

Elastic Stack

David THIBAU – 2025

david.thibau@gmail.com

Agenda

- **Introduction**

- L'offre ELK
- Concepts de base
- Les APIs

- **Indexation et documents**

- Document API
- Routing
- API Search Lite
- Distribution de la recherche

- **Ingestion de données**

- Alternatives à l'ingestion
- Les Beats
- Elastic Agent

- **Pipelines ES**

- **Logstash**

- Pipeline
- Syntaxe
- Modèle d'exécution

- **Configuration d'index**

- Types de données
- Contrôle du mapping
- Configuration d'index
- Gabarit d'index
- DataStreams

Agenda (2)

- **Recherche avec DSL**
 - Syntaxe DSL et combinaison de clauses
 - Recherche filtre
 - Recherche full-text
 - Agrégations
 - Géolocalisation
- **Vers la production**
 - Architectures Ingestion
 - Monitoring Logstash
 - Architecture Indexation/Recherche
 - Monitoring API ElasticSearch
 - Exploitation
- **Analyse temps-réel avec Kibana**
 - Présentation
 - Mise au point de tableau de bord
 - Types de visualisation
 - Management

Introduction

L'offre Elastic Stack
Concepts de base
L'API Rest

Introduction

- **Elastic Stack** (avant ELK) est un groupe d'outils facilitant l'analyse et la visualisation temps-réel de données volumineuses
- Ces outils libres sont développés par la même structure, la société *Elastic*, qui encadre le développement communautaire et propose des services complémentaires (support, formation, intégration et hébergement cloud)

Principaux composants de la pile

- Elastic Stack est composé des outils suivants :
 - **Elasticsearch** : Base documentaire NoSQL basée sur le moteur de recherche Lucene
 - **Logstash** : Outil de traitement des traces qui exécutent différentes transformations, et exporte les données vers des destinations diverses dont les index ElasticSearch
 - **Beats** : Agents spécialisés permettant de fournir les données à logstash
 - **Kibana** est une application Angular permettant de visualiser les données présentes dans Elasticsearch

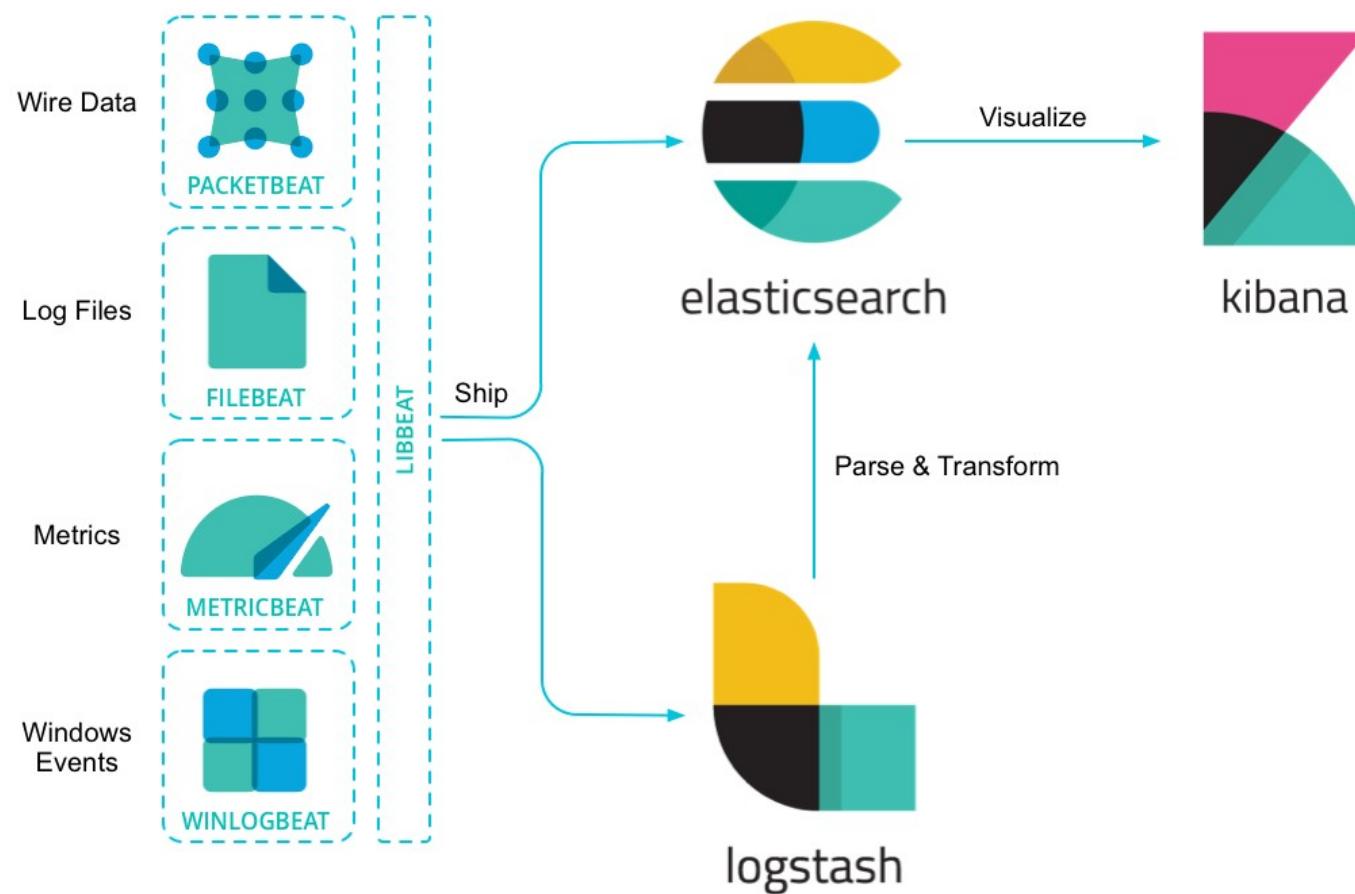
ElasticSearch

- ElasticSearch est un serveur offrant une API RestFul qui permet de :
 - Stocker et indexer des données
(Contenu web ou bureautique, tweets, fichiers de traces, métriques de performance, données structurées RDBMS...)
 - D'exécuter des requêtes de recherche
(structuré, full-text, langage naturel, geo-localisation)
 - Aggréger les données afin de calculer des KPI ou métriques

Caractéristiques

- ✓ Architecture massivement **distribuée et scalable** en volume et en charge
=> Adapté au Big Data, performance remarquable
- ✓ **Haute disponibilité** grâce à la réPLICATION
- ✓ **Muti-tenancy** ou multi-index. Les données sont stockés dans des indexEs dont les cycle de vie sont complètement indépendants
- ✓ Support pour **différents types de requête**
=> Recherche exacte, Full-text, vectoriel, agrégations
- ✓ **Flexibilité** : Stockage structuré des documents sans schéma préalable, possibilité d'ajouter des champs sur un schéma existant
- ✓ **API RESTful** très cohérente et complète
- ✓ **Open Source et commercial**

Architecture



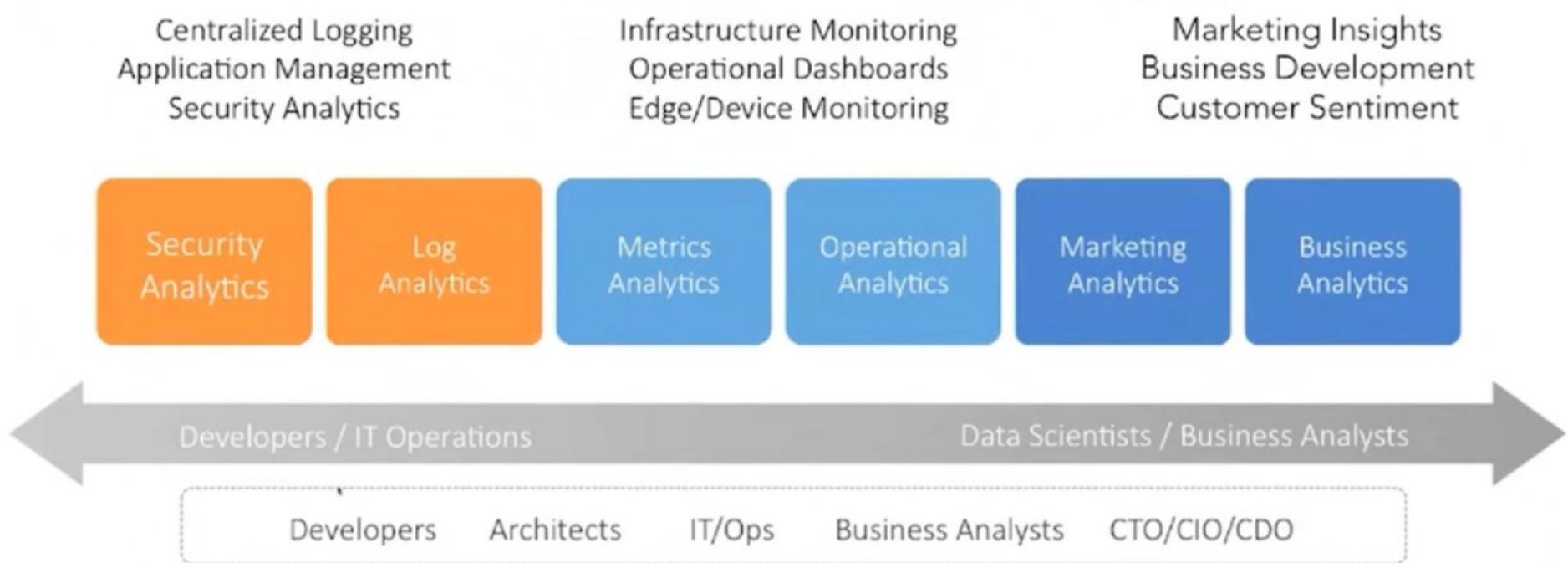
Offres connexes

- La société Elastic propose également des services payants :
 - **X-Pack** : fonctionnalités d'entreprise
 - Sécurité, Monitoring (avant release 8.x)
 - Alertes, Machine Learning
 - Application Performance Management
 - Reporting et planification
 - **Elastic Cloud** : Delegation de l'exploitation d'un cluster à Elastic

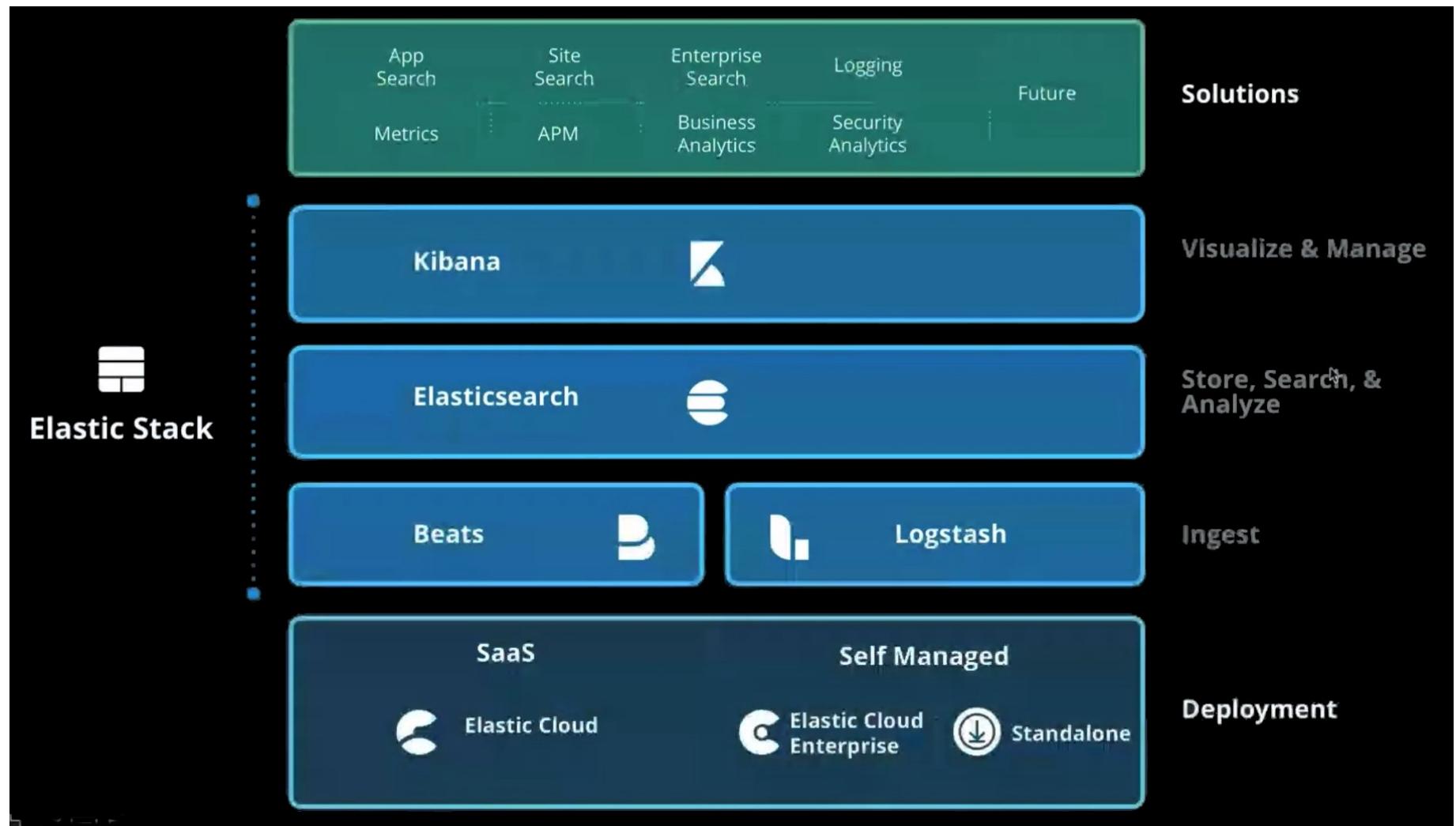
Solutions sur étagère

- ***Enterprise Search*** : Définir les sources de contenu (web, autre..), indexer le contenu et fournir des fonctionnalités de recherche
- ***Elastic Observability*** : Surveillez la consommation de CPU et de mémoire, agrégez les fichiers journaux, inspectez le trafic réseau, surveillez la des performances et l'usage des applications.
Machine Learning pour détecter les anomalies et anticiper le futur
- ***Elastic Security*** : Détecter et répondre aux menaces

Elastic Use cases



En résumé



Distributions

- La distribution permettant d'installer *ElasticSearch* est disponible sous plusieurs formes :
 - ZIP, TAR, DEB ou RPM
 - Également image Docker

Versions

- 8.0.0 : Février 2022
- 7.x : Avril 2019
- 6.x : Novembre 2017 à Décembre 2019
- 5.0.0 : Octobre 2016
- 2.4.1 : Septembre 2016

Installation ELS 8.x

1.Dézipper l'archive

2.Démarrer ELS
bin/elasticsearch

3. Noter les crédentiels elastic et accéder
à ELS

curl <https://localhost:9200/>

Configuration

- Il existe plusieurs fichiers principaux de configuration :
 - ***elasticsearch.yml*** : Propriétés propres au noeud ELS
 - ***jvm.options*** : Options de la JVM
 - ***log4j2.properties*** : Verbosité et support de traces
 - Annuaire utilisateur et certificat SSL

La configuration par défaut permet de démarrer rapidement après une indexation.

En production, il faut quand même modifier certains paramètres. Par exemple :

- Le nom du nœud : ***node.name***
- Les chemins : ***paths***
- Le nom du cluster ***cluster.name***
- L'interface réseau : ***network.host***

elasticsearch.yml

- Les variables d'environnement sont accessibles :
 $\${ENV_VAR}$
- Certaines propriétés peuvent être demandées au démarrage :
 $\${prompt.text}$, $\${prompt.secret}$
- Les valeurs des propriétés peuvent être passées en ligne de commande.
./elasticsearch -Enode.name=David

Bootstrap check

- Au démarrage ELS effectue des vérifications sur l'environnement . Si ces vérifications échouent :
 - En mode développement, des warning sont affichés dans les logs
 - En mode production (écoute sur une adresse publique), ELS ne démarre pas
- Ces vérifications concernent :
 - Dimensionnement de la heap pour éviter les redimensionnements et le swap
 - Limite sur les nombre de descripteurs de fichiers très élevée (65,536)
 - Autoriser 2048 threads
 - Taille et zones de la mémoire virtuelle (pour le code natif Lucene)
 - Le type de JVM (interdit les JVM client)
 - Le garbage collector (interdit la collecte série), la collecte Garbage First si la JVM est trop vieille
 - Filtre sur les appels système
 - Vérification sur le comportement lors d'erreur JVM ou de *OutOfMemory*

Introduction

L'offre
Concepts de base
Les APIs

Cluster

- Un **cluster** est un ensemble de serveurs (nœuds) qui contient l'intégralité des données et offre des capacités de recherche sur les différents nœuds
 - Il est identifié par son nom unique sur le réseau local (par défaut : "elasticsearch").
- => Un cluster peut être mono-noeud
- => Un nœud ne peut pas appartenir à 2 clusters distincts

Noeud

- Un **nœud** est un processus *els* identifié par un nom unique qui a rejoint un cluster
 - Le nombre de nœuds dans un cluster n'est pas limité, les nœud peuvent être ajoutés supprimés sans (trop) perturber le cluster
 - Un nœud peut avoir un ou plusieurs rôles :
 - **master** : Peut être élu pour coordonner le cluster
 - **data** : Stocke les données
 - **ingest** : Ingère les données
 - **ml** : Exécute des jobs de Machine Learning
 - ...
-

Nœud maître

- Dans un cluster un nœud est élu comme **nœud maître**, c'est lui qui est en charge de gérer la configuration du cluster comme la création d'index, l'ajout de nœud dans le cluster
- Pour toutes les opérations sur les documents (indexation, recherche), chaque nœud ayant le rôle adéquat est **interchangeable** et un client peut s'adresser à n'importe lequel des nœuds

Index

- Un **index** est une collection de documents qui ont des caractéristiques similaires
 - Par exemple un index pour les données client, un autre pour le catalogue produits et encore un autre pour les commandes
- Un index est identifié par un nom (en minuscule)
 - Le nom est utilisé pour les opérations de mise à jour ou de recherche
- Dans un cluster, on peut définir autant d'index que l'on veut

Shard

- Un index peut stocker une très grande quantité de documents qui peuvent excéder les limites d'un simple nœud.
- Pour pallier ce problème, ELS permet de sous-diviser un index en plusieurs parties nommées **shards**
 - Le nombre de shards est défini à la création de l'index
- Chaque *shard* est un index indépendant contenant un sous-ensemble des données hébergé sur un nœud *data* du cluster

Apports du sharding

- Le sharding permet :
 - De **scaler** le volume de contenu
 - De **distribuer** et paralléliser les opérations
=> augmenter les performances
- La mécanique interne de distribution lors de l'indexation et d'agrégation de résultat lors d'une recherche est complètement gérée par ELS et donc transparente pour l'utilisateur

Réplica

- Pour pallier à toute défaillance, un shard peut être **répliqué**
- Le nombre de réplique est défini à la création mais peut être modifié dynamiquement
- La réPLICATION permet
 - La **haute-disponibilité** dans le cas d'une défaillance d'un nœud (Une réplique ne réside jamais sur le nœud hébergeant le shard primaire)
 - De **scaler** le volume des requêtes car les recherches peuvent être exécutées sur toutes les répliques en parallèle .
-

Document

- Un **document** est l'unité basique d'information stocké par les shards.
- Il est constitué d'un ensemble de champs typés
- Il peut être représenté avec JSON

Data Stream

- Un flux de données ou ***data stream*** stocke des évènements sur plusieurs index tout en offrant une seule ressource nommée pour les requêtes d'indexation ou de recherche
 - Les flux de données sont bien adaptés aux données générées en continu.
- Le data stream achemine automatiquement les requêtes d'indexation vers le bon index.
- Des stratégies d'ILM¹ permettent de préciser le cycle de vie des données.

Introduction

L'offre Elastic Stack
Concepts de base
Les APIs

API d'ELS

- ELS expose donc une API REST utilisant JSON
- L'API est divisée en catégorie
 - API **document** (CRUD sur document)
 - API **d'index** (CRUD sur Index)
 - API **datastream** : Flux de données (collections d'index datés)
 - API de **recherche** (*_search*)
 - API **cluster** : monitoring du cluster (*_cluster*)
 - API **cat** : Format de réponse tabulée pour gestion du cluster (*_cat*)
 - API **ingest** : Gestion des pipelines ELS
 - API **Logstash** : Gestion des pipelines logstash
 - API **ML** : Gestion des jobs de ML
 - ...

Conventions

- Les différentes API REST d'ELS respectent un ensemble de conventions
 - Possibilité d'indiquer plusieurs indexs dans l'URL
 - Support des *Date*,*Math* dans les noms d'index. (Utilisation de timestamp dans les noms d'index et calculs de date)
 - Options/paramètres communs

Multiple index

- La plupart des APIs qui référencent un index peuvent s'effectuer sur plusieurs index :

test1,test2,test3

*test

+test*, -test3

_all

Noms d'index avec Date Math

- La résolution d'index avec Date Math permet de restreindre les index utilisés en fonction d'une date.
- Il faut que les index soient nommés avec des dates
- La syntaxe est :

`<static_name{date_math_expr{date_format|time_zone}}>`

- date_math_expr : Expression qui calcul une date
- date_format : Format de rendu
- time_zone : Fuseau horaire

- Exemples :

GET /<logstash-{now/d}>/_search => **logstash-2017.03.05**

GET /<logstash-{now/d-1d}>/_search => **logstash-2017.03.04**

GET /<logstash-{now/M-1M}>/_search => **logstash-2017.02**

Options communes (1)

- Les paramètres utilisent l'**underscore casing**
- **?pretty=true** : JSON bien formatté
- **?format=yaml** : Format yaml
- **?human=false** : Si la réponse est traitée par un outil
- **Date Math** : *+1h* : Ajout d'une heure, *-1d* : Soustraction d'une journée, */d* : Arrondi au jour le plus proche
- **?filter_path** : Filtre de réponse, permettant de spécifier les données de la réponse
Ex : GET /_cluster/state?
pretty&filter_path=nodes

Options communes (2)

- **?flat_settings=true** : Les settings sont renvoyés en utilisant la notation « `.` » plutôt que la notation imbriquée
- **?error_trace=true** : Inclut la stack trace dans la réponse
- Formats humains
 - Unités de temps : **d, h, m, s, ms, micros, nanos**
 - Unité de taille : **b, kb, mb, gb, tb, pb**
 - Nombre sans unités : **k, m, g, t, p**
 - Unités de distance : **km, m, cm, mi, yd, ft**

Clients possibles

- Il est possible d'utiliser les clients REST classiques : curl, navigateurs avec add-ons, SOAPUI, ...
- Elastic fournit des librairies en différents types de langage (Java, Javascript, Python, Groovy, Php, .NET, Perl) pour intégrer les appels ELS dans vos applications. Ces librairies offrent du load-balancing
- Enfin, le client le plus confortable est la Dev Console de Kibana

DevConsole Kibana

- Kibana, via sa **DevConsole** offre une interface utilisateur permettant d'interagir avec l'API REST *d'Elasticsearch*.
- Elle est composée de 2 onglets :
 - L'éditeur permettant de composer les requêtes
 - L'onglet de réponse
- La console comprend une syntaxe proche de Curl

```
GET /_search
{
  "query": {
    "match_all": {}
  }
}
```

Fonctionnalités

- La console permet une traduction des commandes CURL dans sa syntaxe
- Elle permet l'auto-complétion
- L'auto indentation
- Passage sur une seule ligne de requête (utile pour les requêtes BULK)
- Permet d'exécuter plusieurs requêtes
- Peut changer de serveur ElasticSearch
- Raccourcis clavier
- Historique des recherche
- Elle peut être désactivée (*console.enabled: false*)

Installation

- La version de Kibana doit correspondre rigoureusement à celle d'ElasticSearch
- La distribution inclut également la bonne version de Node.js
- Elle est disponible sous différents formats :
 - Archive compressée
 - Package debian ou rpm
 - Image Docker

Installation à partir d'une archive

```
wget  
https://artifacts.elastic.co/downloads/kibana/kibana-  
5.0.1-linux-x86_64.tar.gz  
sha1sum kibana-5.0.1-linux-x86_64.tar.gz  
tar -xzf kibana-5.0.1-linux-x86_64.tar.gz  
cd kibana/  
.bin/kibana
```

Structure des répertoires

- ***bin*** : Script binaires dont le script de démarrage et le script d'installation de plugin
- ***config*** : Fichiers de configuration dont *kibana.yml*
- ***data*** : Emplacement des données, utilisé par Kibana et les plugins
- ***optimize*** : Code traduit.
- ***plugins*** : Emplacement des plugins. Chaque plugin correspond à un répertoire

Configuration

- Les propriétés de Kibana sont lues dans le fichier ***conf/kibana.yml*** au démarrage
- Les propriétés principales sont :
 - ***server.host, server.port*** : défaut localhost:5601
 - ***elasticsearch.hosts***: **http://localhost:9200**
 - ***kibana.index*** : Index d'ElasticSearch utilisé pour stocker les recherches et les tableaux de bord
 - ***kibana.defaultAppId*** : L'application utilisée au démarrage Par défaut *discover*
 - ***logging.dest*** : Fichier pour les traces. Défaut *stdout*
 - ***logging.silent, logging.quiet, logging.verbose*** : Niveau de log

Indexation et Documents

Document API
Routing
API Search Lite
Distribution de la recherche

Introduction

- *ElasticSearch* est une base documentaire distribuée.
- Il est capable de stocker et retrouver des structures de données (sérialisées en documents JSON) en temps réel
- Les documents sont constitués de champs.
- Chaque champ a un type
- Certains champs de type texte sont indexés

Structure de données

- Un document est donc une **structure de données**.
 - Il contient des champs et chaque champ a une ou plusieurs valeurs (tableau)
- Un champ peut également être une structure de données. (Imbrication)
- Le format utilisé par ELS est le format JSON

Exemple

```
{  
  "name": "John Smith",      // String  
  "age": 42,                 // Nombre  
  "confirmed": true,         // Booléen  
  "join_date": "2014-06-01",  // Date  
  "home": {                  // Imbrication  
    "lat": 51.5,  
    "lon": 0.1  
  },  
  "accounts": [               // tableau de données  
    {  
      "type": "facebook",  
      "id": "johnsmith"  
    }, {  
      "type": "twitter",  
      "id": "johnsmith"  
    }  
  ]  
}
```

Méta-données

- Des méta-données sont également associées à chaque document.
- Les principales méta-données sont :
 - **_index** : L'emplacement où est stocké le document
 - **_id** : L'identifiant unique
 - **_version** : Version du document
 - ...

Introduction

- L'API document est l'API permettant les opérations CRUD sur la base documentaire
 - Création : *POST*
 - Récupération à partir de l'ID : *GET*
 - Mise à jour : *PUT*
 - Suppression : *DELETE*

Indexation et *id* de document

- Un document est identifié par ses métadonnées `_index` , `_type` et `_id`.
- Lors de l'indexation (insertion dans la base Elastic), il est possible de fournir l'ID ou de laisser ELS le générer

Exemple avec Id

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

```
{
    "field": "value",
    ...
}

...
{
    "_index": {index},
    "_type": "_doc",
    "_id": {id},
    "_version": 1,
    "created": true
}
```

Exemple sans Id

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

```
{  
    "field": "value",  
    ...  
}  
...  
{  
    "_index": {index},  
    "_type": "_doc",  
    "_id": "wM00SFhDQXGZAWDf0-drSA",  
    "_version": 1,  
    "created": true  
}
```

Récupération d'un document

- La récupération d'un document peut s'effectuer en fournissant l'identifiant complet :

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

- La réponse contient le document et ses méta-données.
Exemple :

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

Mise à jour

- Les documents stockés par ELS sont immuables.
=> La mise à jour d'un document consiste à indexer une nouvelle version et à supprimer l'ancienne.
- La suppression est asynchrone et s'effectue en mode batch (suppression de toutes les répliques)

Mise à jour d'un document

```
PUT /website/_doc/123
{
  "title": "My first blog entry",
  "text": "I am starting to get the hang of this...",
  "date": "2014/01/02"
}

...
{
  "_index" : "website",
  "_type": "_doc",
  "_id" : "123",
  "_version" : 2,
  "created": false
}
```

Suppression d'un document

```
DELETE /website/_doc/123
```

```
...
```

```
{
```

```
  "found" : true,  
  "_index" : "website",  
  "_type": "_doc",  
  "_id" : "123",  
  "_version" : 3
```

```
}
```

Mise à jour partielle

- Les documents sont immuables : ils ne peuvent pas être changés, seulement remplacés
 - La mise à jour de champs consiste donc à réindexer le document et supprimer l'ancien
- L'API **_update** utilise le paramètre **doc** pour fusionner les champs fournis avec les champs existants

Exemple

```
POST /website/_update/1
{
  "doc" : {
    "tags" : [ "testing" ],
    "views": 0
  }
}
...
{
  "_index" : "website",
  "_type": "_doc",
  "_id" : "1",
  "_version" : 3
}
```

Utilisation de script

- Un script (*Groovy* ou *Painless*) peut être utilisé pour changer le contenu du champ `_source` en utilisant une variable de contexte :
`ctx._source`
- Exemples :

```
POST /website/_update/1/ {  
    "script" : "ctx._source.views+=1"  
}  
POST /website/_update/1 {  
    "script" : {  
        "source" : "ctx._source.tags.add(params.new_tag)",  
        "params" : {  
            "new_tag" : "search"  
        } } }
```

- Les scripts peuvent également être chargés à partir de l'index particulier `.scripts` ou du disque du serveur. Ils peuvent être utilisés dans d'autres contextes qu'une mise à jour.

Bulk API

- L'API **bulk** permet d'effectuer plusieurs ordres de mise à jour (création, indexation, mise à jour, suppression) en 1 seule requête
 - => C'est le mode batch d'ELS.
- Attention : Chaque requête est traitée séparément. L'échec d'une requête n'a pas d'incidence sur les autres. L'API Bulk ne peut donc pas être utilisée pour mettre en place des transactions.

Format de la requête

- Le format de la requête est :

```
{ action: { metadata } }\n
```

```
{ request body } \n
```

```
{ action: { metadata } }\n
```

```
{ request body } \n
```

...

