ELASTICSEARCH

MOTEUR DE RECHERCHE

Arthur VEYS, Vincent COUTURIER

MOTEUR DE RECHERCHE ?

- Et ce n'est pas faux ! Ce sont des moteurs de recherche appliqués aux pages web.
- Le but d'un moteur de recherche est de deviner ce que l'utilisateur souhaite voir comme contenu et de trouver ce qui correspond le plus
- C'est un vocabulaire différent de celui des SGBDR et du NoSQL :
  - Je cherche tous les films ayant comme titre « Batman » VS Je cherche tous les films ayant dans le titre un mot qui ressemble à Batman
  - Exactitude vs ressemblance

POURQUOI UN MOTEUR DE RECHERCHE ?

- Comprendre le langage humain :
  - USA = United States of America
  - Saut, sautant, sautillant, sauter, … sont des mots de la même famille, du même sens
  - Diminuer les erreurs typographiques et phonétiques (ex : « Johnie Halliday »)
- A l'ère du Big Data, un moteur de recherche se doit d'être capable de faire des requêtes sur plusieurs millions de documents en moins de quelques centaines de ms
  - Scalabilité et résilience (=capacité d'un système à continuer de fonctionner en cas de panne, d'incident intentionnelle ou non et/ou de sollicitation extrême) sont des valeurs clés
- Quelle utilité ?
  - Recherche rapide sur les données de l'entreprise (suggestion, recherche de documents)
  - Centralisation, stockage et indexation de logs systèmes.

LES MOTEURS DE RECHERCHE DU MARCHÉ

- Open source
  - Lucene
    - Elasticsearch
    - Solr
  - …
- Propriétaire
  - Exalead
  - Oracle Text
  - …

ELASTICSEARCH

LE PROJET

- Créé en 2010 par Shay Bannon des cendres de Compass (solution de recherche textuelle)
  - Produit open-source
- Création d'une entreprise de services commerciaux et produits supplémentaires en 2012 (Siège social à Amsterdam)
  - Aujourd'hui valorisé à plus de 700M de $
  - Rachat et intégration d'outil pour elastic (Logstash, Kibana, Beats, Found, …)
- Produit dispo en SaaS (cher mais pratique)
- Version actuelle : 6.6.0

ELASTICSEARCH

- Basé sur le moteur d'indexation Apache Lucene
  - Moteur de recherche distribué
  - Multi-entité (Documents JSON)
  - Accessible via API REST
- N'a besoin que d'une JVM 1.8 pour fonctionner
- Utilisé par toutes les grandes entreprises du web
  - Netflix, Discord, Facebook, Mozilla, GitHub, OVH, …

UN PEU DE VOCABULAIRE

- Faire abstraction de toutes ses connaissances en SQL

Relational DB ⇒ Databases ⇒ Tables ⇒ Rows ⇒ Columns
Elasticsearch ⇒ Index ⇒ (Types)* ⇒ Documents ⇒ Fields

- Index
  - Index (**Nom**)
    - Un index est l'équivalent d'une base de donnée sur les SGBD relationnel
  - Indexer (**Verbe**)
    - *Indexer un document* est le fait de le stocker dans un index (nom) pour y être sauvegardé et requêté.

* *Les types sont en train de disparaitre*

REST ET CRUD SUR ELASTICSEARCH

- Fonctionne sur API REST
  - Implémentation du CRUD sur HTTP
    - GET : récupération de document ou exécution de requête sans payload
    - POST : exécution de requête avec payload (document)
    - DELETE : suppression de document ou index
  - Chemin d'accès : <HOST>:<PORT>/<PATH>?<QUERY_STRING>

EXEMPLE DE DOCUMENT

GET INDEX/TYPE/ID

DE L'INDEXATION …

- Rappel sur l'indexation
  - Un index est une sorte d'annuaire où une clé décrit un ensemble de documents qui ont un lien plus ou moins fort envers cette clé
  - Les premiers index étaient une simple liste des occurrences de mots dans un texte liées à leur localisation exacte.
  - Aujourd'hui, on construit les index de façon à ce qu'ils représentent mieux le contenu
- Elasticsearch indexe chaque document selon un mapping (une sorte de descripteur du type de chaque champ et de leurs propriétés).
  - Il ajoute à chaque champ des métadonnées: un ID auto généré unique dans tout le cluster, un champ version et un champ *all* de type string qui contient la concaténation de tous les champs de type string

