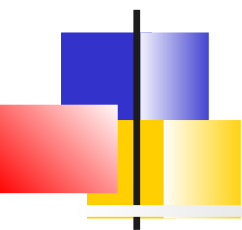


Other logs

The logfile audit output is the only output for auditing.

It writes data to the

<clustername>_audit.json file in the logs directory.



Monitoring and Operations

Logs

Low level Cluster API

Monitoring with beats

Reindexation, Snapshot and Restore

Resizing and restarting the cluster



Introduction

ELS expose an API to monitor the cluster (*_cluster*, *_nodes*)

In addition, metricbeats allows aggregation of the node's metrics inside indices and it provides built-in dashboards to monitor an ELS cluster



Cluster Health API

```
GET _cluster/health
{
  "cluster_name": "elasticsearch_zach",
  "status": "green", // green, yellow or red
  "timed_out": false,
  "number_of_nodes": 1,
  "number_of_data_nodes": 1,
  "active_primary_shards": 10,
  "active_shards": 10,
  "relocating_shards": 0,
  "initializing_shards": 0,
  "unassigned_shards": 0
}
```



Indices information

```
GET _cluster/health?level=indices
{
  "cluster_name": "elasticsearch_zach",
  "status": "red",
  ...
  "unassigned_shards": 20
  "indices": {
    "v1": {
      "status": "green",
      "number_of_shards": 10,
      "number_of_replicas": 1,
      "active_primary_shards": 10,
      "active_shards": 20,
      "relocating_shards": 0,
      "initializing_shards": 0,
      "unassigned_shards": 0
    },
```



node-stats API

GET `_nodes/stats`

```
{  
  "cluster_name": "elasticsearch_zach",  
  "nodes": {  
    "UNr6ZMf5Qk-YCPA_L18B0Q": {  
      "timestamp": 1408474151742,  
      "name": "Zach",  
      "transport_address":  
        "inet[zacharys-air/192.168.1.131:9300]",  
      "host": "zacharys-air",  
      "ip": [  
        "inet[zacharys-air/192.168.1.131:9300]",  
        "NONE"  
      ],  
    },  
  },  
}
```



Indices section

The **indices** section displays aggregated statistics for each index of a node:

- **docs** : how many documents reside on the node, the number of deleted documents that have not yet been purged
- **store** indicates the storage space used by the node
- **indexing** the number of indexed documents
- **get** : Get-by-ID request statistics
- **search** : the number of active searches, total number of queries the cumulative execution time of queries
- **merges** : merging segments of Lucene
- **filter_cache** : the memory occupied by the filter cache
- **id_cache** memory usage distribution
- **field_data** memory used for temporary calculation data
- **segments** the number of Lucene segments. (should be 50-150)



OS, Process, JVM sections

These are the classic metrics on CPU load and memory usage at the system level plus classic metrics on JVM.

Concerning the JVM, the main metrics to monitor is the time spent in garbage collecting : ***collection_time_in_millis***



Thread pools section

ELS maintains thread pools for its internal tasks.

Typically, there is no need to configure these pools.



I/O : File system and network

ELS provides information about your file system: Free space, data directories, disk IO statistics

There are also sections on **network**

- **transport** : basic statistics on inter-node or client communications (TCP port 9300)

- **http** : HTTP port statistics.

If we observe a very large number of constantly increasing open connections, this means that one of the HTTP clients is not using keep-alive connections which is very important for the performance of ELS



Index stats API

The **index stats API** allows you to view statistics about an index

GET my_index/_stats

The result is similar to the node-stats output: Search count, get count, segments, ...



Update of the configuration

Most ELS configurations are dynamic, they can be updated by the API.