- Le format consiste à des documents JSON sur une ligne concaténer avec le caractère \n.
 - Chaque ligne (même la dernière) doit se terminer par \n simple ligne
 - Les lignes ne peuvent pas contenir d'autre \n => Le document JSON ne peut pas être joliment formattés

Syntaxe

- La ligne action/metadata spécifie :
 - l'action :
 - **create** : Crédation d'un document non existant
 - **index** : Crédation ou remplacement d'un document
 - **update** : Mise à jour partielle d'un document
 - **delete** : Suppression d'un document.
 - Les méta-données : `_id`, `_index`, `_type`

Exemple

```
POST /website/_bulk // website est l'index par défaut
{ "delete": { "_type": "_doc", "_id": "123" }}
{ "create": { "_index": "website2", "_type": "_doc", "_id": "123" }}
{ "title": "My first blog post" }
{ "index": { "_type": "_doc" }}
{ "title": "My second blog post" }
{ "update": { "_type": "_doc", "_id": "123"} }
{ "doc" : {"title" : "My updated blog post"} }
```

Réponse

```
{  
  "took": 4,  
  "errors": false,  
  "items": [  
    { "delete": { "_index": "website",  
                 "_id": "123",  
                 "_version": 2,  
                 "status": 200,  
                 "found": true  
               },  
    { "create": { "_index": "website",  
                 "_id": "123",  
                 "_version": 3,  
                 "status": 201  
               },  
    { "create": { "_index": "website",  
                 "_id": "EiwfApScQiiy7TIKFxRCTw",  
                 "_version": 1,  
                 "status": 201  
               },  
    { "update": { "_index": "website",  
                 "_id": "123",  
                 "_version": 4,  
                 "status": 200  
               } } ] } }
```

Indexation et Documents

Document API
Routing
API Search Lite
Distribution de la recherche

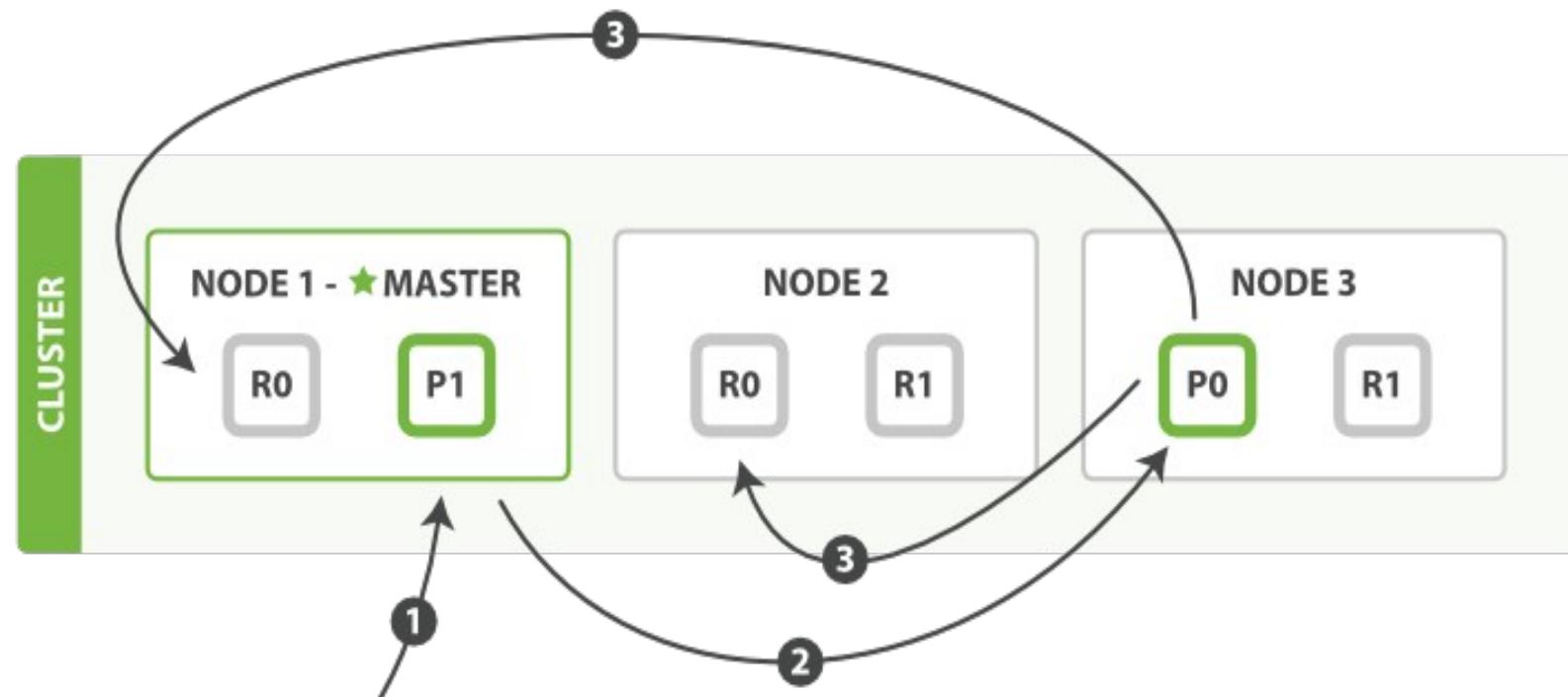
Résolution du shard primaire

- Lors de l'indexation, le document est d'abord stocké sur le shard primaire. La résolution du n° de shard s'effectue grâce à la formule :

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

- La valeur du paramètre **routing** est une chaîne arbitraire qui par défaut correspond à l'*id* du document mais peut être explicitement spécifiée
=> Le nombre de shards primaires est donc fixé à la création de l'index et ne peut plus être changé

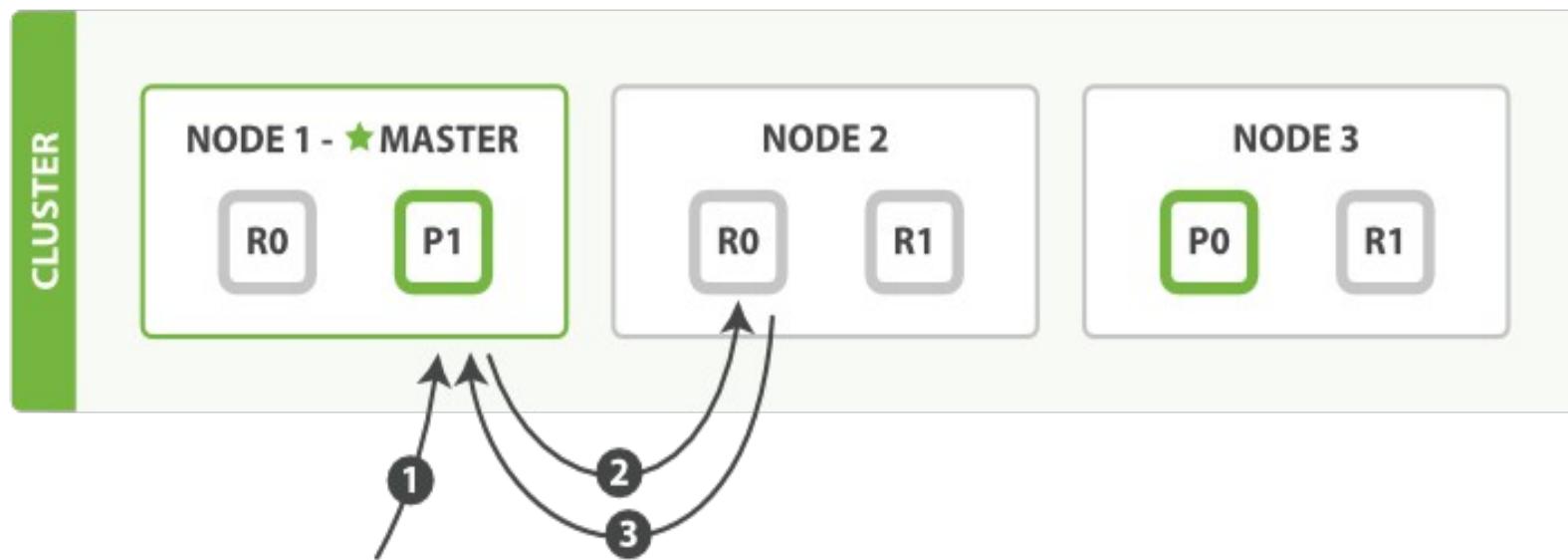
Séquence d'une mise à jour sur une architecture cluster



Séquence d'une mise à jour sur une architecture cluster (2)

- 1. Le client envoie une requête d'indexation ou suppression vers le nœud 1
- 2. Le nœud utilise l'id du document pour déduire que le document appartient au shard 0 . Il transfert la requête vers le nœud 3 , où la copie primaire du shard 0 réside.
- 3. Le nœud 3 exécute la requête sur le shard primaire. Si elle réussit, il transfère la requête en parallèle aux répliques résidant sur le nœud 1 et le nœud 2 . Une fois que les ordres de mises à jour des réplicas aboutissent, le nœud 3 répond au nœud 1 qui répond au client

Séquence pour la récupération d'un document



Séquence pour la récupération d'un document

- 1. Le client envoie une requête au nœud 1.
- 2. Le nœud utilise l'*id* du document pour déterminer que le document appartient au shard 0. Des copies de shard 0 existent sur les 3 nœuds Pour cette fois-ci, il transfert la requête au nœud 2 .
- 3. Le nœud 2 retourne le document au nœud 1 qui le retourne au client.
- Pour les prochaines demandes de lecture, le nœud choisira un shard différent pour répartir la charge (algorithme Round-robin)

Indexation et Documents

Document API
Routing
API Search Lite
Distribution de la recherche

APIs de recherche

- Il y a donc 2 API de recherche :
 - Une version **simple** qui attend que tous ses paramètres soient passés dans la chaîne de requête
 - La version **complète** composée d'un corps de requête JSON qui utilise un langage riche de requête appelé DSL

Search Lite

- La version ***lite*** est cependant très puissante, elle permet à n'importe quel utilisateur d'exécuter des requêtes lourdes portant sur l'ensemble des champs des index.
- Ce type de requêtes peut être un trou de sécurité permettant à des utilisateurs d'accéder à des données confidentielles ou de faire tomber le cluster.
- => En production, on interdit généralement ce type d'API au profit de DSL

Recherche vide

GET /_search

- Retourne tous les documents de tous les index du cluster

Réponse

```
{  
    "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" : 1  
    }, tooks : 4,  
    "_shards" : {  
        "failed" : 0,  
        "successful" : 10,  
        "total" : 10  
    },  
    "timed_out" : false  
}
```

Champs de la réponse

- **hits** : Le nombre de document qui répondent à la requête, suivi d'un tableau contenant l'intégralité des 10 premiers documents. Chaque document a un élément `_score` qui indique sa pertinence. Par défaut, les documents sont triés par pertinence
- **took** : Le nombre de millisecondes pris par la requête
- **shards** : Le nombre total de shards ayant pris part à la requête. Certains peuvent avoir échoués
- **timeout** : Indique si la requête est tombée en timeout. Il faut avoir lancé une requête de type :
`GET /_search?timeout=10ms`

Limitation à un index

- **`_search`** : Tous les index
- **`/gb/_search`** : Tous les documents de l'index gb
- **`/gb,us/_search`** : Tous les documents de l'index gb et us
- **`/g*,u*/_search`** : Tous les documents des index commençant par g ou u

Pagination

- Par défaut, seul les 10 premiers documents sont retournés.
- ELS accepte les paramètres *from* et *size* pour contrôler la pagination :
 - **size** : indique le nombre de documents devant être retournés
 - **from** : indique l'indice du premier document retourné

Paramètre q

- L'API search lite prend le paramètre q qui spécifie la requête dans la syntaxe Lucene adéquate pour les champs full-text
- La chaîne est parsée en une série de
 - Termes : (Mot ou phrase)
 - Et d'opérateurs (AND/OR)
- La syntaxe support :
 - Les caractères joker et les expressions régulières
 - Le regroupement (parenthèses)
 - Les opérateurs booléens (OR par défaut, + : AND, - : AND NOT)
 - Les intervalles : Ex : [1 TO 5]
 - Les opérateurs de comparaison

Exemples search lite

GET /_all/_search?q=tweet:elasticsearch

Tous les documents de type tweet dont le champ full text *match* « elasticsearch »

+name:john +tweet:mary

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

- Tous les documents dont le champ full-text *name* correspond à « john » et le champ *tweet* à « mary »
- Le préfixe **-** indique des conditions qui ne doivent pas matcher

Champ `_all`

- Lors de l'indexation d'un document, ELS concatène toutes les valeurs de type string dans un champ full-text nommé **_all**
- C'est ce champ qui est utilisé si la requête ne précise pas de champ

GET /_search?q=mary

- Si le champ `_all` n'est pas utile, il est possible de le désactiver

Exemple plus complexe

- La recherche suivante utilise les critères suivants :
- Le champ *name* contient « mary » ou « john »
- La date est plus grande que « 2014-09-10 »
- Le champ *_all* contient soit les mots « aggregations » ou « geo »

+name:(mary john) +date:>2014-09-10 +(aggregations geo)

- Voir doc complète :

<https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html#query-string-syntax>

Autres Paramètres de la query string

- Les autres paramètres disponibles sont :
 - **df** : Le champ par défaut, utilisé lorsque aucun champ n'est précisé dans la requête
 - **default_operator** (AND/OR) : L'opérateur par défaut. Par défaut OR
 - **explain** : Une explication du score ou de l'erreur pour chaque hit
 - **_source** : *false* pour désactiver la récupération du champ `_source`.
Possibilité de ne récupérer que des parties du document avec `_source_include` & `_source_exclude`
 - **sort** : Le tri. Par exemple `title:desc,_score`
 - **timeout** : Timeout pour la recherche. A l'expiration du timeout, les hits trouvés sont retournés

Indexation et Documents

Document API

Routing

API Search Lite

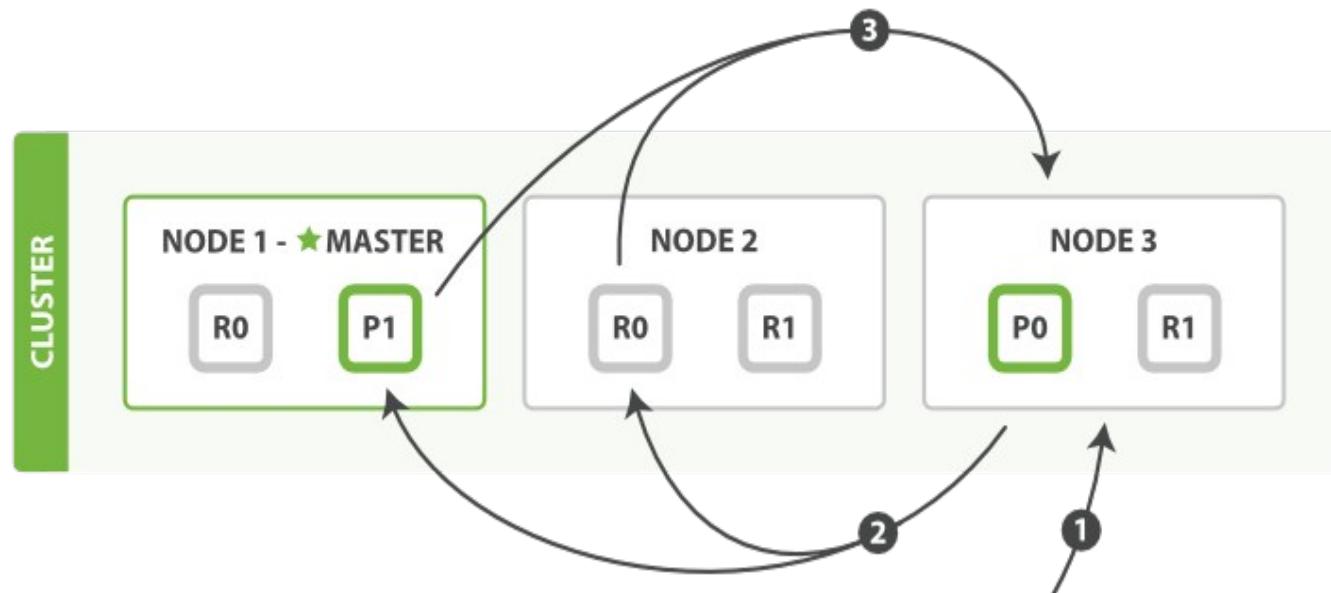
Distribution de la recherche

Introduction

- La recherche nécessite un modèle d'exécution complexe les documents correspondant à la recherche sont dispersés sur les shards
- Une recherche doit consulter une copie de chaque shard de l'index ou des index sollicités
- Une fois trouvés ; les résultats des différents shards doivent être combinés en une liste unique afin que l'API puisse retourner une page de résultats
- La recherche est donc exécutée en 2 phases :
 - *query*
 - *fetch.*

Phase de requête

- Durant la 1ère phase, la requête est diffusée à une copie (primaire ou réplique) de tous les shards. Chaque shard exécute la recherche et construit une file à priorité des documents qui matchent

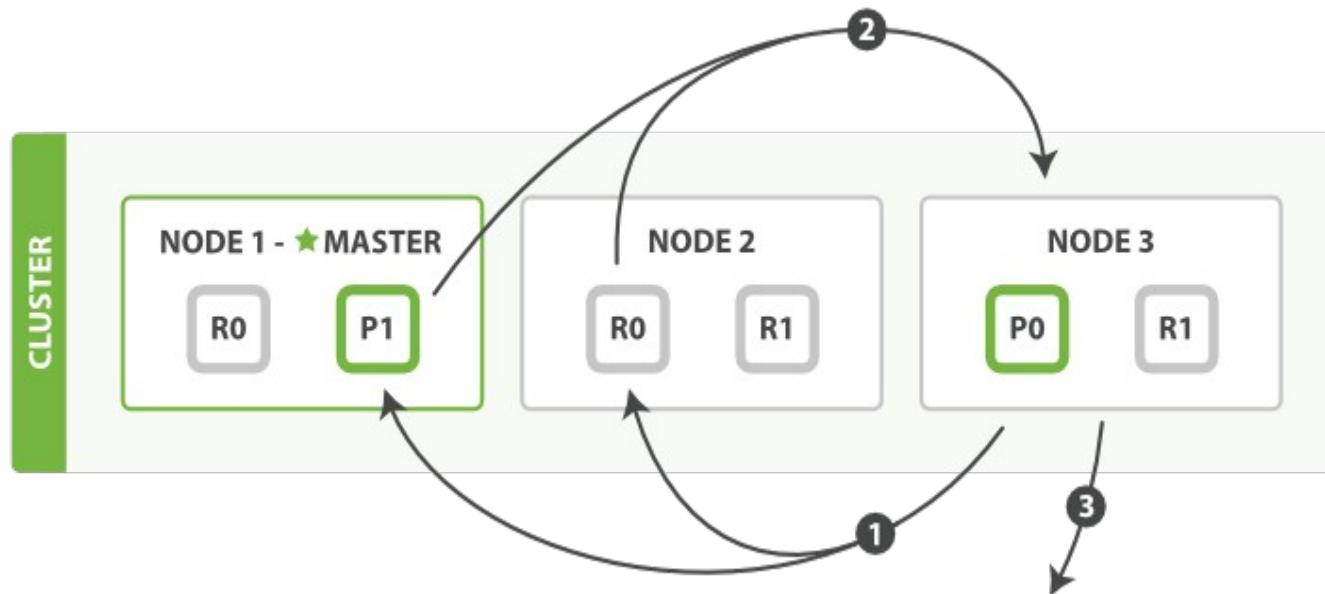


Phase de requête

- 1. Le client envoie une requête de recherche au nœud 3 qui crée une file à priorité vide de taille *from + size* .
- 2. Le nœud 3 transfère la requête à une copie de chaque shard de l'index. Chaque shard exécute la requête localement et ajoute le résultat dans une file à priorité locale de taille *from + size* .
- 3. Chaque shard retourne les IDs et les valeurs de tri de tous les documents de la file au nœud 3 qui fusionne ces valeurs dans sa file de priorité

Phase fetch

- La phase de fetch consiste à récupérer les documents présents dans la file à priorité.



Phase fetch

- 1. Le nœud coordinateur identifie quels documents doivent être récupérés et produit une requête multiple GET aux shards.
- 2. Chaque shard charge les documents, les enrichi si nécessaire (surbrillance par exemple) et les retourne au nœud coordinateur
- 3. Lorsque tous les documents ont été récupérés, le coordinateur retourne les résultats au client.

Ingestion de données

Alternatives à l'ingestion

Les beats
Elastic Agent

Introduction

- Lors du cas d'usage d'analyse temps-réel, Elastic propose 2 méthodes principales pour envoyer des données à Elasticsearch :
 - Les **Beats** sont des convoyeurs de données installés sur les machines ciblent qui envoient régulièrement leur données vers logstash ou ElasticSearch.
 - **Elastic Agent** est un agent unique qui peut être déployé dans deux modes différents :
 - Gestion centralisée via Fleet : La configuration de l'ingestion s'effectue via Kibana.
 - Mode autonome : Configuration manuelle via fichier YAML.

Beats vs Agent

- Elastic Agent et Beats ont des fonctionnalités similaires mais Elastic Agent présente des avantages :
 - Plus facile à déployer. Au lieu de déployer plusieurs Beats, vous déployez un seul Elastic Agent.
 - Plus facile à configurer. Vous n'avez plus besoin de définir et de gérer des fichiers de configuration distincts pour chaque Beat exécuté sur un hôte.
=> Une seule stratégie d'agent qui spécifie les paramètres d'intégration à utiliser, et l'Elastic Agent génère la configuration requise par les programmes sous-jacents, comme Beats.
 - Gestion centralisée. Fleet dans Kibana .
 - Protection des endpoints: Protection des endpoints contre les menaces.

Beats vs Agent

Output	Beats	Fleet-managed Elastic Agent	Standalone Elastic Agent
Elasticsearch Service	✓	✓	✓
Elasticsearch	✓	✓	✓
Logstash	✓	✓	✓
Kafka	✓	✓	✓
Remote Elasticsearch	✓	✓	✓
Redis	✓	✗	✗
File	✓	✗	✗
Console	✓	✗	✗

Ingestion de données

Alternatives à l'ingestion

Les beats

Elastic Agent

Types de beats

- La distribution de beats fournie par ELK contient :
 - **Packetbeat** : Un analyseur de paquets réseau qui convoie des informations sur les transactions entre les différents serveurs applicatifs
 - **Filebeat** : Convoie les fichiers de trace des serveurs.
 - **Metricbeat** : Un agent de monitoring qui collecte des métriques sur l'OS et les services qui s'y exécutent
 - **Winlogbeat** : Événements Windows
- Il est possible de créer ses propre beats. ELK offre la librairie *libbeat* (écrit en Golang) comme support

Beats communautaires

- De nombreux beats communautaires sont également disponibles :
 - **Apachebeat** : Statut des serveurs Apache HTTPD
 - **Elasticbeat** : Statut d'un cluster Elasticsearch. Il envoie ses données directement à Elasticsearch.
 - **Execbeat** : Exécute des commandes shells périodiquement et envoie la sortie standard vers Logstash ou Elasticsearch.
 - **hsbeat** : Métriques de performance de la VM Java HotSpot
 - **httpbeat** : Interroge périodiquement des endpoints HTTP(S) et envoie les réponses vers Logstash ou Elasticsearch.
 - **jmxproxybeat** : Lit les métriques JMX de Tomcat.
 - **journalbeat** : Convoie les logs de systemd/journald sur les systèmes Linux.

Beats communautaires (2)

- **logstashbeat** : Collecte les données de l'API de monitoring de Logstash et les index dans *Elasticsearch*.
- **mysqlbeat** : Exécute des requêtes SQL sur MySQL et envoie les résultats à *Elasticsearch*.
- **nagioscheckbeat** : Vérification Nagios et données de performance
- **pingbeat** : Envoie des ping ICMP pings à des cibles et stocke le *round trip time* (RTT) dans *Elasticsearch*.
- **springbeat** : Collecte les métriques de performance et de santé d'une application Spring Boot avec le module actuator activé.
- **twitterbeat** : Lit des tweets
- **udpbeat** : Envoie des traces via UDP
- **wmibeat** : Utilise WMI pour récupérer des métriques Windows configurable.

Caractéristiques communes

- Tous les beats ont des caractéristiques communes :
 - Leur fichiers de configuration sont des fichiers YAML
 - Ils contiennent les configurations par défaut pour des sorties vers *logstash* ou *elasticsearch*
 - Un fichier **-all.yml* contient toutes les options possibles et fait figure de documentation
 - Lors de leur première utilisation, ils mettent à jour dans ELS un *gabarit de mapping* correspondant aux types des données qu'ils produisent
 - Ils sont distribués avec un ensemble de tableaux de bord Kibana prêt à l'emploi. Ces tableaux de bord sont importés manuellement ou automatiquement lors de la première utilisation

Metricbeats

- *Metricbeat* utilise des modules pour collecter les métriques. Chaque module se configure individuellement
 - *system* : Informations sur le cpu, le système de fichier, la mémoire, les processus, le réseau
 - Apache, nginx
 - Mysql, postgresql
 - MongoDb, CouchBase
 - Redis, Kafka, RabbitMQ
 - Docker, Kubernetes, ZooKeeper
 - Windows
 - ELS, Kibana, Logstash
- *Metricbeat* envoie ses données vers logstash ou ELS

Configuration

- La configuration consiste à :
 - Spécifier quels modules à exécuter
 - Spécifier les options générales de Metricbeat, les options communes à tous les beats
 - Configurer la file d'attente de traitement d'événements
 - Configurer la sortie
 - Filtrer et enrichir les données
 - Spécifier éventuellement une pipeline ELS
 - Indiquer l'hôte Kibana
 - Charger les tableaux de bord Kibana
 - Charger les gabarits d'index ELS
 - Fixer le niveau de trace

Ingestion de données

Alternatives à l'ingestion

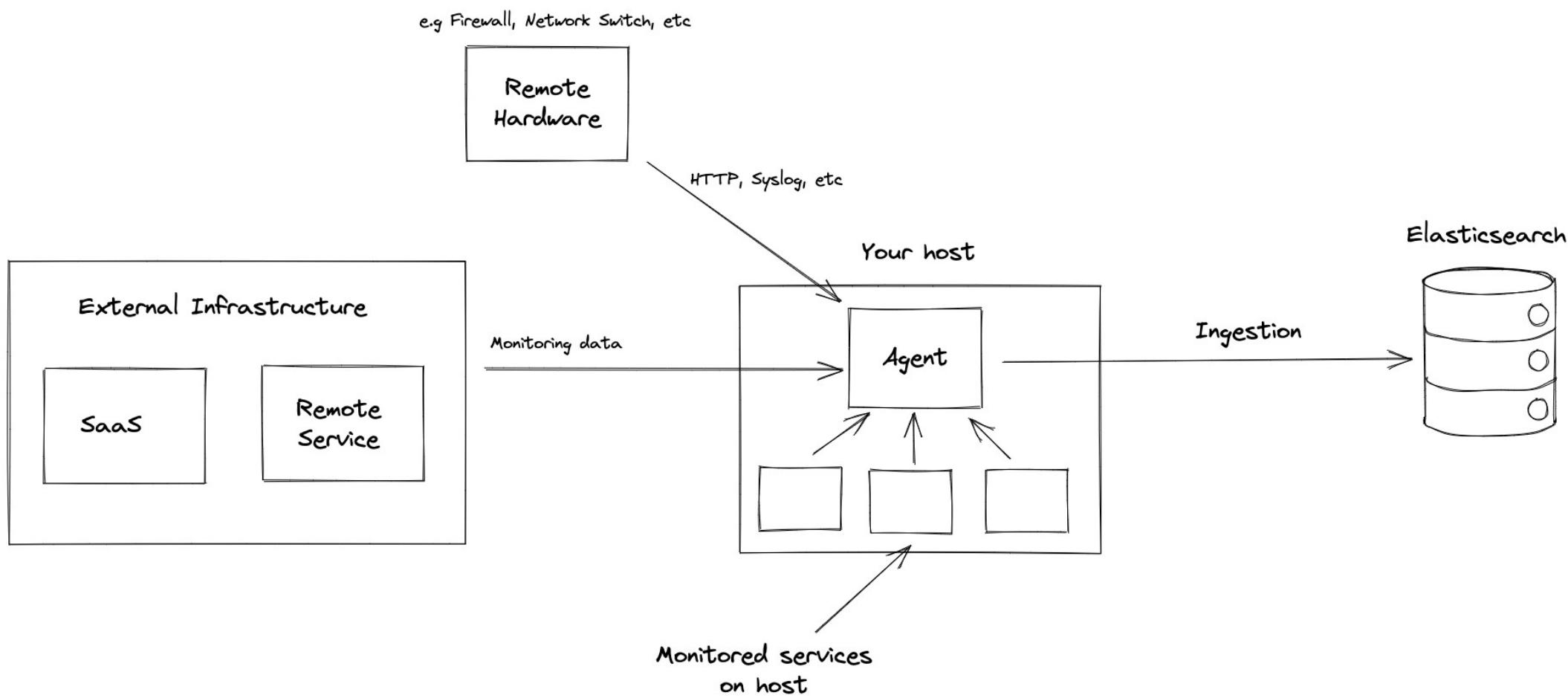
Les beats

Elastic Agent

Elastic Agent

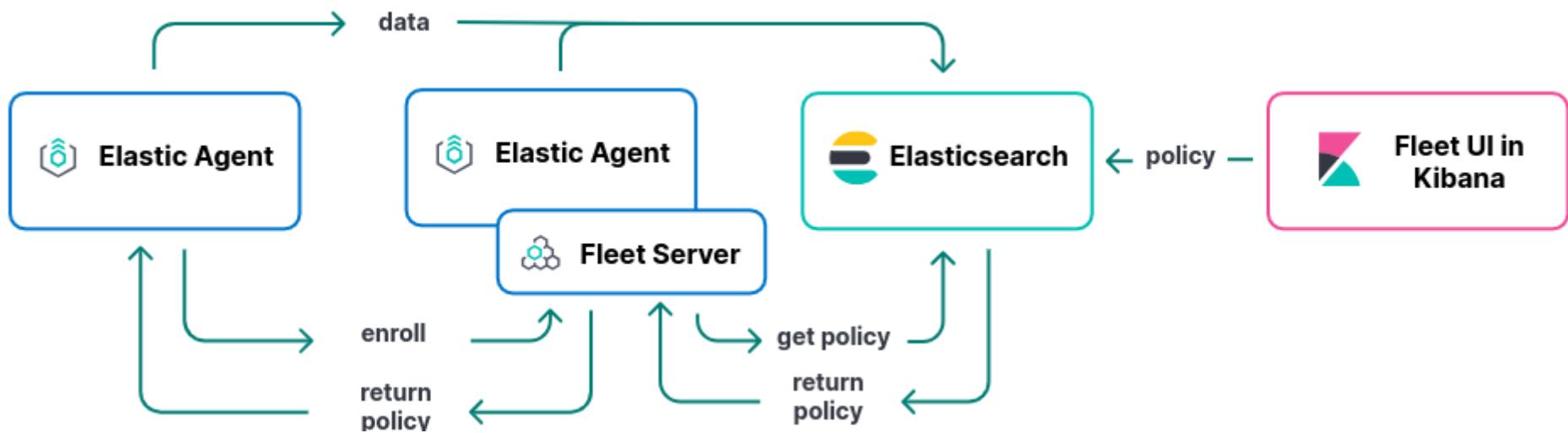
- **Elastic Agent** offre un moyen unifié d'ajouter une surveillance pour les journaux, les métriques et d'autres types de données à un hôte.
- Il peut également protéger les hôtes contre les menaces de sécurité, interroger les données des systèmes d'exploitation, transférer les données des services ou du matériel distants, etc.
- Un agent unique facilite et accélère le déploiement de la surveillance sur l'ensemble de votre infrastructure.
- Chaque agent dispose d'une politique unique que vous pouvez mettre à jour pour ajouter des intégrations pour de nouvelles sources de données, des protections de sécurité, etc.

Architecture



Fleet server

- Pour pour pouvoir gérer de façon centralisée vos agent avec Fleet, il est nécessaire d'installer **Fleet Server**.
 - Un Elastic Agent qui fonctionne dans un mode Fleet Server.
 - Solution scalable. (plusieurs Fleet Server)



Integrations

- Les **intégrations** Elastic offrent un moyen simple de connecter Elastic à des services et systèmes externes.
- Elles sont souvent livrées avec des ressources prêtes à l'emploi telles que des tableaux de bord, des visualisations et des pipelines pour extraire des champs structurés des journaux et des événements.
 - Disponibles pour les services et plateformes classiques Nginx ou AWS, fichiers journaux.
- Kibana fournit une interface utilisateur Web pour gérer les intégrations

Cas d'utilisation

- Typiquement, après avoir installé des Elastic Agent gérés par un Fleet Server :
- Dans le menu Integrations, rechercher l'integration voulue et l'ajouter
 - En lui donnant un nom
 - Et en l'ajoutant à un agent policy (nouveau ou existant)
=> Les agents associés à cette policy remonte les informations dans ElasticSearch
- Dans le menu Integrations, visualiser les assets ajoutés par l'intégration

Pipeline ELS

Concepts de l'ingestion

- Certains nœuds du cluster peuvent être spécialisés en nœud de type ***ingest-nodes***. Leurs CPU sont alors utilisés pour pré-traiter des documents avant leur indexation
- Les pré-traitements sont identifiés par une ***pipeline***
- Le nom de la pipeline est précisée via le paramètre *pipeline* sur une requête bulk ou d'indexation
`PUT my-index/_doc/my-id?
pipeline=my_pipeline_id`
- `{ "foo": "bar"}`

Pipeline

- Une **pipeline** est définie par 2 champs :
 - Une description
 - Une liste de **processseurs**
- Les processseurs correspondent au ré-traitement effectués et s'exécutent dans leur ordre de déclaration

Processeurs

- ELS fournit de nombreux processeurs.
 - Enrichissement
append, date_index_name, attachment, geo_ip, ...
 - Transformation
convert, grok, rename, gsub, split, ...
 - Filtre
drop, remove, ...
 - Gestionnaire d'erreurs ou de routing
fail, pipeline, reroute, ...
 - Scripting
for_each, script, ...