DE L'INDEXATION …

- Elasticsearch reçoit en PUT INDEX/TYPE un nouveau document.
- Lucene va donc commencer par indexer les documents qui arrivent dans l'index
  - Il va commencer par compter le nombre d'occurrences d'un mot dans le texte et les classer par le nombre d'apparitions
  - Il va ensuite supprimer les « stop-words ». Ce sont les mots les plus fréquents dans une langue mais qui apportent peu de sens à la donnée (ex : « un », « de », « les », « of », « the », … .
  - Il va ensuite réduire les mots à leurs racine en enlevant tout ce qui est conjugaison, genre, déclinaison, … Cela permet de cumuler les fréquences sur les mots ayant la même racine et donc le même sens.
- Les deux dernières étapes sont faites par un *analyzer* que l'on peut définir
- Ce travail d'indexation permet de créer ce que l'on appelle un index inversé (*Inverted Index*). (cf. slide suivant)

INDEX INVERSÉ

- Un index inversé est la correspondance entre du contenu et sa position dans un ensemble de données
- Exemple:

"D1" = "c'est ce que c'est"
"D2" = "c'est ceci "
"D3" = "ceci est une banane"

  - Un index inversé sur les mots ou groupes de mots donnerait (le critère d'indexation est la présence du mot dans la phrase) :

"c'" {D1, D2} "est" {D1, D2, D3}
"ce" {D1} "que" {D1}
"ceci" {D2, D3} "une" {D3}
"banane" {D3}

… À LA RECHERCHE

- Etapes d'une recherche Lucene :
  1. Récupération du type de recherche et du mot-clé recherché
  2. Calcul pour chaque document indexé d'une pondération (score) qui correspond à la vraisemblance de son contenu vis-à-vis de la recherche
  3. Application des opérateurs de recherche
  4. Résultats triés par score
- On peut configurer la façon dont le score est calculé
- Le résultat de la recherche dépend du type de champs sur lequel on effectue celle-ci (cf. slide suivant).

EXACT VALUES VS FULL-TEXT

- Les données contenues dans Elasticsearch peuvent être de 2 types différents :
  - Donnée exacte (ex : un id, un username, une adresse mail). Ainsi, « Avion » et « avion » sont deux valeurs différentes.
  - Donnée textuelle : C'est souvent du texte écrit par un humain.
- La recherche sur une donnée exacte est facile pour un ordinateur, c'est une simple vérification d'exactitude.
- A l'inverse, faire une recherche sur du texte est bien plus complexe :
  - « Cerise est sympa mais je préfère Prune » est une phrase ambigüe pour un ordinateur qui ne sait pas si on se réfère à des personnes ou à un fruit.
  - La recherche n'est plus « Est-ce que le document correspond à la recherche ? » mais « Est-ce que le document est semblable à la recherche ? »

MAPPING

- Le mapping (objet JSON) définit la structure des champs d'un index
  - Le type des champs (String, integer, geo_ip, ip, date, …)
  - Les propriétés des champs
  - Les propriétés de l'index
- Lorsque qu'une donnée sera indexée, elle sera traitée selon le type défini dans le mapping.
  - Lorsque qu'un champ ne correspond à aucune configuration du Mapping, ES va appliquer le type qui semble correspondre le plus à la donnée.
- Une fois les données indexées avec un certain Mapping, il est impossible de le changer. On peut néanmoins utiliser une API de réindexation depuis la V5 mais elle nécessite de créer un index cible et son mapping.

```
{
  "mappings": {
    "Fare": {
      "properties": {
        "approachDuration": {
          "type": "long"
        },
        "approachingAt": {
          "type": "date",
                "format":"epoch_millis"
        },
        "approachingAtFromBookedAt": {
          "type": "long"
        },
        "approachingAtFromCreatedAt": {
          "type": "long"
        },
        "approachingAtFromDepartureAt": {
          "type": "long"
        },
        "approachingPosition": {
          "type":"geo_point"
        },
        "attributedBy": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "bonus": {
          "type": "boolean"
        },
        "bookedAt": {
          "type": "date",
                "format":"epoch_millis"
        },
        "bookedAtFromCreatedAt": {
          "type": "long"
        },
        "bookedAtFromDepartureAt": {
          "type": "long"
        },
```