The cluster-update API operates in 2 modes:

- ***transient*** : Changes are reverted on restart
- ***persistent*** : The changes are permanent. On reboot, they overwrite the values of the configuration files



Example

```
PUT /_cluster/settings
{
  "persistent" : {
    "logger.discovery" : "WARN"
  },
  "transient" : {
    "indices.recovery.*" : null
  }
}
```

Resetting all the properties under *indices.recovery* to default values



Monitoring and Operations

Logs

Low level Cluster API

Monitoring with beats

Reindexation, Snapshot and Restore

Resizing and restarting the cluster



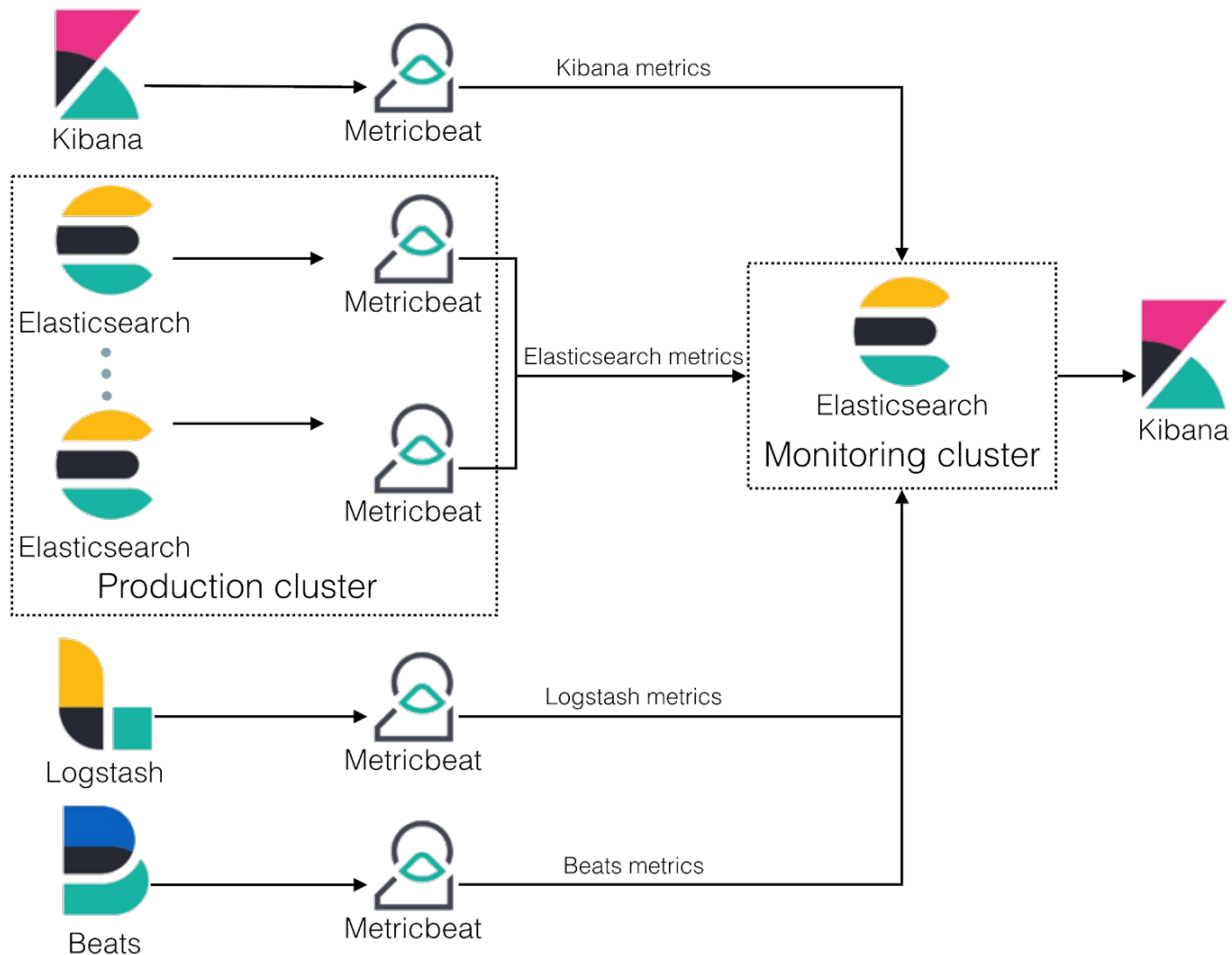
Introduction

Beats are the more natural way to centralize monitoring data into ES indices.

- Metricbeats conveys most of performance data
- Filebeat can aggregate logs
- Other beats like PacketBeat, Auditbeat can also be used

In a production environment, it is recommended to set up a dedicated cluster to monitoring.