Processeur attachment

- Le processeur **attachment** permet d'extraire le texte des fichiers dans les formats bureautiques les plus communs
- Il utilise *Tika*
-

Ingest API

- L'API ***ingest*** permet de gérer les pipelines :
 - ***PUT*** pour ajouter ou mettre à jour une pipeline
 - ***GET*** pour retourner une pipeline
 - ***DELETE*** pour supprimer
 - ***SIMULATE*** pour simuler un appel à une pipeline

Usage

#Création de la pipeline nommée attachment

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

#Utilisation de la pipeline lors de l'indexation d'un doc.

```
PUT my_index/_doc/my_id?pipeline=attachment
{
  "data": "e1xydGYxXGFuc2kNCkxvcmVtIGlw3VtIGRvbG9yIHNpdCBhbWV0DQpc
cGFyIH0="
}
```

Résultat

```
GET my_index/my_type/my_id
{
  "found": true,
  "_index": "my_index",
  "_type": "my_type",
  "_id": "my_id",
  "_version": 1,
  "_source": {
    "data": "e1xydGYxXGFuc2kNCkxvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbW0DQpccGFyIH0=",
    "attachment": {
      "content_type": "application/rtf",
      "language": "ro",
      "content": "Lorem ipsum dolor sit amet",
      "content_length": 28
    }
  }
}
```

Logstash

Concepts

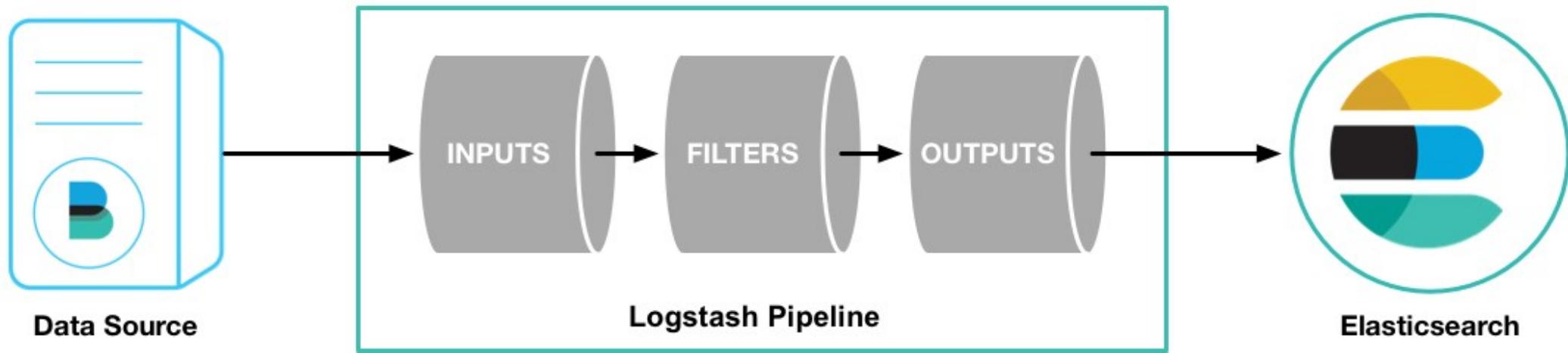
Syntaxe pipeline
Modèle d'exécution

Logstash

- *Logstash* est un outil de **collecte de données temps-réel** capable d'unifier et de normaliser les données provenant de sources différentes
- A l'origine centré sur les fichiers de traces, il peut s'adapter à d'autres types d'événements
- *Logstash* est basé sur la notion de **pipeline** de traitement.
Il permet de filtrer, transformer, enrichir mixer et orchestrer différentes entrées
- Extensible via des plugins, il se connecte à de nombreuses sources de données

Pipeline Logstash

- Une **pipeline** logstash a au minimum :
 - Un plugin d'entrée pour lire les données
 - Une plugin de sortie pour écrire les données
 - Optionnellement des filtres entre les 2



Configuration

- Les pipelines sont généralement configurées dans des fichiers :

```
# The # character at the beginning of a line indicates a comment. Use
# comments to describe your configuration.
input {
}
# The filter part of this file is commented out to indicate that it is
# optional.
# filter {
#
# }
output {
}
```

- Ou la configuration peut être précisée sur la ligne de commande

```
bin/logstash -e 'input { stdin {} } output { stdout {} }'
```

Exemple

```
input {
    beats { port => "5043" } // plugin beats
}

filter {
    grok { // plugin grok, splitter un champ en plusieurs champs avec des regexp
        match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
    geoip { // plugin permettant d'enrichir avec les données géo, à partir d'1 IP
        source => "clientip"
    }
}

output {
    elasticsearch { // Permet d'écrire dans un index elasticsearch
        hosts => [ "localhost:9200" ]
    }
}
```

Multiple entrées/sorties

- *Logstash* est capable de traiter plusieurs entrées et les acheminer vers plusieurs sorties

```
input {  
    twitter {  
        consumer_key => "enter_your_consumer_key_here"  
        consumer_secret => "enter_your_secret_here"  
        keywords => ["cloud"]  
        oauth_token => "enter_your_access_token_here"  
        oauth_token_secret => "enter_your_access_token_secret_here"  
    }  
    beats {  
        port => "5043"  
    }  
}  
  
output {  
    elasticsearch {  
        hosts => ["IP Address 1:port1", "IP Address 2:port2", "IP Address 3"]  
    }  
    file {  
        path => "/path/to/target/file"  
    }  
}
```

Les entrées

- Les entrées permettent d'insérer des données dans les pipelines *Logstash*.
- Les plus courant sont :
 - **file** : Lecture d'une fichier ($\Leftrightarrow tail -f$)
 - **syslog** : Écoute les messages syslog sur le port 514 et les parse au format RFC3164
 - **redis** : Lecture d'un serveur redis. Redis est souvent utilisé pour centraliser les événements provenant de différents installation Logstash
 - **beats** : Traite les événements de *FileBeats*

Champs

- Chaque événement traité par logstash est constitué de **champs**
 - Les champs deviendront les champs des documents indexés par ElasticSearch
 - Les filtres permettent de faire des transformations sur des champs :
 - Splitter un champ en plusieurs (grok)
 - Convertir le type de données
 - Ajouter/Supprimer des champs
 - ...
 - Il est possible de conditionner l'exécution d'un filtre ou d'un plugins de sortie à la présence d'un champ

Les filtres

- Les filtres sont les traitements intermédiaires d'une pipeline Logstash
- Il peuvent s'appliquer selon des conditions, i.e. les champs d'un événement remplissent certains critères
- Les filtres les plus courants :
 - **grok**: Parse et structure un texte arbitraire. 120 patterns sont prédéfinis dans *Logstash* correspondant aux formats de logs les plus courant
 - **mutate**: Effectue des transformations sur les champs de l'événement (Renommage, suppression, remplacement, modification)
 - **drop**: Supprime un événement Ex : DEBUG
 - **clone**: Effectue une copie de l'événement en ajoutant ou enlevant des champs
 - **geoip**: Ajoute des informations géographiques à partir de l'adresse IP

Les sorties

- Les sorties sont la phase finales d'une pipeline
- Les sorties les plus courantes sont :
 - **elasticsearch**: Envoie les données à ElasticSearch pour indexation
 - **file**: écrit l'événement sur un fichier
 - **graphite**: Envoie les données à graphite, un outil open source pour stocker des données de graphiques
<http://graphite.readthedocs.io/en/latest/>
 - **statsd**: Envoie vers le démon *statsd* qui écoute sur UDP, agrège des données et envoie à des services backend pluggable
 - **tcp/udp** : Envoie vers des sockets tcp ou udp

Codecs

- Les entrées et les sorties peuvent appliquer des **codecs** qui permettent d'encoder ou décoder les données sans utiliser de filtres particuliers
- Les codecs permettent de facilement séparer le transport de message du processus de sérialisation
- Les codecs les plus utilisés sont :
 - **json** : Encode ou décode les données au format JSON
 - **multiline** : fusionne des événements textes multi-ligne en une seule ligne. Ex : Exception Java et leur stacktrace
 - **rubydebug** : Utilisée pour le debugging. Permet de voir les champs trouvés par logstash

Options au démarrage

- Les autres options intéressantes de logstash sont :
 - ***-f –path.config*** : Chemin vers le fichier de configuration
 - ***--log.level LEVEL*** : *fatal, error, warn, info, debug, trace*
 - ***--config.debug*** : Si *log.level = debug*. Affiche dans le log les événements envoyés en sortie au format json (Sortie ruby)
 - ***-t, --config.test_and_exit*** : Vérifie la syntaxe de la configuration
 - ***-r, --config.reload.automatic*** : Permet des changements dynamiques de la configuration. On peut également activer à posteriori le rechargement automatique par ***kill -1 <pid>***

Atelier : Installation logstash, premiers pas

Logstash

Concepts
Syntaxe pipeline
Modèle d'exécution

Introduction

- Le fichier de configuration spécifie les plugins à utiliser et leur configuration
- La configuration d'un plugin consiste en
 - le nom du plugin
 - suivi par un block spécifiant des valeurs pour les propriétés de configuration
- Les valeurs fournies doivent respecter le type attendu
- Une syntaxe particulière permet de référencer les champs des événements
- Des structures de contrôle (test if) peuvent conditionner l'exécution d'un traitement

Types supportés

- Les blocs de configuration sont différents selon les plugins, mais les valeurs de configuration appartiennent aux types suivants :
 - Types simples : Booléen, nombre ou chaînes de caractères
`ssl_enable => true name => "Hello world"`
 - Table de Hash
`match => { "field1" => "value1" "field2" => "value2" }`
 - Taille en octets
`my_bytes => "10MiB" # 10485760 bytes`
 - Chaîne avec gabarits : Uri, password, path
`my_path => "/tmp/logstash" my_uri => "http://foo:bar@example.net" my_password => "password"`
 - Codec (valeur prédéfinie)
`codec => "json"`
 - Commentaires
`# this is a comment`

Référence des champs

Pour référencer un champ de haut niveau,

- il suffit de spécifier le champ. Ex : **agent**.
- Ou utiliser la notation []. Ex : **[agent]**

Pour les champs imbriqués, il faut utiliser la notation []. Ex **[ua][os]**

```
{  
  "agent": "Mozilla/5.0 (compatible; MSIE 9.0)",  
  "ip": "192.168.24.44",  
  "request": "/index.html"  
  "response": {  
    "status": 200,  
    "bytes": 52353  
  },  
  "ua": {  
    "os": "Windows 7"  
  }  
}
```

Exemples *sprintf*

La notation `%{}`, nommée ***sprintf***, permet de concaténer des valeurs statiques et les valeurs des champs

- Une notation spécifique permet de récupérer la date du jour dans un format particulier.

```
output {  
    statsd {  
        increment => "apache.%{[response][status]}"  
    }  
}
```

```
output {  
    file {  
        path => "/var/log/%{type}.%{+yyyy.MM.dd.HH}"  
    }  
}
```

Configuration conditionnelle

- La configuration conditionnelle s'obtient en utilisant des instructions *if then else*

```
if EXPRESSION {  
    ...  
} else if EXPRESSION {  
    ...  
} else {  
    ...  
}
```

Exemples

```
filter {  
  if [action] == "login" {  
    mutate { remove_field => "secret" }  
  } }  
  
output {  
  # Send production errors to pagerduty  
  if [loglevel] == "ERROR" and [deployment] == "production" {  
    pagerduty {  
      ...  
    } } }  
  
if [foo] in ["hello", "world", "foo"] {  
  mutate { add_tag => "field in list" }  
}  
}
```

L'expression `if [foo]` retourne false si :

- `[foo]` n'existe pas dans l'évènement,
- `[foo]` existe mais est false ou null

Le champ @metadata

- Il existe un champ spécial : **@metadata**
- Ce champ ne sera pas écrit sur les sorties par contre il peut être utilisé pendant tout le traitement Logstash

```
input { stdin { } }

filter {
    mutate { add_field => { "show" => "This data will be in the output" } }
    mutate { add_field => { "[@metadata][test]" => "Hello" } }
    mutate { add_field => { "[@metadata][no_show]" => "This data will not be in
the output" } }
}

output {
    if [@metadata][test] == "Hello" {
        stdout { codec => rubydebug }
    }
}
```

Variables d'environnement

- une variable environnement peut être référencée par **`${var}`** dans le fichier de configuration
 - Des références à des variables indéfinies provoquent des erreurs
 - Les variables sont sensibles à la casse
 - Une valeur par défaut peut être fournie par **`${var:default value}`**.

Logstash

Concepts
Syntaxe pipeline
Modèle d'exécution

Modèle d'exécution d'une pipeline

- Le modèle d'exécution de Logstash distingue :
 - Les **threads d'input** traitant la réception des messages sur les différentes entrées. Une thread par input
 - Les **worker threads** exécutant les filtres et les sorties
- Les threads d'input écrivent les événements dans une file **synchronisée**, i.e les écritures se terminent lorsque la lecture par une worker thread s'effectue. (Pas de buffer)
 - Si les workers threads sont occupées, la thread d'input est bloquée
- Les worker threads traitent les événements par **lots**
 - Ils remplissent une file d'attente, lorsqu'elle atteint une certaine taille. Le lot d'événements est envoyé dans la pipeline
 - Ils utilisent également une file d'attente pour l'écriture sur les sorties
- Par défaut, les files d'attente des workers sont des files d'attente **mémoire**.
 - Lors d'un arrêt brusque de logstash, les événements dans la file mémoire sont perdus
 - Il est possible de configurer une file persistante

Options configurables de la pipeline

- 3 options de la commande en ligne jouant sur les performances sont configurable pour une pipeline
 - **--pipeline.workers** ou **-w** : Le nombre de workers par défaut 4
 - Généralement supérieur au nombre de CPU disponibles. Il faut augmenter ce chiffre afin que les CPU travaillent au maximum
 - **--pipeline.batch.size** ou **-b** : Nombre maximum d'événements qu'un worker peut collecter. Par défaut 125
 - Plus le nombre est grand, plus le débit et plus la mémoire augmente. Si on augmente trop ce chiffre, les performances se dégradent à cause des collectes mémoire
 - **--pipeline.batch.delay** : La latence de la pipeline. Par défaut 5 ms
 - Si aucun nouveau événement n'arrive sous ce délai. Les événements dans le batch sont traités. Ce paramètre nécessite rarement un tuning

Plusieurs pipelines

- Logstash permet d'exécuter plusieurs pipelines dans le même processus.
- Cela permet d'avoir des configurations de performance ou de durabilité différentes en fonction des entrées
- La configuration s'effectue par un fichier additionnel ***pipelines.yml*** placé dans *path.settings*
- *Logstash* est alors lancé sans l'argument ***-f***

Exemple *pipeline.yml*

- pipeline.id: my-pipeline_1
path.config: "/etc/path/to/p1.config"
pipeline.workers: 3
- pipeline.id: my-other-pipeline
path.config: "/etc/different/path/p2.cfg"
queue.type: persisted

API

- Logstash fournit une API REST pour la surveillance port par défaut 9600
- ***GET /_node/plugins***
Informations sur les plugins installés
- ***GET /_node/pipelines***
Configuration des pipelines, le nombre de workers, la taille et le délai de batch
- ***GET /_node/stats/pipelines***
Information sur l'exécution des pipelines. Nombre d'évènements traités, erreurs, ...

Exemple pipeline

```
{  
  "pipeline": {  
    "events": { "duration_in_millis": 7863504, "in": 100, "filtered": 100, "out": 100 },  
    "plugins": {  
      "inputs": [],  
      "filters": [  
        {  
          "id": "grok_20e5cb7f7c9e712ef9750edf94aefb465e3e361b-2",  
          "events": { "duration_in_millis": 48, "in": 100, "out": 100 },  
          "matches": 100,  
          "patterns_per_field": { "message": 1 },  
          "name": "grok"  
        },  
        {  
          "id": "geoip_20e5cb7f7c9e712ef9750edf94aefb465e3e361b-3",  
          "events": { "duration_in_millis": 141, "in": 100, "out": 100 },  
          "name": "geoip"  
        }  
      ],  
      "outputs": [  
        {  
          "id": "20e5cb7f7c9e712ef9750edf94aefb465e3e361b-4",  
          "events": { "in": 100, "out": 100 },  
          "name": "elasticsearch"  
        }  
      ]  
    },  
    " reloads": { "last_error": null, "successes": 0, "last_success_timestamp": null, "last_failure_timestamp": null, "failures": 0 }  
  }  
}
```

Exactly-once

- Logstash fournit 2 mécanismes pour garantir qu'aucun événement ne sera oublié même en cas d'arrêt brusque ou d'indisponibilité d'une sortie :
 - **File persistante** : Événements en cours de traitement persistés sur le disque
 - **Boite des dead letters** : Événements non délivrés persistés sur le disque

Bénéfices des files persistantes

- En fait, les files persistantes apportent 2 bénéfices :
 - Capable d'absorber un pic de charge sans un autre mécanisme de buffering comme Redis ou Kafka
 - Garantie de livraison unique de chaque événement même lors d'un arrêt brutal.

Configuration

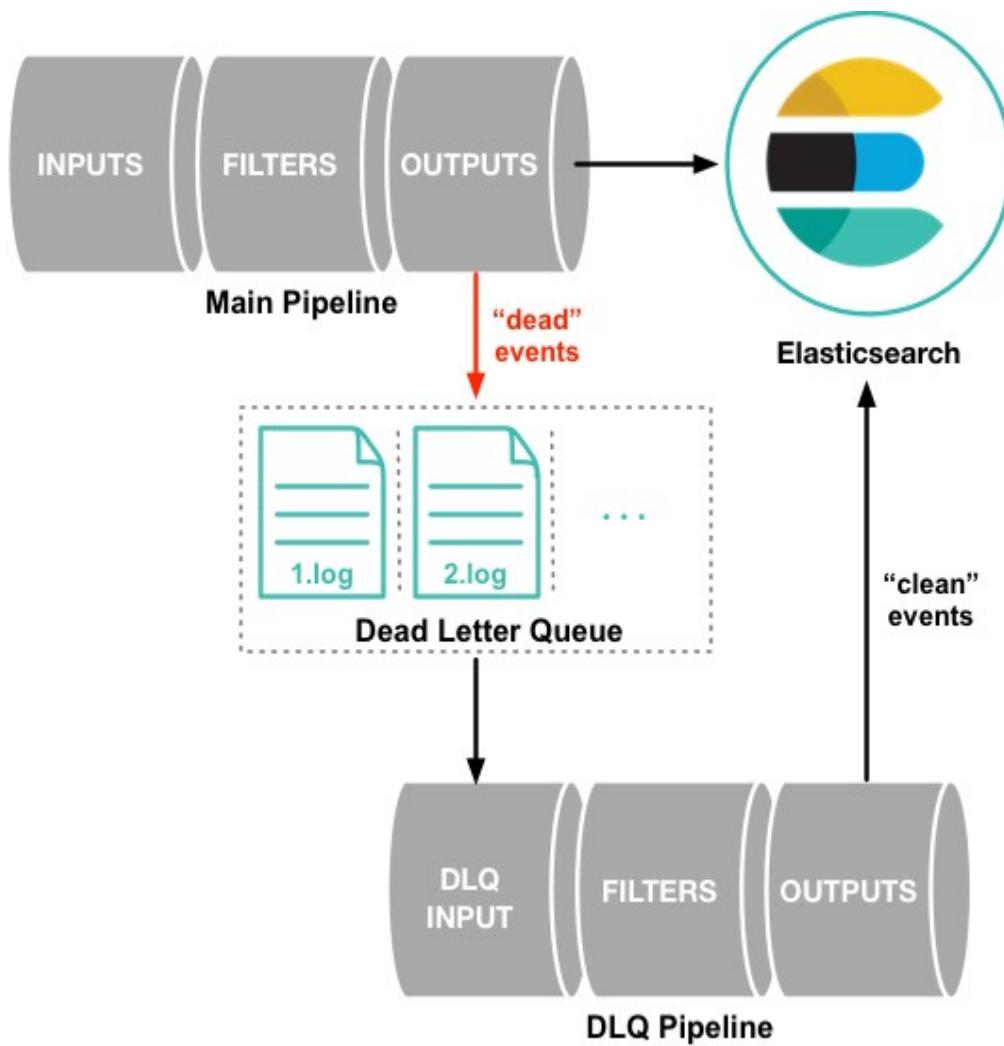
- **queue.type**: *persisted* pour autoriser les files persistantes.
- **path.queue**: Chemin où sont stockés les messages
- **queue.page_capacity**: La taille maximale d'une *page* (i.e un fichier de stockage). Par défaut 64mb.
- **queue.drain**: true => Logstash vide la file avant son arrêt total
- **queue.max_events**: Le nombre maximum événements dans la file. Par défaut illimité (0)
- **queue.max_bytes**: La capacité totale de la file. Par défaut 1gb

Bufferisation

- Lorsque la file est pleine, les entrées de logstash sont bloquées et les nouveaux événements ne rentrent pas dans la pipeline
- Cela permet de ne pas surcharger les sorties (elasticsearch)

Dead letter

- Par défaut, lorsque logstash rencontre un événement qui provoque une erreur , l'événement est supprimé.
- Cela peut être évité si l'on configure une dead letter (seulement pour un output *elasticsearch*)
- L'événement fautif est alors stocké sur disque avec des méta-données additionnelles donnant la cause de l'erreur.
- Pour traiter les événements en erreur, il suffit alors d'utiliser le plugin d'entrée ***dead_letter_queue***



Configuration

```
dead_letter_queue.enable: true  
path.dead_letter_queue:  
"path/to/data/dead_letter_queue"  
dead_letter_queue.max_bytes : 1024mb
```

Pipeline de traitement

```
input {  
    dead_letter_queue {  
        path =>  
        "/path/to/data/dead_letter_queue"  
        commit_offsets => true  
        pipeline_id => "main"  
    }  
}  
  
output {  
    stdout {  
        codec => rubydebug { metadata => true }  
    }  
}
```

Atelier 4.3: Exécution des 2 pipelines

Mapping et types de données

Types de données

Contrôle du mapping

Configuration et création d'index

Gabarits d'index

Datastreams

Types simples supportés

- ELS supporte :
 - Les chaînes de caractères : *string*
 - *text* ou *keyword* (pas analysé)
 - Les numériques : *byte* , *short* , *integer* , *long*, *float* , *double*, *token_count*
 - Les booléens : *boolean*
 - Les dates : *date*
 - Les octets : *binary*
 - Les intervalles : *integer_range*, *float_range*, *long_range*, *double_range*, *date_range*
 - Les adresses IP : *IPV4* ou *IPV6*
 - Les données de géo-localisation : *geo_point*, *geo_shape*
 - Un requête (structure JSON) : *percolator*

Valeur exacte ou full-text

- Les chaînes de caractère peuvent être de deux types :
 - **keyword** : La valeur est prise telle quelle, des opérateurs de type filtre peuvent être utilisés lors de la recherche
Foo!= foo
 - **text** : La valeur est analysée et découpée en termes ou token. Des opérateurs de recherche full-text peuvent être utilisés lors des recherche. Cela concerne des données en langage naturelS

Index inversé

- Afin d'accélérer les recherches, ELS utilise une structure de donnée nommée **index inversé**
- Cela consiste en une liste de mots unique où chaque mot est associé aux documents dans lequel il apparaît.

	A	B
1	term	docs
2	pizza	3, 5
3	solr	2
4	lucene	2, 3
5	sourcesense	2, 4
6	paris	1, 10
7	tomorrow	1, 2, 4, 10
8	caffè	3, 5
9	big	6
10	brown	6
11	fox	6
12	jump	6
13	the	1, 2, 4, 5, 6, 8, 9

Analyseurs

- Les **analyseurs** utilisés à l'indexation et lors de la recherche, transforment un champ texte en un flux de "token" ou mots qui constituent l'index inversé.
- ELS propose des analyseurs prédéfinis :
 - **Analyseur Standard** : (défaut). Il consiste à :
 - Séparer le texte en mots
 - Supprimer la ponctuation
 - Passer tous les mots en minuscule
 - **Analyseurs de langues** : Ce sont des analyseurs spécifiques à la langue. Ils incluent les « stop words » (enlève les mots les plus courant) et extrait la racine d'un mot. C'est le meilleur choix si le champ texte est dans une langue fixe

Test des analyseurs

```
GET /_analyze?analyzer=standard
```

```
Text to analyze
```

Réponse :

```
{  
  "tokens": [ {  
    "token": "text",  
    "start_offset": 0,  
    "end_offset": 4,  
    "type": "<ALPHANUM>",  
    "position": 1  
  }, {  
    "token": "to",  
    "start_offset": 5,  
    "end_offset": 7,  
    "type": "<ALPHANUM>",  
    "position": 2  
  }, {  
    "token": "analyze",  
    "start_offset": 8,  
    "end_offset": 15,  
    "type": "<ALPHANUM>",  
    "position": 3  
  } ] }
```

Comportement par défaut

- Lorsque ELS détecte un nouveau champ *String* dans un type de document, il le configure automatiquement comme 2 champs :
 - 1 champ *text* utilisant l'analyseur standard.
 - 1 champ *keyword*
- Si ce n'est pas le comportement voulu, il faut explicitement spécifier le **mapping** lors de la création de l'index

Types complexes

- En plus des types simples, ELS supporte
 - Les **tableaux** : Il n'y a pas de mapping spécial pour les tableaux. Chaque champ peut contenir 0, 1 ou n valeurs

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

 - Les valeurs d'un tableau doivent être de même type
 - Un champ à *null* est traité comme un tableau vide
 - Les **objets** : Il s'agit d'une structure de données embarquées dans un champ
 - Les **nested** : Il s'agit d'un tableau de structures de données embarquées dans un champ. Chaque élément du tableau est stocké séparément

Mapping des objets embarqués

- ELS détecte dynamiquement les champs objets et les mappe comme objet. Chaque champ embarqué est listé sous **properties**

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

Mapping et types de données

Types de données

Contrôle du mapping

Configuration et création d'index

Gabarits d'index

Datastreams

Mapping

- ELS est capable de générer dynamiquement le mapping
 - Il devine alors le type des champs
- On peut voir le mapping d'un type de données dans un index par :

GET /gb/_mapping

Réponse _mapping

```
{ "gb": {  
  "mappings": {  
    "properties": {  
      "date": {  
        "type": "date",  
        "format": "dateOptionalTime"  
      },  
      "name": {  
        "type": "text"  
      },  
      "tweet": {  
        "type": "text"  
      },  
      "user_id": {  
        "type": "long"  
      }  
    }  
  } } } }
```

Mapping

- Un mapping définit pour chaque champ d'un type de document
 - Le type de donnée
 - Si champ est de type *text*, l'analyseur associé
 - Les méta-données associées

Mapping personnalisé

- Un mapping personnalisé permet (entre autre) de :
 - Spécifier le type d'un champ. Exemple *geo_point*
 - Faire une distinction entre les champs string de type full-text ou valeur exacte (*keyword*)
 - Utiliser des analyseurs spécifiques
 - Définir plusieurs types et analyseurs pour le même champ
 - Spécifier des formats de dates personnalisés
 - ...

Spécification du mapping

- On peut spécifier le mapping :
 - Lors de la création d'un index
 - Lors de l'ajout d'un nouveau champ dans un index existant

=> On ne peut pas modifier un champ déjà indexé

Le point d'entrée de l'API est *_mapping*

Exemple (création)

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

Exemple (Ajout)

```
PUT /gb/_mapping/
{
  "properties" : {
    "tag" : {
      "type" : "text",
      "index": "not_analyzed"
    }
  }
}
```

Mapping et types de données

Types de données

Contrôle du mapping

Configuration et création d'index

Gabarits d'index

Datastreams

Introduction

- ELS permet de démarrer sans mise en place particulière
- Par contre, pour une mise en production, il est nécessaire de tuner finement les processus d'indexation et de recherche afin de s'adapter à son cas d'utilisation
- La plupart de ces personnalisations concernent les index

Création manuelle de l'index

- Lors de la création, 3 blocs peuvent être configurées :
 - **aliases** : Les alias pour cet index
 - **settings** : Principalement répliques et shards
 - **mappings** : Les types de champs et les analyseurs

```
PUT /my_index
{
  "settings": { ... any settings ... },
  "mappings": {
    "type_one": { ... any mappings ... },
    ...
  }
}
```

Alias

- Un alias est un nom secondaire pour un groupe de datastream ou d'index.
- Les API Elasticsearch acceptent un alias à la place d'un flux de données ou d'un nom d'index.
- L'API _alias permet de gérer les alias

Création d'alias

```
POST _aliases
```

```
{
```

```
  "actions": [
```

```
    {
```

```
      "add": {
```

```
        "index": "logs-*",
```

```
        "alias": "logs"
```

```
      }
```

```
    }
```

```
  ]
```

```
}
```

Répliques et shards

- Les 2 principales configuration sont :
 - Le **nombre de shards** : nombre de shards primaire qu'un index a. La valeur par défaut est 1 . Cette configuration ne peut pas être changée après la création
 - Le **nombre de répliques** : Par défaut 1, cette configuration peut être changé à tout moment

```
PUT /my_temp_index
{
  "settings": {
    "number_of_shards" : 1,
    "number_of_replicas" : 0
  }
}
```

Dynamic Mapping

- La détection et l'ajout automatique de nouveau champs est le ***dynamic mapping***
- Les règles d'affectation des types de données peuvent être customisées via 2 moyens :
 - L'activation ou la désactivation du *dynamic mapping* et les règles de détection de nouveaux types
 - L'édition de gabarits dynamique (***dynamic templates***) spécifiant les règles de mapping pour les nouveaux champs

La propriété *mappings/_default_* est dépréciée dans la version 6.x

Activation/Désactivation du dynamic mapping

- La propriété **dynamic** permet de spécifier le comportement du *dynamic mapping* :
 - **true** (défaut) : Chaque nouveau champ est automatiquement ajouté
 - **false** : Tout nouveau champ est ignoré
 - **strict** : Tout nouveau champ lève une exception
- La spécification de *dynamic* peut s'effectuer sur l'objet racine ou sur une propriété particulière

```
PUT /my_index
{
  "mappings": {
    "my_type": {
      "dynamic": "strict",
      "properties": {
        "title": { "type": "string" },
        "stash": { "type": "object", "dynamic": true }
      }
    }
  }
}
```

Règles de détection par défaut

Par défaut, en fonction du type JSON, ElasticSearch applique des correspondances :

- **null** : Pas d'ajout de champ
- **true** ou **false** : Champ *boolean*
- **Point flottant** : Champ *float*
- **integer** : Champ *long*
- **object** : champ *object*
- **array** : Dépend de la première valeur non nulle du tableau
- **String** :
 - Soit un champ *date* (Possibilité de configurer les formats de détection)
 - Soit un *double* ou *long* (Possibilité de configurer les formats de détection)
 - Soit un champ *text* avec un sous-champ *keyword*.