EXEMPLE DE MAPPING

GET INDEX/_mapping

REQUÊTES

- 2 systèmes de requête :
  - Requête lucene
  - Query DSL

RECHERCHE LUCENE

- Système de requêtage simple REST, appliquant un filtre sur les données (GET INDEX/TYPE*/_search?q=<Requete lucene> *optionnel
  - Recherche textuelle
    - paris hilton
      - Recherche de chaque mot dans tous les champs
    - "paris hilton"
      - Recherche de la phrase exacte dans tous les champs
  - Recherche sur un champ particulier
    - name:"hilton"
      - Recherche tous les enregistrements ayant le mot *hilton* dans le champ *name*

RECHERCHE LUCENE

- Recherche générique
  - « ? » est un joker pour un caractère
  - « * » est un joker pour plusieurs caractères
- Recherche entre deux bornes numériques
  - PerfData:[400 TO 10000]
    - Recherche avec le champ PerfData compris entre 400 et 10000
- Opérateurs logiques
  - AND / OR / NOT / + / -
    - Le NOT ne peut pas être utilisé seul, il faut utiliser le « - »
    - Le + signifie que le terme doit être présent obligatoirement dans le document

RECHERCHE LUCENE

- Exemple de requête lucene :
  - status:[400 TO 499] AND (extension:php OR extension:html)
    - Recherche de toutes les lignes de logs apache en erreur (400 à 499) avec le champ extension qui est PHP ou HTML
  - "changing folder" AND ResponseTime:[10000 TO 100000000]
    - Recherche de la chaîne "Changing folder" dans tous les champs et uniquement avec ResponseTime compris entre 10 000 et 10 000 000.

QUERY DSL

- Le Query DSL d'Elasticsearch s'exprime en JSON via requête POST sur INDEX/TYPE*/_search *optionnel
  - Commence toujours par une query

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

  - Suivie par une clause de recherche simple (ci-dessus) ou une clause composée :

```
{
    "bool": {
        "must":     { "match": { "tweet": "elasticsearch" }},
        "must_not": { "match": { "name":  "mary" }},
        "should":   { "match": { "tweet": "full text" }},
        "filter":   { "range": { "age" : { "gt" : 30 }} }
    }
}
```

LES DIFFÉRENTES QUERY

- Match
  - Fait une recherche textuelle sur un champ
  - { "match": { "tweet": "About Search"}}
- Range
  - Recherche de nombre ou dates qui entrent dans une échelle donnée
  - {"range": {"age": {"gte":20,"lt":30}}}
- Term
  - Comme Match mais ne cherche qu'en valeur exacte
  - { "term": { "public": true }}
- Exist et missing
  - Filtre sur l'existence ou non d'un champ
  - { "exist": { "field": "age" }}

COMBINER LES REQUÊTES ENTRE ELLES

- On va utiliser la query Bool
  - Composée de quatre clauses dans lesquelles on peut mettre une ou plusieurs query
    - Must : les documents retournés devront correspondre à toutes les requêtes de ce champ
    - Must_not : Les documents retournés ne devront pas correspondre à ces requêtes
    - Should : Les documents qui correspondent à ces requêtes ont un boost de score.
    - Filter : ressemble au must mais n'influe pas sur le score
- Il est possible d'assembler des bool query dans des bool query

EXEMPLE DE BOOL QUERY

```
{
    "bool": {
        "must":     { "match": { "title": "how to make millions" }},
        "must_not": { "match": { "tag":   "spam" }},
        "should": [
            { "match": { "tag": "starred" }}
        ],
        "filter": {
          "bool": {
              "must": [
                  { "range": { "date": { "gte": "2014-01-01" }}},
                  { "range": { "price": { "lte": 29.99 }}}
              ],
              "must_not": [
                  { "term": { "category": "ebooks" }}
              ]
          }
        }
    }
}
```

