



# Elastic Stack

David THIBAU – 2018

david.thibau@gmail.com

# Agenda

- **Introduction**

- L'offre ELK et ses cas d'usage
- Les composants de la pile
- Distributions et installation
- Concepts de base
- Conventions API ELS, Dev console de Kibana

- **Indexation et documents**

- Qu'est ce qu'un Document ?
- Document API
- Routing
- API Search Lite
- Distribution de la recherche

- **Ingestion de données : Beats et Logstash**

- Les Beats
- Concepts logstash
- Syntaxe Configuration pipeline
- Principaux input, filter, codecs, outputs

- **Configuration d'index**

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

# Agenda (2)

- **Recherche avec DSL**

- Syntaxe DSL et combinaison de clauses
- Recherche filtre
- Recherche full-text
- Agrégations
- Géolocalisation

- **Analyse temps-réel avec Kibana**

- Présentation
- Discover
- Visualisations et tableaux de bord
- Consoles, Management et plugins
- Timelion

- **Vers la production**

- Architectures Logstash
- Monitoring API Logstash
- Replica et Shards
- Monitoring API Elasticsearch
- Points de surveillance
- Exploitation

- **Annexes X-Pack**

- Installation - Fonctionnalités
- Machine Learning

# L'offre Elastic Stack

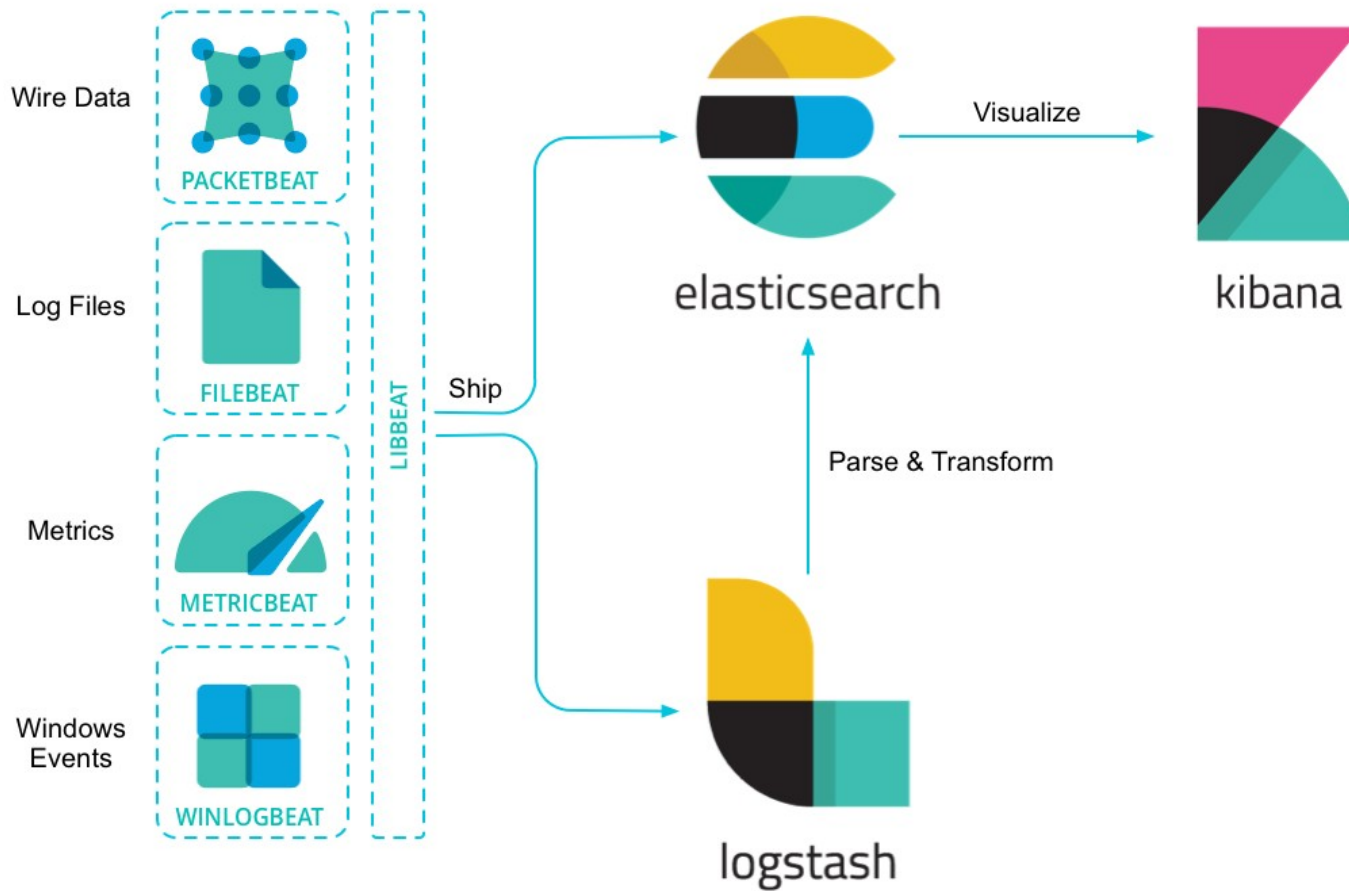
# Introduction

- **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)

# 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 d'Elasticsearch

# Architecture





# Autres outils

- D'autres outils connexes peuvent être ajoutés :
  - ***X-Pack*** : Fonctionnalités d'entreprise à la pile (Sécurité, Monitoring, Alerte, Machine Learning)
  - ***ES-Hadoop*** : Stockage BigData
  - ***Elastic Cloud / Enterprise*** : ELK As A Service
  - ***APM*** (Application Performance Management) : Tracing de requête ou de transactions

# Versions

- Depuis la 5.0, les différents produits de ELK ont la même numérotation en terme de versions
  - Tous les produits font partie du même processus de release, ainsi lorsqu'un produit est released tous les produits de la pile sont releaseds.
- => Il faut donc juste s'assurer que tous les produits installés ont la même version.

# Ordre d'installation

Il est recommandé d'installer la pile en respectant l'ordre suivant :

1. Elastic Search  
Éventuellement X-Pack pour ELS
2. Kibana  
Éventuellement X-Pack pour Kibana
3. Logstash
4. Beats
5. Éventuellement Elasticsearch Hadoop

# Raisons du succès

- ELK est une plate-forme simple robuste et opensource. Ses principales atouts sont
  - Accès aux données en temps-réel
  - Scalable en ressources (CPU, RAM) et en volume (Disque)
  - API REST
  - Intégration aisée avec langages et framework. Nombreux clients, (Java, Javascript, PHP, Python, ...)
  - Documentation disponible et complète
  - Importante communauté
  - Netflix, Facebook, Microsoft, LinkedIn, Salesforce et Cisco l'ont choisi !

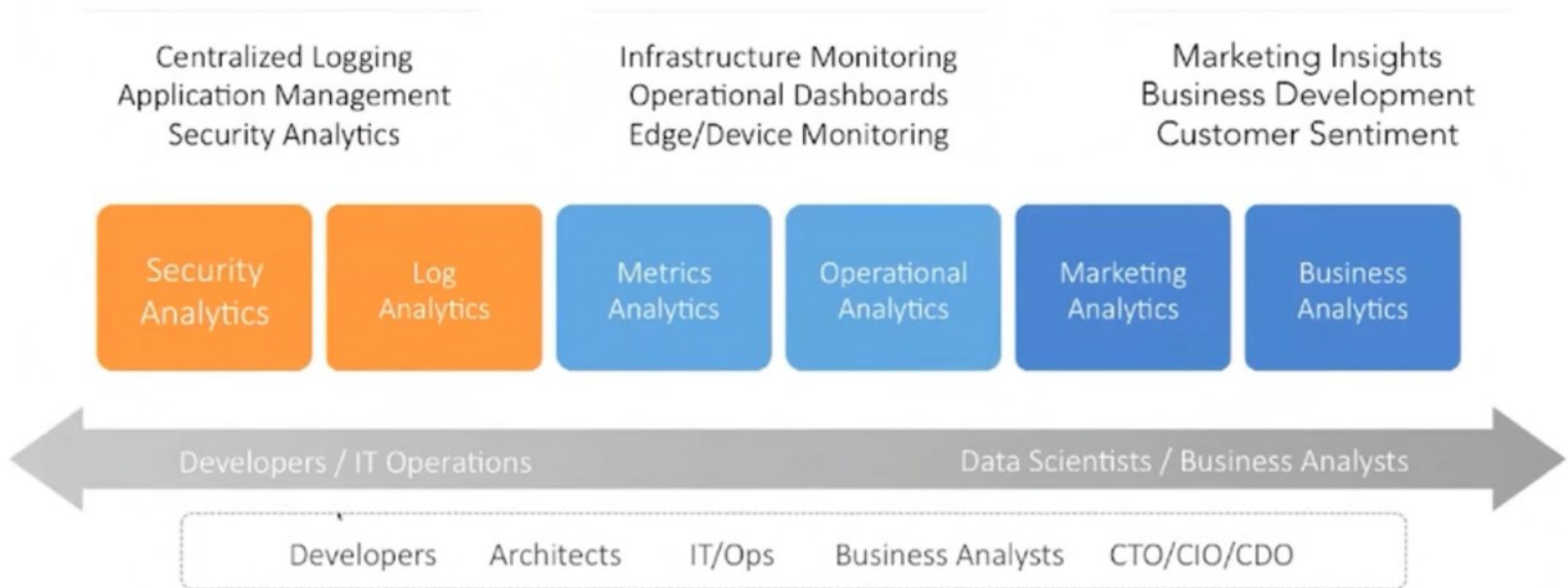
# Analyse des traces

- Les SI des entreprises sont désormais fortement distribuées et les problèmes pouvant y survenir sont multiples  
=> L'ensemble de l'infrastructure doit continuellement être surveillé afin que les problèmes soient détectés au plus tôt. Cela implique les métriques temps réel et l'analyse des fichiers journaux
- Le volume des traces étant énorme (BigData). Des outils de recherche, d'agrégation et d'analyse sont nécessaires.
- Ces outils sont aussi bien à destination de l'exploitation technique que métier

# Autres Cas d'usage

- La pile a également d'autres cas d'usage
  - BI : Elle permet d'avoir une vue métier à 360°
  - Solution de recherche pour les applications webs ou mobiles
  - Comportement utilisateur, fréquentation, segmentation marketing
  - Gestion de risque métier et détection de fraude
  - L'Internet des objets connectés, (capteurs de santé, ...)
  - Big Data