Exemple : Configuration des formats de détection

```
PUT my_index
{
  "mappings": {
    "dynamic_date_formats": [
      "MM/dd/yyyy"
    ]
  }
}
```

```
PUT my_index
{
  "mappings": {
    "numeric_detection": true
  }
}
```

Exemple

```
PUT my_index
{ "mappings": {
    "dynamic_templates": [
        { "integers": {
            "match_mapping_type": "long",
            "mapping": {
                "type": "integer"
            }
        },
        { "strings": {
            "match_mapping_type": "string",
            "mapping": {
                "type": "text",
                "fields": {
                    "raw": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            }
        } ]
    }
}
```

Gabarits dynamiques

- Les gabarits d'index définissent en général des **gabarits dynamiques**
- Il permettent de définir des règles de mapping personnalisées en fonction :
 - Du **datatype** détecté par ELS avec *match_mapping_type*.
 - Du **nom** du champ avec les propriétés *match*, *unmatch* ou *match_pattern*.
 - Du **chemin** complet du champ avec *path_match* et *path_unmatch*.
- Le nom d'origine du champ *{name}* et le type détecté *{dynamic_type}* peuvent être utilisés comme variable lors de la spécification des règles

Spécification

- Les gabarits dynamiques sont donc des collections de règles nommées.
- Ils peuvent être associés à un index ou un gabarit d'index
- La syntaxe est la suivante :

```
"dynamic_templates": [
  {
    "my_template_name": { // Nom de la règle
      ... match conditions ... // match_mapping_type, match, match_pattern, ...
      "mapping": { ... } // Le mapping à utiliser
    }
  },
  ...
]
```

Les règles sont traitées dans l'ordre et la première qui match gagne

Exemple

```
PUT my_index
{ "mappings": {
    "dynamic_templates": [
        { "integers": {
            "match_mapping_type": "long",
            "mapping": {
                "type": "integer"
            }
        }},
        { "strings": {
            "match_mapping_type": "string",
            "mapping": {
                "type": "text",
                "fields": {
                    "raw": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            }
        }}
    ]
}}
```

Mapping et types de données

Types de données

Contrôle du mapping

Configuration et création d'index

Gabarits d'index

Datastreams

Introduction

- Les gabarits d'index permettent de définir des gabarits qui s'appliquent lors de la création d'index, lorsque son nom respecte un pattern (propriété ***index-pattern***)
- Ils définissent 2 aspects:
 - ***settings*** : Nombre de shards, de répliques, etc
 - ***mappings*** : Types des champs

Gabarits prédéfinis

- Elasticsearch installe automatiquement les gabarits d'index suivants :
 - logs-*-*
 - metrics-*-*
 - synthetics-*-*
 - profiling-*

API Rest (legacy)

Les gabarits d'index utilise l'API Rest
_template. Ex :

```
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" } } } }
```

Exemple pattern logstash

```
"mappings" : {
    "dynamic_templates" : [
        {
            "message_field" : {
                "path_match" : "message",
                "match_mapping_type" : "string",
                "mapping" : {
                    "type" : "text",
                    "norms" : false
                }
            },
            "string_fields" : {
                "match" : "*",
                "match_mapping_type" : "string",
                "mapping" : {
                    "type" : "text",
                    "norms" : false,
                    "fields" : {
                        "keyword" : {
                            "type" : "keyword",
                            "ignore_above" : 256
                        }
                    }
                }
            }
        ],
        "properties" : {
            "@timestamp" : {"type" : "date"},
            "@version" : {"type" : "keyword"},
            "geoip" : {
                "dynamic" : true,
                "properties" : {
                    "ip" : { "type" : "ip" },
                    "location" : { "type" : "geo_point" },
                    "latitude" : { "type" : "half_float" },
                    "longitude" : { "type" : "half_float" }
                }
            }
        }
}
```

Component templates

- Depuis la version 7.8, les gabarits d'index sont créés à partir de gabarits de composants
- Les gabarits de composants sont des blocs de construction réutilisables qui configurent le mapping, les settings et les alias.
- Ils ne sont pas directement appliqués à un ensemble d'index.
- Les gabarits d'index peuvent contenir une collection de gabarits de composants, ainsi que spécifier directement des paramètres, des mappages et des alias.

Exemple création gabarits de composant

```
PUT _component_template/component_template1
{
  "template": {
    "mappings": {
      "properties": {
        "@timestamp": {
          "type": "date"
        }
      }
    }
  }
}

PUT _component_template/runtime_component_template
{
  "template": {
    "mappings": {
      "runtime": {
        "day_of_week": {
          "type": "keyword",
          "script": {
            "source": "emit(doc['@timestamp'].value.dayOfWeekEnum.getDisplayName(TextStyle.FULL,
Locale.ENGLISH))"
          }
        }
      }
    }
  }
}
```

Création de gabarit composé

```
PUT _index_template/template_1
{
  "index_patterns": ["te*", "bar*"],
  "template": {
    "settings": {
      "number_of_shards": 1
    },
    "mappings": {
      "_source": {
        "enabled": true
      },
      "properties": {
        "host_name": {
          "type": "keyword"
        },
        "created_at": {
          "type": "date"
        }
      }
    },
    "aliases": {
      "mydata": { }
    }
  },
  "priority": 500,
  "composed_of": ["component_template1", "runtime_component_template"],
  "version": 3,
  "_meta": {
    "description": "my custom"
  }
}
```

Mapping et types de données

Types de données

Contrôle du mapping

Configuration et création d'index

Gabarits d'index

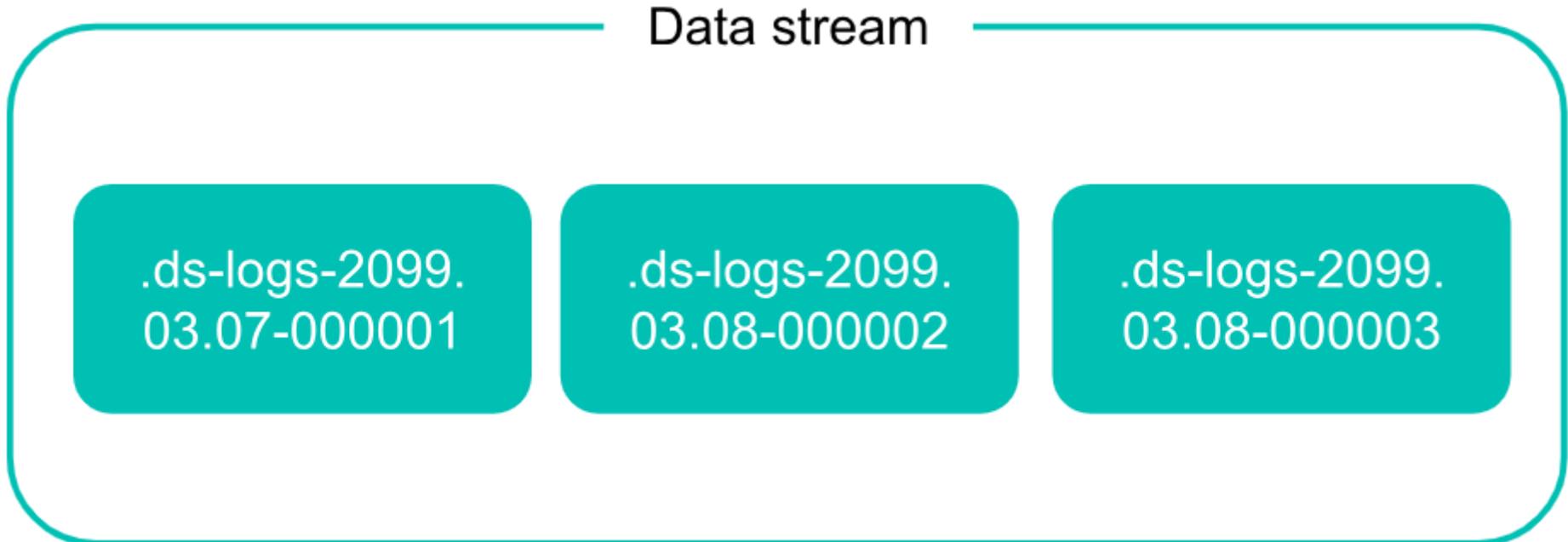
Datastreams

Introduction

- Les datastream permettent de stocker des évènements
- Chaque document doit contenir un champ @timestamp, de type date ou date_nanos.
- Un datastream est associé à un gabarit d'index
- Il existe un api **_datastream** qui permet de créer , supprimer un *datastream*

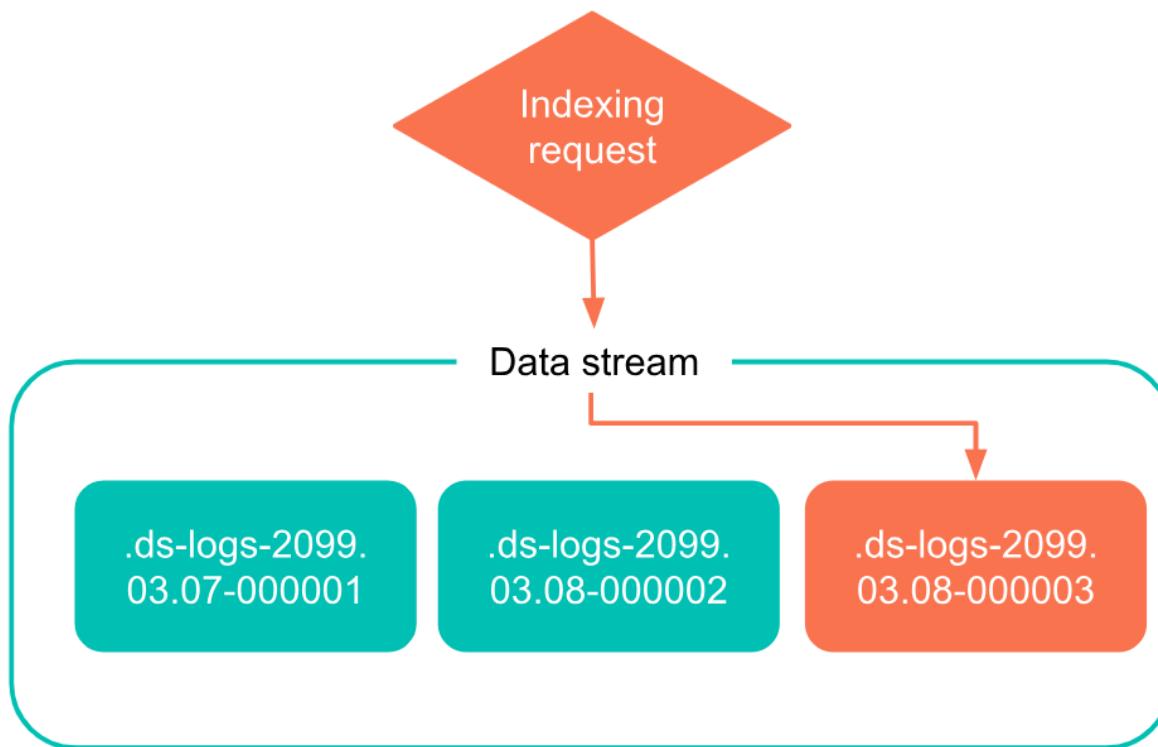
Index

- Un datastream se compose d'un ou de plusieurs index de sauvegarde masqués et générés automatiquement.



Write index

- Un des index du datastream est l'index d'écriture. Les nouveaux documents sont écrits vers cet index



Rollover

- Un rollover crée un nouvel index de sauvegarde qui devient le nouvel index d'écriture.
- Les ILM permettent de basculer automatiquement les flux de données lorsque l'index d'écriture atteint un âge ou une taille spécifiés.
- Il est possible de basculer manuellement un flux de données.

Gestion du cycle de vie des index

- Il est possible d'automatiser des actions de gestion du cycle de vie des index comme :
 - **Rollover** : Redirigez un alias pour commencer à écrire dans un nouvel index lorsque l'index existant atteint un certain âge, nombre de documents ou taille.
 - **Shrink** : Réduire le nombre de shards primaires dans un index.
 - **Force merge** : Réduire le nombre de segments dans chaque shards d'un index et libérer l'espace utilisé par les documents supprimés.
 - **Freeze** : Rendre un index read-only et minimiser son empreinte mémoire
 - **Delete** : Supprimer un index

ILM

- En pratique, il faut associer une **stratégie de cycle de vie, i.e ILM, avec un gabarit d'index**, afin qu'il soit appliqué aux index nouvellement créés.
- La stratégie de cycle de vie définit les conditions de transition entre les différents statuts d'un index :
 - **Hot** : L'index est mis à jour et interrogé
 - **Warm** : L'index n'est plus mis à jour mais encore interrogé.
 - **Cold** : L'index n'est plus mis à jour et est rarement interrogé. Les informations sont toujours consultables, mais les requêtes sont plus lentes
 - **Delete** : L'index n'est plus utilisé et peut être supprimé.

Stratégies

- Une stratégie de cycle de vie consiste à spécifier :
 - Taille ou âge maximum pour effectuer le rollover.
 - Le moment où l'index n'est plus mis à jour et où le nombre de shards primaires peut être réduit.
 - Le moment pour forcer la fusion
 - Le moment où on peut déplacer l'index vers un matériel moins performant.
 - Le moment où la disponibilité n'est pas aussi critique et le nombre de répliques peut être réduit.
 - Le moment où l'index peut être supprimé en toute sécurité.

Création

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

Mise en place de datastream

- La mise en place d'un datastream s'effectue généralement en plusieurs étapes :
 - Créer un ILM
 - Créer des gabarits de composants
 - Créer un gabarit d'index
 - Créer le datastream
 - Sécuriser le data stream

Recherche avec DSL

Syntaxe DSL

Principaux opérateurs

Agrégations

Géolocalisation

Introduction DSL

- Les recherches avec un corps de requête offrent plus de fonctionnalités qu'une recherche simple.
- En particulier, elles permettent :
 - De combiner des clauses de requêtes plus facilement
 - D'influencer le score
 - De mettre en surbrillance des parties du résultat
 - D'agréger des sous-ensemble de résultats
 - De retourner des suggestions à l'utilisateur
 - ...

GET ou POST

- La RFC 7231 qui traite de la sémantique HTTP ne définit pas des requêtes GET avec un corps
 - => Seuls certains serveurs HTTP le supportent
- ELS préfère cependant utiliser le verbe GET car cela décrit mieux l'action de récupération de documents
- Cependant, afin que tout type de serveur HTTP puisse être mis en frontal de ELS, les requêtes GET avec un corps peuvent également être effectuées avec le verbe POST

Recherches vides

```
GET /_search
```

```
{}
```

```
GET /index_2014*/_search
```

```
{}
```

```
GET /_search
```

```
{
```

```
  "from": 30,
```

```
  "size": 10
```

```
}
```

Requête DSL

- Le langage DSL permet d'exposer toute la puissance du moteur Lucene avec une interface JSON
- Pour utiliser DSL, il faut passer une requête dans le paramètre ***query*** :

```
GET /_search  
{ "query": YOUR_QUERY_HERE }
```

Principe DSL

- DSL peut être vu comme un arbre syntaxique qui contient :
 - Des clauses de requête **feuille**.
Elles correspondent à un type de requête (*match*, *term*, *range*) s'appliquant à un champ. Elles peuvent s'exécuter seules
 - Des clauses **composées**.
Elles combinent d'autres clauses (feuille ou composée) avec des opérateurs logiques (*bool*, *dis_max*) ou altèrent leurs comportements (*constant_score*)
- De plus, les clauses peuvent être utilisées dans 2 contextes différents qui modifient leur comportement
 - Contexte **requête ou full-text**
 - Contexte **filtre**

Exemple Combinaison

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

Distinction entre requête et filtre

- DSL permet d'exprimer deux types de requête
 - Les **filtres** sont utilisés pour les champs contenant des valeurs exactes. Leur résultat est de type booléen. Un document satisfait un filtre ou pas
 - Les **recherches** calculent un score de pertinence pour chaque document trouvé. Le résultat est trié par le score de pertinence

Activation des contextes

- Le contexte requête est activée dès lors qu'une clause est fournie en paramètre au mot-clé **query**
- Le contexte filtre est activée dès lors qu'une clause est fournie en paramètre au mot-clé **filter** ou **must_not**

Exemple

```
GET /_search
{
  "query": { // activation du contexte requête
    "bool": { // Les clauses bool, must et match sont exécutées dans un contexte requête
      "must": [
        { "match": { "title": "Search" } },
        { "match": { "content": "Elasticsearch" } }
      ],
      "filter": [ // activation du contexte filtre
        { "term": { "status": "published" } }, // exécuté dans un contexte filtre
        { "range": { "publish_date": { "gte": "2015-01-01" } } } // contexte filtre
      ]
    }
  }
}
```

Recherche avec DSL

Syntaxe DSL

Principaux opérateurs

Agrégations

Géolocalisation

Opérateurs dans le contexte filtres

- **term** : Utilisé pour filtrer des valeurs exactes :
`{ "term": { "age": 26 } }`
- **terms** : Permet de spécifier plusieurs valeurs :
`{ "terms": { "tag": ["search", "full_text", "nosql"] } }`
- **range** : Permet de spécifier un intervalle de date ou nombre :
`{ "range": { "age": { "gte": 20, "lt": 30 } } }`
- **exists** et **missing** : Permet de tester si un document contient ou pas un champ
`{ "exists": { "field": "title" } }`
- **bool** : Permet de combiner des clauses avec :
 - **must** équivalent à ET
 - **must_not** équivalent à NOT
 - **should** équivalent à OU

match

- La recherche **match** est la recherche standard pour effectuer une recherche exacte ou full-text sur presque tous les champs.
 - Si la requête porte sur un champ full-text, il analyse la chaîne de recherche en utilisant le même analyseur que le champ,
 - si la recherche porte sur un champ à valeur exacte, il recherche pour la valeur exacte
- { "match": { "tweet": "About Search" }}

OR par défaut

- *match* est une requête booléene qui par défaut analyse les mots passés en paramètres et construit une requête de type OR. Elle peut être composée de :
 - **operator** (and/or) :
 - **minimum_should_match** : le nombre de clauses devant matcher
 - **analyzer** : l'analyseur à utiliser pour la chaîne de recherche
 - **lenient** : Pour ignorer les erreurs de types de données

Exemple

```
GET /_search
{
  "query": {
    "match" : {
      "message" : {
        "query" : "this is a test",
        "operator" : "and"
      }
    }
  }
}
```

multi_match

- La requête ***multi_match*** permet d'exécuter la même requête sur plusieurs champs :

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

- Les caractères joker peuvent être utilisés pour les champs
- Les champs peuvent être boostés avec la notation ^

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

Filtre *prefix*

- Le filtre ***prefix*** est une recherche s'effectuant sur le terme. Il n'analyse pas la chaîne de recherche et assume que l'on a fourni le préfixe exact

```
GET /my_index/address/_search
{
  "query": {
    "prefix": { "postcode": "W1" }
  }
}
```

Wildcard et regexp

- Les recherche **wildcard** ou **regexp** sont similaires à *prefix* mais permet d'utiliser des caractère joker ou des expressions régulières

```
GET /my_index/address/_search
{
  "query": {
    "wildcard": { "postcode": "W?F*HW" }
  }
}
GET /my_index/address/_search
{
  "query": {
    "regexp": { "postcode": "W[0-9].+" }
  }
}
```

match_phrase

- La recherche ***match_phrase*** analyse la chaîne pour produire une liste de termes mais ne garde que les documents qui contient tous les termes dans le même position.

```
GET /my_index/my_type/_search
{
  "query": {
    "match_phrase": { "title": "quick brown
fox" }
  }
}
```

Proximité avec *slop*

- Il est possible d'introduire de la flexibilité au phrase matching en utilisant le paramètre ***slop*** qui indique à quelle distance les termes peuvent se trouver.
- Cependant, les documents ayant ces termes les plus rapprochés auront une meilleure pertinence.

```
GET /my_index/my_type/_search
{
  "query": {
    "match_phrase": {
      "title": {
        "query": "quick fox",
        "slop": 20 // ~ distance en mots
      }
    }
  }
}
```

match_phrase_prefix

- La recherche ***match_phrase_prefix*** se comporte comme *match_phrase*, sauf qu'il traite le dernier mot comme un préfixe
- Il est possible de limiter le nombre d'expansions en positionnant *max_expansions*

```
{  
  "match_phrase_prefix" : {  
    "brand" : {  
      "query": "johnnie walker bl",  
      "max_expansions" : 50  
    }  
  }  
}
```

Requête fuzzy

- La recherche fuzzy ou recherche floue permet de gérer des erreurs de typo en récupérant les termes approchant (Distance de Levenshtein)
- Elle est équivalente à une recherche par terme

```
GET /my_index/my_type/_search
{
  "query": {
    "fuzzy": {"text": "surprise" }
  }
}
```

2 paramètres peuvent être utilisées pour limiter l'impact de performance :

- ***prefix_length*** : Le nombre de caractères initiaux qui ne seront pas modifiés.
- ***max_expansions*** : Limiter les options que la recherche floue génère. La recherche fuzzy arrête de rassembler les termes proches quand elle atteint cette limite

Recherche avec DSL

Syntaxe DSL

Principaux opérateurs

Agrégations

Géolocalisation

Introduction

- Les agrégations sont extrêmement puissantes pour le reporting et les tableaux de bord
- Des énormes volumes de données peuvent être visualisées en temps-réel
 - Le reporting change au fur et à mesure que les données changent
- L'utilisation de la stack Elastic, Logstash et Kibana démontre bien tout ce que l'on peut faire avec les agrégations.

Exemple (Kibana)



Syntaxe DSL

- Une agrégation peut être vue comme une unité de travail qui construit des informations analytiques sur une ensemble de documents.
- En fonction de son positionnement dans l'arbre DSL, il s'applique sur l'ensemble des résultats de la recherche ou sur des sous-ensembles
- Dans la syntaxe DSL, un bloc d'agrégation utilise le mot-clé **aggs**

```
// Le max du champ price dans tous les documents
POST /sales/_search?size=0
{
  "aggs" : {
    "max_price" : { "max" : { "field" : "price" } }
  }
}
```

Types d'agrégations

- Plusieurs concepts sont relatifs aux agrégations :
 - **Groupe ou Buckets** : Ensemble de document qui ont un champ à la même valeur ou partagent un même critère. Les groupes peuvent être imbriqués. ELS propose des syntaxes pour définir les groupes et compter le nombre de documents dans chaque catégorie
 - **Métriques** : Calculs de métriques sur un groupe de documents (min, max, avg, ...)
 - **Matrice** : Opérations de classification selon différents champs, produisant une matrice des différentes possibilités. Le scripting n'est pas supporté pour ce type d'agrégation
 - **Pipeline** : Une agrégation s'effectuant sur le résultat d'une agrégation

Exemple Bucket

```
GET /cars/_search
{
  "aggs" : {
    "colors" : {
      "terms" : { "field" : "color.keyword" }
    } } }
```

Réponse

```
{  
  ...  
  "hits": { "hits": [] },  
  "aggregations": {  
    "colors": {  
      "doc_count_error_upper_bound": 0, // incertitude  
      "sum_other_doc_count": 0,  
      "buckets": [  
        { "key": "red", "doc_count": 4 },  
        { "key": "blue", "doc_count": 2 },  
        { "key": "green", "doc_count": 2 }  
      ]  
    } } }
```

Exemple métrique

```
GET /cars/_search
{
  "size": 0,
  "aggs" : {
    "avg_price" : {
      "avg" : {"field" : "price"}
    }
  }
}
```

Réponse

```
"hits": {  
    "total": 8,  
    "max_score": 0,  
    "hits": []  
},  
"aggregations": {  
    "avg_price": {  
        "value": 26500  
    }  
}
```

Juxtaposition Bucket/Métriques

```
GET /cars/_search
{
  "size": 0,
  "aggs" : {
    "colors" : {
      "terms" : { "field" : "color.keyword"
    }
    },
    "avg_price" : {
      "avg" : {"field" : "price"}
    }
  }
}
```

Réponse

```
{  
  ...  
  "hits": { "hits": [] },  
  "aggregations": {  
    "avg_price": {  
      "value": 26500  
    },  
    "colors": {  
      "doc_count_error_upper_bound": 0,  
      "sum_other_doc_count": 0,  
      "buckets": [  
        { "key": "red", "doc_count": 4 },  
        { "key": "blue", "doc_count": 2 },  
        { "key": "green", "doc_count": 2 }  
      ]  
    }  
  }  
}
```

Imbrication

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

Réponse

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

Agrégation et recherche

- En général, une agrégation est combinée avec une recherche. Les buckets sont alors déduits des seuls documents qui matchent.

```
GET /cars/_search
{
  "query" : {
    "match" : { "make" : "ford" }
  }, "aggs" : {
    "colors" : {
      "terms" : { "field" : "color" }
    } } }
```

Spécification du tri

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

Types de bucket

- Différents types de regroupement sont proposés par ELS
 - Par terme, par filtre : Nécessite une tokenization du champ
 - Par intervalle de valeurs
 - Par intervalle de dates, par histogramme
 - Par intervalle d'IP
 - Par absence d'un champ
 - Par le document parent
 - Significant terms
 - Par géo-localisation
 - ...

Intervalle de valeur

```
GET /cars/_search
{
  "aggs": {
    "price": {
      "histogram": {
        "field": "price",
        "interval": 20000
      },
      "aggs": {
        "revenue": {
          "sum": { "field": "price" }
        }
      }
    }
  }
}
```

Histogramme de date

```
GET /cars/_search
{
  "aggs": {
    "sales": {
      "date_histogram": {
        "field": "sold",
        "interval": "month",
        "format": "yyyy-MM-dd"
      }
    }
  }
}
```

significant_terms

- L'agrégation ***significant_terms*** est plus subtile mais peut donner des résultats intéressants (proche du machine-learning).
- Cela consiste à analyser les données retournées et trouver les termes qui apparaissent à une fréquence *anormalement* supérieure
Anormalement signifie : par rapport à la fréquence pour l'ensemble des documents
=> Ces anomalies statistiques révèlent en général des choses intéressantes

Fonctionnement

- *significant_terms* part d'un résultats d'une recherche et effectue une autre recherche agrégé
- Il part ensuite de l'ensemble des documents et effectue la même recherche agrégé
- Il compare ensuite les résultats de la première recherche qui sont anormalement pertinent par rapport à la recherche globale
- Avec ce type de fonctionnement, on peut :
 - Les personnes qui ont aimé ... ont également aimé ...
 - Les clients qui ont eu des transactions CB douteuses sont tous allés chez tel commerçant
 - Tous les jeudi soirs, la page untelle est beaucoup plus consultée
 - ...

Exemple

```
{  
    "query" : {  
        "terms" : {"force" : [ "British Transport Police" ]}  
    },  
    "aggs" : {  
        "significantCrimeTypes" : {  
            "significant_terms" : { "field" : "crime_type" }  
        }  
    }  
}
```

Réponse

```
"aggregations" : {  
    "significantCrimeTypes" : {  
        "doc_count": 47347, // Total résultat de requête  
        "buckets" : [  
            {  
                "key": "Bicycle theft",  
                "doc_count": 3640, // Nbr docs le résultat de requête  
                "score": 0.371235374214817,  
                "bg_count": 66799 // Nbr pour le total de l'index  
            }  
            ...  
        ]  
    }  
}
```

=> Le taux de vols de vélos est anormalement élevé pour « British Transport Police »

Types de métriques

- ELS propose de nombreux métriques :
 - ***avg,min, max, sum***
 - ***value_count, cardinality*** : Comptage de valeur distinctes
 - ***top_hit*** : Les meilleurs documents
 - ***extended_stats*** : Fournit plein de métriques (count, sum, variance, ...)
 - ***percentiles*** : percentiles

Recherche avec DSL

Syntaxe DSL

Principaux opérateurs

Agrégations

Géolocalisation

Introduction

- ELS permet de combiner la géo-localisation avec les recherches full-text, structurées et les agrégations
- ELS a 2 modèles pour représenter des données de géolocalisation
 - Le type ***geo_point*** qui représente un couple latitude-longitude. Cela permet principalement le calcul de distance
 - Le type ***geo_shape*** qui définit une zone via le format GeoJSON. Cela permet de savoir si 2 zones ont une intersection

Geo-point

- Les Geo-points ne peuvent pas être détectés automatiquement par le *dynamic mapping*. Ils doivent être explicitement spécifiés dans le mapping:

```
PUT /attractions

{ "mappings": { "restaurant": {

    "properties": {

        "name": { "type": "string" },

        "locationlocation": "40.715, -74.011" }

PUT /attractions/restaurant/2

{ "name": "Pala Pizza", "location": { "lat":40.722,"lon": -73.989 } }

PUT /attractions/restaurant/3

{ "name": "Mini Munchies Pizza","location": [ -73.983, 40.719 ] }
```

Filtres

- 4 filtres peuvent être utilisés pour inclure ou exclure des documents vis à vis de leur *geo-point*:
 - ***geo_bounding_box*** : Les geo-points inclus dans le rectangle fourni
 - ***geo_distance*** : Distance d'un point central inférieur à une limite. Le tri et le score peuvent être relatif à la distance
 - ***geo_distance_range*** : Distance dans un intervalle
 - ***geo_polygon*** : Les geo-points incluent dans un polygone

Exemple

```
GET /attractions/restaurant/_search
{
  "query": {
    "bool": {
      "filter": {
        "geo_bounding_box": {
          "location": { "top_left": { "lat": 40.8, "lon": -74.0 },
                        "bottom_right": { "lat": 40.7, "lon": -73.0 }
                      }
        }
      }
    }
  }
}
```

Agrégation

- 3 types d'agrégation sur les geo-points sont possibles
 - ***geo_distance*** (*bucket*): Groupe les documents dans des ronds concentriques autour d'un point central
 - ***geohash_grid*** (*bucket*): Groupe les documents par cellules (geohash_cell, les carrés de google maps) pour affichage sur une map
 - ***geo_bounds*** (*metrics*): retourne les coordonnées d'une zone rectangle qui engloberait tous les geo-points. Utile pour choisir le bon niveau de zoom

Exemple

```
GET /attractions/restaurant/_search
{
  "query": { "bool": { "must": {
    "match": { "name": "pizza" }
  }},
  "filter": { "geo_bounding_box": {
    "location": { "top_left": { "lat": 40.8, "lon": -74.1 },
                  "bottom_right": { "lat": 40.4, "lon": -73.7 }
                }
  } } } },
  "aggs": {
    "per_ring": {
      "geo_distance": {
        "field": "location",
        "unit": "km",
        "origin": {
          "lat": 40.712,
          "lon": -73.988
        },
        "ranges": [
          { "from": 0, "to": 1 },
          { "from": 1, "to": 2 }
        ]
      }
    }
  }
}
```

Geo-shape

- Comme les champs de type `geo_point`, les **geo-shape** doivent être mappés explicitement :

```
PUT /attractions

{
  "mappings": { "landmark": {
    "properties": {
      "name": { "type": "string" },
      "location": { "type": "geo_shape" }
    } } } }

---
PUT /attractions/landmark/dam_square

{
  "name" : "Dam Square, Amsterdam",
  "location" : {
    "type" : "polygon",
    "coordinates" : [[[ 4.89218, 52.37356 ], [ 4.89205, 52.37276 ], [ 4.89301, 52.37274 ], [ 4.89392, 52.37250 ], [ 4.89218, 52.37356 ] ]
  } }
```

Exemple

```
GET /attractions/landmark/_search
{
  "query": {
    "geo_shape": {
      "location": {
        "shape": {
          "type": "circle",
          "radius": "1km"
        },
        "coordinates": [ 4.89994,
          52.37815]
      }
    }
  }
}
```

Kibana

Introduction

Data views et discover

KQL

ES|QL

Tableaux de bord

Introduction

- Kibana est une plateforme d'analyse et de visualisation fonctionnant avec ElasticSearch
- Il est capable de rechercher les données stockées dans les index d'ElasticSearch
- Il propose une interface web permettant de créer des tableaux de bord dynamiques affichant le résultat des requêtes en temps réel
- Il s'exécute sur Node.js

Dashboard Kibana



Usages

- Kibana à des destinations de tous profils : Métier, Exploitation, Développeur
- Une fois les données ingérées dans des index ElasticSearch, il permet :
 - D'explorer les données afin de mieux les comprendre
 - Créer des visualisation et les inclure dans des tableaux de bord
 - Créer des présentations afin d'animer des réunions
 - Partager via mail, page web ou documents PDF
 - Appliquer des modèles de Machine Learning pour détecter des anomalies ou anticiper le futur
 - Générer des graphes visualisation les relations entre vos données
 - Gérer les index ElasticSearch
 - Générer des alertes relatives à vos données
 - Organiser les assets Kibana en « Espace » lié au modèle de sécurité et de permissions d'ES

Surveillance, Analyse et réaction aux évènements

- Surveillez les services et les applications en temps réel en collectant des informations sur les performances : **APM**
- Surveillez la disponibilité de vos sites et services : **Uptime**
- Recherchez, filtrez et suivez tous vos journaux : **Logs**
- Analysez les métriques de votre infrastructure, de vos applications et de vos services : **Metrics**

Prévenir, détecter et répondre aux menaces

- Créez et gérez des règles pour les événements sources suspects et affichez les alertes créées par ces règles.
Detections
- Affichez tous les hôtes et les événements de sécurité liés à un hôte. **Hosts**
- Affichez les principales mesures d'activité du réseau via une carte interactive. **Network**
- Enquêtez sur les alertes et les menaces complexes, telles que le mouvement latéral de logiciels malveillants entre les hôtes de votre réseau. **Timelines**
- Créer et suivre les problèmes de sécurité. **Cases**
- Afficher et gérer les hôtes qui offre des points d'accès sécurisés. **Administration**

Espaces

- Kibana permet d'organiser son contenu via des **espaces**
 - Permet de regrouper les visualisations, les tableau de bords et les index ES dans des dossiers indépendants
 - Des permissions et des droits d'accès y sont souvent associées
- Parallèlement, un système de tags permet également d'organiser le contenu



Select your space

You can change your space at anytime

D

Default

This is your Delightful Default Space!