PARAMÈTRES D'UNE QUERY DSL

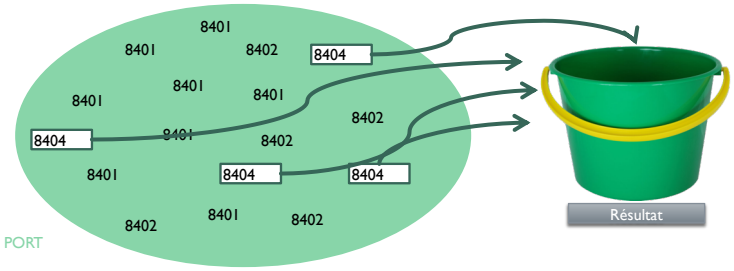
- On peut ajouter à la racine du JSON divers paramètres à la query
- *Size*
  - "size": 20 => on peut définir le nombre de résultats à retourner (10 par défaut)
- *Fields*
  - On peut définir les champs que l'on veut garder dans le retour de la requête

LES AGRÉGATIONS

- Les agrégations permettent d'analyser et de résumer notre set de données par des métriques
  - Combien d'aiguilles y'a t-il dans une botte de paille ?
  - Quelle est la longueur moyenne de mes aiguilles ?
  - Quelle est la longueur médiane de mes aiguilles par producteur ?
  - Combien d'aiguilles ont été rajoutées dans ma botte de paille par mois ?
  - Quelle est le producteur avec le plus d'aiguilles dans ma botte de paille ?
- On peut mélanger query et agrégations pour filtrer les documents à prendre en compte dans cette l'agrégation

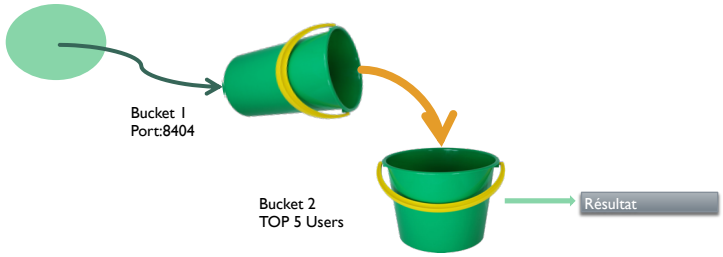
AGRÉGATIONS

- Une agrégation est un filtre permettant de traiter et/ou sélectionner un groupe ou de trouver une valeur particulière (par métrique) tel qu'un minimum.
  - ex : Avoir les données dont le champ Port est égal à 8404
- Les résultats des agrégations sont regroupés dans un « *bucket* »



AGRÉGATIONS

- La puissance d'Elasticsearch vient de la possibilité de faire de multiples sous-agrégations:
  - Le résultat du bucket de la précédente agrégation est utilisé comme source pour le suivant
  - Ex : Récupérer les 5 utilisateurs faisant le plus de transactions sur le port 8404



LES OPÉRATEURS D'AGRÉGATION

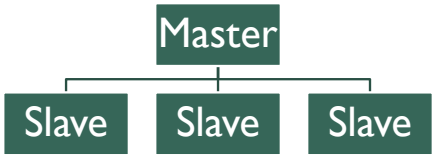
- Agrégation de type métrique
  - Moyenne (Average), médiane
  - Valeur unique (Cardinality)
  - Somme (Sum)
  - Min, max, centile, …
- Agrégation par bucket
  - date_histogram : découpe les données selon une échelle de temps (ex : nombre de documents par tranche de 10min)
  - Histogram : donne le nombre de valeurs contenues dans un intervalle donné (ex: nb de document ayant des perfs entre 0 et 500ms, 500ms et 1000ms, …)
  - Range : retourne les documents contenus dans un ou plusieurs intervalles de temps ou numérique
  - IPV4_range : Retourne les données dont le champ IP est contenu dans un intervalle donné
  - Terms : Catégorise les données selon un champ et sélectionne les X premiers ou derniers (ex : classe les documents par utilisateur et donne les 5 utilisateurs ayant le plus de documents)

LES OPÉRATEURS D'AGRÉGATION (SUITE)