# Elastic Use cases

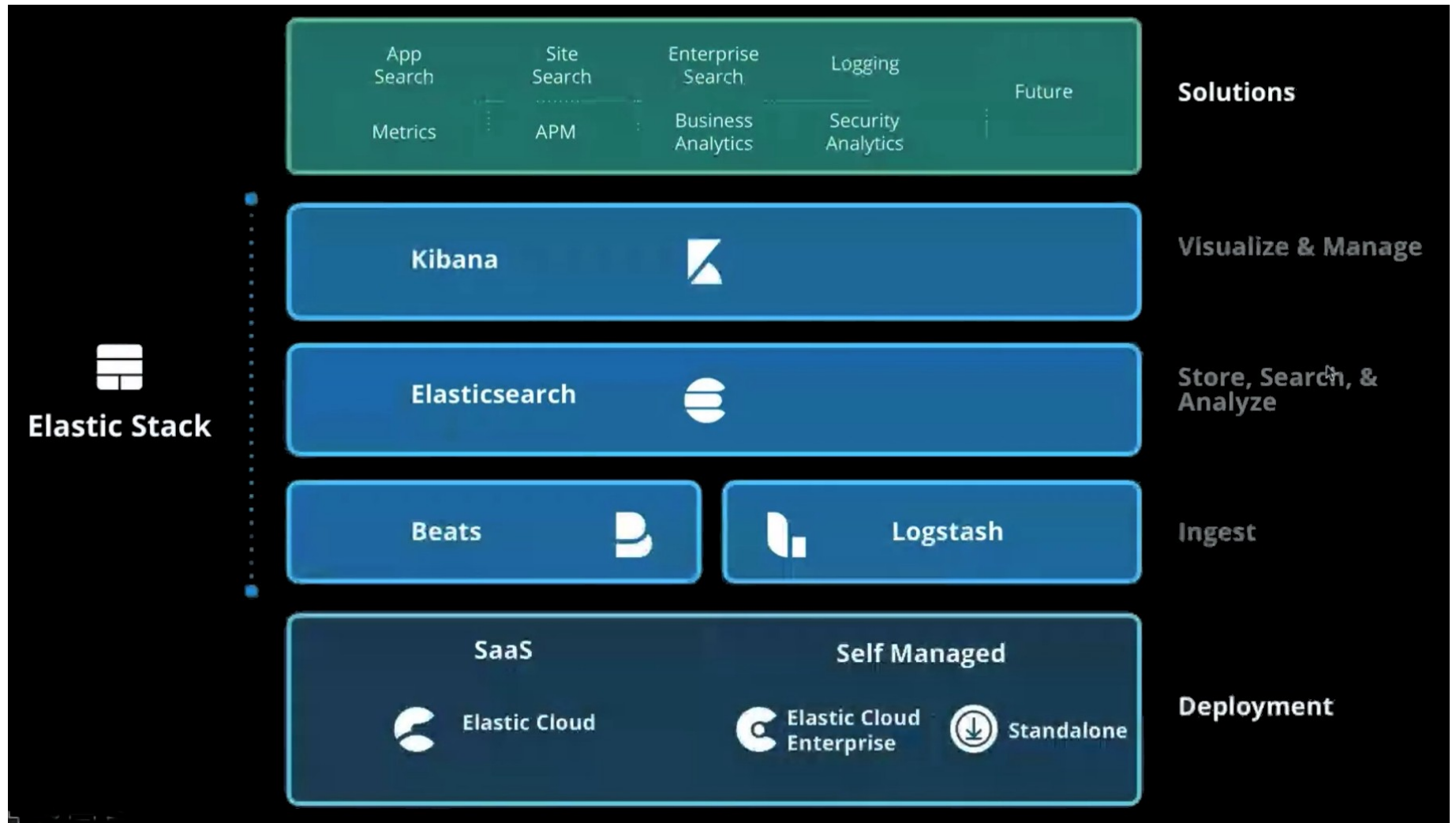


# Déploiement

- ELK peut être installé dans une entreprise
  - Cependant, l'exploitation de plate-forme peut être consommatrice de ressources surtout si celle-ci a tendance à scaler continuellement en prenant en compte de nouveaux cas d'usage
- L'autre option est de choisir un fournisseur *ELK as a Service*. (*Elastic* ou autres) ou d'installer Elastic Cloud dans son SI



# En résumé



# Distributions et installation

# Distributions

- La distribution permettant d'installer *ElasticSearch* est disponible sous plusieurs formes :
  - ZIP, TAR, DEB ou RPM
  - Également image Docker
- Versions
  - 6.4.0 : Août 2018
  - 6.3.0 : Juin 2018 : Ouverture des fonctionnalités X-Pack
  - 6.2.0 : Février 2018
  - 6.0.0 : Novembre 2017
  - 5.0.0 : Octobre 2016
  - 2.4.1 : Septembre 2016
  - 2.4.0 : Août 2016

# Installation

1. Pré-requis Java > 6

2. Dézipper l'archive

3. Exécuter ELS

`bin/elasticsearch`

4. Accéder à ELS

`curl http://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*** : Verbose 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*

- Le format YAML permet de fixer les valeurs des propriétés de configuration.
- Le format prend en compte les **:** et les **tabulations**

path:

```
  data: /var/lib/elasticsearch
  logs: /var/log/elasticsearch
```

- Est équivalent à

```
path.data: /var/lib/elasticsearch
```

```
path.logs: /var/log/elasticsearch
```

- Les variables d'environnement sont accessibles `${ENV_VAR}`
- Certaines propriétés peuvent être demandées au démarrage :  
`${prompt.text}` , `${prompt.secret}`
- Les valeurs par défaut peuvent être positionnées par la commande en ligne. Elles écrasent la valeur définie dans *elasticsearch.yml*  
`./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 le 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*

# Concepts de base



# 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 constitué que d'un seul nœud
- => Un nœud ne peut pas appartenir à 2 clusters distincts

# Nœud

- Un **nœud** est un simple serveur faisant partie d'un cluster.
- Un nœud stocke des données, et participe aux fonctionnalités d'indexation et recherche du cluster.
  - Un nœud est également identifié par un nom unique (généré automatiquement si pas renseigné)
- Le nombre de nœuds dans un cluster n'est pas limité

# 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 du cluster 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

# Type

- A l'intérieur d'un index, on peut définir un ou plusieurs **types** de documents
- Un type est une partition logique de l'index
- Il définit des documents qui ont les mêmes champs

**Déprécié dans la version 6.x, encore présent dans l'API mais ne pas s'appuyer dessus !**

**=> Les index sont mono-type !!**

# Document

- Un **document** est l'unité basique d'information qui peut être indexée.
  - Le document est un ensemble de champs (clé/valeur) exprimé avec le format JSON
- A l'intérieur d'un index/type, on peut stocker autant de documents que l'on veut
- Un document d'un index doit être assigné à un type

# 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*
  - A la création de l'index, il est possible de définir le nombre de shards
- Chaque *shard* est un index indépendant qui peut être hébergé sur un des nœuds 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, il est recommandé d'utiliser des mécanismes de failover dans le cas où un nœud défaille
- ELS permet de mettre en place des copies des shards : les **répliques**
- 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)
  - Il permet de **scaler** le volume des requêtes car les recherches peuvent être exécutées sur toutes les répliques en parallèle .
-

# Résumé

- En résumé chaque index peut être divisé sur plusieurs shards.
- Il peut être répliqué plusieurs fois
- Une fois répliqué, chaque index a des shards primaires et des shards de réplication
- Le nombre de shards et de répliques peuvent être spécifiés au moment de la création de l'index
- Après sa création, le nombre de répliques peut être changé dynamiquement mais pas le nombre de shards
- Par défaut, ELS alloue 5 shards primaires et une réplique

# Conventions de l'API REST

# 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 de **recherche** (*\_search*)
  - API **cluster** : gestion de l'architecture (*\_cluster*)
  - ...

# Introduction

- 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 formaté
- **?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
- 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.url*** : <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

Qu'est-ce qu'un document

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
  - ***\_type*** : La classe de l'objet que le document représente. **(Déprécié avec 6.x)**
  - ***\_id*** : L'identifiant unique
  - ...

# Document API

# 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 méta-données *\_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}/{type}/{id}

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

# Exemple sans Id

**POST** /{index}/{type}

```
{
  "field": "value",
  ...
}
...
{
  "_index": {index},
  "_type": {type},
  "_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}}/{{type}}/{{id}}?pretty`

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

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

# Partie d'un document

- Il est possible de ne récupérer qu'une partie des données.
- Exemples :

**# Les méta-données + les champs title et text**

GET /website/blog/123?\_source=title,text

**# Juste la partie source sans les méta-données**

GET /website/blog/123/\_source

**# Vérifier qu'un document existe (Retour 200)**

HEAD /website/blog/123

# 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/blog/123**

```
{
  "title": "My first blog entry",
  "text": "I am starting to get the hang of this...",
  "date": "2014/01/02"
}
...
{
  "_index" : "website",
  "_type" : "blog",
  "_id" : "123",
  "_version" : 2,
  "created": false
}
```

# Création ... if not exist

- 2 syntaxes sont possible pour créer un document seulement si celui-ci n'existe pas déjà dans la base :

PUT /website/blog/123?**op\_type=create**

- Ou

PUT /website/blog/123/**\_create**

- Si le document existe, ELS répond avec un code retour 409

# Suppression d'un document

```
DELETE /website/blog/123
```

```
...
```

```
{
```

```
  "found" : true,
```

```
  "_index" : "website",
```

```
  "_type" : "blog",
```

```
  "_id" : "123",
```

```
  "_version" : 3
```

```
}
```



# Concurrence des mises à jour

- Elasticsearch gère la concurrence des accès à sa base par une approche **optimiste**
- En s'appuyant sur le champ **\_version**, il s'assure qu'une requête de mise à jour s'applique sur la dernière version du document. (ELS peut également s'appuyer sur un champ version externe)
- Si ce n'est pas le cas (cela veut dire que le document a été mis à jour par une autre thread entre-temps), il répond avec un code d'erreur 409

```
{  
  "error" : "VersionConflictEngineException[[website][2] [blog][1]:  
version conflict, current [2], provided [1]]",  
  "status" : 409  
}
```

# 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

# Example

POST /website/blog/1/\_update