E

Engineering

This is the Elegant Engineering Space!

M

Marketing

This is the Magical Marketing Space!

So

Security operations

This is the Super Security operations Space!

Sécurité

- Kibana permet via son modèle de sécurité de définir les accès des utilisateurs
- Des rôles et des privilèges peuvent être définis et associés aux utilisateurs
- L'authentification peut se faire sur l'annuaire embarqué ou sur d'autre fournisseurs d'identité externes à la stack (OpenID par exemple)
- Des données d'audit sont disponibles pour voir qui a fait quoi

Recherche

- Un moteur de recherche est disponible pour retrouver facilement les objets Kibana via leur type, leur nom ou leur tag

The screenshot shows a search interface with a search bar at the top containing the text 'index'. Below the search bar is a list of search results. The first result is 'Data / Index Management' with a 'Management' icon, followed by a 'Go to' button. The second result is 'Data / Index Lifecycle Policies' with a 'Management' icon. The third result is 'Kibana / Index Patterns' with a 'Management' icon. The fourth result is 'Machine Learning / Index Data Visualizer' with an 'Analytics' icon. At the bottom of the interface, there are buttons for 'Filter by type: or tag:' and a 'Shortcut Command + /'.

Type	Name	Management
Data / Index	Data / Index Management	Management
Data / Index	Data / Index Lifecycle Policies	Management
Kibana / Index	Kibana / Index Patterns	Management
Machine Learning / Index	Machine Learning / Index Data Visualizer	Analytics

Filter by **type:** or **tag:** Shortcut **Command + /**

Gestion de la stack

- Kibana permet également de gérer la stack Elastic :
 - Gestion des pipelines
 - Gestion des index, politique d'archivage et de suppression, sauvegarde, réPLICATION sur des clusters distants
 - Gestion des alertes
 - Des jobs de Machine Learning

Analysez et visualisez vos données

- Connaître ses données. **Discover**
- Créer des graphiques et d'autres visualisations. **Dashboard**
- Afficher vos données sous différents angles. **Dashboard**
- Travailler avec des données de localisation. **Maps**
- Créer une présentation de vos données. **Canvas**
- Générez des modèles pour le comportement de vos données.
Machine Learning
- Explorer les connexions entre vos données. **Graph**
- Partager vos données. **Dashboard, Canvas**

Administre l'ElasticStack

- Gérer les données ElasticSearch.
Stack Management > Data
- Mettre en place des règles.
Stack Management > Rules and Connectors
- Organiser l'espace de travail et les utilisateurs.
Stack Management > Spaces
- Définir les rôles et permissions.
Stack Management > Users
- Personnaliser Kibana.
Stack Management > Advanced Settings

Edition de DataView

- La vue détaillé d'une data view permet :
 - De voir les caractéristiques des champs, en particulier les colonnes Searchable et Agregatable
 - D'éditer les champs en particulier leur format d'affichage par défaut
 - De rechercher un champ particulier
 - D'ajouter des champs scriptés (champs calculés à partir d'autre champs)
 - D'exclure des champs présents dans l'index
 - De visualiser les objets sauvegardés liés à la Data View : Requêtes, Visualisations, Dashboard, Alertes, ...

Kibana

Introduction
Data views et discover
KQL
ES|QL
Tableaux de bord

Data views

- Kibana nécessite une **Data View** pour indiquer à quelles données Elasticsearch on souhaite accéder et si les données sont basées sur le temps.
- Une data view référence un ou plusieurs data stream ou index contenant généralement un champ **@timestamp** qui horodate le document
- Elles sont généralement créées par un administrateur dans Stack Management.
- Kibana montre la liste des champs correspondant à une data view ainsi que leurs types
- Il est possible de personnaliser le nom d'affichage et le format de chaque champ.

Création

- La création de DataView s'effectue dans la partie StackManagement

Create data view

Name
my-data-view

Index pattern
k*

Timestamp field
@timestamp

Select a timestamp field for use with the global time filter.

Show advanced settings

× Close Use without saving Save data view to Kibana

Your index pattern matches 3 sources.

	All sources	Matching sources
kibana_sample_data_ecommerce		Index
kibana_sample_data_flights		Index
kibana_sample_data_logs		Data stream

Rows per page: 10 ▾

Vue détaillée

logstash

Index pattern: logstash* Time field: @timestamp ★ Default

[Fields \(64\)](#) [Scripted fields \(0\)](#) [Field filters \(0\)](#) [Relationships \(0\)](#)

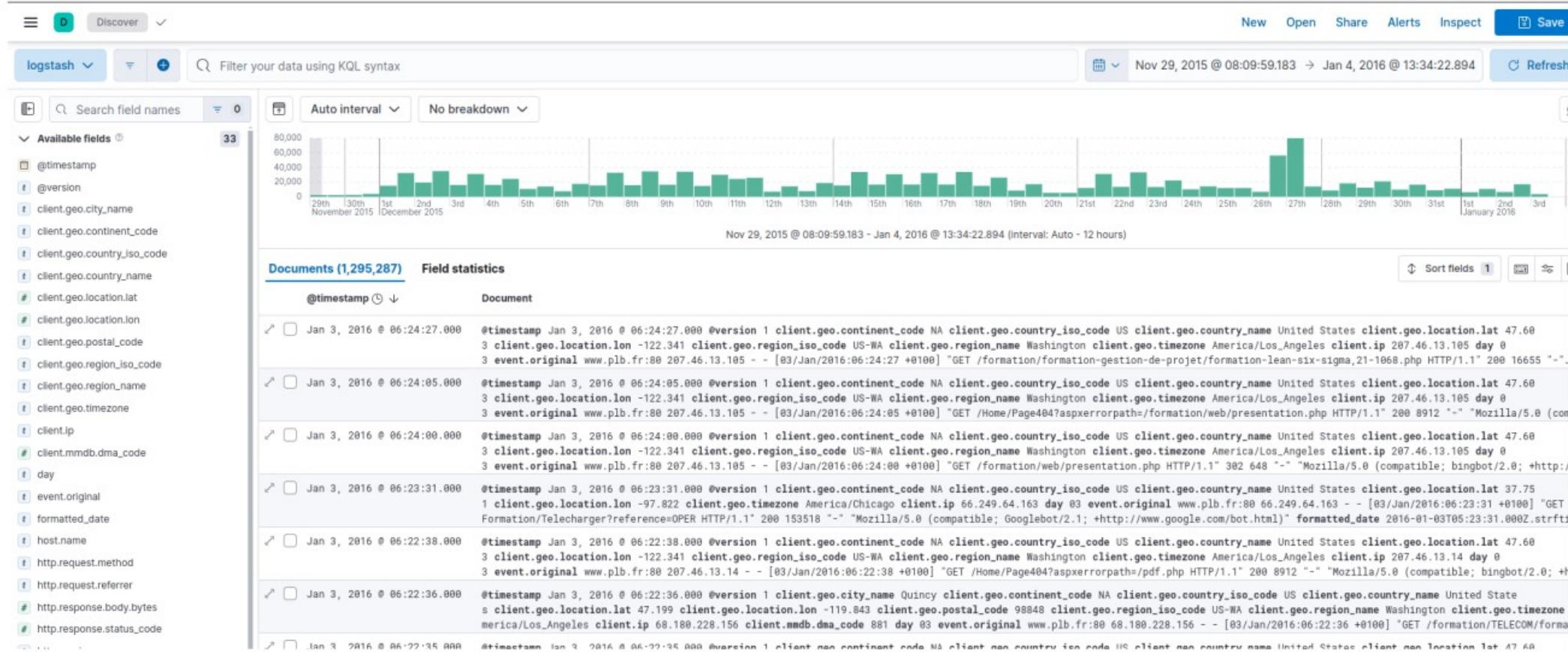
Search Field type 8 Schema type 2 [Refresh](#) [Add field](#)

Name ↑	Type	Format	Searchable	Aggregatable	Excluded
@timestamp ⓘ	date		●	●	Edit
@version	text		●		Edit
@version.keyword	keyword		●	●	Edit
_id	_id		●		Edit
_index	_index		●	●	Edit
_score					Edit
_source	_source				Edit
client.geo.city_name	text		●		Edit
client.geo.city_name.keyword	keyword		●	●	Edit
client.geo.continent_code	text		●		Edit

Rows per page: 10 < 1 2 3 4 5 6 7 >

Discover

- Une fois le Data View mis en place on peut explorer ses données via le menu Discover



Recherche

- Kibana propose plusieurs méthodes pour créer des requêtes de recherche.
 - un filtre temporel
 - Des filtres sur les données structurées
 - Des recherches via les syntaxes KQL, ES/QL et Lucene
- Les recherches peuvent être sauvegardées

The screenshot shows the Kibana search interface with the following components:

- Data view:** A dropdown menu showing "kibana_sample_data_ecommerce".
- Save query & add filter:** Buttons for saving the query and adding filters.
- Semi-structured search:** A text input field with placeholder "Filter your data using KQL syntax".
- Time filter:** A dropdown menu showing "Last 7 days".
- Filters:** A section at the bottom showing applied filters:
 - "geoip.country_iso_code: US" with a remove button.
 - "NOT day_of_week: Wednesday" with a remove button.
- Extra filters with AND:** A label indicating additional filters are present.

Filtre temporel

- Le filtre temporel permet de limiter la plage de temps que l'on visualise
 - Il est présent dans la plupart des pages Kibana
- Plusieurs moyens sont disponibles pour spécifier la plage
 - Quick Select. Les dernières ou les prochaines 15 minutes par exemple
 - Les plages fréquentes
 - Les plages de dates récemment utilisées.
- Il est possible d'activer le rafraîchissement automatique.

Sélecteur de DataView

Kibana Sample Data eCommerce

Add a field to this data view

Manage this data view

Data views + Create a data view

Find a data view

Kibana Sample Data Logs

Kibana Sample Data Flights

Kibana Sample Data eCo... ↔

Exploration des champs

ma

Popular fields 1

Available fields 3

k email

t manufacturer

t products.manufacturer

Agrégation

customer_birth_date

customer_first_name

customer_full_name

customer_gender

customer_id

customer_last_name

customer_phone

day_of_week

day_of_week_i

email

event.dataset

geoip.city_name

geoip.continent_name

geoip.country_iso_code

Top values

day_of_week	19.8%
Monday	19.8%
Friday	14.4%
Tuesday	13.8%
Thursday	13.4%
Sunday	13.3%
Wednesday	12.8%
Saturday	12.4%

Calculated from 1,134 records.

Visualize

Ajout de champ calculé

t log.file.path

Add a field

Ajout de filtres

Add filter Technical preview

day_of_week is not Wednesday

Preview
NOT day_of_week: Wednesday

Custom label (optional)
Add a custom label here

Cancel Add filter

Vue détaillée du document

1,000 hits Reset search

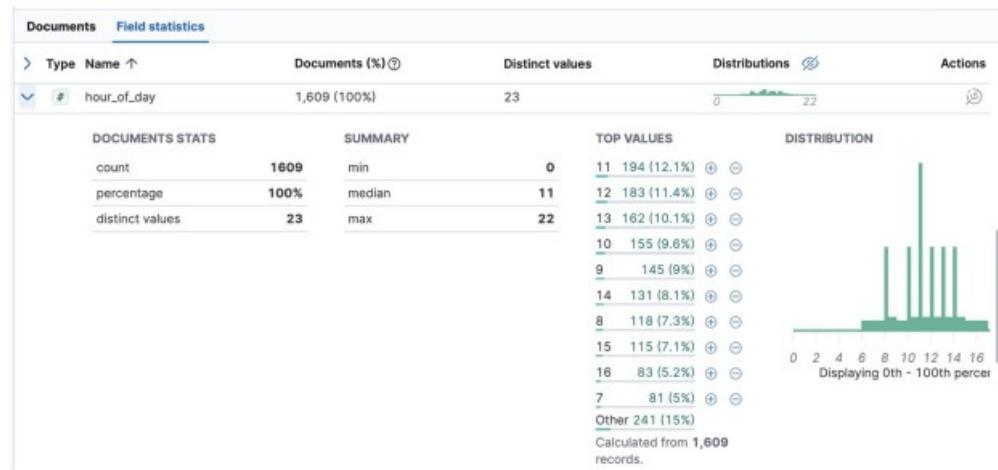
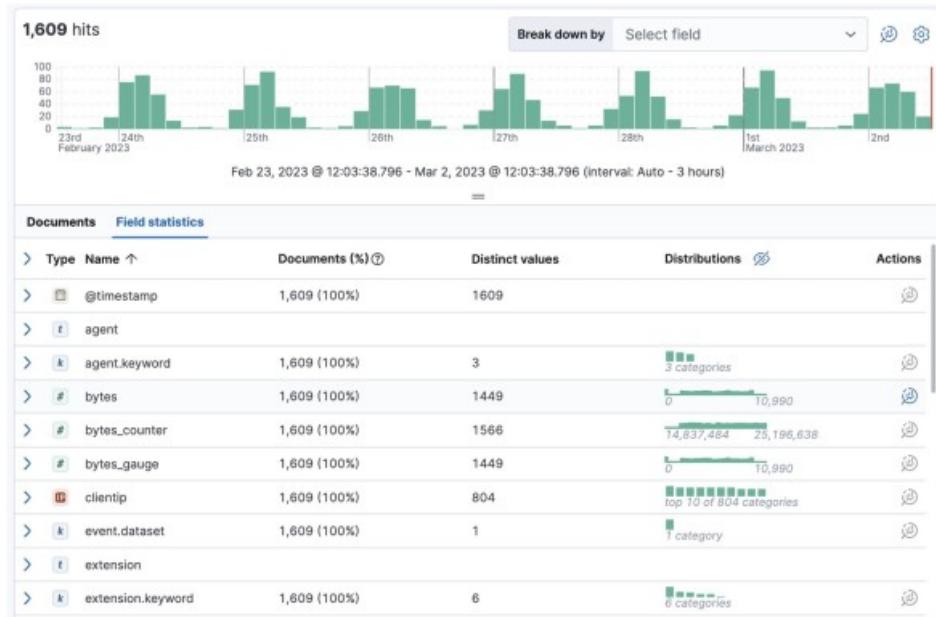
order_date

Actions	Field	Value
x _id	_id	ub1puacBn0KvY_NfNt_
x _index	_index	kibana_sample_data_eCommerce
x _score	_score	-
x category	category	(Men's Clothing, Men's Accessories, Men's Shoes)
x currency	currency	EUR
x customer	customer	Abbott, J
x customer.first_name	customer.first_name	Jackson
x customer.full_name	customer.full_name	Jackson Abbott
x customer.gender	customer.gender	MALE
x customer.id	customer.id	12
x customer.last_name	customer.last_name	Abbott

Modifier la table de résultats :

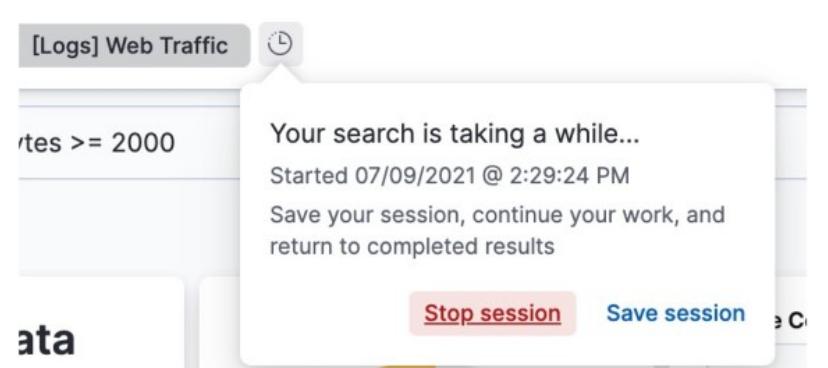
- Ordonner et dimensionner les colonnes
- Ajuster la hauteur de ligne, le nombre de lignes d'une page
- Ordonner
- Editer le format de visualisation d'un champ
- N'afficher que les documents sélectionnés

Statistiques sur les Champs



Requêtes en tâche de fond

- Quelquefois, il est nécessaire de rechercher sur de très gros volumes de données
- Si la requête est très longue, on peut l'exécuter en tâche de fond
- La requête est alors stocké et visible dans la partie StackManagement



The screenshot shows the 'Search Sessions' page. It has a search bar at the top and a table below it. The table columns are App, Name, # Searches, Status, Created, and Expiration. There are two entries:

App	Name	# Searches	Status	Created	Expiration
[Logs]	Web Traffic - 07/09/2021 @ 2:24 PM	11	Complete	9 Jul, 2021, 14:24:06	16 Jul, 2021, 14:24:06
[Logs]	Flights out of Warsaw or Venice - 07/09/2021 @ 2:22 PM	1	Complete	9 Jul, 2021, 14:22:36	16 Jul, 2021, 14:22:36

A context menu is open over the second row, with options: 'Inspect' (highlighted), 'Edit name', 'Extend', and 'Delete'.

Kibana

Introduction
Data views et discover
KQL
ES|QL
Tableaux de bord

Recherche semi-structurée KQL

- ***Kibernate Query Language*** sert à filter les données. (Il n'effectue pas de tri ni d'agrégations)
- Il permet de combiner la recherche en full-text avec la recherche par champ via (KQL).
- Lors d'une recherche full-text les résultats sont triables par pertinence.

Exemples

- Filtrer les documents pour lesquels un champ existe :
http.request.method: *
- Filtrer les documents correspondant à une valeur
 - Documents dont le champ est égale une valeur
http.request.method: GET
 - Documents dont le champ est égale une valeur ou une autre valeur
http.request.method: GET POST
 - Documents dont un des champs est égal à une valeur
GET
- Valeur exacte si le champ n'est pas un texte, sinon recherche full-text.
 - Le champ texte contient les mots null et pointer
http.request.body.content: null pointer
 - Le champ texte contient la phrase « null pointer »
http.request.body.content: "null pointer"
- Recherche d'une phrase :
http.response.body.content.text:"quick brown fox"

Intervalles et wildcard

- La recherche par intervalle est utilisée principalement pour les chiffres ou les dates
 - *http.response.bytes < 10000*
 - *http.response.bytes > 10000 and http.response.bytes <= 20000*
 - *@timestamp < now-2w*
- La recherche par wildcard permet de filtrer des documents par le préfixe d'une valeur :
 - *http.response.status_code: 4**
- Les wildcards peuvent être utilisés pour indiquer plusieurs champs :
 - *datastream.*: logs*

Opérateurs booléens

- La négation s'effectue avec NOT
 - ***NOT http.request.method: GET***
- On peut combiner plusieurs requêtes avec AND , OR et les parenthèses
 - ***http.request.method: GET OR http.response.status_code: 400***
 - ***http.request.method: GET AND http.response.status_code: 400***
 - ***(http.request.method: GET AND http.response.status_code: 200) OR (http.request.method: POST AND http.response.status_code: 400)***

Champs imbriqués

Exemple de document :

```
{  
  "user" : [  
    {  
      "first" : "John",  
      "last" : "Smith"  
    },  
    {  
      "first" : "Alice",  
      "last" : "White"  
    } ] }
```

- Pour recherche Alice White :

user:{ first: "Alice" and last: "White" }

Champs imbriqués (2)

- Si il y a plusieurs niveaux d'imbrication, il faut donner le chemin complet du champ.

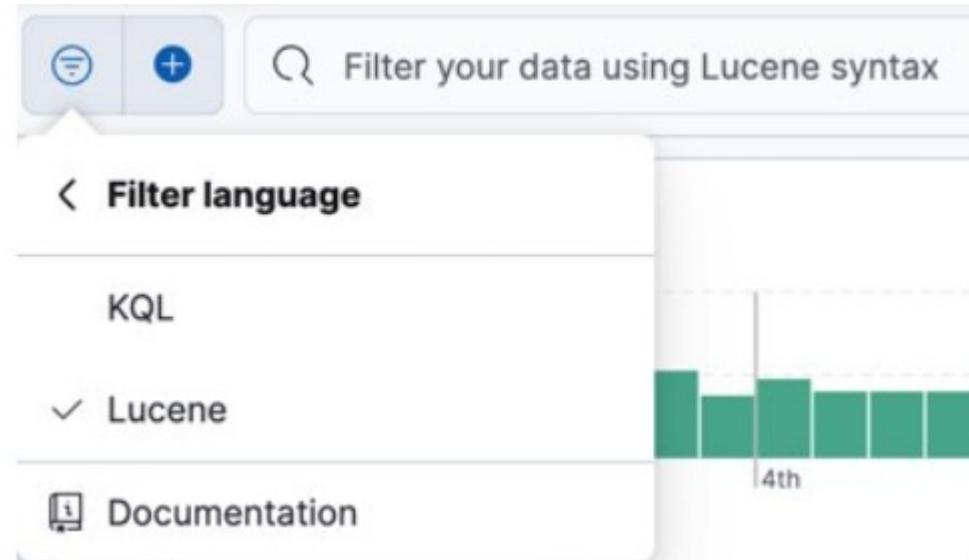
```
{  
  "user": [ {  
    "names": [ { "first": "John", "last": "Smith"},  
              { "first": "Alice", "last": "White" }  
            ]  
  }]  
}
```

- La requête pour Alice devient :

user.names:{ first: "Alice" and last: "White" }

Syntaxe Lucene

- Lucene permet des fonctionnalités avancées, telles que les expressions régulières ou les fuzzy-search (mot approchant).
- Cependant, la syntaxe Lucene ne permet pas de rechercher des objets imbriqués ou des champs scriptés.



Kibana

Introduction
Data views et discover
KQL
ES|QL
Tableaux de bord

ES|QL

- **Elasticsearch Query Language, ES|QL** peut accélérer l'exploration des données
- ES|QL permet d'enchaîner plusieurs commandes
 - Avec une seule requête, on peut rechercher, agréger, calculer et effectuer des transformations de données.
- ES|QL est disponible dans Discover ou dans DevTools avec l'API _eq/
- De l'aide est apportée via :
 - Un assistant pour la complétion
 - Un éditeur multiligne pour les requêtes longues
 - Un historique des requêtes

Éléments de syntaxe

- Chaque requête démarre avec une commande indiquant la source des données :
from kibana_sample_data_logs
- Une commande source peut être complétée par des commandes de traitement :
from kibana_sample_data_logs | limit 10
- Le résultat de la requête est un tableau de résultat.
 - Il est possible de sélectionner les colonnes que l'on veut garder :
***FROM kibana_sample_data_logs
| KEEP @timestamp, bytes, geo.dest***
 - De trier :
***FROM kibana_sample_data_logs
| KEEP @timestamp, bytes, geo.dest
| SORT bytes DESC***
 - De filtrer :
***FROM kibana_sample_data_logs
| WHERE bytes > 10000***

Analyse et visualisation

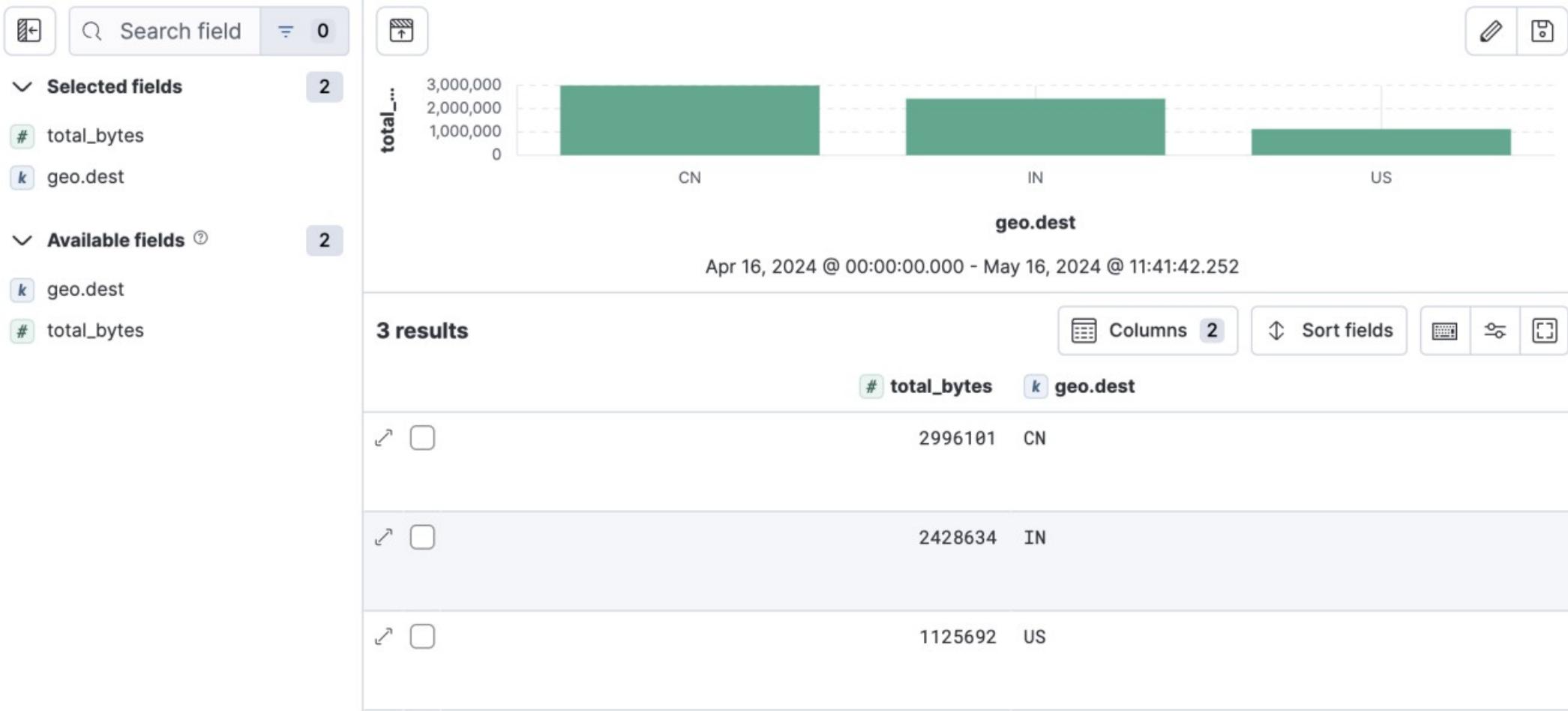
- La commande **STATS** permet des calculs statistiques comme SUM, AVG, MAX, MIN, COUNT, etc.
- Elle s'utilise avec le mot-clé **BY** pour regrouper les données.
- La visualisation de Discover¹ s'adapte à la query ES|QL
- La visualisation est éditable : Changement des couleurs, des axes, etc.
- Et peut être sauvegardée pour utilisation dans un Tableau de bord

1. Par défaut un histogramme comptant le nombre de document groupé via le temps



```
1 FROM kibana_sample_data_logs
2 | STATS total_bytes = SUM(bytes) BY geo.dest
3 | SORT total_bytes DESC
4 | LIMIT 3
```

4 lines @timestamp found

[Submit feedback](#) [Show recent queries](#)

Edition et sauvegarde

- L'histogramme peut être édité et sauvegardé via la barre de bouton 
- L'édition permet de modifier le type de visualisation, de couleur,
- La sauvegarde permet d'ajouter la visualisation à un dashboard
- A l'intérieur du dashboard, on peut ré-éditer la requête

Enrichissement

- La commande ***ENRICH*** permet d'enrichir le jeu de données avec des champs provenant d'un autre jeu de données.
- Il est nécessaire d'avoir préalablement mis au point le jeu de données et la jointure dans une ***ENRICH POLICY***

Enrich Policy

Create enrich policy

Specify how to retrieve and enrich your incoming data.

Configuration

Policy name

countries

Policy type

Match

Source indices

countries

Query (optional)

{

}

Defaults to: [match_all](#) query.

[Next >](#)

2

Field selection

Match field

geo.dest

Enrich fields

country_name

< Back

Next >

Usage

FROM kibana_sample_data_logs

| STATS total_bytes = SUM(bytes) BY geo.dest

| SORT total_bytes DESC

| LIMIT 3

| ENRICH countries

- Sans ENRICH Policies on peut également écrire la requête comme suit :

FROM kibana_sample_data_logs

| ENRICH WITH country_index

ON kibana_sample_data_logs.geo.dest = country_index.country_name

FIELDS country_code, country_area

Alertes

- ES|QL pour servir à créer des alertes.

Discover → Alerts → Create search threshold rule

- La requête ES|QL est associée à une fenêtre temporelle.
- Si la requête retourne des documents l'alerte est déclenchée

Create rule

Name

High traffic

Tags (optional)

Elasticsearch query

Alert when matches are found during the latest query run. [Learn more](#)

Define your query using ES|QL

```
1 FROM kibana_sample_data_logs
2 | STATS total_bytes = SUM(bytes) BY geo.dest
3 | SORT total_bytes DESC
4 | WHERE total_bytes>10000000
```

4 lines @timestamp detected

Select a time field

@timestamp

Set the time window

5

minutes

▶ Test query

Query matched 2 documents in the last 5m.

total_bytes	geo.dest
14888980	CN
12974807	IN

Alerts generated query matched

Cancel

✓ Save

Kibana

Introduction
Data views et discover
KQL
ES|QL
Les tableaux de bord

Tableaux de bords et panels

- Les tableaux de bord agrège en une page un ensemble de panels qui clarifient les données et qui permettent de se concentrer uniquement sur les données qui sont importantes.
- Les panels affichent les données sous forme de graphiques, de tableaux, de cartes, etc..
- Un tableau de bord peut être partagé sous forme de lien.
- Il s'actualise automatiquement en fonction de son filtre temporel

Types de panel

- Kibana propose différents types de panels :
 - **Éditeurs** : Visualisations des données.
 - **Cartes** : Affichages de données géographiques.
 - **Swimlane des anomalies** : Adapté aux jobs de ML détectant les anomalies.
 - **Graphique d'anomalies** : Graphique d'anomalies à partir de l'explorateur d'anomalies. (ML)
 - **Flux de journaux** : Tableau de journaux en direct.
 - **Outils** : Filtres interactifs avec les panneaux de contrôle.
 - **Texte** : Ajout de simple texte.
 - **Image** : Ajout d'image personnalisée.