- Agrégation par bucket
  - Filters : regroupe les données par requêtes Lucene (ex : Nombre de documents correspondant à une recherche de supervision, de transaction, …)
  - Significant Terms : Equivalent à Terms mais ressort les données les plus intéressantes et inhabituelles

RÉSILIENCE

- Elasticsearch a été crée dès le début pour être un système scalable et résilient. Le but est de fournir un service continu et sans perte de données
- Il s'appuie pour cela sur un fonctionnement en cluster et sur une partition des données
- Un cluster est un ensemble de nœud ES communiquant ensemble sous la supervision d'un nœud maitre (*Master*). Un cluster permet une scalabilité horizontale



Exemple de cluster 4 nœuds

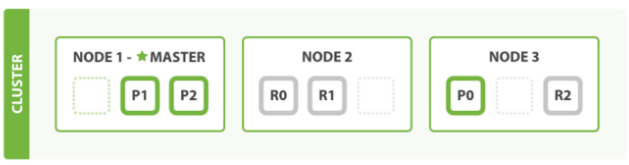
CLUSTERING

- Chaque nouvelle instance d'Elasticsearch va essayer de se connecter à son cluster lors de son démarrage
  - Si aucun nœud ne répond, il devient automatiquement master si sa configuration le permet
  - Si un master répond, il rejoint le cluster
- La charge de recherche et d'indexation est répartie de façon intelligente entre les nœuds (temps de réponse au ping, charge réseau, …)
- Si le serveur du nœud master vient à se déconnecter, les nœuds vont automatiquement réélire un nouveau nœud pour le remplacer.

SHARDING ET REPLICAS

- Chaque donnée indexée par ES va être découpée en morceaux de données (*Shard*) lesquelles seront par la suite réparties sur les différents nœuds du cluster
- Elasticsearch va aussi créer des *replicas* qui seront des copies parfaites des *shards*, qui iront se positionner sur un nœud différent en tant que sauvegarde.
- En cas de défaillance d'un nœud, le master va activer les *shards* de backups et immédiatement faire de nouveaux *replicas*

Figure 4. A three-node cluster—shards have been reallocated to spread the load



AUTRES FONCTIONNALITÉS

- Système de snapshot (backup / restore)
- X-PACK (payant)
  - Sécurité (Shield)
  - Monitoring (Marvel)
  - Alerting (Watcher)
  - Machine Learning (ML)

LOGSTASH

LES CONCEPTS

- C'est un ETL fonctionnant sur un système de plugins
- Traitement en 3 parties :
  - Input : Capture et collecte des données
  - Filter : traitement et normalisation des données
  - Output : Transport de ces données vers une autre destination
- La configuration de Logstash est entièrement programmable et repose sur un système de plugins
  - N'importe qui peut développer son propre plugin (en Ruby)
- Développé par Elastic, il n'est pas forcément exclusif à leurs produits

PRINCIPAUX INPUTS

Plugin	Description
elasticsearch	Reads query results from an Elasticsearch cluster
exec	Captures the output of a shell command as an event
eventlog	Pulls events from the Windows Event Log
file	Streams events from files
http	Receives events over HTTP or HTTPS
http_poller	Decodes the output of an HTTP API into events
imap	Reads mail from an IMAP server
jdbc	Creates events from JDBC data
...	

PRINCIPAUX FILTERS

Plugin	Description
anonymize	Replaces field values with a consistent hash
csv	Parses comma-separated value data into individual fields
checksum	Creates a checksum based on fields in an event
date	Parses dates from fields to use as the Logstash timestamp for an event
geoip	Adds geographical information about an IP address
json	Parses JSON events
json_encode	Serializes a field to JSON
kv	Parses key-value pairs
mutate	Performs mutations on fields
multiline	Merges multiple lines into a single event
sleep	Sleeps for a specified time span
split	Splits multi-line messages into distinct events
grok …	

LES OUTPUTS

Plugin	Description
csv	Writes events to disk in a delimited format
email	Sends email to a specified address when output is received
elasticsearch	Stores logs in Elasticsearch
exec	Runs a command for a matching event
file	Writes events to files on disk
http	Sends events to a generic HTTP or HTTPS endpoint
irc	Writes events to IRC
syslog	Sends events to a syslog server
…	