```
{
  "doc" : {
    "tags" : [ "testing" ],
    "views": 0
  }
}
...
{
  "_index" : "website",
  "_id" : "1",
  "_type" : "blog",
  "_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/blog/1/_update {
  "script" : "ctx._source.views+=1"
}
POST /website/blog/1/_update {
  "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*** : Création d'un document non existant
    - ***index*** : Création 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": "blog", "_id": "123" }}
{ "create": { "_index": "website2", "_type": "blog", "_id": "123" }}
{ "title": "My first blog post" }
{ "index": { "_type": "blog" }}
{ "title": "My second blog post" }
{ "update": { "_type": "blog", "_id": "123", "_retry_on_conflict" :
3} }
{ "doc" : {"title" : "My updated blog post"} }
```



# Réponse

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

# Routing

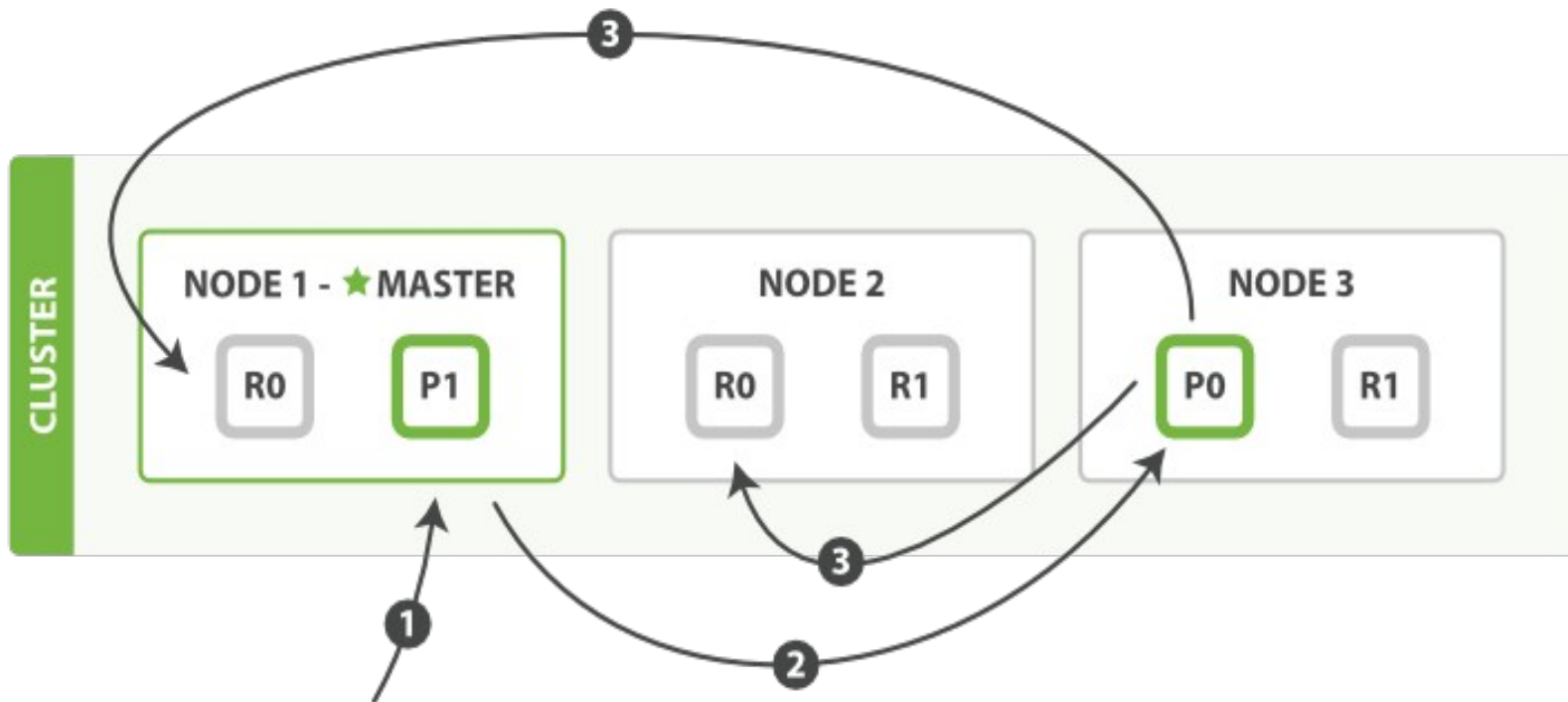
# Résolution du shard primaire

- 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

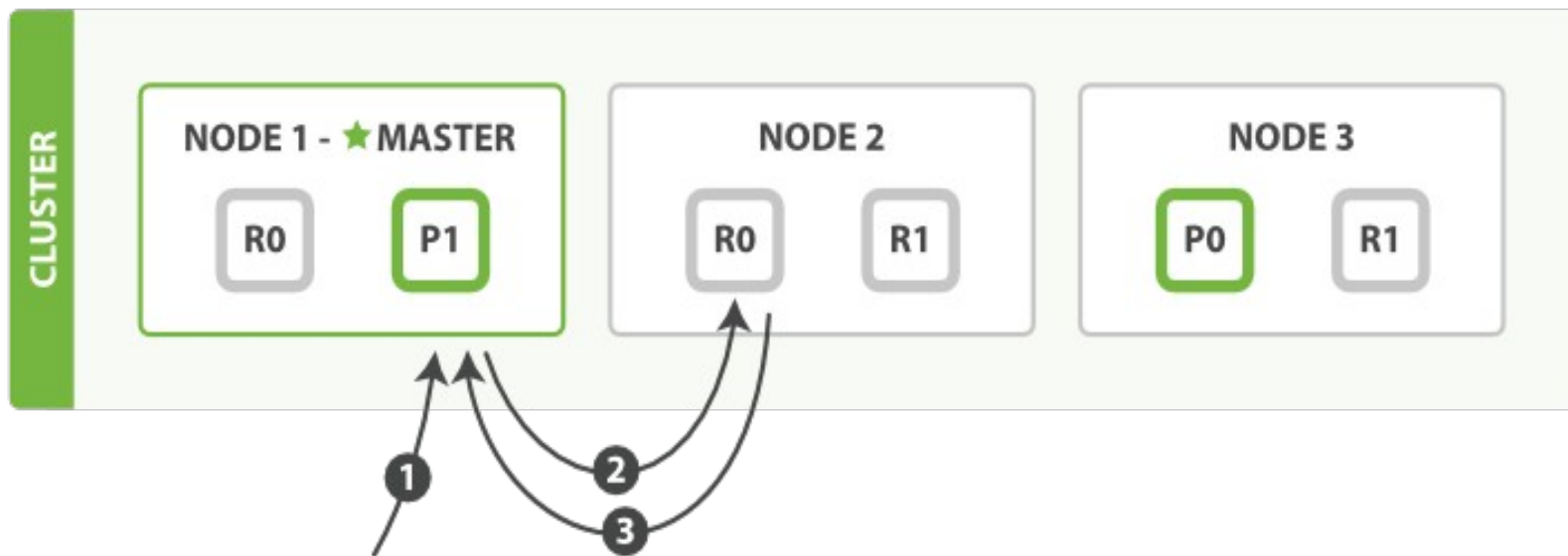
# Séquence d'une mise à jour sur une architecture cluster



# 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 transfère la requête vers le nœud 3 , ou 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 répliques 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 transfère 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)

# API Search Lite



# 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, un type

- ***\_search*** : Tous les index, tous les types
- ***/gb/\_search*** : Tous les types de l'index gb
- ***/gb,us/\_search*** : Tous les types de l'index gb et us
- ***/g\*,u\*/\_search*** : Tous les types des index commençant par g ou u
- ***/gb/user/\_search*** : Tous les documents de type user dans l'index
- ***/gb,us/user,tweet/\_search*** : Tous les documents de type user ou tweet présents dans les index gb et us
- ***/\_all/user,tweet/\_search*** : Tous les documents de type user ou tweet présents dans tous les index

# 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 indique la chaîne de requête
- 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/tweet/\_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

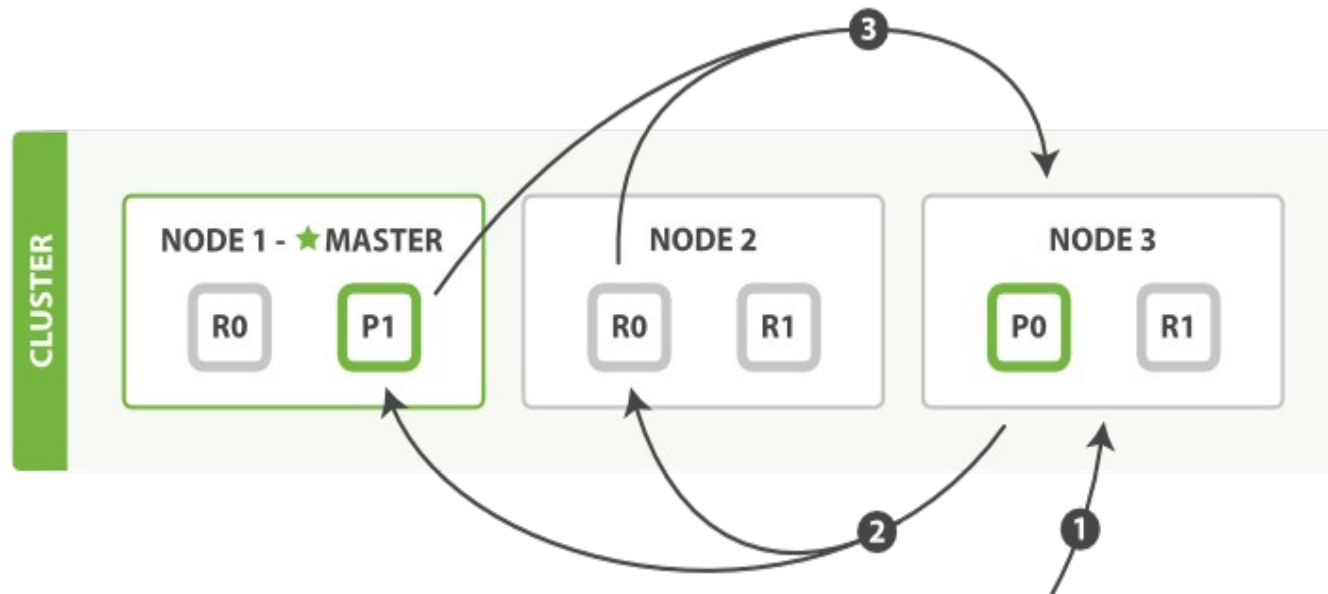
# Distribution de la recherche

# 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



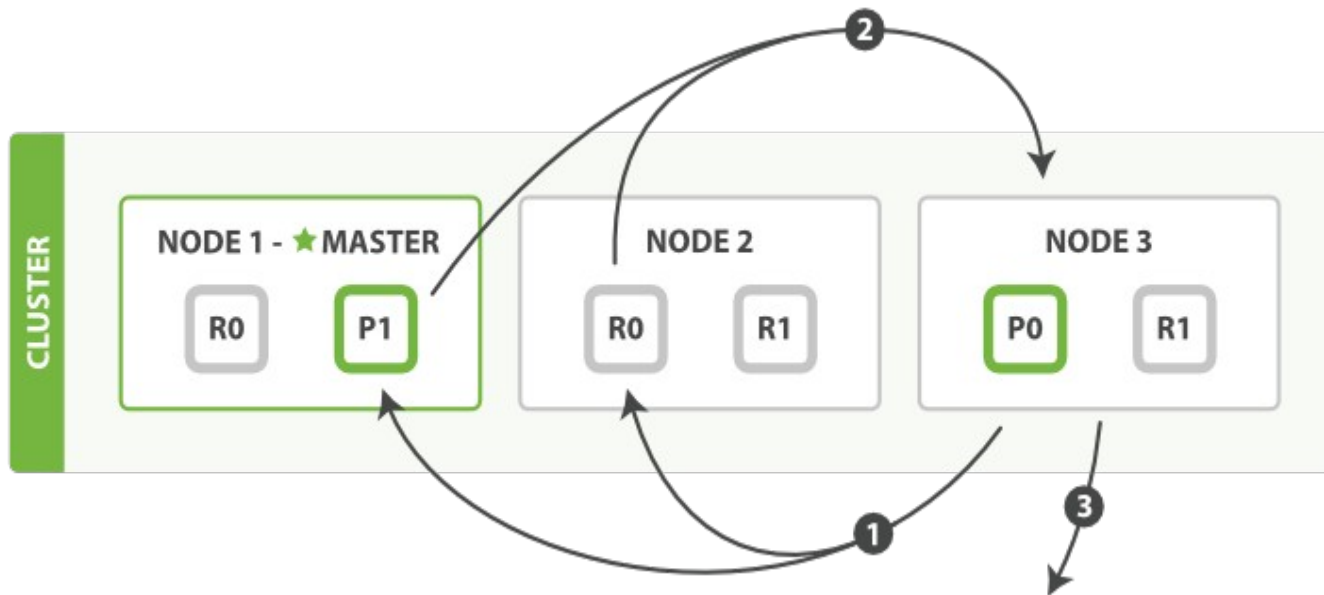
# Phase de requête

- 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

Beats

Concepts Logstash

Syntaxe configuration Logstash

Principaux inputs, filtres, outputs

# Introduction Beats

- Les **beats** sont des convoyeurs de données
- Ils sont installés comme agents sur les différents serveurs et envoient leur données
  - Directement à *ElasticSearch*
  - Ou à des pipeline *logstash* qui peuvent effectuer certaines transformations sur les données

# 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

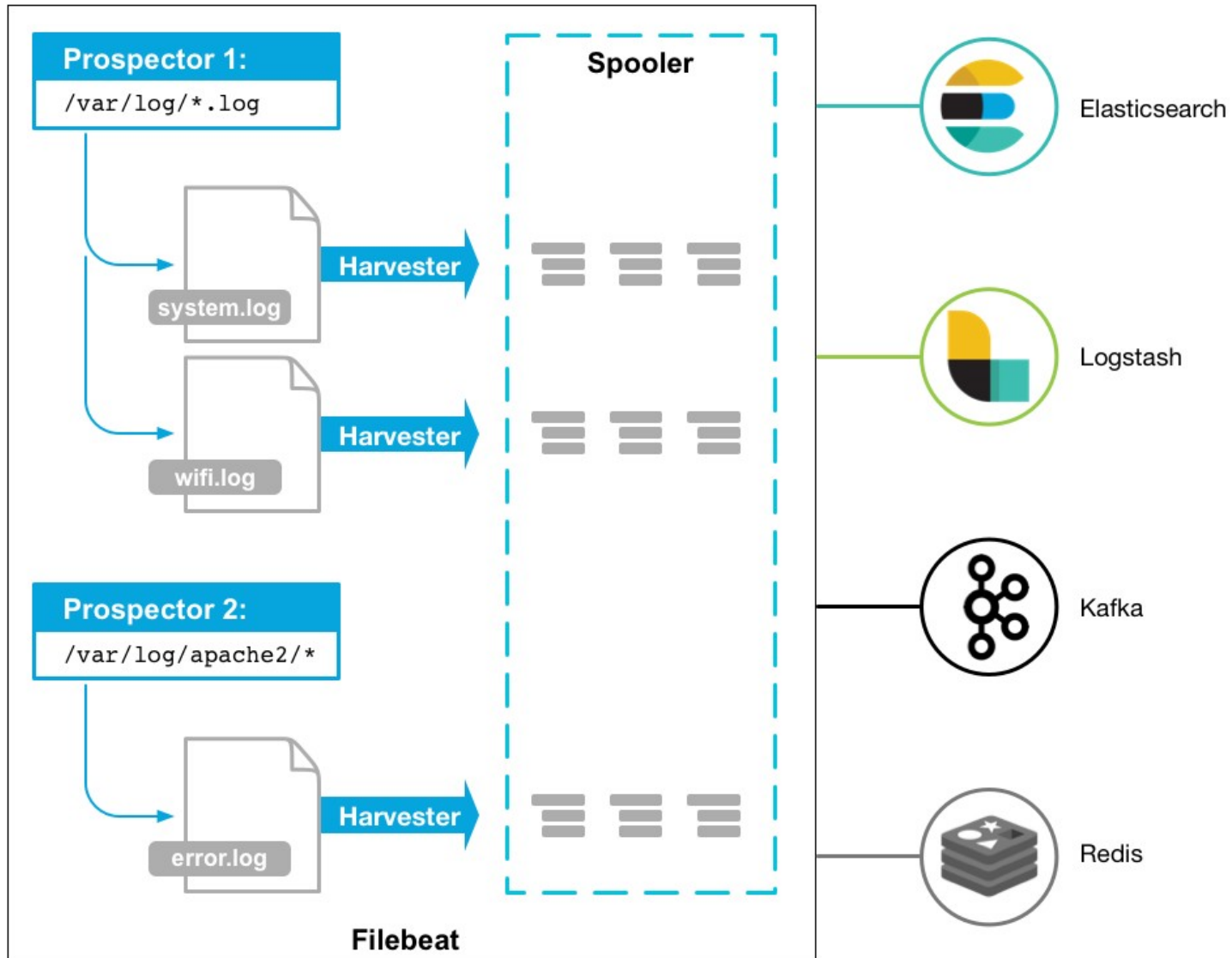


# FileBeat

# 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 bords Kibana
    - soit manuellement, soit automatiquement au 1<sup>er</sup> démarrage
  - Charger les gabarits d'index ELS
    - soit manuellement, soit automatiquement au 1<sup>er</sup> 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
```