Visualisations

- Les visualisations sont basées sur des recherches ES
- En utilisant des agrégations, on peut créer des graphiques qui montrent des tendances, des pics, ...
- Différents éditeurs pour les visualisations :
 - **Lens** : Drag and Drop
 - **Maps** : Géolocalisation
 - **TSVB** : Adaptées aux séries temporelles
 - Visualisations personnalisées : **Vega**
 - Visualisations basées sur des **agrégations**
 - **Timelion** : Séries temporelles avec un langage d'expression

Mise en place de Dashboard

- ***Dashboard → Create Dashboard***
Lors de la création, On est tout de suite en mode édition
- Les panneaux sont créés à l'aide des éditeurs, accessible
 - via la barre d'outils du tableau de bord
 - Ou via la bibliothèque de visualisation (Visualize Library)
- Ils peuvent être ajoutés à un dashboard via :
 - La bibliothèque de visualisation
 - Directement à partir des résultats de recherche de Discover
- A la sauvegarde d'un panel, on a le choix de le sauvegarder :
 - Dans la bibliothèque (pour réutilisation)
 - Directement dans le tableau de bord
- Lors que l'on modifie un panel provenant de la bibliothèque, on a le choix entre :
 - Modifier le panel de la librairie
 - Dissocier le panel de la librairie

Disposition des panels

- Les panels peuvent ensuite être déplacés, redimensionnés
- Chaque panel contient un titre, une description et un intervalle de temps
- Des panels de texte ou d'image permettent de contextualiser les informations
- Sur chaque panneau, il est possible via le menu inspect de voir la requêtes correspondantes et les données présentées au format CSV

Finalisation du dashboard

- Lorsque les panels du tableau de bord sont mis au point, on peut renseigner les Settings du tableau de bord :
 - Titre et description
 - Les tags
 - D'autres options :
 - Sauvegarder le filtre temporel avec le dashboard
 - Jouer sur les marges entre les panneaux
 - Afficher ou non les titres de panneaux
 - Synchroniser les panneaux entre eux : palette de couleur, forme de curseur, tooltip
- Il ne reste plus alors que partager le tableau de bord
- On peut également l'exporter au format JSON pour le réimporter sur une autre instance Kibana

Options de partage

- Les tableaux de bord, les requêtes sauvegardées, les librairies et les Canvas peuvent être partagés.
- Différents formats de partage sont possibles :
 - Rapports PDF ou PNG : Version payante
 - Rapports CSV pour les recherches enregistrées et de visualisations Lens.
 - Sous forme de lien : recherches enregistrées, des tableaux de bord et visualisations.
 - Sous forme d'iframe : le tableau de bord peut être intégré dans une autre page Web.

Lens

- Lens permet de travailler via Glisser/Déposer des champs.
- Il propose alors les meilleures pratiques de visualisation pour appliquer les champs et créer une visualisation qui affiche au mieux les données.
- Il émet également des suggestions permettant de changer de type de visualisation
- Avec Lens, il est possible de :
 - Créer des graphiques en aires, en courbes et à barres avec des calques pour afficher plusieurs indices et types de graphiques.
 - Modifier la fonction d'agrégation pour modifier les données dans la visualisation.
 - Créer des tables personnalisées.
 - Effectuer des calculs sur les agrégations à l'aide de formules.
 - Utiliser des décalages temporels pour comparer les données dans deux intervalles de temps, par exemple d'un mois à l'autre.
 - Ajouter des annotations et des lignes de référence.

TSVB

- ***Time Series Visual Builder (TSVB)*** est un outil plus spécialisé, conçu spécifiquement pour la création de visualisations de séries temporelles complexes.
- Outre les graphiques de lignes classiques, TSVB supporte des types de visualisations comme les "gauge" (jauges), "markdown" (texte enrichi), et des barres d'état, ce qui le rend idéal pour les tableaux de bord de monitoring et d'observation.
- TSVB offre des options avancées pour l'agrégation et le filtrage des données

[Time Series](#)[Metric](#)[Top N](#)[Gauge](#)[Markdown](#)[Table](#) Auto apply

The changes will be automatically applied.

...

[Data](#) [Panel options](#) [Annotations](#)

Label

Metrics Options

Aggregation

Sum bytes

Group by

Everything

Threshold

Label

Vega

- Vega et Vega-Lite sont deux grammaires permettant de créer des visualisations personnalisées.
- Il est nécessaire d'écrire des requêtes Elasticsearch manuellement.
- Vega et Vega-Lite utilise JSON

Agrégations

- Avec les visualisations basées sur ***Aggregations***, vous pouvez :
 - Diviser les graphiques jusqu'à trois niveaux d'agrégation, ce qui est plus que Lens et TSVB
 - Créer une visualisation avec des données non chronologiques
 - Utiliser une recherche enregistrée comme entrée
 - Trier les tableaux de données et utiliser les fonctionnalités de ligne de résumé et de colonne de pourcentage
 - Attribuer des couleurs aux séries de données
 - Étendre les fonctionnalités avec des plugins
- Elles ont les limitations suivantes :
 - Options de style limitées
 - Math n'est pas pris en charge
 - Les indices multiples ne sont pas pris en charge

Agrégations

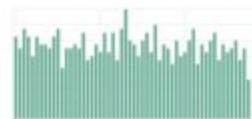
- **Area** : Visualise les contributions totales de plusieurs séries
- **Table de données** : Affichage en tableau des données agrégées
- **Line** : Compare différente séries
- **Metric**: Affichage d'un seul métrique
- **Gauge/Goal**: Une unique valeur avec des intervalles de bonnes/mauvaises valeurs ou par rapport à un objectif
- **Pie** : Camembert .
- **Heat map** : Tableau coloré
- **Nuage de tags** : Les tags les plus importants ont la plus grande police
- **Horizontal / Vertical bar** : Histogramme permettant la comparaison de séries



\$446.23
Avg. Ticket Price



Heavy Fog Clear
Cloudy Rain Sunny
Hail Thunder & Lightning



Timelion

- **Timelion** est un visualiseur dédié aux données temporelles qui permet de combiner des sources de données complètement indépendantes dans le même graphique.
- Il est piloté par un langage d'expression simple permettant de récupérer les données et d'effectuer des calculs.
- Avec timelion, il est possible de répondre à ce type de question :
Quel est le pourcentage de la population japonaise qui a visité mon site ?

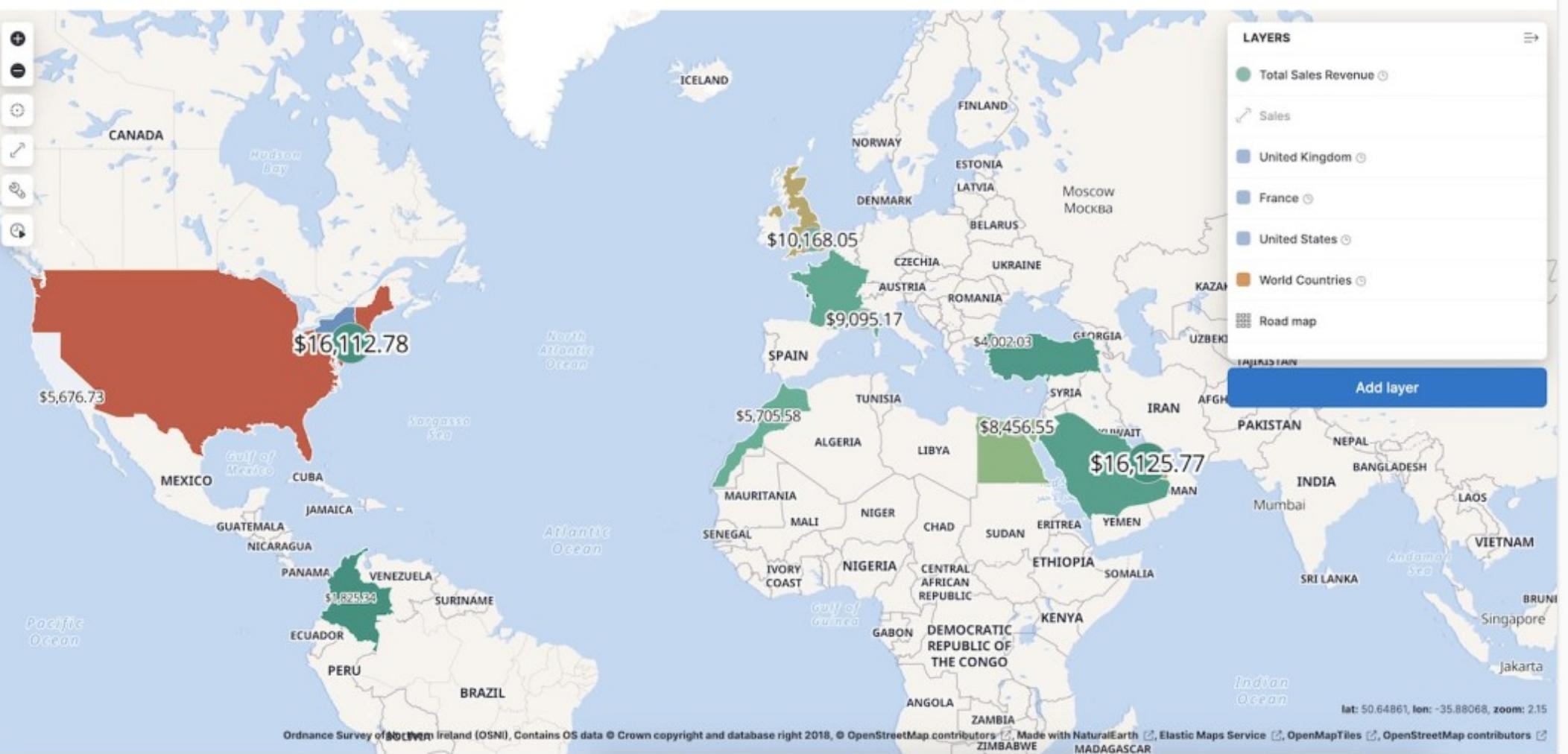
Exemples

```
# Affichage en temps réel de la moyenne du
# pourcentage d'utilisation CPU user à partir
# d'un index metricbeats d'elastic search
.es(index=metricbeat-*, timefield='@timestamp',
metric='avg:system.cpu.user.pct')

# La même chose en ajoutant une série affichant
# les données pour un offset d'1h
.es(index=metricbeat-*, timefield='@timestamp',
metric='avg:system.cpu.user.pct'), .es(offset=-1h,
index=metricbeat-*, timefield='@timestamp',
metric='avg:system.cpu.user.pct')
```

Maps

- L'éditeur **Maps** permet de :
 - Créez des cartes avec plusieurs couches et index.
 - Animez des données spatio-temporelles.
 - Téléchargez des fichiers GeoJSON et des calques de forme.
 - Intégrez votre carte dans des tableaux de bord.
 - Utiliser des symboles en fonction des valeurs d'une donnée.



Dimensionnement et exploitation

Architectures ingestion

Monitoring Logstash

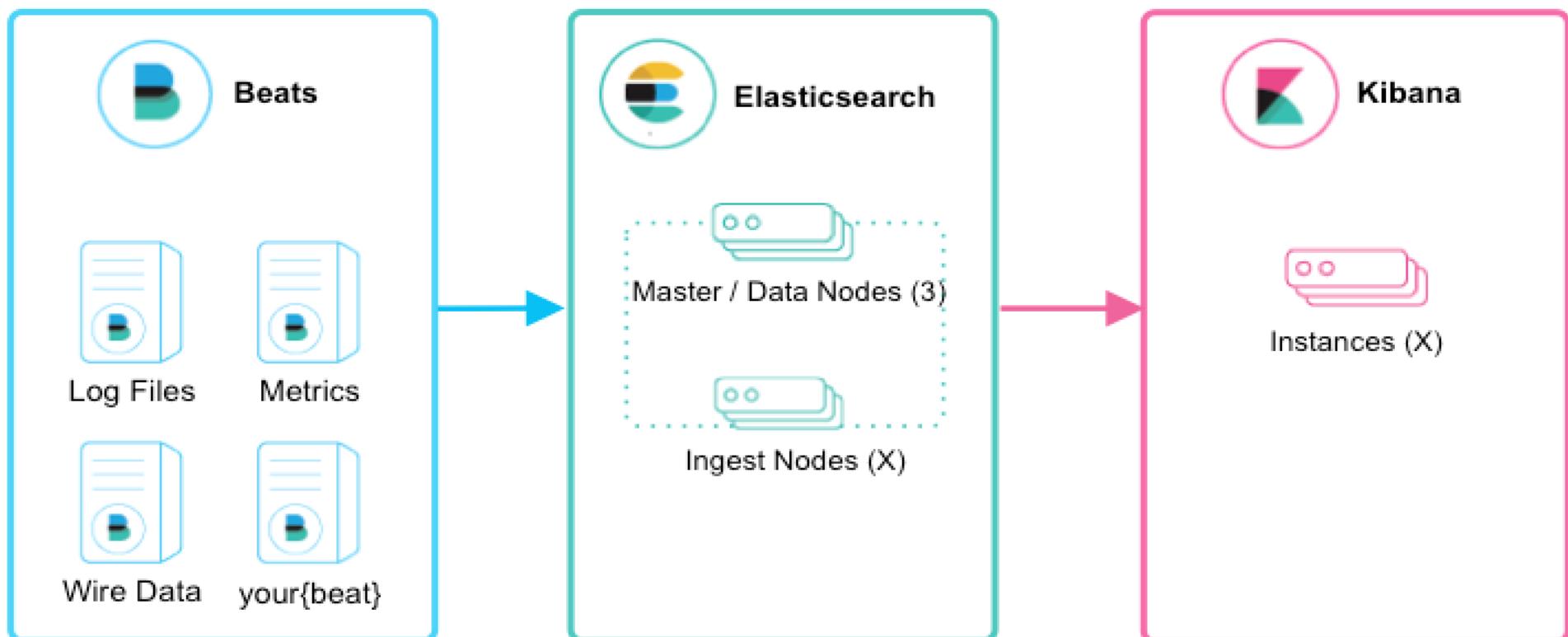
Architecture Indexation/Recherche

Gestion des shards

Monitoring ES

Exploitation

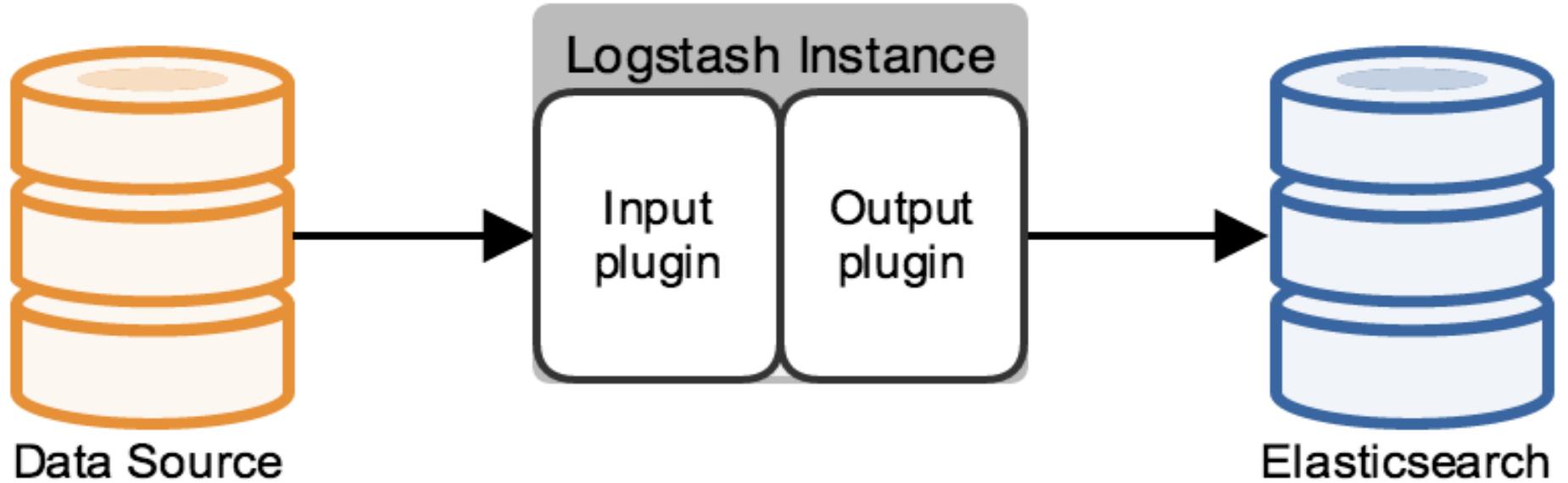
Architecture sans logstash



Apports de logstash

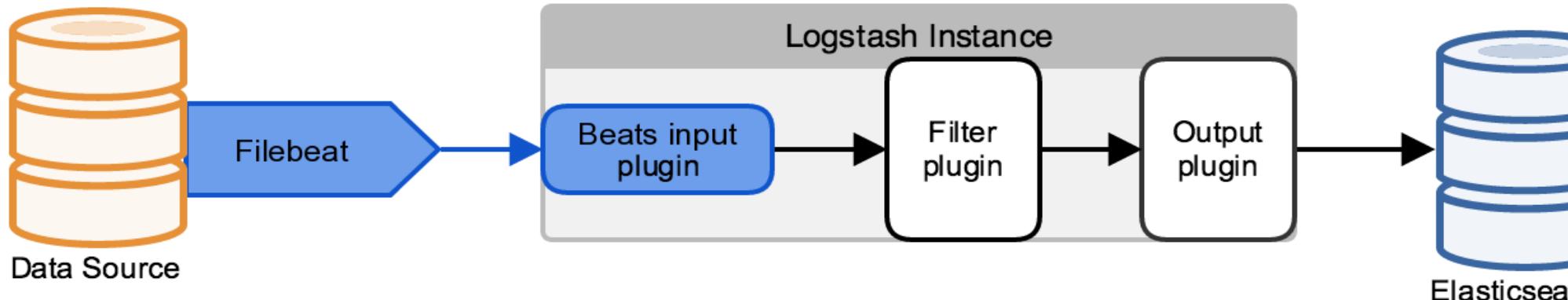
- S'adapter à des pics de charge, via le système de bufferisation intrinsèque de Logstash
- Ingérer des données provenant d'autres sources de données : BD, S3, ou files de message
- Émettre des données vers plusieurs destinations : S3, HDFS ou fichier
- Insérer de la logique conditionnelle dans le traitement des événements

Architecture minimale

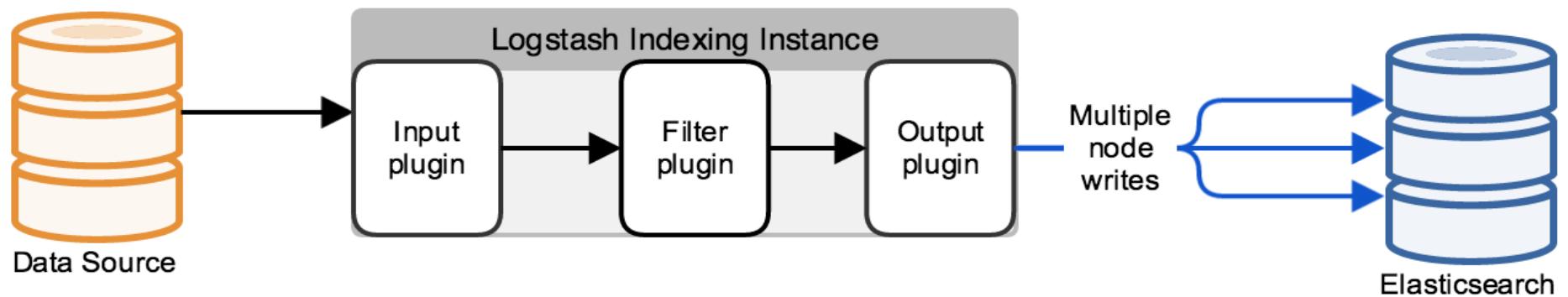


Architecture avec un Beat

- L'utilisation d'un *Beat* permet de déporter du traitement sur la machine source



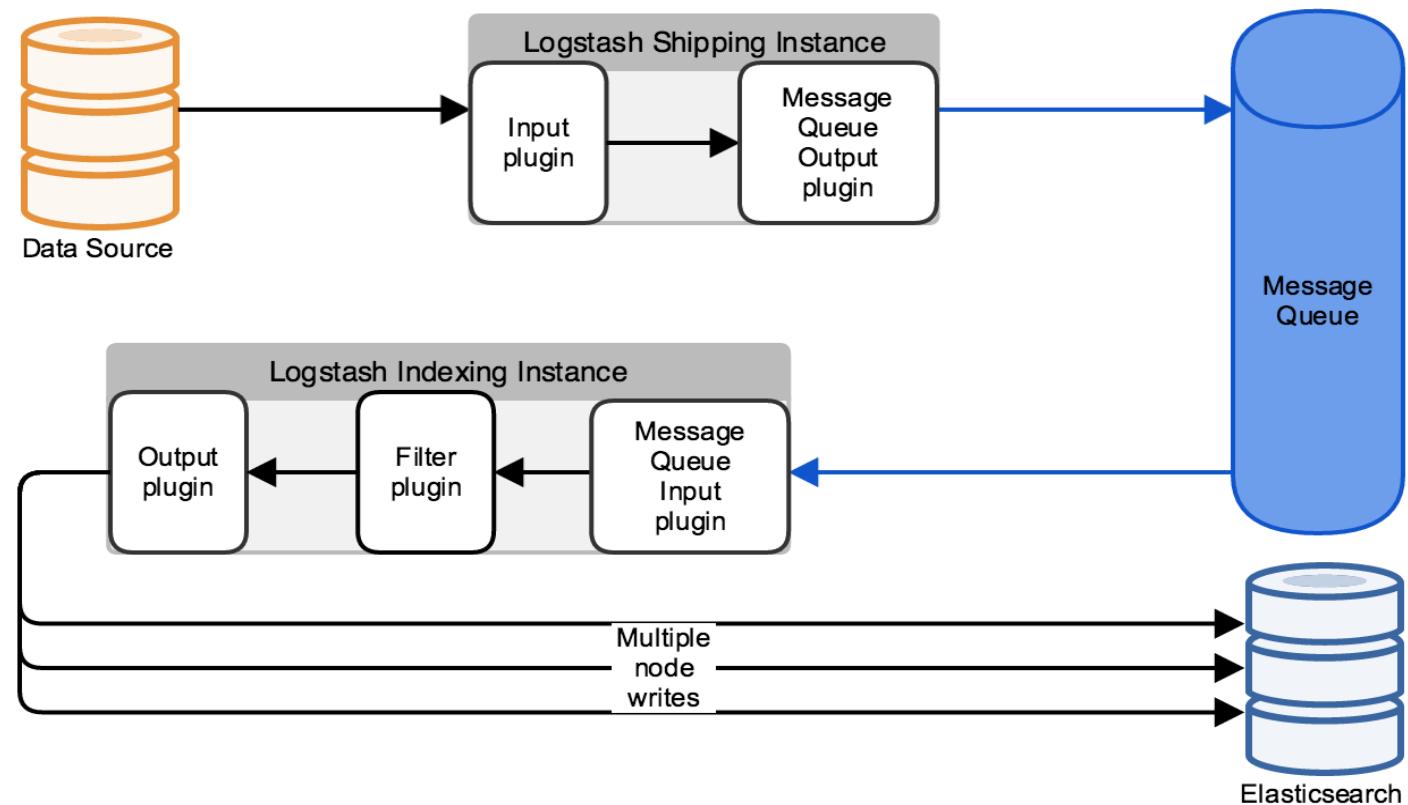
Load balancing sur les noeuds de ElasticSearch



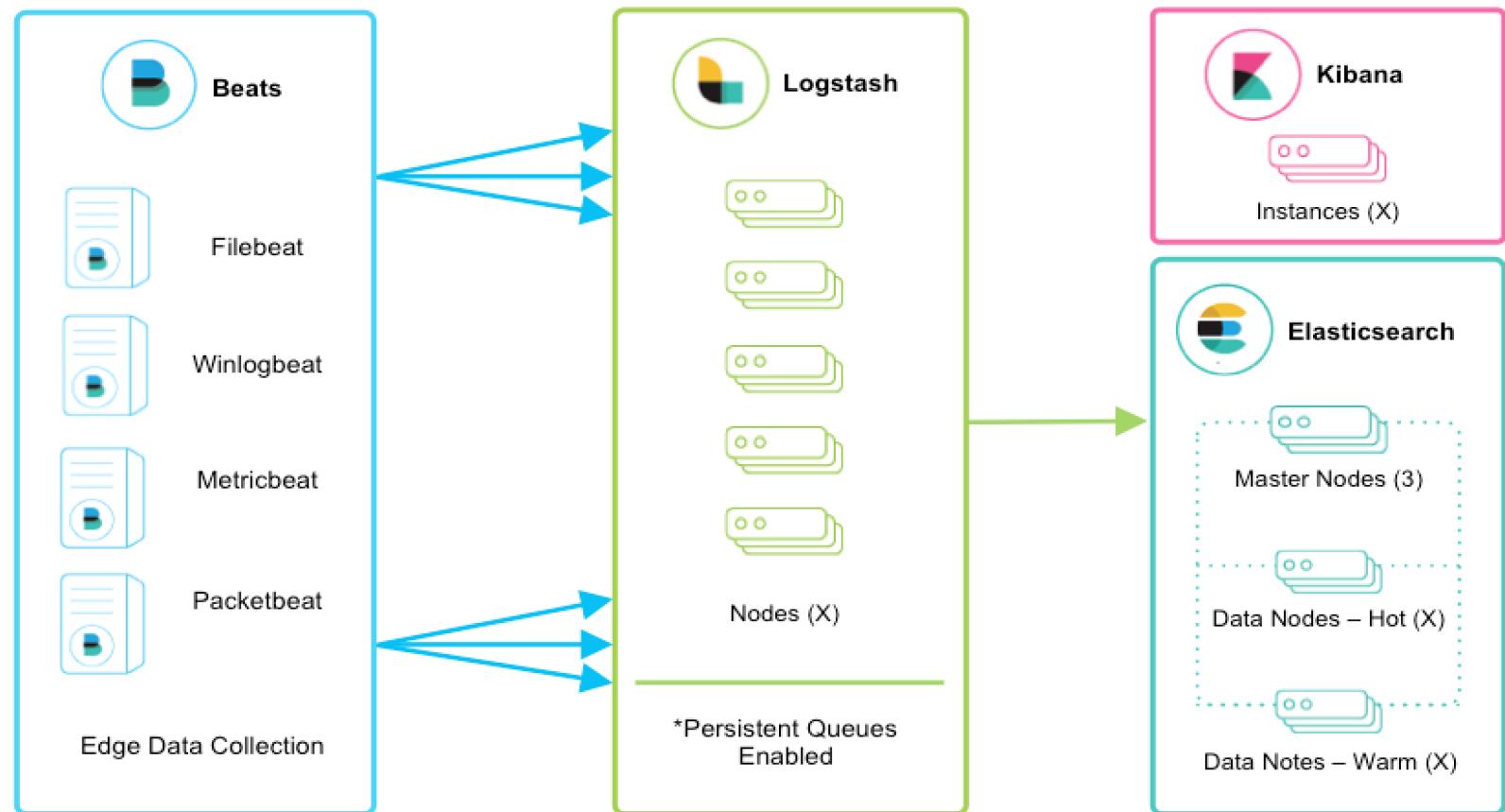
Message broker

- Afin de faire face à des pics de débit, l'architecture peut inclure des message brokers (Redis, Kafka, RabbitMQ).
 - Cela peut soulager le travail d'indexation d'ElasticSearch
- Des instances de logstash écrivent vers une file de message
- D'autres lisent de la file, effectue les traitements et envoient vers ElasticSearch

Message broker



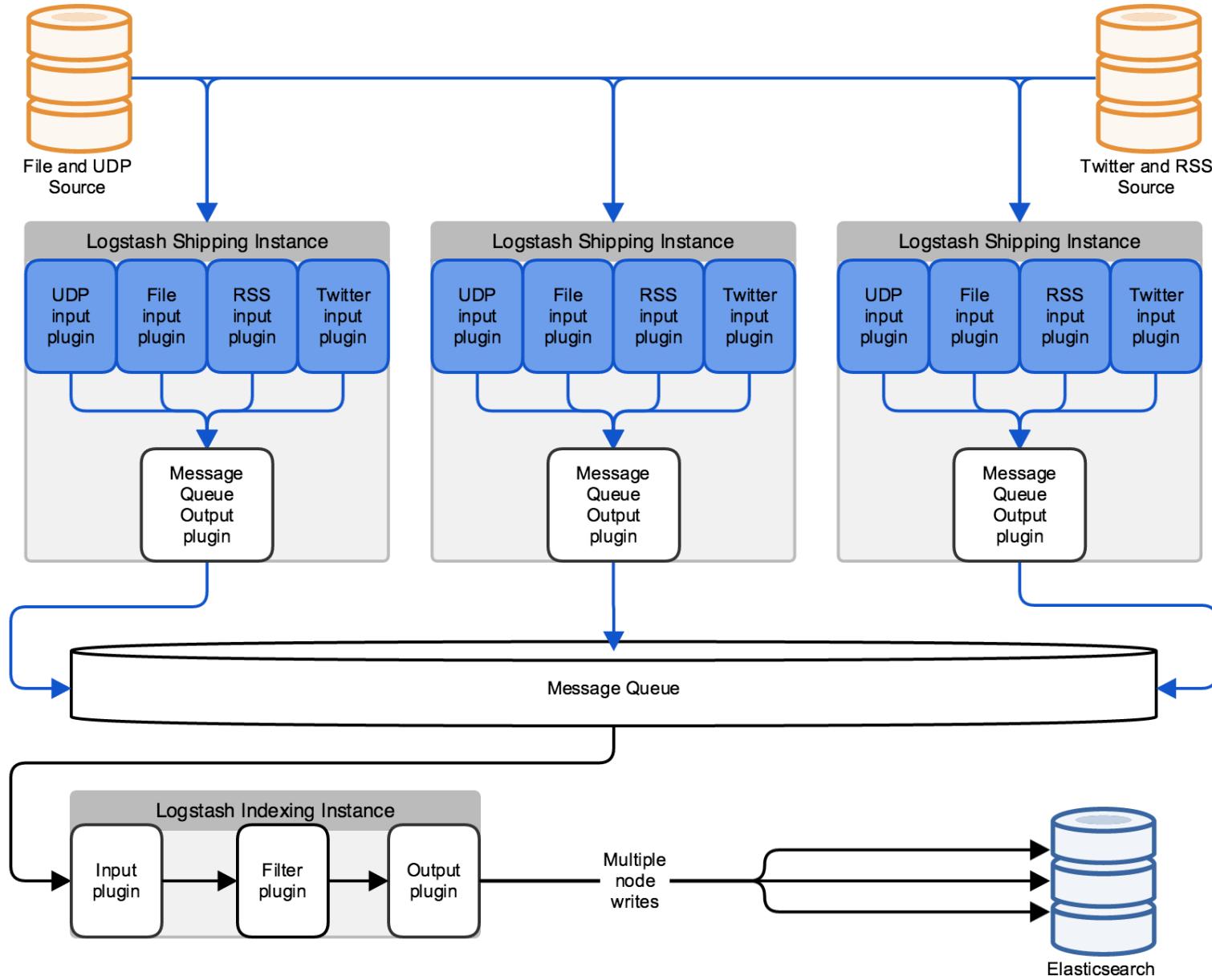
Répartir la charge sur plusieurs Logstash



