**ELASTICSEARCH**

Table des matières

[Chapitre 1 Introduction 2](#_Toc105575878)

[Chapitre 2 Ecosystème ELK (Es Logstash Kibana) 2](#_Toc105575879)

[1-Avant : 2](#_Toc105575880)

[2-Après : 3](#_Toc105575881)

[Chapitre 3 Use cases 4](#_Toc105575882)

[Chapitre 4 : CLUSTER ELASTICSEARCH : 4](#_Toc105575883)

[1-Master node : (Mâtre) 4](#_Toc105575884)

[2-Data Node : 4](#_Toc105575885)

[3-Ingest : 4](#_Toc105575886)

[Chapitre 5 Schéma de données : 5](#_Toc105575887)

[1-Indexe : 5](#_Toc105575888)

[2-Type : 5](#_Toc105575889)

[3-le document : 5](#_Toc105575890)

[4-Field : 5](#_Toc105575891)

[Chapitre 6 Types de field : 5](#_Toc105575892)

[Chapitre 7 Shards Replicats & segments : 6](#_Toc105575893)

[1-Sharding : 6](#_Toc105575894)

[2-Replication : 6](#_Toc105575895)

[3-Segment : 6](#_Toc105575896)

[Chapitre 8 Indexation des documents : 6](#_Toc105575897)

[Chapitre 9 Lecture, Update & suppression d’un document 6](#_Toc105575898)

[1-lecture : 6](#_Toc105575899)

[2-Mises à jour (update) : 7](#_Toc105575900)

[3-Suppression : 7](#_Toc105575901)

[Chapitre 10 Basic Crud 7](#_Toc105575902)

# 

# Introduction

Elasticsearch = ES

ES est une base de données noSql orientée documents distribuée, moteur d’indexation de recherche.

ES n’est pas sur HDFS ( a son propre cluster), peut être déployé sur windows

Indexe = base de données.

Apache Lucene : moteur d’indexation non distribué (un seul nœud.

Es basé sur Appache Lucene (chaque nœud) : intéressant mais compliqué à gérer (code)

Google à chaque nouveau site, il le détecte, il indexe les éléments importants du site, par des requêtes https puis l’algorithme

Recherche multi critères très compliquée sur une page (Lucene) (il se passe beaucoup de traitements derrière code java) et ES masque la complexité qui existe derrière.

ES stock les données en Json.

API Rest est supportée par tous les langages d’api Rest ( postman, SOPWI)

Pugin pour tout

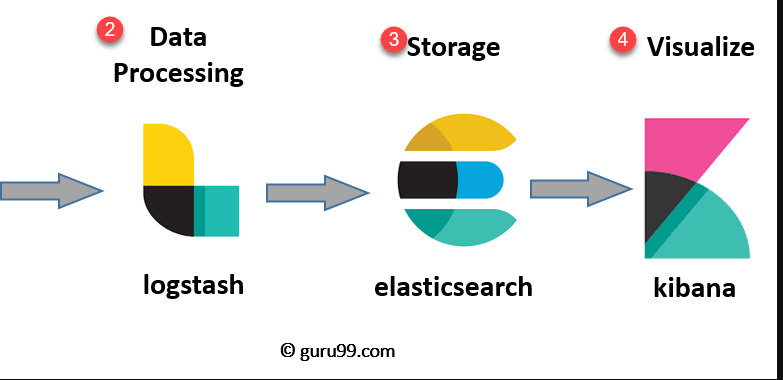
CMS (Content management system) : framework qui permet de créer des pages web facilement

# Ecosystème ELK (Es Logstash Kibana)

## 1-Avant :

Au début ES est dédié à Logstash des logs

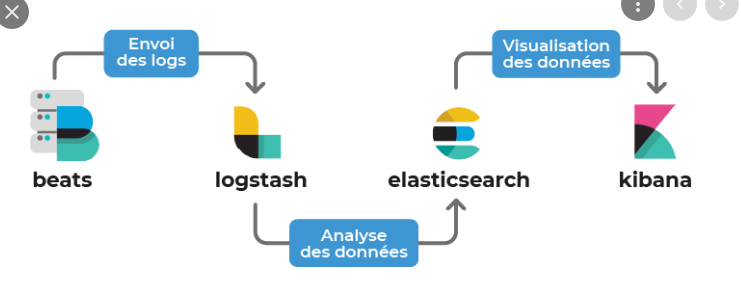
Logstash = ETL : récupère les logs les transformes et les charger dans ES.