Dedicated cluster





Set up metricbeat

1. Enable monitoring on the production cluster

```
PUT _cluster/settings {  
  "persistent": {  
    "xpack.monitoring.collection.enabled": true  
  }  
}
```

2. Install metricbeats :

- Either on each production node
- Or a single installation with *scope: cluster* and configure hosts to point to an endpoint (e.g. a load-balancing proxy) which directs requests to the master-eligible nodes in the cluster

3. Enable the Elasticsearch module in Metricbeat

```
metricbeat modules enable elasticsearch-xpack
```

4. Configure the Elasticsearch module in Metricbeat. Mainly Ips of the monitored cluster and the output to the dedicated cluster

5. Optionally disable the module system

```
metricbeat modules disable system
```



Set up metricbeat (2)

6. Start metricbeat on each node

7. Disable the default collection of Elasticsearch monitoring metrics on the monitoring cluster

```
PUT _cluster/settings
```

```
{  
  "persistent": {  
    "xpack.monitoring.elasticsearch.collection.enabled": false  
  }  
}
```

8. View the monitoring data in kibana



Kibana's configuration

By default, data is retrieved from the cluster specified in the *elasticsearch.hosts* value in the *kibana.yml* file.

If we want to retrieve it from a different cluster, use the property *monitoring.ui.elasticsearch.hosts*

Then go to *Stack Monitoring*

Kibana dashboard

Elasticsearch

Overview

| | |
|-----------------------|--|
| Health | ● Healthy |
| Version | 8.0.0 |
| Uptime | 12 days |
| Machine learning jobs | 4 |
| License | Platinum expires on December 31, 2022 |

Nodes: 13

| | |
|----------------|-----------------------------|
| Disk Available | 71.26% 4.1 TB / 5.8 TB |
| JVM Heap | 50.16% 16.4 GB / 32.6 GB |

Indices: 407

| | |
|----------------|---------------|
| Documents | 2,403,684,500 |
| Disk Usage | 1.7 TB |
| Primary Shards | 407 |
| Replica Shards | 167 |

Logs

🔔 No logs for Elasticsearch
Follow [these directions](#) to set up Elasticsearch.

Kibana ● Healthy

Overview

| | |
|--------------------|------|
| Requests | 0 |
| Max. Response Time | 0 ms |

Instances: 2

| | |
|--------------|-----------------------------|
| Connections | 0 |
| Memory Usage | 10.96% 908.5 MB / 8.1 GB |



Set up filebeat (1)

Filebeat can be used to monitor the Elasticsearch log files, and ship them to the monitoring cluster.

Recent logs are visible on the *Monitoring* page in Kibana.

1. Verify that Elasticsearch is running and that the monitoring cluster is ready to receive data from Filebeat.

2. Enable monitoring on the production cluster

```
PUT _cluster/settings {  
  "persistent": { "xpack.monitoring.collection.enabled": true } }
```

3. Identify which logs you want to monitor.

Filebeat can handle audit, deprecation, gc, server, and slow logs.

4. Install Filebeat on each node of the production cluster

5. Identify where to send data. *output.elasticsearch* property



Set up filebeat (2)

6. Identify the instance of kibana to visualize logs

Property : `setup.kibana`

7. Enable the Elasticsearch module and set up the initial Filebeat environment on each node.

```
filebeat modules enable elasticsearch
```

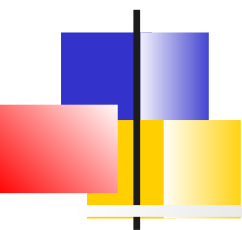
```
filebeat setup -e
```

8. Configure the Elasticsearch module in Filebeat on each node.
In particular path to logs

9. Start *filebeat* on each node

10. Check that the indices are created (filebeat-*)

11. View the monitoring data in kibana



Monitoring and Operations

Logs

Low level Cluster API

Monitoring with beats

**Reindexation, Snapshot and
Restore**

Resizing and restarting the cluster



Reindexing

It is not possible to make some changes a posteriori to an index. (addition of analyzer for example)

To reorganize an index, the only solution is to reindex, i.e. create a new index with the new configuration and copy all documents from the old index to the new one.

The easiest way is to use the `_reindex` API



API *_reindex*

The ***_reindex*** API consists of copying documents from one index to another index.