MetricsBeat

# 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

# Concepts Logstash

# 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, mixer et orchestrer différentes entrées
- Extensible via des plugins, il se connecte à de nombreuses sources de données

# Types de source

- Traces et métriques : Apache, log4j, syslog, Windows Event, JMX, ...
- Web : Requêtes HTTP, Twitter, Hook pour GitHub, JIRA, Polling d'endpoint HTTP
- Support de persistance : JDBC, NoSQL, \*MQ
- Capteurs diverses : Mobiles, Domotique, Véhicules connectés, Capteur de santé

# Enrichissement de données

- *Logstash* permet également d'enrichir les données d'entrées.
  - Géo-localisation à partir d'une adresse IP. (lookup vers un service)
  - Encodage/décodage de données
  - Normalisation de dates
  - Requêtes Elastic Search pour l'enrichissement
  - Anonymiser des informations sensibles



# Structure répertoires

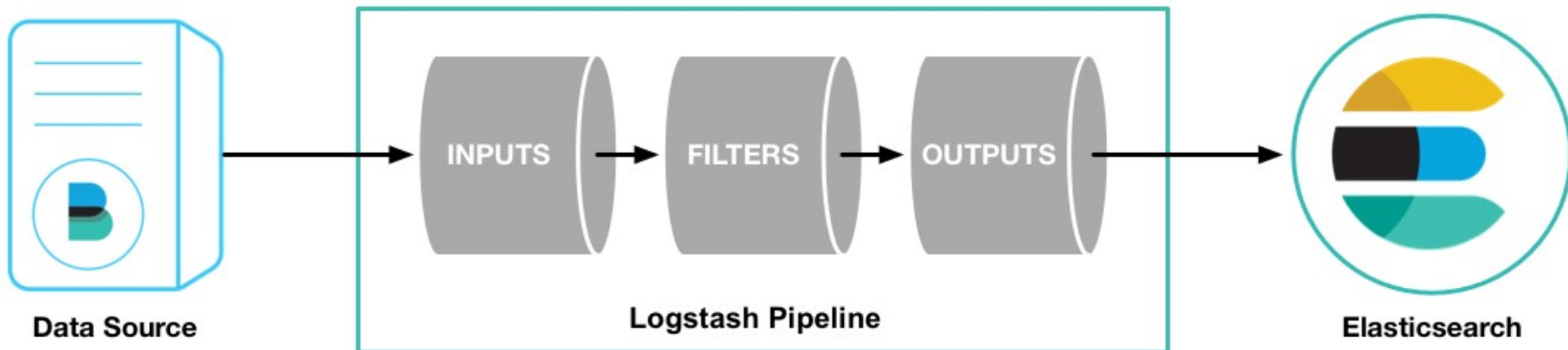
- **bin** : Scripts ; en particulier
  - *logstash* le script de démarrage
  - et *logstash-plugin* pour installer des plugins)
- **config**: Fichiers de configuration ; en particulier
  - *logstash.yml*
  - et *jvm.options*
- **logs** : Fichiers de traces, configurable via : *path.logs*
- **plugins** : plugins locaux (non Ruby-Gem). Chaque plugin est contenu dans un répertoire configurable via *path.plugins*

# Plugins

- Il est possible d'ajouter des plugins à l'installation :
  - en utilisant le dépôt *RubyGems.org* Exemple :  
`bin/logstash-plugin install logstash-output-kafka`
  - En utilisant un chemin local :  
`bin/logstash-plugin install  
/path/to/logstash-output-kafka-1.0.0.gem`
- Les plugins peuvent ensuite être mis à jour  
`bin/logstash-plugin update`

# 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 { } }'
```

# Example

```
input {  
  beats { port => "5043" }  
}  
filter {  
  grok {  
    match => { "message" => "%{COMBINEDAPACHELOG}" }  
  }  
  geoip {  
    source => "clientip"  
  }  
}  
output {  
  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'un 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 différentes installations 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>***

**TP : Installation logstash, premiers pas**

# Syntaxe configuration Logstash

# 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 {  
    ...  
}
```

# Examples

```
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}`***.