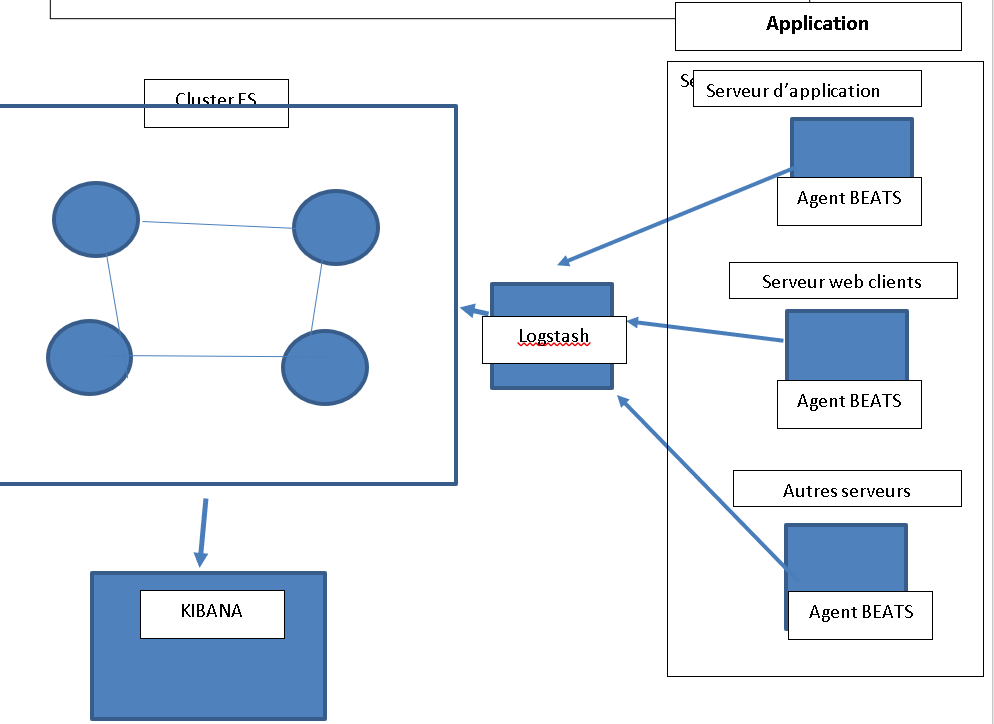


## 2-Après :

Agent logstash ES consomme de la mémoire donc remplacé par beats (agent light)

Beats : log shipping (sans transformation)





# Use cases

On peut utiliser Es dans plusieurs uses cases :

* Analyse de log
* Gestion d’évènements : alertes, notification, publication
* Recherche textuelle
* Analytics & agrégation : données dénormalisées : un document json peut contenir une ligne résultante de jointure de plusieurs tables d’une base de données relationnelle.
* Alrting grace à l’API Percolator
* Analyse des times series
* Etc …

# : CLUSTER ELASTICSEARCH :

Un cluster peut avoir un ou plusieurs nœuds,

Un nœud est une instance de ES en cours.

Les nœuds communiquent grace au protocole TCP (parce les communication TCP sont plus rapides que les requetes https)

Par contre le client est obligé de faire appel à une requête rest pour pouvoir communiquer avec ES..

Un shard = l’unité atomique qui stock dans un noeud

Un cluster ES est composé de 3 types de nœuds :

## 1-Master node : (Mâtre)

* En plus d’être un nœud il gère la création/suppression d’indexes.
* Tracking des nœuds et de leurs états.
* Allocation de shards aux nœuds.
* A chaque instant, il n’ y a qu’un seul Master (par defaut tous les nœuds peuvent devenir master).
* Il est important d’avoir plusieurs nœuds éligibles à devenir master

## 2-Data Node :

* Gere les shards
* Il réalise les opérations crud, & d’aggrégation

## 3-Ingest :

On peut s’en passer de ne pas avoir celui la.

On le met en place dans le cas où on a beaucoup à une haute fréquence de pipelines de données de même type, donc ce nœud sera chargé a faire des opérations de préprocessing, filtres par exemple (sélectionner les données qu’on ingeste dans ES) : cette stratégie on la met en place pour décharger log stash que pour la transformation.

# Schéma de données :

## 1-Indexe :

1ère chose à créer dans ES.

Indexe est une collection : un ensemble de documents Json. On peut les comparer un Base de données relationnelle car un fichier Json peut contenir des informations de plusieurs tables dans une BDD-R

## 2-Type :

Vient après l’index

C’est un premier filtre ( si on fait de requête sur un filtre on ne reçoit pas de réponse de la par de ES) => une abstraction permettant de partitionner les documents similaires mais pas identiques.

C’est comme une colonne family dans HBASE (schéma défini à l’insertion).

A partir de la version 1 chaque Indexe a un seul type.

## 3-le document :

Format Json, conserne 1 seul type d’enregistrement.