The target index can have a different configuration than the source index (replica, shard, mapping)

```
POST _reindex?wait_for_completion=false
```

```
{
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitter"
  }
}
```



API *_reindex*

It is possible to filter the copied documents and even to query a remote cluster.

```
POST _reindex
{
  "source": {
    "remote": {
      "host": "http://otherhost:9200", // Cluster distant
      "socket_timeout": "1m",
      "connect_timeout": "10s"
    },
    "index": "source", // Sélection de l'index
    "size": 10000, // limitation sur la taille
    "query": {
      "match": { // limitation par une query
        "test": "data"
      }
    }
  },
  "dest": {
    "index": "dest"
  }
}
```




Sample : resizing script

```
#!/bin/bash
for index in $(list of indexes); do
  documents=$(curl -XGET http://cluster:9200/${index}/_count 2>/dev/null | cut -f 2 -d : | cut -f 1 -d ',')
  # Sizing of the number of shards according to the number of documents
  if [ $counter -lt 4000000 ]; then
    shards=1
  elif [ $counter -lt 5000000 ]; then
    shards=2
  else
    shards=$(( $counter / 5000000 + 1 ))
  fi

  new_version=$(( $(echo ${index} | cut -f 1 -d _) + 1 ))
  index_name=$(echo ${index} | cut -f 2 -d _)

  curl -XPUT http://cluster:9200/${new_version}${index_name} -d '{
    "number_of_shards" : '${shards}'
  }'
  curl -XPOST http://cluster:9200/_reindex -d '{
    "source": {
      "index": "'${index}'"
    },
    "dest": {
      "index": "'${new_version}${index_name}'"
    }
  }'
done
```



Backup

To backup a cluster, the snapshot API can be used

This takes the current state of the cluster and its data and stores it in a shared repository

The first snapshot is full, the others save the deltas

Repositories can be of different types

- Shared directory (NAS for example)
- Amazon S3
- HDFS (Hadoop Distributed File System)
- Azure Cloud



Simple usage

```
PUT _snapshot/my_backup
```

```
{
```

```
"type": "fs",
```

```
"settings": {
```

```
  "location": "/mount/backups/my_backup"
```

```
} }
```

Then

```
PUT _snapshot/my_backup/snapshot_1
```

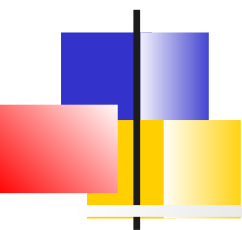


Restoring

`POST _snapshot/my_backup/snapshot_1/_restore`

The default behavior is to restore all existing indexes in the snapshot

It is also possible to specify the indexes that we want to restore



Monitoring and Operations

Logs

Low level Cluster API

Monitoring with beats

Reindexation, Snapshot and Restore

Resizing and restarting the cluster



Adding new nodes

Without security adding a new node consist start ELS on a new machine after have configured correctly :

- ***cluster.name*** : A cluster is a group of nodes that have the same cluster.name attribute
- ***discovery.seed_hosts*** : In order that the new node can discover the rest of its cluster.
- Configure a pingable IP from the other nodes in ***transport.host***

With security, you must enroll the new node in the cluster with an enrolment token generated by ***elasticsearch-create-enrollment-token***.

This token allows to copy all the certificates files needed for inter-nodes communication

Specific attention is required for master-eligible nodes



Master eligible nodes

It is recommended to have a small and fixed number of master-eligible nodes in a cluster, and to scale the cluster up and down by adding and removing master-ineligible nodes only.

If we had to add/remove master nodes, the cluster's voting configuration, (the set of master-eligible nodes whose responses are counted) must be updated

Process of adding a master-eligible nodes is equivalent to a normal node

But during removing, we must be sure that voting can be performed.

If we have 3 master, just stop one at a time otherwise the cluster becomes unavailable



Full-cluster Restart

The procedure is as follows :

1) Disable shard allocation.

```
PUT _cluster/settings
{ "persistent": { "cluster.routing.allocation.enable": "primaries" } }
```

2) Stop indexing and perform a flush.

```
POST /_flush
```

3) Shut down all nodes

4) Perform changes

5) Restart each nodes

6) Wait for all nodes to join the cluster and report a status of yellow

7) Re-enable allocation.

```
PUT _cluster/settings
{ "persistent": { "cluster.routing.allocation.enable": null } }
```