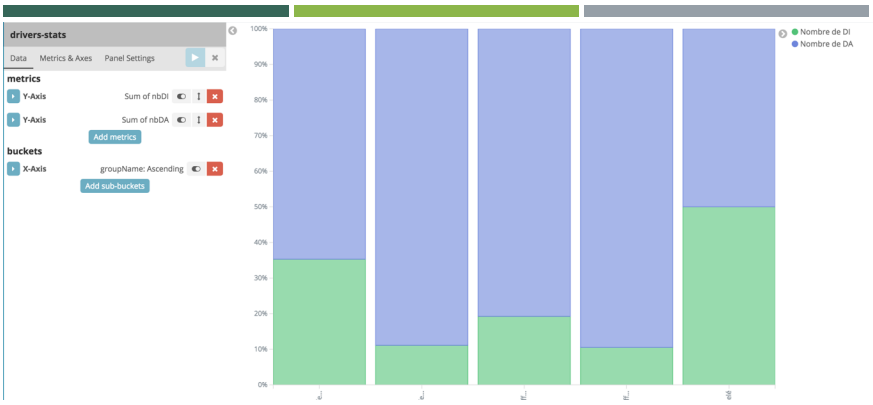
```
input {
    file{
        type => "rkhunter-log"
        path => ["/var/log/rkhunter.log"]
        sincedb_path => "/opt/logstash/.sincedb/rkhunter.since"
        add_field =>{
            "service" => "Braavos"
        }
    }
}
filter {
    if [type] == "audit-syslog" {

        grok {
            match => {"message" => "%{AUDIT}" }
        }

        grok {
            match => {"message" => "%{AUDITLOGIN}" }
        }

        mutate {
            remove_field => [ "tags", "message" ]
        }
    }
}
output {
    tcp {
       host => "hcjnc118"
       mode => "client"
       codec => "json_lines"
       port => "5001"
    }
}
```

CONFIGURATION LOGSTASH

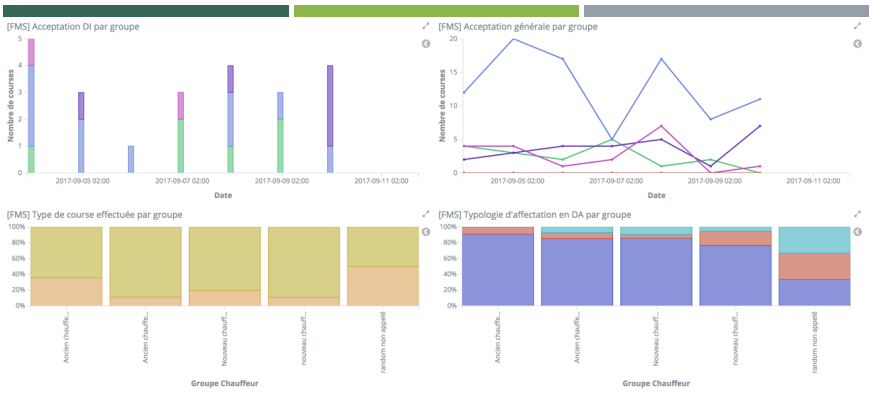
KIBANA

- C'est une UI pour Elasticsearch
  - Interface de recherche de document
  - Interface de création de graphiques venant des agrégations
  - Création de Dashboard interactifs regroupant les graphiques
- Ne se connecte qu'à Elasticsearch
  - Grafana est un fork open-source d'une précédente version de Kibana qui se connecte à d'autre DB
- Nécessite que Node.js soit présent sur la machine hôte

GRAPHIQUES



GRAPHIQUES


[FMS] Acceptation DI par groupe
Nombre de courses
Date
2017-09-05 02:00
2017-09-07 02:00
2017-09-09 02:00
2017-09-11 02:00
[FMS] Acceptation générale par groupe
Nombre de courses
Date
[FMS] Type de course effectuée par groupe
Groupe Chauffeur
[FMS] Typologie d'affectation en DA par groupe
Groupe Chauffeur

DASHBOARD