Inputs, filters et outputs Logstash

# 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 événements au hasard pour le test
  - **heartbeat** : Génère des événements *heartbeat* pour le test
  - **jdbc** : Événements à partir de JDBC
  - Kafka, redis, rabbitmq, jms : Messaging
  - Tcp, udp, log4j, unix, syslog : Événements 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 -0F* 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 *since.db*
- 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** : Parse 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 formaté.
  - ***json\_lines*** : Lit du contenu JSON formaté 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 que 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 événement correspond
  - **file** : Ecrit dans un fichier
  - **stdout** : Ecrit sur la sortie standard
  - **pipe** : Envoie vers l'entrée standard d'un autre programme
  - **http** : Envoie sur un endpoint HTTP ou HTTPS
  - **nagios, nagios\_nasca** : 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 template 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}(?:\\[%{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 }
}
```

***TP : Traitements de logs applicatifs***

# 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

# Résilience de données

# Introduction

- 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

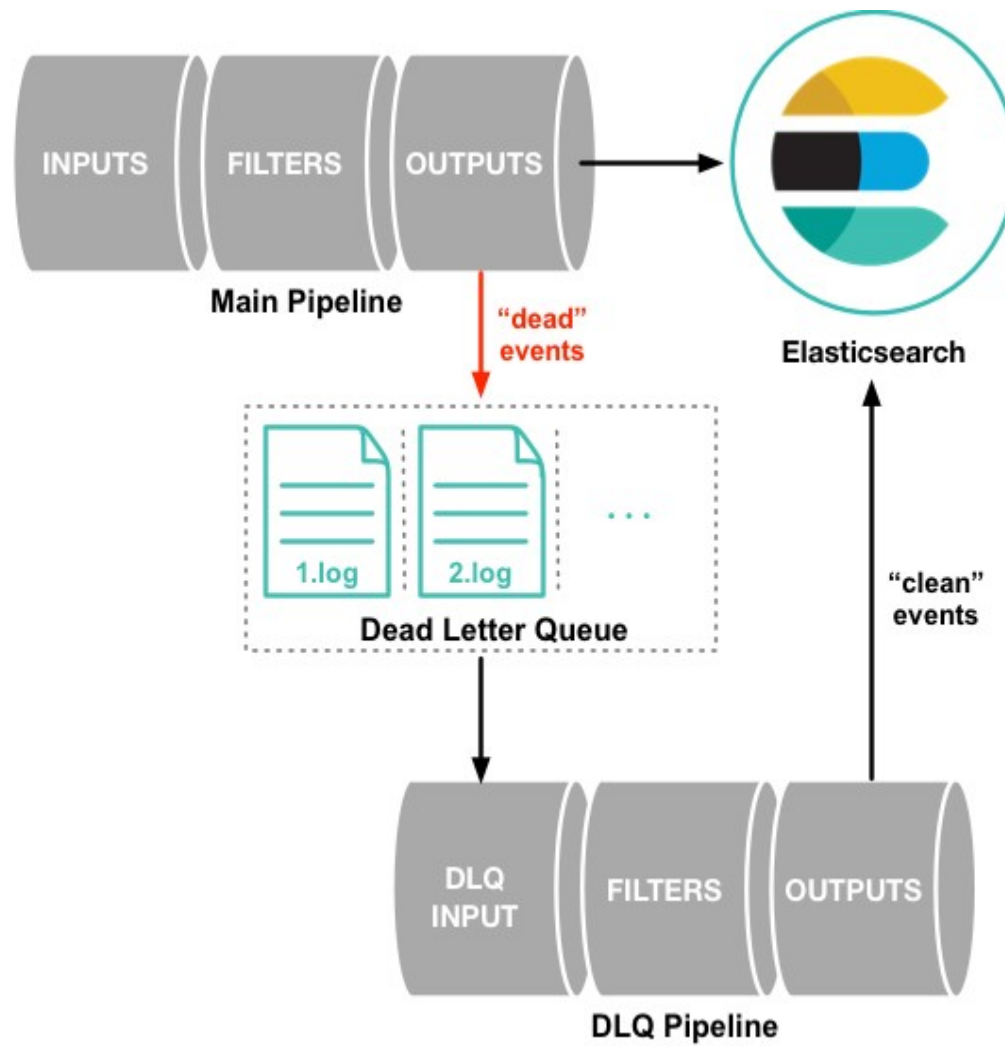
# Bufferisation

- 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 }  
  }  
}
```

***TP : Exécution des 2 pipelines***

# Mapping

Types de données  
Contrôle du mapping  
Configuration d'index  
Gabarits d'index

# Types de données

# 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.

# Example

	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

# 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 indexées par ELS 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

# 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** : C'est l'analyseur par défaut. Le meilleur choix lorsque le champ texte est dans des langues diverses. 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" }  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

# Indexation des objets embarqués

- Un document Lucene consiste d'une liste à plat de paires clé/valeurs. Les objets embarqués sont alors convertis comme suit :

```
{  
"tweet": [elasticsearch, flexible, very],  
"user.id": [@johnsmith],  
"user.age": [26],  
"user.name.first": [john],  
"user.name.last": [smith]  
}
```

# Attention !

```
"followers": [  
  { "age": 35, "name": "Mary White"},  
  { "age": 26, "name": "Alex Jones"},  
  { "age": 19, "name": "Lisa Smith"}]
```

Après indexation, cela donne :

```
{  
  "followers.age": [19, 26, 35],  
  "followers.name": [alex, jones, lisa, smith, mary, white]  
}
```

=> Lien entre age et nom perdu !

*q=age:19 AND name:Mary* retourne des résultats

=> Solution les ***Nested objects***



# Type nested

- Pour garder l'indépendance de chaque objet, il faut déclarer un type *nested*
- Cela a pour effet de créer des « sous-documents » indépendants et chaque sous-documents peut être recherché indépendamment (Voir nested query)

```
PUT my_index
{
  "mappings": {
    "my_type": {
      "properties": {
        "followers": {
          "type": "nested"
        }
      }
    }
  }
}
```

# Contrôle du mapping

# 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/tweet

# Réponse *\_mapping*

```
{ "gb": {  
  "mappings": {  
    "tweet": {  
      "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
  - 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": {
    "tweet" : {
      "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"
    }
  }
}
```

# Création et configuration d'index

# 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, 2 blocs peuvent être configurées :
  - Settings : Principalement répliques et shards
  - Mappings : Les types de champs et les analyseurs

```
PUT /my_index
```

```
{  
  "settings": { ... any settings ... },  
  "mappings": {  
    "type_one": { ... any mappings ... },  
    ...  
  } }  
}
```

Il est également possible de désactiver la création automatique

```
# config/elasticsearch.yml
```

```
action.auto_create_index: false
```

# 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 5 . 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
  }
}
```

# Champ *\_source*

- ELS stocke la string JSON (compressée) représentant le document dans le champ ***\_source***.
- Ce champ apporte plusieurs fonctionnalités :
  - Le document complet est accessible à partir des résultats de recherche
  - Les requêtes de mise à jour partielle sont possibles
  - Si une réindexation est nécessaire, le champ *\_source* peut être utilisé
  - Les valeurs individuelles des champs peuvent être extraites du champ source lors des requêtes GET ou SEARCH
  - Cela facilite le debugging

# Configuration de *\_source*

- Si le comportement par défaut n'est pas désirable :
- Il est possible de désactiver le stockage du document source:

```
PUT /my_index
{ "mappings": { "my_type":
{ "_source": { "enabled": false }
} } }
```

Ou d'affiner les champs à stocker via les attributs *excludes*,  
*includes*

```
PUT /my_index
{ "mappings": { "my_type":
{ "_source": { "excludes": ["data","content"] }
} } }
```

- Mais cependant, attention plus de surbrillance dans les recherches full-text

# Champ *\_all*

- Le champ *\_all* concaténant tous les champs n'est souvent plus utile lorsque l'on affine son index.
- Pour le désactiver :

```
PUT /my_index/_mapping/  
{ "my_type": { "_all": { "enabled": false } } }
```

- Pour spécifier les champs constituant *\_all* :

```
PUT /my_index/my_type/_mapping  
{ "my_type": { "include_in_all": false,  
  "properties": {  
    "title": { "type": "string", "include_in_all": true  
  },  
  ...  
} } }
```



# 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": {
    "_doc": {
      "dynamic_date_formats": ["MM/dd/yyyy"]
    }
  }
}
```

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

# Dynamic templates

- Les **gabarits dynamiques** 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 des collections de règles nommées.
- Ils peuvent être associés à un index ou un gabarit d'index (voir plus loin)
- 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

# Example

PUT my\_index

```
{ "mappings": { "my_type": {  
  "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 d'index



# Introduction

- Les gabarits d'index permettent de contrôler la configuration lorsqu'un index est créé en fonction de son nom
- Si le nom de l'index respecte un pattern (propriété ***index-pattern***), le gabarit s'applique
- Ils définissent les 2 aspects de configuration d'un index :
  - ***settings*** : Nombre de shards, de répliques, etc
  - ***mappings*** : Types des champs

# API Rest

- 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" : {
  "_default_" : {
    "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" } } } }
    }
  }
}
```

# Recherche avec DSL

Syntaxe DSL

Principaux opérateurs

Agrégations

Géolocalisation

# Syntaxe DSL

# 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*/type1,type2/_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
      ]
    }
  }
}
```

# Principaux opérateurs

# 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 booléenne 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

# Example

```
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 meilleur 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": "surprize" }
  }
}
```

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

# Agrégations

# 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 un 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 (Expérimental pour l'instant)

# Example Bucket

```
GET /cars/transactions/_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/transactions/\_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/transactions/\_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/transactions/_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/transactions/_search
```

```
{  
  "query" : {  
    "match" : { "make" : "ford" }  
  }, "aggs" : {  
    "colors" : {  
      "terms" : { "field" : "color" }  
    }  
  }  
}
```



# Spécification du tri

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

# Histogramme de date

```
GET /cars/transactions/_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
  - ...

# Example

```
{  
  "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  
        "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

# Géo-localisation

# 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" },  
    "location": { "type": "geo_point" }  
  }  
} } }
```

```
----
```

```
PUT /attractions/restaurant/1
```

```
{ "name": "Chipotle Mexican Grill", "location": "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

# Example

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

# Example

```
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 ] ]  
  } }  
}
```

# Example

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

# Analyse temps-réel avec Kibana

Introduction

Management

Discover

Visualisations et tableau de bord

Timelion

# 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 dynamique affichant le résultat des requêtes en temps réel
- Il s'exécute sur Node.js

# Dashboard Kibana



# Premier accès

- Lors du premier accès, Kibana demande avec quels index d'ElasticSearch il doit se connecter
  - Les index sont résolus à partir d'un motif (pattern) ; ex : /\* ou logstash-\*
  - Eventuellement, indiquer le champ timestamp de l'index
- Kibana utilise le dynamic mapping d'ElasticSearch.  
=> Si cette fonctionnalité est désactivée :
  - Il faut fournir explicitement le mapping pour la visualisation
  - Il faut autoriser le dynamic mapping pour l'index .kibana