8) You can monitor process with

```
GET _cat/health
GET _cat/recovery
```




Rolling restart

The aim is to have an uninterrupted service

- 1) Disable shard allocation.
- 2) Stop non-essential indexing and perform a flush
- 3) Shut down a single node
- 4) Perform any needed changes.
- 5) Restart the node you changed.
- 6) Reenable shard allocation.
- 7) When the node has recovered and the cluster is stable, repeat these steps for each node that needs to be changed



Security

Automatic security with ELS 8.x

Manual configuration

User authentication and authorization

Audit logging, IP filtering



Security principles

Never run an Elasticsearch cluster without security enabled

Never run Elasticsearch as the root user

Never expose Elasticsearch to public internet traffic

Define roles for your users and assign appropriate privileges to ensure that users have access only to the resources that they need



First start

With ELS8.x, at the first start, the security configuration occurs automatically:

- Certificates and keys for TLS are generated for the transport and HTTP layers.
- The TLS configuration settings are written to `elasticsearch.yml`.
- A password is generated for the elastic user.
- An enrollment token is generated for Kibana.

Then Kibana can be started with the enrollment token which

- applies the security settings from the Elasticsearch cluster,
- authenticates to Elasticsearch with the built-in kibana service account
- writes the security configuration to *kibana.yml*



Enroll additional nodes

At first start, the transport layer is bound to localhost

Before adding other nodes, we must

- Bound to an adress other than localhost
- Satisfy bootstrap checks
- Create an enrollment token
`bin/elasticsearch-create-enrollment-token -s node`
- Start the new node with this enrollment token
`bin/elasticsearch --enrollment-token <token>`

Then, ELS generates certificates and keys in
config/certs



Elasticsearch clients

Elasticsearch Clients (Custom programs, logstash, beats, ...) must trust the certificate used for HTTPS

- Either with its fingerprint
- Either with the certificate itself



Certificates and keys

http_ca.crt : The CA certificate that is used to sign the certificates for the HTTP layer of this Elasticsearch cluster.

http.p12 : Password-protected keystore that contains the key and certificate for the HTTP layer for this node.

To retrieve the password :

```
bin/elasticsearch-keystore show  
xpack.security.http.ssl.keystore.secure_password
```

transport.p12 : Password-protected keystore that contains the key and certificate for the transport layer for all the nodes in your cluster.

To retrieve the password :

```
bin/elasticsearch-keystore show  
xpack.security.transport.ssl.keystore.secure_password
```



Security

Automatic security with ELS 8.x

Manual configuration

User authentication and authorization

Audit logging, IP filtering



Introduction

3 layers :

- Authentication of users
- TLS internode communications
- TLS for http communications



Command line tools

ELS provide many command line tools to easy the manual set up of security :

- ***elasticsearch-certutil*** : creation of certificates to use with Transport Layer Security (TLS).
- ***elasticsearch-create-enrollment-token*** : Creation of enrollment tokens for nodes and Kibana instances.
- ***elasticsearch-keystore*** : Manages secure settings in the ELS keystore.
- ***elasticsearch-reset-password*** : Resets the passwords of users in the native realm and built-in users
- ***elasticsearch-syskeygen*** : Creation of a system key file in the elasticsearch config directory. (Symmetric cryptography)
- ***elasticsearch-users*** : Add/remove users, assign roles, and manage passwords per node



Authentication of users

The process enables to enable Elasticsearch security features and then create passwords for built-in users. More users can be added later.

1) On every node of the cluster set

```
xpack.security.enabled: true
```

2) On any node, set passwords for built-in users.

```
./bin/elasticsearch-reset-password -u elastic
```

```
./bin/elasticsearch-reset-password -u kibana_system
```



Kibana's configuration

Users may log in to kibana with a valid username and password but kibana will use the *kibana_system* account to access ELS

1) In kibana.yml

```
elasticsearch.username: "kibana_system"
```

2) Store the password in a keystore

```
./bin/kibana-keystore create
```

```
./bin/kibana-keystore add elasticsearch.password
```

3) Restart kibana

4) Log with elastic user in order to create new users



TLS between nodes

It is the basic security mechanism to prevent unauthorized nodes from accessing to the cluster.

Inter-communications are then crypted and authenticated

First Step : Certificate

1) On any single node, generate the certificate authority

`./bin/elasticsearch-certutil ca`

=> *elastic-stack-ca.p12* which contains the public certificate for the CA and the private key used to sign certificates

2) On any single node, generate a certificate and private key for the nodes in your cluster.