Caractéristiques

- **Scalabilité** : plusieurs instances de logstash exécutent la même pipeline, des nœuds peuvent être ajoutés, les beats doivent distribuer la charge sur le cluster
- **At-least-one** : les plugins Filebeat et Winlogbeat le garantissent, les autres non
- **Persistent-queue** : Par défaut, logstash utilise des files mémoire pour le traitement batch des événements ; utiliser des persistent-queue pour éviter des pertes de données en cas de crash
- **Secure transport** : X-Pack
- **Monitoring** : UI inclut dans X-Pack mais libre d'utilisation, API

Haute disponibilité via des connections multiples



Architecture en tiers

- Un déploiement Logstash a typiquement un pipeline composé de différents tiers :
 - Le tiers d'entrée consomme les données des sources et est composé d'instance logstash avec les bons plugins d'entrée
 - Le broker de messages sert de buffer et de protection contre des pannes
 - Le tiers de filtrage traite et normalise les données
 - Le tiers d'indexation déplace les données traitées vers ElasticSearch
- Chaque tiers peut être scalés selon les besoins

Dimensionnement et exploitation

Architectures ingestion
Monitoring Logstash

Architecture Indexation/Recherche
Gestion des shards
Monitoring ES
Exploitation

Introduction

- Logstash fournit une API REST pour la surveillance
- L'API est divisé en 4 domaines
 - **Node Info** : Informations de configuration d'un nœud
 - **Plugins** : Les plugins installés
 - **Node stats** : Métriques sur les nœuds
 - **Hot threads** : Threads avec un gros usage CPU
- Les réponses JSON peuvent être formattés par les paramètres :
 - *pretty=true*
 - *human=true*

Node info

- Sur une pipeline, le nombre de workers, la taille et le délai de batch

GET /_node/pipelines

- Sur l'OS, les versions et les processeurs disponibles

GET /_node/os

- Sur la JVM, processus, Heap et garbage collector

GET /_node/jvm

Plugins info

GET /_node/plugins

```
{  
  "total": 91,  
  "plugins": [  
    {  
      "name": "logstash-codec-collectd",  
      "version": "3.0.2"  
    },  
    {  
      "name": "logstash-codec-dots",  
      "version": "3.0.2"  
    },  
    .  
    .  
    .  
  ]}
```

Node stats

GET /_node/stats/<types>

- Soit tous les métriques, soit limité à un type qui peut être :
 - **jvm** : JVM stats, threads, usage mémoire et garbage collectors.
 - **process** : processus, descripteurs de fichiers, consommation mémoire et usage CPU
 - **mem** : Usage mémoire .
 - **pipelines** : Métrique sur la pipeline Logstash

Exemple pipeline

```
{  
  "pipeline": {  
    "events": { "duration_in_millis": 7863504, "in": 100, "filtered": 100, "out": 100 },  
    "plugins": {  
      "inputs": [],  
      "filters": [  
        {  
          "id": "grok_20e5cb7f7c9e712ef9750edf94aefb465e3e361b-2",  
          "events": { "duration_in_millis": 48, "in": 100, "out": 100 },  
          "matches": 100,  
          "patterns_per_field": { "message": 1 },  
          "name": "grok"  
        },  
        {  
          "id": "geoip_20e5cb7f7c9e712ef9750edf94aefb465e3e361b-3",  
          "events": { "duration_in_millis": 141, "in": 100, "out": 100 },  
          "name": "geoip"  
        }  
      ],  
      "outputs": [  
        {  
          "id": "20e5cb7f7c9e712ef9750edf94aefb465e3e361b-4",  
          "events": { "in": 100, "out": 100 },  
          "name": "elasticsearch"  
        }  
      ]  
    },  
    " reloads": { "last_error": null, "successes": 0, "last_success_timestamp": null, "last_failure_timestamp": null, "failures": 0 }  
  }  
}
```

Hot Threads

GET /_node/hot_threads

- Retourne des informations sur les threads qui prennent le plus de CPU
- Pour chaque thread :
 - Le pourcentage de CPU
 - Son état
 - Sa stack trace

Dimensionnement et exploitation

Architectures ingestion
Monitoring Logstash

Architecture Indexation/Recherche

Gestion des shards
Monitoring ES
Exploitation

Matériel

- **RAM** : Le talon d'Achille de ELS. Une machine avec 64 GB est idéale ; 32 GB et 16 GB sont corrects
- **CPU** : Favoriser le nombre de cœur plutôt que la rapidité du CPU
- **Disques** : Si possible rapides, SSDs ? Éviter NAS (network-attached storage)

JVM

- Utiliser la version embarquée de la distribution
- Surtout ne pas modifier la configuration de la JVM fournie par ELS. Elle est issue de l'expérience

Gestion de configuration

- Utiliser de préférence des outils de gestion de configuration comme Puppet, Chef, Ansible ...
=> Sinon la configuration d'un cluster avec beaucoup de nœuds devient rapidement un enfer

Personnalisation d'une configuration

- La configuration par défaut défini déjà beaucoup de choses correctement, Il y a donc peu à personnaliser. Cela se passe dans le fichier `elasticsearch.yml`
 - Le **nom du cluster** (le changer de `elasticsearch`)
`cluster.name: elasticsearch_production`
 - Le **nom des nœuds**
`node.name: elasticsearch_005_data`
 - Les **chemins** (hors du répertoire d'installation de préférence)
`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`

Spécialisation des nœuds

- Tous les nœuds d'un cluster se connaissent mutuellement et peuvent rediriger des requêtes HTTP vers le nœud approprié. Il est possible de spécialiser les nœuds afin de répartir la puissance entre la charge de gestion des données et la charge d'ingestion.
- Les différents types de nœuds sont :
 - Nœuds pouvant être **maître** : ***node.master*** à *true*
 - Nœuds de **données** : ***node.data*** à *true*. Détient les données et effectue les tâches d'indexation et de recherche
 - Nœuds **d'ingestion** : ***node.ingest*** à *true*. Exécute les pipelines d'ingestion
 - Nœuds de **coordination** : Nœuds acceptant les requêtes et redirigeant vers les nœuds appropriés
 - Nœuds **tribe** ou **cross-cluster** (Version 6.x). Propriétés ***tribe.**** Nœuds pouvant effectuer des recherches vers plusieurs cluster.

Noeuds maître

- Le nœud maître est responsable d'opérations légères :
 - Création ou suppression d'index
 - Surveillance des nœuds du cluster
 - Allocations des shards
- Dans un environnement de production, il est important de s'assurer de la stabilité du nœud maître.
- Il est conseillé pour de gros cluster de ne pas charger les nœuds maîtres avec des travaux d'ingestion, d'indexation ou de recherche.

```
node.master: true  
node.data: false  
node.ingest: false  
search.remote.connect: false
```

Configurations des nœuds

- Configuration d'un nœud de **données**

```
node.master: false  
node.data: true  
node.ingest: false  
search.remote.connect: false
```

- Configuration d'un nœud **d'ingestion**

```
node.master: false  
node.data: false  
node.ingest: true  
search.remote.connect: false
```

- Configuration d'un nœud de **coordination**

```
node.master: false  
node.data: false  
node.ingest: false  
search.remote.connect: false
```

Bootstrapping ELS

- Lors du premier démarrage du cluster en production, il est nécessaire de spécifier ***cluster.initial_master_nodes*** qui liste la liste des nœuds maîtres éligibles. (En général, les noms des nœuds sont précisés)
- Ensuite, lors d'un redémarrage de nœud, la recherche du maître s'effectue avec :
 - Les hôtes listés dans ***discovery.seed_hosts***
 - Le dernier maître connu

Mémoire heap

- L'installation par défaut d'ELS est configuré avec 1 GB de heap (mémoire JVM). Ce nombre est bien trop petit
- 3 façons pour changer la taille de la heap .
 - Le fichier **jvm.options**
 - Via la variable d'environnement **ES_HEAP_SIZE**
export ES_HEAP_SIZE=10g
 - Via la commande en ligne
.bin/elasticsearch -Xmx=10g -Xms=10g
- 2 recommandations standard :
 - donner 50% de la mémoire disponible à ELS et laisser l'autre moitié vide. En fait Lucene occupera allègrement l'autre moitié
 - Ne pas dépasser 32Go

Swapping

- Éviter le swapping à tout prix.
- Éventuellement, le désactiver au niveau système

```
sudo swapoff -a
```

- Ou au niveau ELS

```
bootstrap.memory_lock: true
```

(Anciennement `bootstrap.mlockall`)

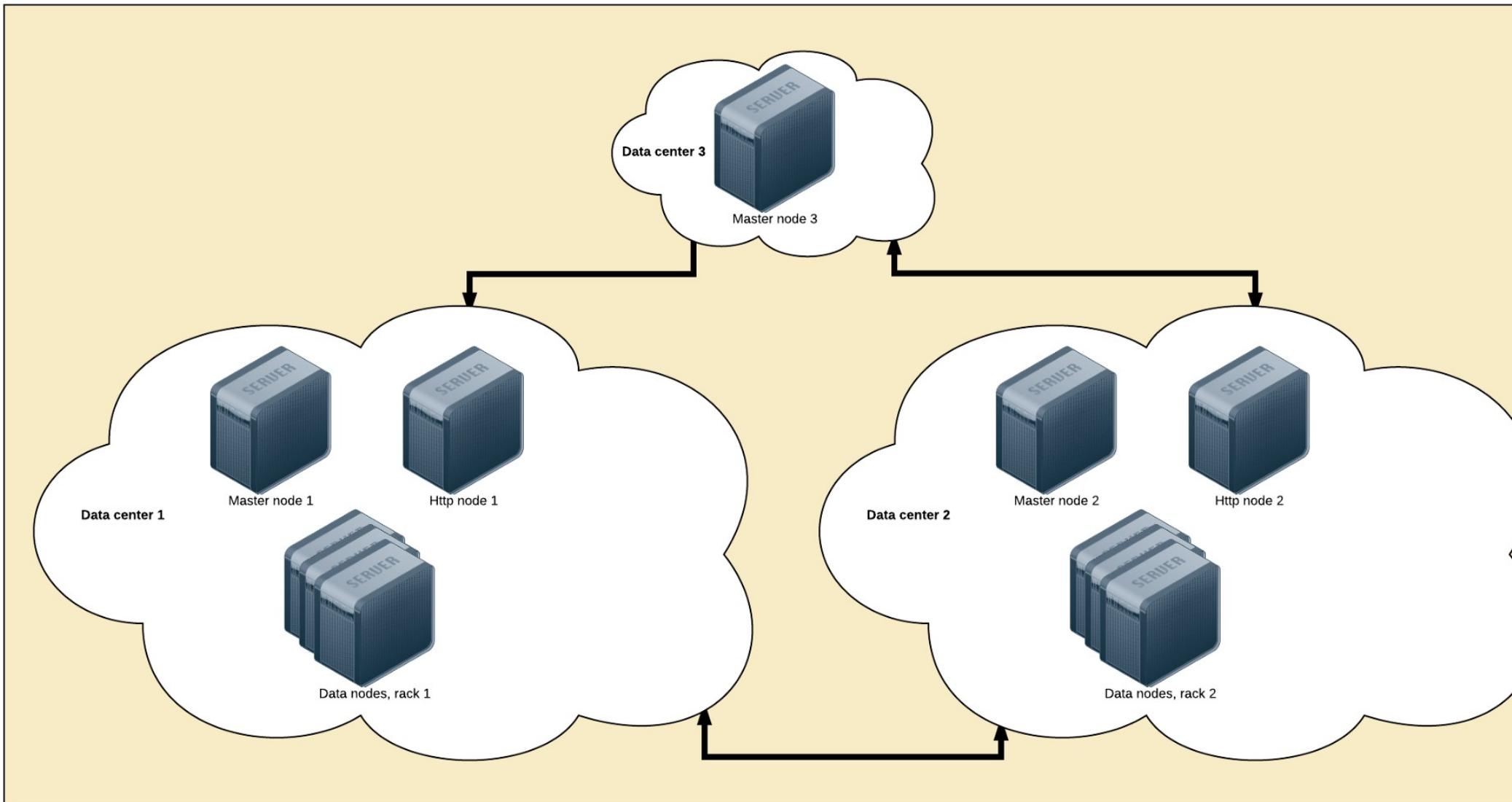
Descripteurs de fichiers

- Lucene utilise énormément de fichiers. ELS énormément de sockets
- La plupart des distributions Linux limite le nombre de descripteurs e fichiers à 1024.
- Cette valeur doit être augmenté à 64,000

Architecture fault-tolerant

- Une architecture tolérante aux pannes typique est de distribuer les nœuds sur différents data center
 - au minimum 2 data center principaux contenant les nœuds de données et 1 de backup contenant un éventuel master node
 - 3 master node
 - 2 nœuds pour exécuter les requêtes http (1 par data center principal)
 - Des nœuds de données distribués sur les 2 data center principaux

Architecture fault-tolerant



Dimensionnement et exploitation

Architectures ingestion
Monitoring Logstash

Architecture Indexation/Recherche

Gestion des shards

Monitoring ES
Exploitation

Segments Lucene

- Chaque shard d'Elasticsearch est un index Lucene.
- Un index Lucene est divisé en de petits fichiers : les **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

Fusion de segments

- Lucene crée des segments lors de l'indexation. Les segments sont immuables
- Lors d'une recherche, les segments sont traités de façon séquentielle
=> Plus il y a de segments, plus les performances de recherche diminuent
- Pour optimiser la recherche, Lucene propose l'opération **merge** qui fusionne de petits segments en de plus gros.
 - C'est une opération assez lourde qui peut impacter les opérations d'indexation et de recherche. Elle est effectué périodiquement par un pool de threads dédié
 - On peut forcer une opération de merge :
`curl -XPOST 'localhost:9200/logstash-2017.07*/_forcemerge?max_num_segments=1'`
 - Pour que le merge réussisse, il faut que l'espace disque soit 2 fois la taille du shard

Dimensionnement du nombre de shards

- Le nombre de shards est défini lors de la création de l'index. (Par défaut : 1)
- Seulement la charge réelle permet de trouver la bonne valeur pour le nombre de shards.
- Redimensionner un index en production nécessite une réindexation et éventuellement l'utilisation d'alias d'index.

Avantages pour de nombreux shards

- Disposer de beaucoup de shards sur de gros indices et de gros cluster (Plus de 20 nœuds de données) apporte certains avantages :
 - Meilleures allocations entre les nœuds
 - De petits shards sur beaucoup de nœuds rend le processus de recovery plus rapide (Perte d'un nœud de données ou arrêt du serveur).
 - Cela peut régler des problèmes mémoire, lorsque l'on exécute de grosses requêtes.
 - Les gros shards rendent les processus d'optimisation de Lucene plus difficile. Lors d'une fusion de segments Lucene, il faut avoir 2 fois la taille du shard comme espace libre.
- Par contre, avoir de nombreux shards peut surcharger le master et le cluster devient alors très instable

Recommandations

- Repère :
 - Des shards de 10GB semblent offrir un bon compromis entre la vitesse d'allocation, et la gestion du cluster .
- => Pour une moyenne de 2GB pour 1 million de documents :
- De 0 à 4 millions de documents par index: 1 shard.
 - De 4 à 5 million documents par index: 2 shards
 - > 5 millions documents : 1 shards par 5 millions.

Exemple de script de resizing

```
#!/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 ',')
    # Dimensionnement du nombre de shard en fonction du nbre de 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
```

<https://thoughts.t37.net/resizing-your-elasticsearch-indexes-in-production-d7a0402d137e>

Dimensionnement et exploitation

Architectures ingestion
Monitoring Logstash

Architecture Indexation/Recherche

Gestion des shards

Monitoring ES

Exploitation

API et Kibana

- ELS offre une API permettant d'obtenir certains métriques d'un cluster
- Des tableaux de bord Kibana utilise cette API pour la surveillance en continue du 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
}
```

Information au niveau des index

```
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"
      ],
    }
  }
}
```

Sections indices

- La section **indices** liste des statistiques agrégés pour tous les index d'un nœud.
- Il contient les sous-sections suivantes :
 - **docs** : combien de documents résident sur le nœud, le nombre de documents supprimés qui n'ont pas encore été purgés
 - **store** indique l'espace de stockage utilisé par le nœud
 - **indexing** le nombre de documents indexés
 - **get** : Statistiques des requêtes *get-by-ID*
 - **search** : le nombre de recherches actives, nombre total de requêtes le temps d'exécution cumulé des requêtes
 - **merges** fusion de segments de Lucene
 - **filter_cache** : la mémoire occupée par le cache des filtres
 - **id_cache** répartition de l'usage mémoire
 - **field_data** mémoire utilisée pour les données temporaires de calcul (utilisé lors d'agrégation, le tri, ...)
 - **segments** le nombre de segments Lucene. (chiffre normal 50–150)

Section OS et processus

- Ce sont les chiffres basiques sur la charge CPU et l'usage mémoire au niveau système
 - CPU
 - Usage mémoire
 - Usage du swap
 - Descripteurs de fichiers ouverts

Section JVM

- La section **jvm** contient des informations critiques sur le processus JAVA
- En particulier, il contient des détails sur la collecte mémoire (garbage collection) qui a un gros impact sur la stabilité du cluster
- La chose à surveiller est le nombre de collectes majeures qui doit rester petit ainsi que le temps cumulé dans les collectes ***collection_time_in_millis*** .
- Si les chiffres ne sont pas bon, il faut rajouter de la mémoire ou des nœuds.

Section pool de threads

- ELS maintient des pools de threads pour ses tâches internes.
- En général, il n'est pas nécessaire de configurer ces pools.

File system et réseau

- ELS fournit des informations sur votre **système de fichiers** : Espace libre, les répertoires de données, les statistiques sur les IO disques
- Il y a également 2 sections sur le **réseau**
 - **transport** : statistiques basiques sur les communications inter-nœud (port TCP 9300) ou clientes
 - **http** : Statistiques sur le port HTTP. Si l'on observe un très grand nombre de connexions ouvertes en constante augmentation, cela signifie qu'un des clients HTTP n'utilise pas les connexions *keep-alive*. Ce qui est très important pour les performances d'ELS

Index stats API

- L'index stats API permet de visualiser des statistiques vis à vis d'un index

GET `my_index/_stats`

- Le résultat est similaire à la sortie de *node-stats* : Compte de recherche, de get, segments, ...

Overview

david

Elasticsearch

Overview

Health • Healthy

Version 8.16.0

Uptime 3 days

License Basic

Nodes: 3

Disk Available 51.18%
49.8 GB / 97.4 GB

JVM Heap 21.66%
2.6 GB / 12.0 GB

Indices: 67

Documents 5,996,027

Disk Usage 4.1 GB

Primary Shards 71

Replica Shards 74

Logs

No log data found
Set up [Filebeat](#), then configure your Elasticsearch output to your monitoring cluster.

Overview Nodes Indices Ingest Pipelines

Status	Alerts	Nodes	Indices	JVM Heap	Total shards	Unassigned shards	Documents	Data
Green	0	3	67	3.8 GB / 12.0 GB	145	0	5,996,755	4.1 GB

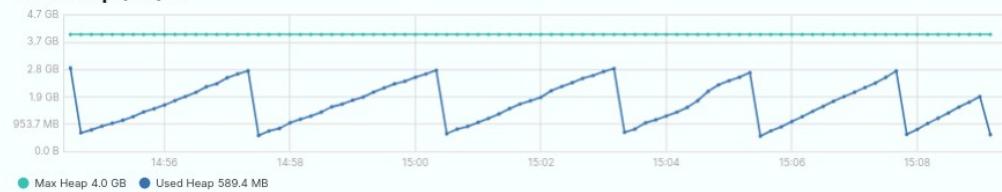
Filter Nodes...

Name	Alerts	Status	Roles	Shards	CPU Usage	Load Average	JVM Heap	Disk Free Space
★ node-1 192.168.21.196:9300	Clear	Online	master data data_content data_hot data_warm +6	49	↑ 2%	↑ 1.4	↓ 32%	↓ 49.9 GB
node-2 192.168.21.196:9301	Clear	Online	master data data_content data_hot data_warm +6	48	↑ 1%	↑ 1.29	↓ 29%	↓ 49.9 GB
node-3 192.168.21.196:9302	Clear	Online	master data data_content data_hot data_warm +6	48	↑ 1%	↑ 1.29	↑ 32%	↓ 49.9 GB

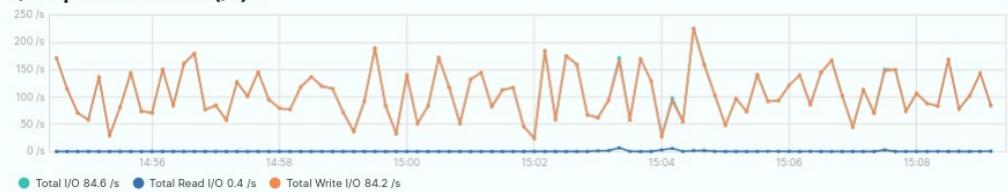
Rows per page: 20 < 1 >

Noeud

JVM Heap (GB) Ⓜ



I/O Operations Rate (/s) Ⓜ



CPU Utilization (%) Ⓜ



System Load Ⓜ



Latency (ms) Ⓜ

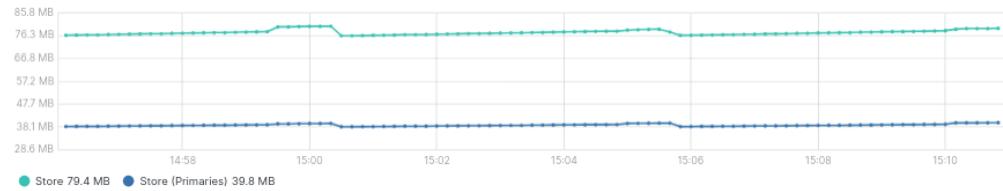


Segment Count Ⓜ

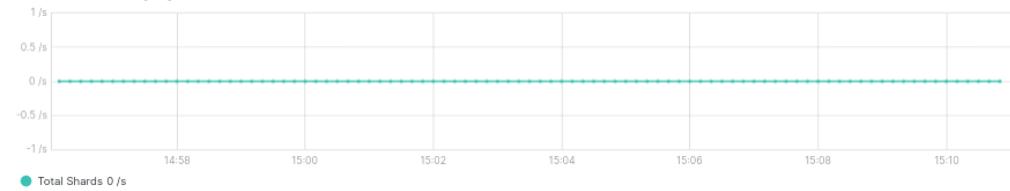


Index

Disk (MB) ⓘ



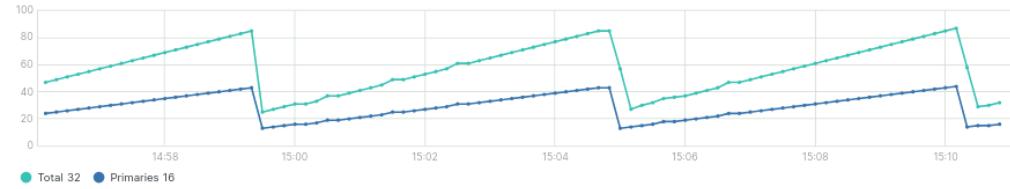
Search Rate (/s) ⓘ



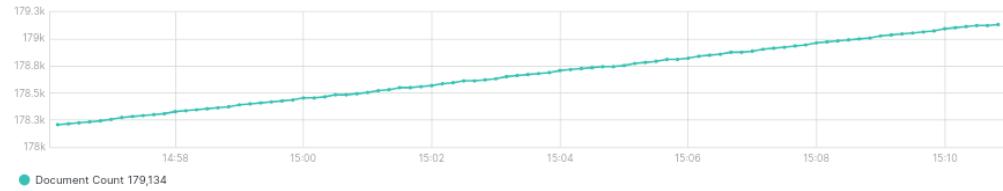
Indexing Rate (/s) ⓘ



Segment Count ⓘ



Document Count ⓘ



Dimensionnement et exploitation

Architectures ingestion
Monitoring Logstash

Architecture Indexation/Recherche
Gestion des shards
Monitoring ES
Exploitation

Changement de configuration

- La plupart des configurations ELS sont dynamiques, elles peuvent être modifiées par l'API.
- L'API *cluster-update* opère selon 2 modes :
 - ***transient*** : Les changements sont annulés au redémarrage
 - ***persistent*** : Les changements sont permanents. Au redémarrage, ils écrasent les valeurs des fichiers de configuration

Exemple

```
PUT /_cluster/settings
{
  "persistent" : {
    "discovery.zen.minimum_master_nodes" : 2
  },
  "transient" : {
    "indices.store.throttle.max_bytes_per_sec" : "50mb"
  }
}
```

Fichiers de trace

- ELS écrit de nombreuses traces dans `ES_HOME/logs`. Le niveau de trace par défaut est `INFO`
- On peut le changer par l'API

```
PUT /_cluster/settings
{
  "transient" : { "logger.discovery" : "DEBUG" }
}
```

Slowlog

- L'objectif du **slowlog** est de logger les requêtes et les demandes d'indexation qui dépassent un certain seuil de temps
- Par défaut ce fichier journal n'est pas activé. Il peut être activé en précisant l'action (query, fetch, ou index), le niveau de trace (WARN , DEBUG, ..) et le seuil de temps
- C'est une configuration au niveau index

```
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"
  } }
```

Backup

- Pour sauvegarder un cluster, l'API snapshot API peut être utilisé
- Cela prend l'état courant du cluster et ses données et le stocke dans une dépôt partagé
- Le premier snapshot est intégral, les autres sauvegardent les deltas
- Les dépôts peuvent être de différents types
 - Répertoire partagé (NAS par exemple)
 - Amazon S3
 - HDFS (Hadoop Distributed File System)
 - Azure Cloud

Usage simple

```
PUT _snapshot/my_backup
{
  "type": "fs",
  "settings": {
    "location": "/mount/backups/my_backup"
  }
}
```

Ensute

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

Restauration

POST _snapshot/my_backup/snapshot_1/_restore

- Le comportement par défaut consiste à restaurer tous les index existant dans le snapshot
- Il est également possible de spécifier les index que l'on veut restaurer

MERCI !!

Pour votre attention

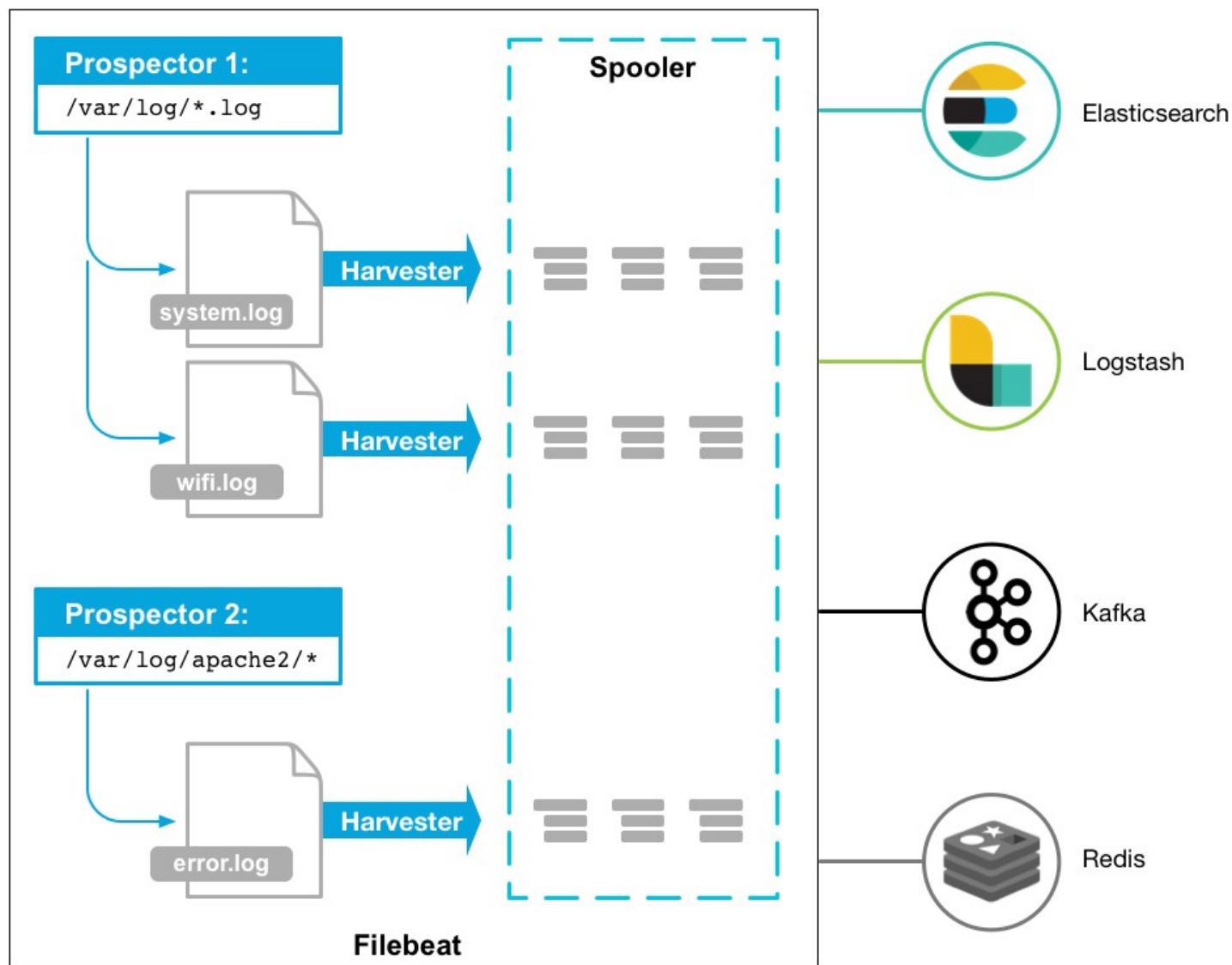
Annexes

FileBeat
Pipelines ElasticSearch
Principaux plugins logstash
Analyseurs
Machine Learning

Prospecteur et Harvester

- 2 concepts dans Filebeat
 - **Harvester** : Composant responsable de lire un fichier. Il garantit que chaque ligne d'un fichier sera envoyé une et une seule fois
 - **Prospector** : Composant responsable de créer les harvester nécessaires

Filebeat



Configuration

- La configuration de *Filebeat* consiste à :
 - Spécifier les modules à exécuter.
 - 1 module correspond à un format classique de trace
 - Spécifier les prospecteurs
 - principalement les chemins vers les fichiers de trace à consommer
 - Gérer les messages multi-lignes
 - en indiquant des expressions régulières permettant de fusionner plusieurs lignes dans un même événement
 - Les options générales de filebeat
 - emplacement du fichier de config
 - et les options communes à tous les beats
 - nom du convoyeur, tags
 - Configurer la file d'attente de traitement des événements

Configuration (2)

- La configuration de *Filebeat* consiste à :
 - Configurer la sortie
 - cluster logstash ou elasticsearch, répartition de la charge, éventuellement SSL
 - Filtrer ou enrichir les données des fichiers
 - Exclure ou ajouter des lignes (moins de possibilités qu'avec logstash)
 - Utiliser une pipeline de traitement côté ELS
 - moins de possibilité qu'avec logstash
 - Spécifier le hôte Kibana pour l'importation des tableaux de bord (6.x)
 - Charger les tableaux de bord Kibana
 - soit manuellement, soit automatiquement au 1^{er} démarrage
 - Charger les gabarits d'index ELS
 - soit manuellement, soit automatiquement au 1^{er} démarrage
 - Configurer le niveau de trace

Activation des modules

- Les modules sont activés de différentes façons :
 - Par les commandes en ligne *modules enable* ou *modules disable*.
C'est la méthode recommandée. Ex :
`./filebeat modules enable apache2 mysql`
 - La commande joue sur le répertoire *modules.d* qui contient toutes les configurations par défaut
 - Lors de l'exécution de *filebeat*
`./filebeat -e --modules nginx,mysql,system`
 - Via le fichier de configuration *filebeat.yml*
filebeat.modules:
 - *module: nginx*
 - *module: mysql*
 - *module: system*

Modules disponibles

- Apache2, Nginx
- System (auth, syslog), Auditd (démon auditd)
- Kafka, Redis,
- Logstash
- Mysql, Postgres
- Traefik

Exemple System Module

- Le module permet :
 - Positionner les chemins par défaut des fichiers de logs
 - Garantir que les traces multi-lignes seront traitées comme un événement unique
 - D'utiliser les nœuds d'ingestion d'ELS
 - Déployer les tableaux de bord

Exemple module System

Mise en place

- Activer le module dans la configuration

```
./filebeat modules enable system
```

- Initialisation, importation des gabarits et des tableaux de bord

```
./filebeat setup -e
```

- Démarrage

```
./filebeat -e
```

Annexes

FileBeat
Pipelines ElasticSearch
Principaux plugins logstash
Analyseurs
Machine Learning

Introduction

- Sans logstash, il est possible d'effectuer des transformations courantes sur les données avant l'indexation en utilisant les pipelines d'ingestion d'ElasticSearch
- Les fonctionnalités sont similaires à logstash mais moins puissantes
- Les pipelines ELS peuvent être utilisés pour supprimer des champs, extraire des valeurs du texte et enrichir vos données.
- Il se compose d'une série de tâches configurables appelées processeurs. Chaque processeur s'exécute de manière séquentielle, apportant des modifications spécifiques aux documents entrants

Pipeline

- Une **pipeline** est définie par :
 - Un nom
 - Une description
 - Et une liste de *processors* (*pré-définis*)
- Les processors sont executés en séquence

Ingest API

- L'API ***ingest*** supporte :
 - ***PUT*** : Ajout ou remplacement d'une pipeline
 - ***GET*** : Récupération d'une pipeline
 - ***DELETE*** : Suppression d'une pipeline
 - ***SIMULATE*** : Simulation de l'exécution d'une pipeline

Usage

#Creation d'une pipeline nommée attachment

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

Utilisation d'une pipeline durant l'indexation.