```
PUT .kibana
{  "index.mapper.dynamic": true }
```

# Fonctionnalités

- Kibana propose 3 fonctionnalités principales :
  - **Discover** : Permet d'effectuer des recherche ES, de sélectionner les champs retournés et d'accéder à un document
  - **Visualisation** : Permet d'agréger les résultats de requêtes et de les afficher sous forme de graphique
  - **Dashboard** : Arranger en une seule page plusieurs visualisations
- Il propose également
  - **Management** : Gérer ses recherches
  - **Console** : Tester des requêtes REST vers ElasticSearch

# Quelques plugins

- ***X-Pack*** : Sécurité, Monitoring, Alerte. Inclus dans l'offre commerciale ou dernières versions
- Apps :
  - ***LogTrail*** : Interface technique spécialement pour les événements de trace
- Visualisations : Swimlanes, Gauge, Nuage de tags, Graphiques 3D

*<https://github.com/elastic/kibana/wiki/Known-Plugins>*



Management

# Management

- L'application ***management*** permet la configuration de Kibana et des objets que l'on peut sauvegarder
- Cette partie est pluggable et les plugins peuvent ajouter leur propres écrans de configuration

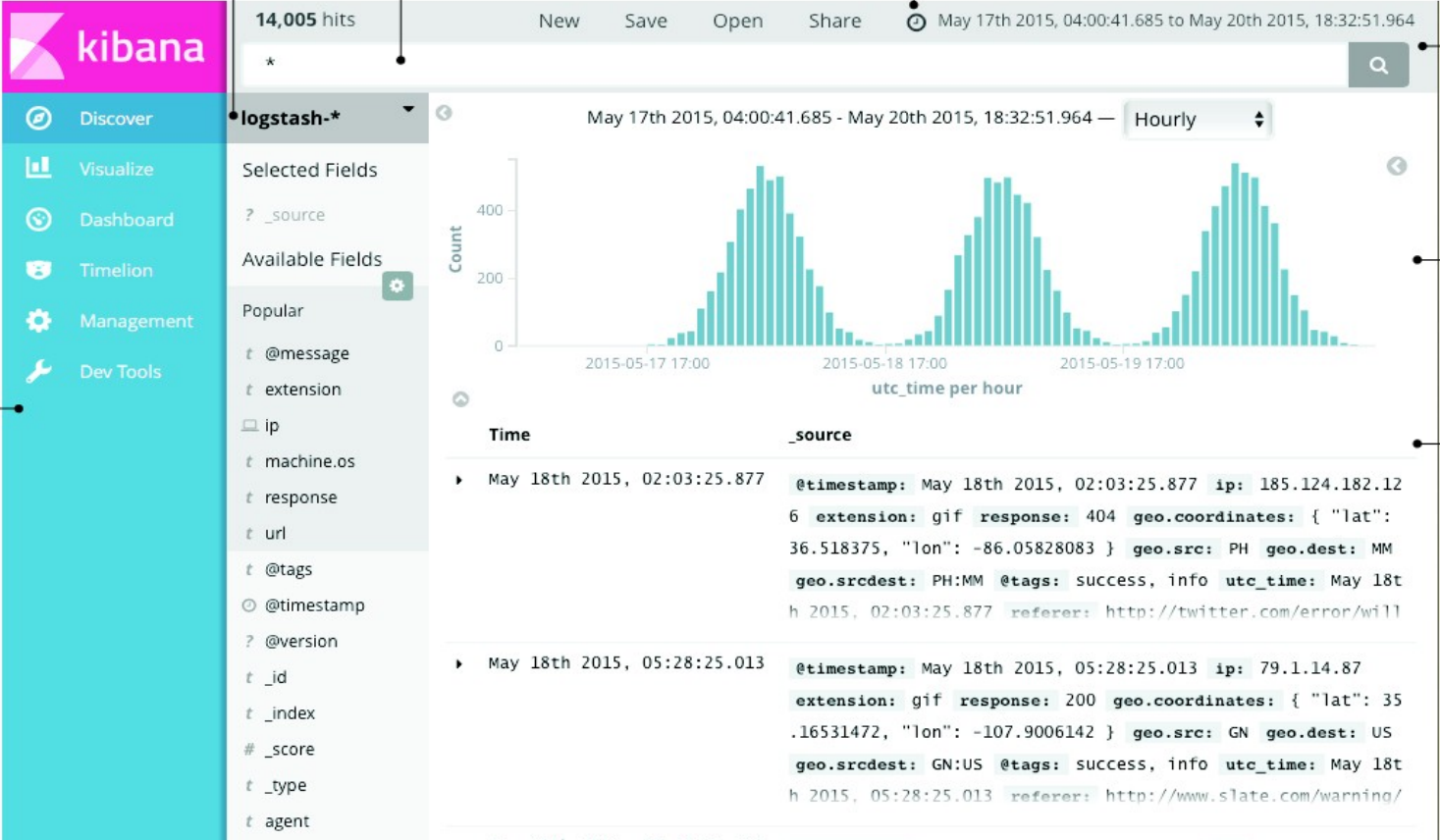
# Fonctionnalités

- Définir les motifs d'index
- Eventuellement le champ comportant le timestamp et le format de la date
- Gérer les champs, le formatage associé
- Créer des champs scriptés (créer et renseigner à la volée). Ces champs peuvent être utilisés dans les visualisations mais pas dans les recherches
- Plein d'options avancées
- Gérer les objets sauvegardés (Recherches, Visualisation et tableaux de bord)

Discover

Diagram illustrating the Kibana interface components and their labels:

- Index Pattern:** Points to the `logstash-*` field in the left sidebar.
- Query bar:** Points to the search bar containing the asterisk `*`.
- Time Picker:** Points to the clock icon in the toolbar.
- Toolbar:** Points to the top navigation bar containing buttons like New, Save, Open, Share, and the Time Picker.
- Side Navigation:** Points to the left sidebar menu containing options like Discover, Visualize, Dashboard, Timelion, Management, and Dev Tools.
- Histogram:** Points to the bar chart visualization showing the distribution of data over time.
- Document Table:** Points to the table displaying individual log entries with fields like `@timestamp`, `ip`, `extension`, `response`, `geo.coordinates`, `geo.src`, `geo.dest`, `geo.srcdest`, `@tags`, `utc_time`, and `referer`.



The screenshot shows the Kibana 'Discover' view. The left sidebar (Side Navigation) has a pink header with the 'kibana' logo and a teal menu with icons for Discover, Visualize, Dashboard, Timelion, Management, and Dev Tools. The main area features a toolbar with 'New', 'Save', 'Open', 'Share', and a clock icon (Time Picker). Below the toolbar is a search bar (Query bar) with the text '14,005 hits' and an asterisk. The index pattern 'logstash-\*' is selected. A histogram shows data distribution over time from May 17th to May 20th, 2015, with a 'Hourly' time range selector. Below the histogram is a table of log documents. The first document is from May 18th, 2015, at 02:03:25.877, with fields like @timestamp, ip, extension, response, geo.coordinates, geo.src, geo.dest, geo.srcdest, @tags, utc\_time, and referer. The second document is from May 18th, 2015, at 05:28:25.013, with similar fields.

# Fenêtre temporelle

- Si un champ *timestamp* existe dans l'index :
  - Le haut de page affiche la distribution des documents sur une période (par défaut 15 mn)
  - La fenêtre de temps peut être changée

# Recherche

- Les critères de recherche peuvent être saisis dans la barre de requête. Il peut s'agir de :
  - Un simple texte
  - Une requête Lucene
  - Une requête DSL avec JSON.
- Le résultat met à jour l'ensemble de la page et seuls les 500 premiers documents sont retournés dans l'ordre chronologique inverse