`./bin/elasticsearch-certutil cert --ca elastic-stack-ca.p12`

3) On every node in your cluster, copy

`elastic-certificates.p12`



TLS (2)

Second Step : Node's configuration

For each node

- 1) In *elasticsearch.yml*, enable internode communication and provide access to the node's certificate.

```
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: certificate
xpack.security.transport.ssl.client_authentication: required
xpack.security.transport.ssl.keystore.path: elastic-certificates.p12
xpack.security.transport.ssl.truststore.path: elastic-certificates.p12
```

- 2) If you entered a password when creating the node certificate, Store the password in the ELS keystore:

```
./bin/elasticsearch-keystore add
xpack.security.transport.ssl.keystore.secure_password
./bin/elasticsearch-keystore add
xpack.security.transport.ssl.truststore.secure_password
```



https

TLS on the HTTP layer ensures that all communications to and from your cluster are encrypted.

Certificates can be

- Self-signed, the CA for TLS intercommunication can be used
- Signed by an external CA. Then you must provide a CSR

Client truststore may be updated



Setup of a self-signed

1) On any single node, generates a .zip which contains certificates and keys used by ELS and Kibana

./bin/elasticsearch-certutil http

- a) Do not generate a CSR.
- b) Use an existing CA
- c) Enter the path of the CA. : elastic-stack-ca.p12 and its password.
- d) Enter an expiration value .
- e) Generate one certificate per node.
 - => Each certificate will have its own private key, and will be issued for a specific hostname or IP address.
- f) Enter the name of the node in your cluster.
- g) Enter all hostnames used to connect to this node.
- h) List every hostname used to connect to your cluster over HTTPS.
- i) Enter the IP addresses that clients can use to connect to your node.
- j) Repeat these steps for each additional node in your cluster.



Setup of a self-signed (2)

- 2) After generating a certificate for each of your nodes, enter a password for your private key
- 3) Unzip the generated *elasticsearch-ssl-http.zip* which contains one directory for both Elasticsearch and Kibana.
- 4) On every node in your cluster,
 - a) Copy the relevant *http.p12* certificate to the conf directory.
 - b) In *elasticsearch.yml* :

```
xpack.security.http.ssl.enabled: true
xpack.security.http.ssl.keystore.path: http.p12
```
 - c) Add the password for the private key

```
./bin/elasticsearch-keystore add
xpack.security.http.ssl.keystore.secure_password
```



Kibana's configuration

The *elasticsearch-ca.pem* file contained in the generated zip is used as a truststore by Kibana

1) Copy *elasticsearch-ca.pem* to the Kibana configuration directory

2) In `kibana.yml`

```
elasticsearch.ssl.certificateAuthorities:  
$KBN_PATH_CONF/elasticsearch-ca.pem
```

```
elasticsearch.hosts:  
https://<your_elasticsearch_host>:9200
```

3) Restart Kibana



Enable TLS to access Kibana

To enable https between browser and Kibana :

1)Generate a server certificate and private key for Kibana.

```
./bin/elasticsearch-certutil csr -name kibana-server -dns  
example.com, www.example.com
```

This will generate a *csr-bundle.zip* which contains a CSR
and a private key

2)Unzip the archive and send the CSR to a CA authority
(Let's Encrypt, ...) for signing.

CA authority will generally provide a crt file

3)Edit kibana.yml

```
server.ssl.certificate: $KBN_PATH_CONF/kibana-server.crt  
server.ssl.key: $KBN_PATH_CONF/kibana-server.key  
server.ssl.enabled: true
```



Updating the certificates

ELS provides an API to check the expiration date of your certificates
GET `/_ssl/certificates`

Generally, you just have to copy at the same location with the same name, the renewed certificate.

ELS will load it automatically without any restart



Security

Automatic security with ELS 8.x

Manual configuration

**User authentication and
authorization**

Audit logging, IP filtering



Realms

ELS comes with several built-in users assigned to static roles:

- *elastic* : Super user
- *kibana-system* : The identity of the Kibana client
- *logstash_system* : The identity of the logstash client
- *beats_system* : The identity used to store monitoring information
- *remote_monitoring_user* : The identity used by Metricbeats to collect and store monitoring information

Super user can access the realm to create new users and assign them role :

- Via the Kibana UI
- The REST API
- The command line tool

It is also possible to integrate the ELS Realm with an external system like *LDAP, saml, kerberos, jwt, oidc, ...*



Authentication

Depending on the configured realms, user credentials must be provided to authenticate clients :

- With the internal realm, usernames and passwords can be provided with basic auth header to the requests.