Document valide : respecte le format Json

Un identifiant unique sera généré de deux façons :

1. Soit d’une façon automatique pour chaque document envoyé (paramètre par défaut de ES)
2. Soit en le faisant nous en ajoutant id\_

Les identifiant servent surtout dans les opérations de mise à jour des documents (cibler le bon document)

## 4-Field :

C’est le contenu de chaque document json

# Types de field :

ES a développé des algorithmes propres à elle pour déterminer des types de données à partir de leurs structures.

Si le format logique de Json n’est pas respecté ES refusera de l’indexer

***NB :***Jason est plus light et moins chiant que XML pour le faire valider,

# Shards Replicats & segments :

## 1-Sharding :

C’est comparable aux partitions dans des topics kafka

Quand on crée l’index on définit le nombre de shards

Si on a un indexe avec 1 seul shard => on exploite pas le cluster car on sollicite tjrs le même nœud

L’indexe n’existe qu’à travers les différents shards.

Si on veut modifier le nombre des shards on doit réindexer tous les documents dans un autre indexe.

***NB :*** Ca ne sert à rien de créer deux shards du même indexe sur un seul nœud

## 2-Replication :

Chaque shard est répliqué une ou plusieurs fois pour assurer la tolérance aux pannes .

Un shard primaire et ses réplicas sont localisés sur des serveurs différents.

Le nombre de réplica peut être modifié même après la création d’index.

## 3-Segment :

Quand on indexe les données sont dans un buffer memory 🡺 flush sur disk (segment = fichier)

Le shard finalement est aussi une abstraction qui n’existe qu’à travers les segments (la réplication se fait après la ségmentation)

Les segments sont régulièrement mergés (comapction major dans hbase).

Shard = repertoire simple

Index = repertoire distribué

# Indexation des documents :

Un client peut soumettre sa reqête à n’importe quel nœud du cluster. Le nœud qui reçoit l’envoie vers le nœud qui gère le shard  primaire auquel appartient le document.

Le document par suite est envoyé par le shard primaire aux shards secondaires

L’acquitement renvoyé au client dépend du paramètre de la requêtede put : « wait\_active\_shard=x »

# Lecture, Update & suppression d’un document

## 1-lecture :

La requête get permet de renvoyer un document et peut être adressé à un shard primaire ou secondaire

## 2-Mises à jour (update) :

Es ne fait pas de mise à jour Es supprime le document et indexe un nouveau document avec les mises à jour appliquées.

Donc il faut faire attention car on n’a pas l’historique

## 3-Suppression :

A la suppression d’un document le shard enregistre dans 1 bitset le fait que le document soit supprimé.

La vraie suppression aura lieu au merge des segments du shard.

Entre le merge et l’enregistrement les opérations de recherche se font en filtrant simplement les documents supprimés.

# Basic Crud

C : create

R : Read,

U : Update

D : Delete

Es expose plusieurs end point (de l’API Rest) pour pour pouvoir intéragir avec elle.

**Idexer un document dont le id est controlé par l’utilisateur :**

La reponse : {‘’\_index ‘’ : ‘’index name’’

‘’\_type’’ : ‘’ type\_name‘’

‘’\_id’’ : ‘’ id\_name‘’

‘’\_version’’ : ‘ 1‘,

‘’\_created’’ : True}

Pour indexer : Put/{ idex } /{ type }/{id}

{

Le corps du fichier json

}

**Idexer un document dont le id est auto généré :**

La reponse : {‘’\_index ‘’ : ‘’index name’’

‘’\_type’’ : ‘’ type\_name‘’

‘’\_id’’ : ‘’ id\_name‘’

‘’\_version’’ : ‘ 1‘,

‘\_’created’’ : True}

Pour indexer : Put/{ idex } /{ type }

{

Le corps du fichier json

}

**Retrouver un document**

La reponse : {‘’\_index ‘’ : ‘’index name’’

‘’\_type’’ : ‘’ type\_name‘’

‘’\_id’’ : ‘’ id\_name‘’

‘’\_version’’ : ‘ 1‘,

‘’\_created’’ : True

‘’\_source’’:{ }}

}

Pour retrouver : GET/{ index}/{ id}

{

Le corps du fichier json

}

**Suppression d’un document :**

La reponse {‘’\_shard’’:

{‘’total’’ : 2,

‘’failed’’:0,

‘’successfully’’: 2},

‘’\_idex’’: ‘’twitter’’

}

}

Pour indexer : DELETE/{ index}/{ type }/{ id}

{

Le corps du fichier json

}

**Mettre à jour  un document :**

# API BULK :

Permet de Regrouper les requêtes (plusieurs requêtes d’indexation / suppression/

+ mise à jour en 1 seul appel