- Des filtres sur les champs peuvent être spécifiés
- La recherche peut être sauvegardée
- L'index pattern peut être modifié
- La recherche peut se rafraîchir automatiquement toutes les X temps

# Gestion des filtres

- Plusieurs boutons sont disponibles sur les filtres :
  - Activation désactivation
  - Pin : Le filtre est conservé même lors d'un changement de contexte
  - Toggle : Filtre positif ou négatif
  - Suppression
  - Edition : On peut alors travailler avec la syntaxe DSL et créer des combinaisons de filtre

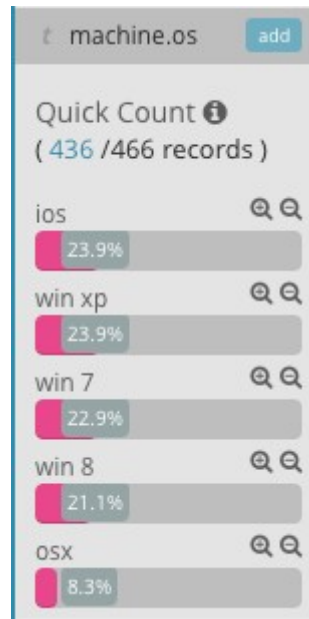


# Visualisation des documents

- La liste affiche par défaut 500 documents (propriété *discover:sampleSize*)
- Il est possible de choisir le critère de tri
- Ajouter/Supprimer des champs
- Des statistiques sont également disponibles

# Statistiques sur la valeur des champs

- Des statistiques sur la valeur des champs permettent de savoir combien de documents ont une valeur particulière dans un champ



Visualisation et tableau de bord

# Introduction

- Les visualisations sont basées sur des recherches ES
- En utilisant des agrégations, on peut créer des graphiques qui montrent des tendances, des pics, ...
- Les visualisations peuvent ensuite être composées en tableau de bord

# Types de visualisations

- **Area** : Visualise les contributions totales de plusieurs séries
- **Table de données** : Affichage en tableau des données agrégées
- **Line** : Compare différentes séries
- **Markdown** : Format libre pour des informations ou des instructions
- **MetricDisplay** : Une unique valeur.
- **Pie** : Camembert .
- **Tile map** : Associe le résultat de l'agrégation à un emplacement géographique
- **Timeseries** : Adopté aux échelles de temps
- **Vertical bar** : Histogramme permettant la comparaison de séries

# Étapes de création

- Les étapes de création consistent donc à :
  1. Choisir un type de graphique
  2. Spécifier les critères de recherche ou utiliser une recherche sauvegardée
  3. Choisir le calcul d'agrégation pour l'axe des Y (*count, average, sum, min, ...*)
  4. Choisir le critère de regroupement des données (bucket expression) (Histogramme de date, Intervalle, Termes, Filtres, Significant terms, ...)
  5. Définir éventuellement des sous-agrégations

# Bucket expressions

- **Histogramme Date** : Un histogramme de date est construit sur un champ entier pour lequel on a précisé une fenêtre temporel
- **Intervalle** : Entier, date ou IPV4
- **Terme** : Permet de présente les n meilleurs (ou plus basse) pertinence ordonné par la valeur agrégée
- **Filtres** : Il est possible d'ajouter des filtres de données à ce niveau
- **Significant Terms** : Agrégation *significant terms*
- **Geohash** : Agrégation sur les coordonnées géographiques (Data Table et carte)

# Options

- Lors de plusieurs agrégations en Y, il est possible de spécifier leur mode d'affichage
  - **Stacked** : Empile les agrégations les unes sur les autres.
  - **Overlap** : Superposition avec transparence
  - **Wiggle** : Ombrage
  - **Percentage** : Chaque agrégation comme une portion du total
  - **Silhouette** : Chaque agrégation comme variance d'une ligne centrale
  - **Grouped** : Groupe les résultats horizontalement par les sous-agrégations (Grapique barre) .



# Types de cartes

- Plusieurs types de cartes sont disponibles :
  - Marqueurs circulaires à l'échelle : Adapte la taille des marqueurs en fonction de la valeur du métrique agrégé
  - Marqueurs circulaires ombragés : Affiche le marqueurs avec différentes ombre en fonction de la valeur du métrique
  - Cellule Geohash ombragé : Affiche des cellules rectangulaires avec différentes ombres en fonction du métrique.
  - Heatmap : Applique du flou au marqueur circulaire et de l'ombrage en fonction du chevauchement

# Tableau de bord

- Un tableau de bord affiche un ensemble de visualisations sauvegardées
- Il est possible de réarranger et retailler les visualisations