```
PUT my_index/my_type/my_id?pipeline=attachment
{
  "data":
"e1xydGYxXGFuc2kNCkxvcmVtIGlw3VtIGRvbG9yIHNpdCBhbW0DQpccGFy
IH0="
}
```

Résultat

```
GET my_index/my_type/my_id
{
  "found": true,
  "_index": "my_index",
  "_type": "my_type",
  "_id": "my_id",
  "_version": 1,
  "_source": {
    "data": "e1xydGYxXGFuc2kNCkxvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbW0DQpccGFyIH0=",
    "attachment": {
      "content_type": "application/rtf",
      "language": "ro",
      "content": "Lorem ipsum dolor sit amet",
      "content_length": 28
    }
  }
}
```

Introduction aux plugins

- Les fonctionnalités d'ELS peuvent être étendues via la notion de plugin
- Les Plugins contiennent des fichiers JAR, mais aussi des scripts et des fichiers de configuration
- Ils peuvent être installés facilement avec
 - `sudo bin/elasticsearch-plugin install [plugin_name]`
 - Ils doivent être installés sur chaque nœud du cluster
 - Après une installation, le nœud doit être redémarré

Ingest plugins

- Un plugin d'ingestion particulier permet d'indexer des documents bureautiques (Word, PDF, XLS, ...):
 - **Attachment Plugin** : Extrait les données textuelles des pièces jointes dans différents formats (PPT, XLS, PDF, etc.). Utilise la librairie Apache Tika.
 -
 - `sudo bin/elasticsearch-plugin install ingest-attachment`

Annexes

FileBeat
Pipelines ElasticSearch
Principaux plugins logstash
Analyseurs
Machine Learning

Inputs

- Quelques inputs :
 - **beats** : Événements d'un beat
 - **elasticsearch** : Lit des résultats de requêtes à partir d'un cluster Elasticsearch
 - **exec** : Sortie d'une commande shell
 - **file** : Lecture de lignes de fichier
 - **generator** : Génère des évènements au hasard pour le test
 - **heartbeat** : Génère des évènements *heartbeat* pour le test
 - **jdbc** : Événements à partir de JDBC
 - Kafka, redis, rabbitmq, jms : Messaging
 - Tcp, udp, log4j, unix, syslog : Evènements sockets bas niveau
 - ...

Exemples

```
input {  
  beats {  
    port => 5044  
  }  
  file {  
    path => "/var/log/*"  
    exclude => "*.gz"  
  }  
  generator {  
    lines => [  
      "line 1",  
      "line 2",  
      "line 3"  
    ]  
    # Emet toutes les lignes 3 fois.  
    count => 3  
  }  
}
```

Input file

- Input **file** est similaire à un *tail -OF* mais peut également lire un fichier depuis son début
- Par défaut, chaque événement correspond à une ligne
- Le plugin garde une trace de la position de lecture courante pour chaque fichier dans un fichier séparé nommé *sincedb*
- Les principales configuration :
 - *path*
 - *start_position*
 - *exclude*
 - *ignore_older*
 - *delimiter*

Quelques filtres

- ***cidr*** : Vérifie des adresses IP
- ***csv*** : Parse le format csv
- ***date*** : Parse des champs afin de déterminer le timestamp de l'événement
- ***dns*** : Effectue un lookup DNS
- ***drop*** : Supprime un événement
- ***elapsed*** : Calcul le temps entre 2 événements
- ***environment*** : Stocke des variables d'environnement comme champ metadata
- ***extractnumbers*** : Extrait des nombres à partir d'une string

Quelques filtres (2)

- **geoip** : Ajoute des informations latitude/longitude à partir d'une IP
- **grok** : Divise un champ en d'autres champs
- **json** : Parses les événements JSON
- **json_encode** : Sérialise un champ au format JSON
- **metrics** : Calcul des agrégations sur un évènements (Comptage, cadence)
- **mutate** : Transforme un champ
- **prune** : Filtre des événements à partir d'une liste rouge ou blanche
- **range** : Vérifie que la taille ou longueur d'un champs est dans un intervalle
- **translate** : Remplace les valeurs des champs à partir d'une table de hash ou fichier YAML
- **truncate** : Tronque les champ supérieur à une certaine longueur
- **urldecode** : Décode les champs URL-encoded
- **useragent** : Parse les chaînes user agent
- **xml** : Parse le format XML

Le filtre *grok*

- **grok** est sûrement le filtre le plus utilisé. Il permet de générer différents champs ELS à partir d'une ligne de log.
- Il s'appuie sur des patterns qui sont déposés par la communauté sur GitHub (+120)
<https://github.com/elastic/logstash/blob/v1.4.2/patterns/grok-patterns>
- Les patterns s'appuient sur les **regexp** et des patterns spécifiques peuvent être déclarés
- Un pattern Grok s'écrit : **%{SYNTAX:SEMANTIC}**
 - SYNTAX est un pattern associé à une expression régulière
 - SEMANTIC est le nom de champ que l'on veut créer
- Il existe un site en ligne permettant de debugger une expression grok
<https://grokdebug.herokuapp.com/>

Le filtre *mutate*

- Permet de multiples transformations, les options sont :
 - **convert** : Conversion de type
 - **copy** : Copie de champ dans un autre champ
 - **gsub** : Remplacement de chaînes de caractères à partir d'expression régulière
 - **lowercase / uppercase** : Passage en minuscule/majuscule
 - **join, merge** : Travaille sur des tableaux
 - **rename** : Renommage de champ
 - **replace, update** : Remplacement de valeur
 - **split** : Transforme un champ en tableau en utilisant un séparateur
 - **strip** : Supprime les espaces avant et après

Le filtre *date*

- Le filtre est utilisé pour parser des dates à partir de champ et utilisé cette date comme timestamp logstash. Les options sont :
 - **match** : Un tableau dont la première valeur est le champ parser et les autres valeurs les formats possibles
 - **locale**, **timezone** : Si pas présent dans le format du match, on peut préciser
 - **target** : Le champ stockant le résultat, par défaut `@timestamp`

Quelques Codecs

- Les codecs changent la représentation d'un événement. Ils sont utilisés à l'intérieur d'une entrée ou d'une sortie
 - ***gzip_lines*** : Lit du contenu encodé avec gzip
 - ***json*** : Lit du contenu JSON formatté.
 - ***json_lines*** : Lit du contenu JSON formatté sur une seule ligne (Bulk API de ELS)
 - ***multiline*** : Fusionne plusieurs lignes en 1 événement
 - ***rubydebug*** : Debug d'évènements

Événements multi-lignes

- Lors d'événements multi-lignes, *Logstash* doit savoir quels lignes doivent faire partie de l'événement unique
- Le traitement des multi-lignes est effectué en général en premier dans le pipeline et utilise le filtre ou le codec ***multiline***
- 3 options importantes de configuration :
 - ***pattern*** : Expression régulière. Les lignes correspondantes sont considérées soit comme la suite des lignes précédentes ou le début d'un nouvel événement Il est possible d'utiliser les gabarits de *grok*
 - ***what*** : Qui peut prendre 2 valeurs : *previous* ou *next*
 - ***negate*** : Qui permet d'exprimer la négation

Exemple stack trace Java

```
Exception in thread "main" java.lang.NullPointerException
        at com.example.myproject.Book.getTitle(Book.java:16)
        at com.example.myproject.Author.getBookTitles(Author.java:25)
        at com.example.myproject.Bootstrap.main(Bootstrap.java:14)
```

- Le filtre suivant indique qui si la ligne démarre avec des espaces, elle est considérée comme la suite de l'événement

```
input {
    stdin {
        codec => multiline {
            pattern => "^\\s"
            what => "previous"
        }
    }
}
```

Les plugins outputs

- Quelques outputs
 - **csv** : Ecrit sur le disque au format CSV
 - **elasticsearch** : Stocke dans Elasticsearch
 - **email** : Envoie un email à une adresse spécifiée
 - **exec** : Exécute une commande si un évènement correspond
 - **file** : Ecrit dans un fichier
 - **stdout** : Ecrit sur la sortie standard
 - **pipe** : Envie vers l'entrée standard d'un autre programmeHttp : Envoie sur un endpoint HTTP ou HTTPS
 - **nagios, nagios_nsca** : Envoi vers Nagios
 - **tcp, udp, syslog, statsd, websocket** : Sockets
 - **kafka, redis, rabbitMQ** : Messaging
 - ...

Output *elasticsearch*

- L'output ***elasticsearch*** contient de nombreuses options. Les plus importantes sont :
 - ***hosts*** : Les hôtes du cluster
 - ***index*** : Le nom de l'index. Exemple : logstash-%{+YYYY.MM.dd}. Attention, le nom influe sur le gabarit d'index utilisé
 - ***template*** : Un chemin vers le fichier template
 - ***template_name*** : Le nom du tempate côté ElasticSearch
 - ***template_overwrite*** (false) : Permet de mettre à jour le template utilisé
 - ***document_id*** : L'id du document (permet les mises à jour d'événements)
 - ***parent*** : Le document parent de l'événement (nested document)
 - ***pipeline*** : La pipeline côté ElasticSearch à utiliser
 - ***routing*** : Peut spécifier le shard à utiliser

Exemple complet log Apache

```
input {
  file {
    path => "/tmp/*_log"
  }
}

# Traitements différents en fonction du type de log
filter {
  if [path] =~ "access" {
    mutate { replace => { type => "apache_access" } }
    grok {
      #gabarit connu par grok
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
    date {
      # Normalisation de la date
      match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
    }
  } else if [path] =~ "error" {
    mutate { replace => { type => "apache_error" } }
  } else {
    mutate { replace => { type => "random_logs" } }
  }
}

output {
  elasticsearch { hosts => ["localhost:9200"] }
  stdout { codec => rubydebug }
}
```

Résultat pour une ligne de log d'accès

```
{  
    "message" => "127.0.0.1 - - [11/Dec/2013:00:01:45 -0800] \"GET /xampp/status.php  
HTTP/1.1\" 200 3891 \"http://cadenza/xampp/navi.php\" \"Mozilla/5.0 (Macintosh; Intel Mac  
OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0\"",  
    "@timestamp" => "2013-12-11T08:01:45.000Z",  
    "@version" => "1",  
    "host" => "cadenza",  
    "clientip" => "127.0.0.1",  
    "ident" => "-",  
    "auth" => "-",  
    "timestamp" => "11/Dec/2013:00:01:45 -0800",  
    "verb" => "GET",  
    "request" => "/xampp/status.php",  
    "httpversion" => "1.1",  
    "response" => "200",  
    "bytes" => "3891",  
    "referrer" => "\"http://cadenza/xampp/navi.php\"",  
    "agent" => "\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0)  
Gecko/20100101 Firefox/25.0\""  
}
```

Exemple complet *syslog*

```
input {
    tcp { port => 5000 type => syslog }
    udp { port => 5000 type => syslog } }

filter {
    if [type] == "syslog" {
        grok {
            match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %"
{SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\T%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" }
            add_field => [ "received_at", "%{@timestamp}" ]
            add_field => [ "received_from", "%{host}" ]
        }
        date { match => [ "syslog_timestamp", "MMM d HH:mm:ss", "MMM dd HH:mm:ss" ] }
    }
}

output {
    elasticsearch { hosts => ["localhost:9200"] }
    stdout { codec => rubydebug }
}
```

Annexes

FileBeat
Pipelines ElasticSearch
Principaux plugins logstash
Analyseurs
Machine Learning

Tokenisation et Normalisation

- Pour construire l'index inversé, ELS doit séparer un texte en mots (**tokenisation**) et ensuite **normaliser** les mots afin que la recherche du terme « envoi » par exemple retourne des documents contenant : « envoi », « Envoi », « envois », « envoyer », ...
- La tokenisation et la normalisation sont appelées **analyse**.
- L'analyse s'applique sur les documents lors de l'indexation **ET** lors de la recherche sur les termes recherchés.

Étapes de l'analyse

- Les **analyseurs** transforment un texte en un flux de “token”. Ils peuvent être constitués d'une seule classe Java ou d'une combinaison de filtres, de tokenizer et de filtres de caractères
 - Les **filtres de caractères** préparent le texte en effectuant du remplacement de caractères (& devient et) ou en supprimant (suppression des balises HTML)
 - Les **tokenizers** splittent un texte en une suite d'unité lexicale : les tokens
 - Les **filtres** prend en entrée un flux de token et le transforme en un autre flux de token

Analyseurs prédéfinis

- ELS propose des analyseurs directement utilisables :
 - **Analyseur Standard** : C'est l'analyseur par défaut. Le meilleur choix lorsque le texte est dans des langues diverses. Il consiste à :
 - Séparer le texte en mots
 - Supprime la ponctuation
 - Passe tous les mots en minuscule
 - **Analyseur simple** : Sépare le texte en token de 2 lettres minimum puis passe en minuscule
 - **Analyseur d'espace** : Sépare le texte en fonction des espaces
 - **Analyseurs de langues** : Ce sont des analyseurs spécifiques à la langue. Ils incluent les « stop words » (enlève les mots les plus courant) et extrait la racine d'un mot. C'est le meilleur choix si l'index est en une seule langue

Test des analyseurs

```
GET /_analyze?analyzer=standard
```

```
Text to analyze
```

Réponse :

```
{  
  "tokens": [ {  
    "token": "text",  
    "start_offset": 0,  
    "end_offset": 4,  
    "type": "<ALPHANUM>",  
    "position": 1  
  }, {  
    "token": "to",  
    "start_offset": 5,  
    "end_offset": 7,  
    "type": "<ALPHANUM>",  
    "position": 2  
  }, {  
    "token": "analyze",  
    "start_offset": 8,  
    "end_offset": 15,  
    "type": "<ALPHANUM>",  
    "position": 3  
  } ] }
```

Spécification des analyseurs

- Lorsque ELS détecte un nouveau champ *String* dans un type de document, il le configure automatiquement comme champ full-text et applique l'analyseur standard.
- Si ce n'est pas le comportement voulu, il faut explicitement spécifier le **mapping** pour le type de document

Analyseurs

- L'analyseur par défaut est l'analyseur standard qui consiste en :
 - Le tokenizer standard qui sépare sur les frontières de mots
 - Le filtre de token standard (qui ne fait rien)
 - Le filtre lowercase qui passe tout en minuscule
 - Le filtre stopwords (qui contient un minimum de stop words)
- Il est possible de surcharger cette configuration par défaut en ajoutant un filtre ou en remplaçant un existant

Création par surcharge

- Dans l'exemple suivant, un nouvel analyseur *fr_std* pour l'index *french_docs* est créé. Il utilise la liste prédéfinie des *stopwords* français :

```
PUT /french_docs
{
  "settings": {
    "analysis": {
      "analyzer": {
        "fr_std": {
          "type": "standard",
          "stopwords": "_french_"
        }
      }
    }
  }
}
```

Création complète

- Il est possible de créer ses propres analyseurs en combinant des filtres de caractères, un tokenizer et des filtres de tokens :
 - 0 ou n : Filtres de caractères
 - 1 : tokenizer
 - 0 ou n : Filtre de tokens

```
PUT /my_index
{
  "settings": {
    "analysis": {
      "char_filter": { ... custom character filters ... },
      "tokenizer": { ... custom tokenizers ... },
      "filter": { ... custom token filters ... },
      "analyzer": { ... custom analyzers ... }
    } } }
```

Exemple

```
PUT /my_index
{
  "settings": {
    "analysis": {
      "char_filter": {
        "&_to_and": {
          "type": "mapping",
          "mappings": [ "&=> and "]
        }, "filter": {
          "my_stopwords": {
            "type": "stop",
            "stopwords": [ "the", "a" ]
          },
          "analyzer": {
            "my_analyzer": {
              "type": "custom",
              "char_filter": [ "html_strip", "&_to_and" ],
              "tokenizer": "standard",
              "filter": [ "lowercase", "my_stopwords" ]
            }
          }
        }
      }
    }
  }
}
```

Affectation à un index

- Il faut ensuite associer l'analyseur à un champ *string*

```
PUT /my_index/_mapping
{
  "properties": {
    "title": {
      "type": "text",
      "analyzer": "my_analyzer"
    }
  }
}
```

Analyseurs prédéfinis

- Elasticsearch fournit des analyseurs pré-définis pouvant être utilisés directement :
 - **Standard** (par défaut) : Sépare en mot, enlève la ponctuation, passe en minuscule, supporte les stop words
 - **Simple** : Sépare en token dès qu'il trouve un caractère qui n'est pas une lettre. Passe en minuscule
 - **Whitespace** : Se base sur les espaces pour la tokenization. Ne passe pas en minuscule
 - **Stop** : Comme l'analyseur simple avec du support pour les stop words.
 - **Keyword** : Ne fait rien. Prend le texte tel quel
 - **Pattern** : Utilise une expression régulière pour la tokenization. Passe en minuscule et supporte les stop words.
 - **Language** : english, french, ...
 - **Fingerprint** : Crée une empreinte du document pouvant être utilisé pour tester la duplication.

Filtres fournis

- ELS fournit de nombreux filtres permettant de mettre au point des analyseurs personnalisés. Citons :
 - **Length Token** : Suppression de mots trop courts ou trop longs
 - **N-Gram** et **Edge N-Gram** : Analyzeur permettant d'accélérer les suggestion de recherche
 - **Stemming filters** : Algorithme permettant d'extraire la racine d'un mot
 - **Phonetic filters** : Représentation phonétique des mots
 - **Synonym** : Correspondance de mots
 - **Keep Word** : Le contraire de stop words
 - **Limit Token Count** : Limite le nombre de tokens associés à un document
 - **Elison Token** : Gestion des apostrophes (français)

Utilisation de synonymes

- Les synonymes peuvent être utilisés pour fusionner des mots qui ont quasiment le même sens.
 - Ex : joli, mignon, beau
- Ils peuvent également être utilisés afin de rendre un mot plus générique.
 - Par exemple, oiseau peut être utilisé comme synonyme à pigeon, moineau, ...

Ajout d'un filtre synonyme

```
• PUT /my_index
• {
•   "settings": {
•     "analysis": {
•       "filter": {
•         "my_synonym_filter": {
•           "type": "synonym",
•           "synonyms": ["british,english", "queen,monarch"]
•         }
•       },
•       "analyzer": {
•         "my_synonyms": { "tokenizer": "standard",
•                         "filter": ["lowercase", "my_synonym_filter"]
•       }
•     }
•   }
• }
```

3 utilisations

- Le remplacement de synonyme peut se faire de 3 façons :
 - Expansion simple : Si un des termes est rencontré, il est remplacé par tous les synonymes listés
"jump, leap, hop"
 - Contraction simple : un des termes rencontré est remplacé par un synonyme
"leap, hop => jump"
 - Expansion générique : un terme est remplacé par plusieurs synonymes
"puppy => puppy, dog, pet"

Annexes

FileBeat
Pipelines ElasticSearch
Principaux plugins logstash
Analyseurs
Machine Learning

Introduction Machine Learning

- Rachat de la société Prevert
- Introduit dans la Version 5.4, fonctionnalités X-Pack
- Permet de se poser les questions :
 - Certains de mes services ont-ils changé de comportement ? »
 - « Y a-t-il des processus inhabituels qui s'exécutent sur mes machines ?
- Basé sur des modèles comportementaux, permet la détection d'anomalies dans des données temporelles

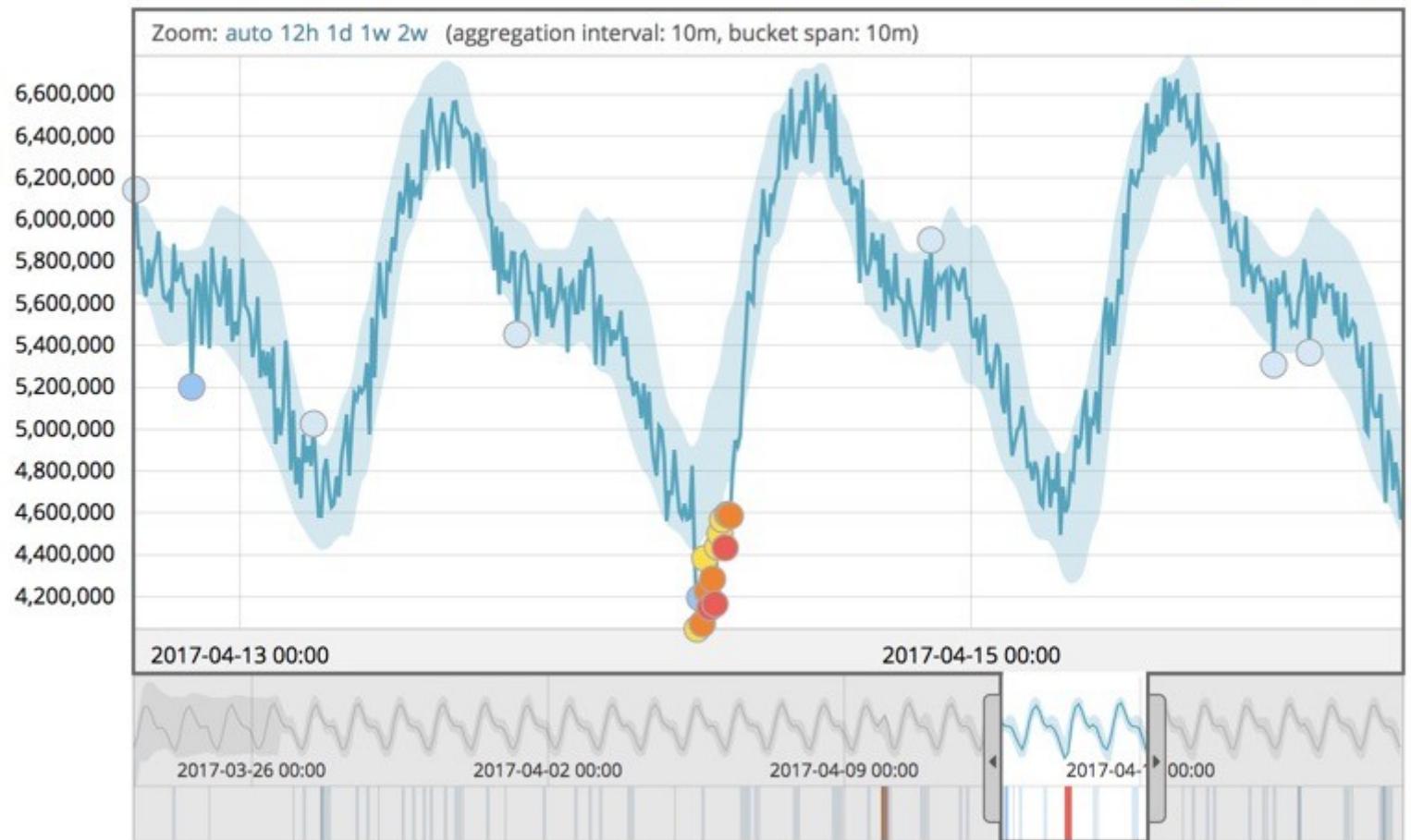
Principe

- Les données relatives au temps sont extraites d'ElasticSearch pour analyse :
 - soit de façon continue
 - soit périodiquement
- Les résultats anormaux sont affichés dans Kibana.
- Différentes situations sont alors remontées :
 - Anomalies liées à des écarts temporels dans les valeurs, les décomptes ou les fréquences
 - Des raretés statistiques
 - Des comportements inhabituels pour un membre d'une population

Valeurs actuelles, limites normales et anomalies

Single time series analysis of sum total

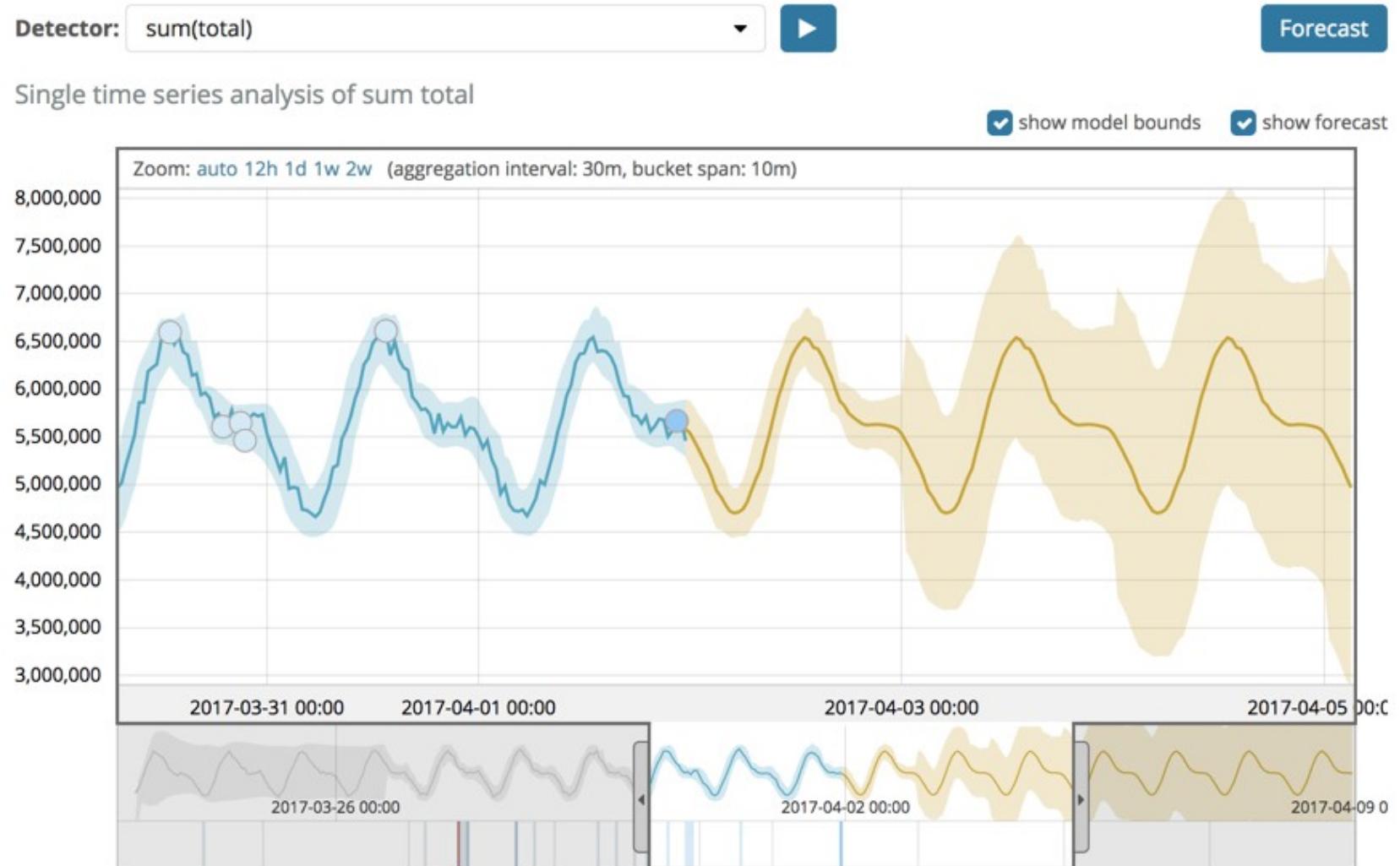
show model bounds



Anticipation

- Les profils extraits du passé peuvent être utiliser pour anticiper le futur.
- Par exemple, prévoir :
 - Le nombre de visites d'un site web dans 1 mois
 - Quand mon disque atteindra 100 % d'utilisation
- Ces prévisions peuvent être visualisées dans Kibana

Intervalle de prévisions en jaune



Tâches d'analyse

- Les **Machine Learning jobs** contiennent les informations de configuration nécessaire à une tâche d'analyse.
 - Chaque job a un ou plusieurs détecteurs correspondant à une fonction analytique sur certains champs de donnée
 - Il contient également des propriétés indiquant quels événements doivent être analysés par rapport à des comportements précédents ou une population
- Kibana propose des assistants permettant de créer ce type de job
- Les jobs peuvent également être groupés afin de visualiser ensemble leurs résultats
- Les jobs sont exécutés sur des nœuds du cluster qui ont les propriétés de configuration **xpack.ml.enabled** et **node.ml** positionnées à *true*

Fonctions analytiques

- Fonction de :
 - **Comptage** : Déetecte des anomalies lorsque le nombre d'événements dans un groupement est anormal
 - **Géographiques** : Déetecte les anomalies dans l'emplacement géographique d'une donnée
 - Sur le **contenu** : Déetecte les anomalies dues au volume d'une donnée String
 - **Métriques** : Anomalies sur des moyennes, des min, des max.
 - Détection de **rareté** : Anomalies qui arrivent rarement sur le temps ou rarement dans une population
 - **Somme** : Anomalies lorsque la somme d'un champ dans un groupement est anormal
 - **Temporelle** : Déetecte des événements qui arrivent à des moments inhabituels. Exemple une week-end

Flux de données

- Les flux de données d'entrée d'un job d'analyse sont créés
 - Soit à partir d'un index pattern ElasticSearch (Typiquement, lorsque l'on utilise Kibana)
 - Soit programmatiquement via une API
- Un job n'est associé qu'à un seul flux de données d'entrée
- Dans le cas d'ELS, les flux de données doivent être démarrés (et arrêtés) ; cela est fait via Kibana

Buckets

- Les **buckets** sont utilisés pour diviser les séries temporelles en traitement par lots
- Ils font partie de la configuration d'un job et déterminent l'intervalle de temps utilisé pour agréger les données.
En particulier, calculer un score d'anomalie

Événements hors norme

- Il est possible de planifier des périodes où l'activité d'événements sera inhabituelle.
(Par exemple : black fridays ou autre)
- Dans ce cas, les tâches d'analyse ne détectent pas d'anomalies