It is also possible to use tokens.

- The token service, the API key service and *service-accounts* can be used to exchange the current authentication for a token or key.
- This token or key can then be used as credentials for authenticating new requests.

The API key service is enabled by default. The token service is enabled by default when TLS/SSL is enabled for HTTP.



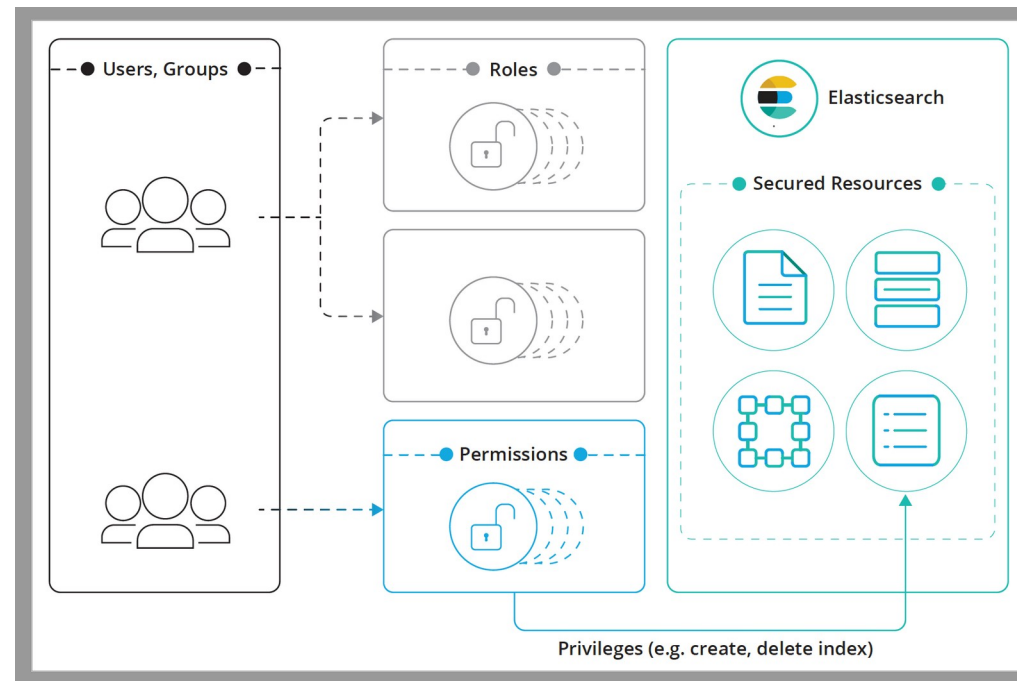
Token based authentication services

3 mechanisms to authenticate via token

- **service-account** is used for external services
- **token-service** use the Get Token API and generate a short-lived token and refresh token based on OAuth2
POST /_security/oauth2/token
with standard OAuth2 parameter provided in the body of the request
- **api-key-service** uses the create API key to generate API keys. By default, keys do not expire
POST /_security/api_key

User authorization

ELK provide a role-based access control (RBAC) mechanism, which enables you to authorize users by assigning privileges to roles and assigning roles to users or groups.





Built-in roles

They are many built-in roles but we can define other roles by giving permissions on ressource.

Built-in roles :

- Superuser
- kibana_admin
- monitoring_user
- ...



Security

Automatic security with ELS 8.x
Manual configuration
User authentication and authorization
Audit logging, IP filtering



Enable audit logging

It is possible to log security-related events such as authentication failures and refused connections

```
xpack.security.audit.enabled : true
```



IP filtering

It is possible to apply IP filtering to application clients, node clients, or transport clients

Configuration is made with :

```
xpack.security.transport.filter.allow  
xpack.security.transport.filter.deny
```

Example :

```
xpack.security.transport.filter.allow: "192.168.0.1"  
xpack.security.transport.filter.deny: "192.168.0.0/24"
```



Thank You for your
attention !!