- Il est possible de partager les tableaux de bord par un simple lien

Timelion

# Introduction

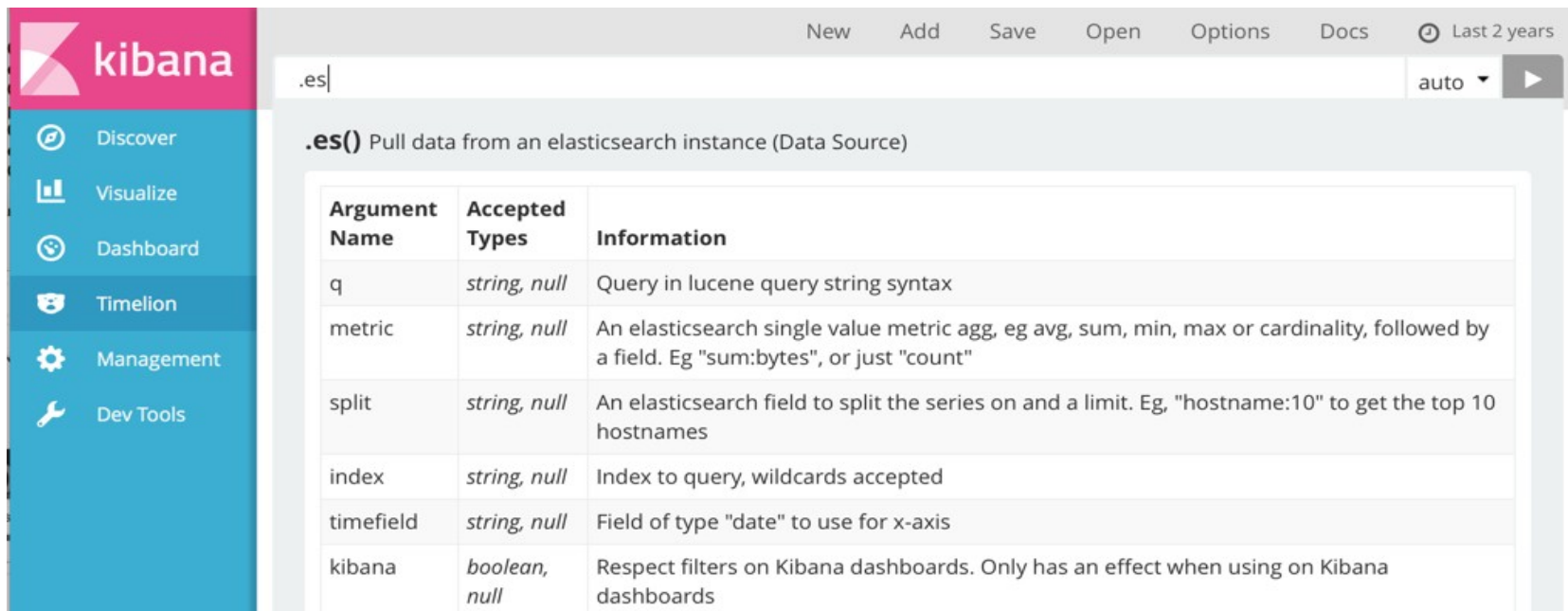
- Timelion est un visualiseur dédié aux données temporelles qui permet de combiner des sources de données complètement indépendantes dans le même graphique.
- Il est piloté par un langage d'expression simple permettant de récupérer les données et d'effectuer des calculs.
- Avec timelion, il est possible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

# Éléments de syntaxe

- L'interface de timelion consiste en :
  - Une zone de saisie d'expression
  - Une zone de visualisation
- Chaque expression timelion démarre avec une fonction identifiant une source de données. Pour *ElasticSearch* :  
`es(*)`
- 2 sources de données peuvent être dessinées côte à côte en utilisant la virgule (,) ;  
`.es(*) , .es(metric=cardinality:user)`
- Elles peuvent être combinées via des fonctions :  
`.es(*).divide(es(metric=cardinality:user))`
- D'autres sources de données peuvent être utilisées (Worldbank's Data API par exemple)  
`.wbi(MZ) // Population du Mozambique`

# Aide en ligne et documentation

- La documentation fait partie de *timelion*
- Elle est disponible par le lien Docs
- De plus, la complétion est très efficace



The screenshot shows the Kibana user interface. On the left is a sidebar with the Kibana logo and navigation links: Discover, Visualize, Dashboard, Timelion (highlighted), Management, and Dev Tools. The main area displays the documentation for the `.es()` function, titled ".es() Pull data from an elasticsearch instance (Data Source)". At the top of the main area is a search bar with ".es|" entered and a dropdown menu set to "auto". Below the title is a table with three columns: Argument Name, Accepted Types, and Information.

Argument Name	Accepted Types	Information
q	string, null	Query in lucene query string syntax
metric	string, null	An elasticsearch single value metric agg, eg avg, sum, min, max or cardinality, followed by a field. Eg "sum:bytes", or just "count"
split	string, null	An elasticsearch field to split the series on and a limit. Eg, "hostname:10" to get the top 10 hostnames
index	string, null	Index to query, wildcards accepted
timefield	string, null	Field of type "date" to use for x-axis
kibana	boolean, null	Respect filters on Kibana dashboards. Only has an effect when using on Kibana dashboards

# 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')
-

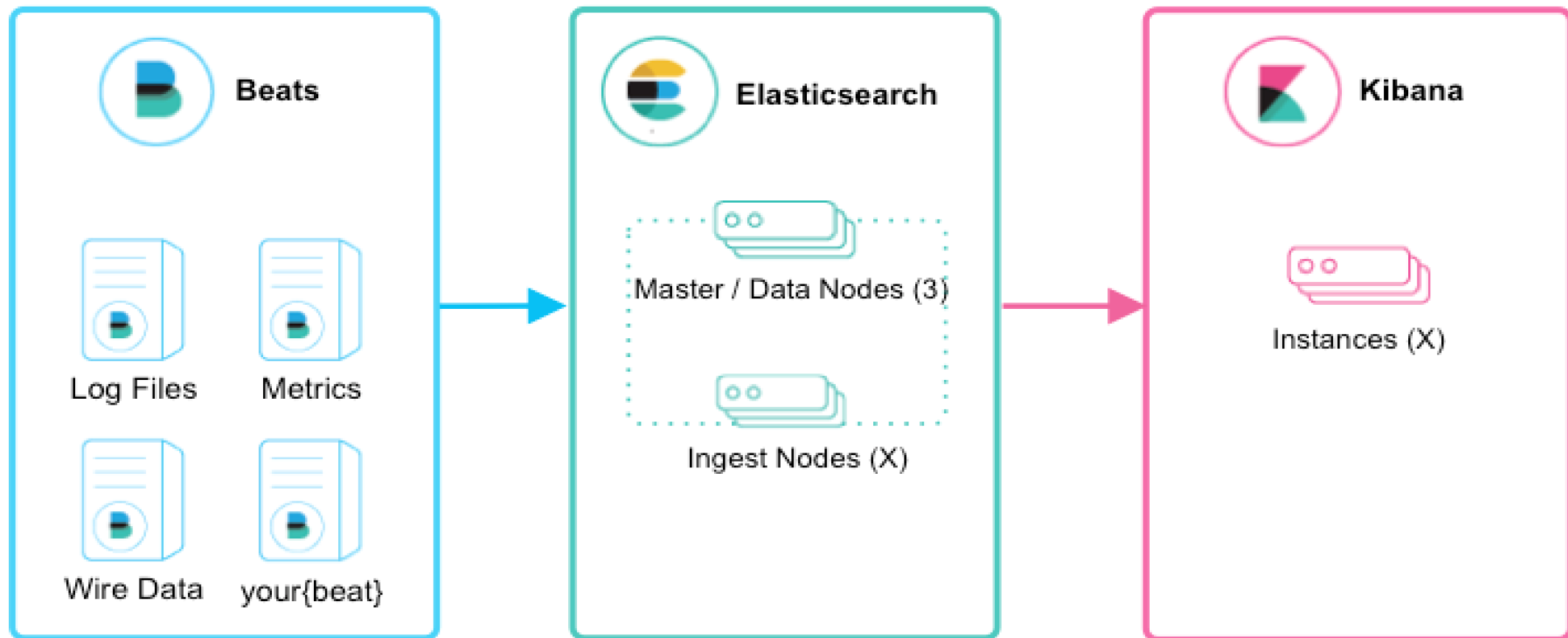
# Déploiement en production

Architectures ingestion  
Monitoring Logstash  
Architecture Indexation/Recherche  
Monitoring ES  
Points de vérifications  
Exploitation



Architectures ingestion

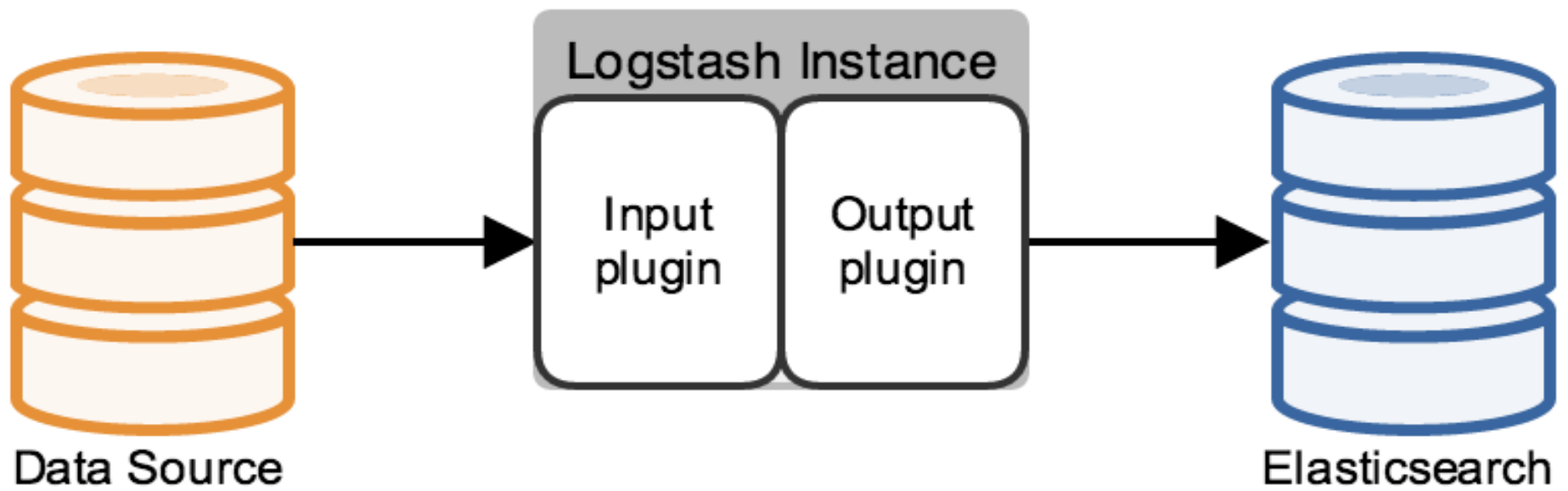
# Architecture sans logstash



# Apports de logstash

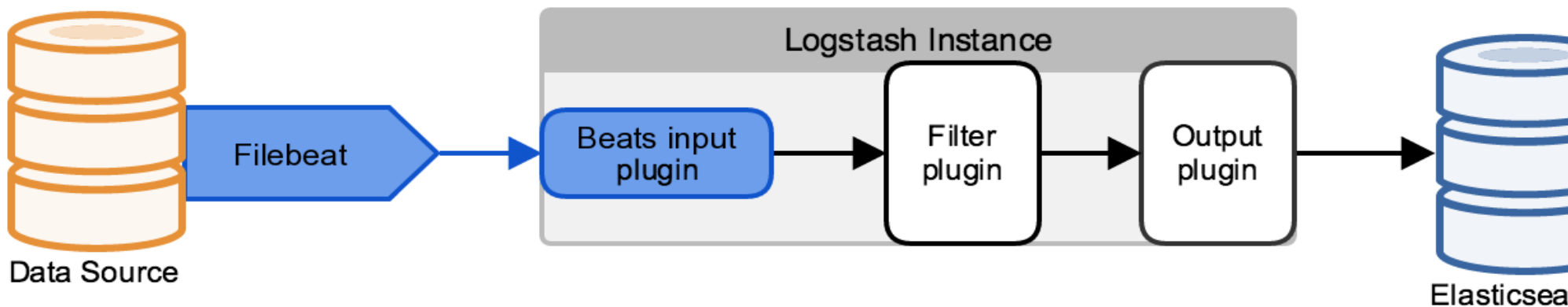
- S'adapter à des pics de charge, via le système de bufferisation intrinsèque de Logstash
- Ingérer des données provenant d'autres sources de données : BD, S3, ou files de message
- Émettre des données vers plusieurs destinations : S3, HDFS ou fichier
- Insérer de la logique conditionnelle dans le traitement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

# Architecture minimale

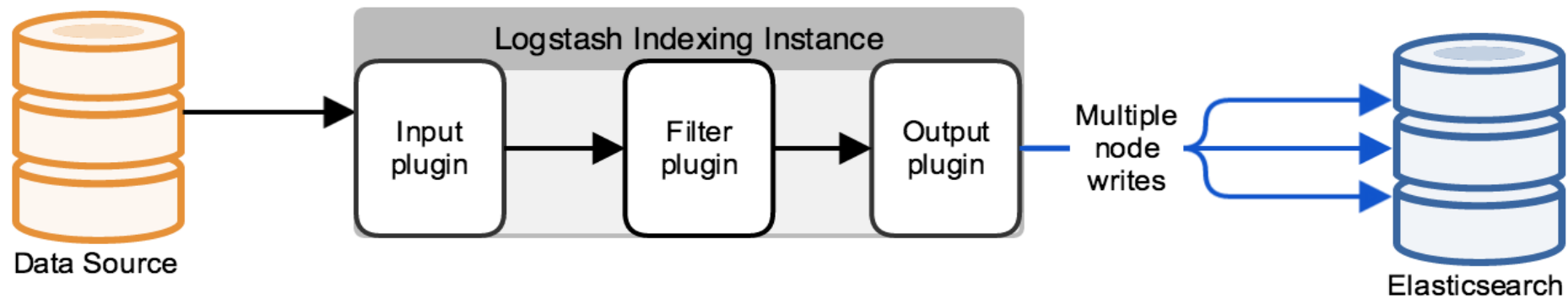


# Architecture avec un Beat

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



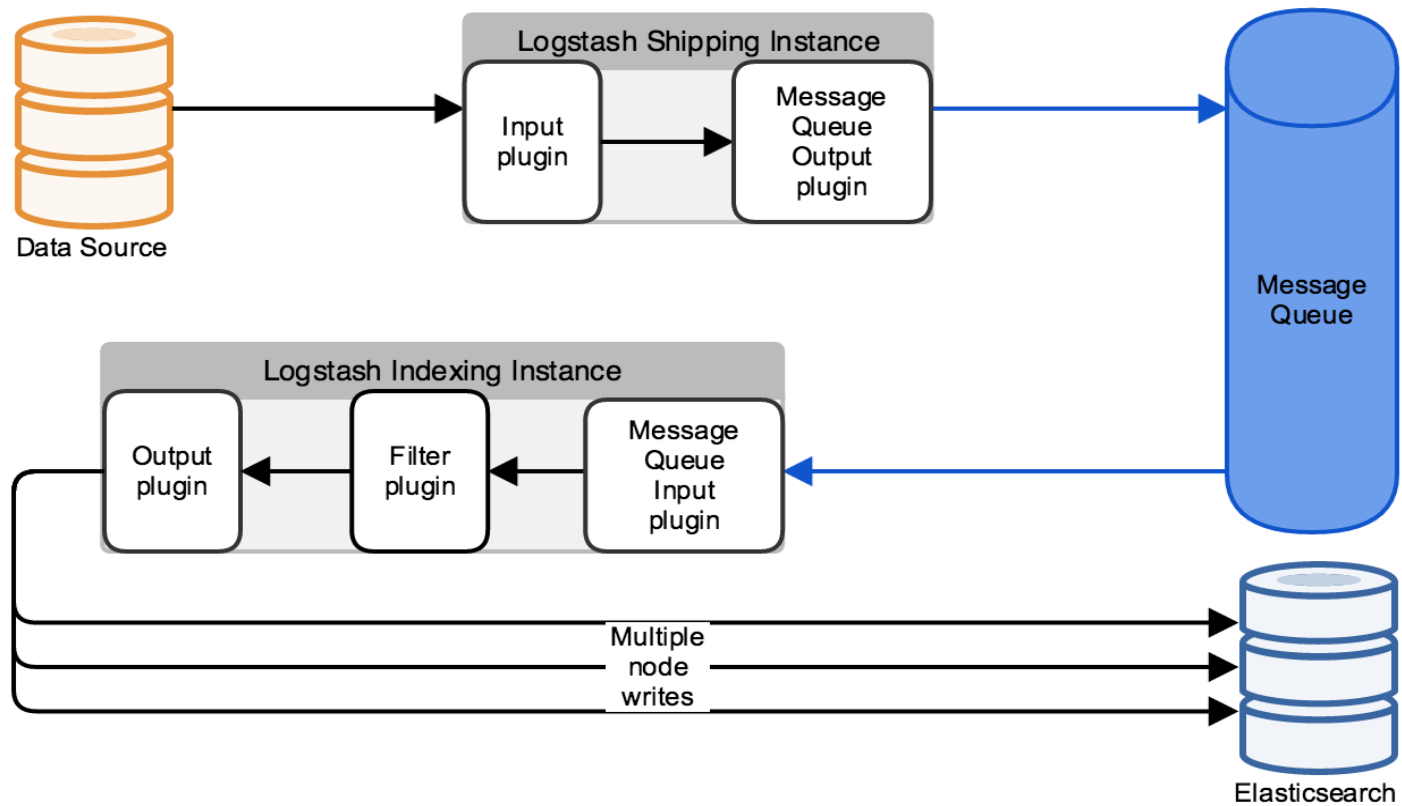
# Load balancing sur les noeuds de ElasticSearch



# Message broker

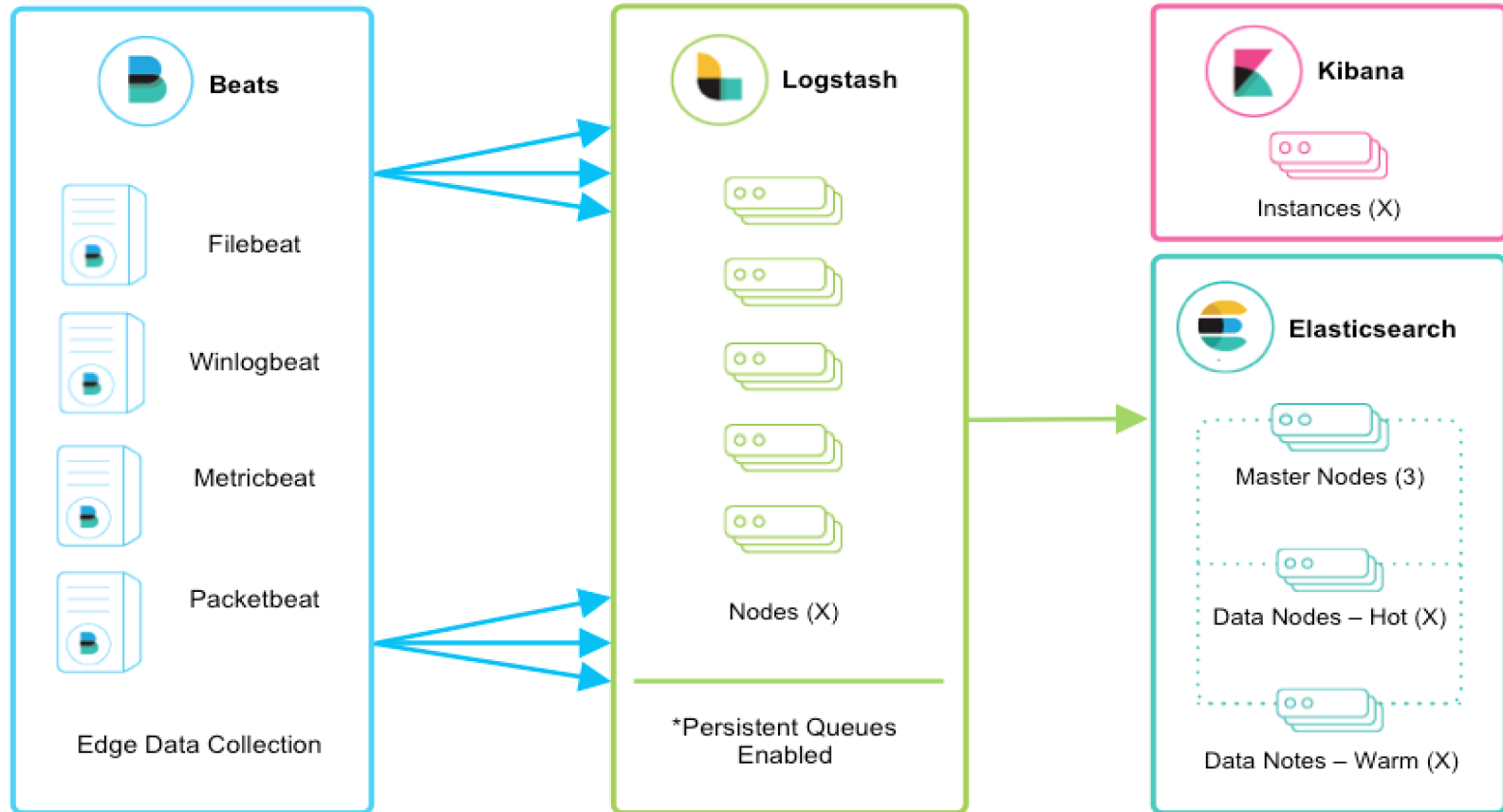
- Afin de faire face à des pics de débit, l'architecture peut inclure des message brokers (Redis, Kafka, RabbitMQ).
  - Cela peut soulager le travail d'indexation d'ElasticSearch
- Des instances de logstash écrivent vers une file de message
- D'autres lisent de la file, effectue les traitements et envoient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

# Message broker





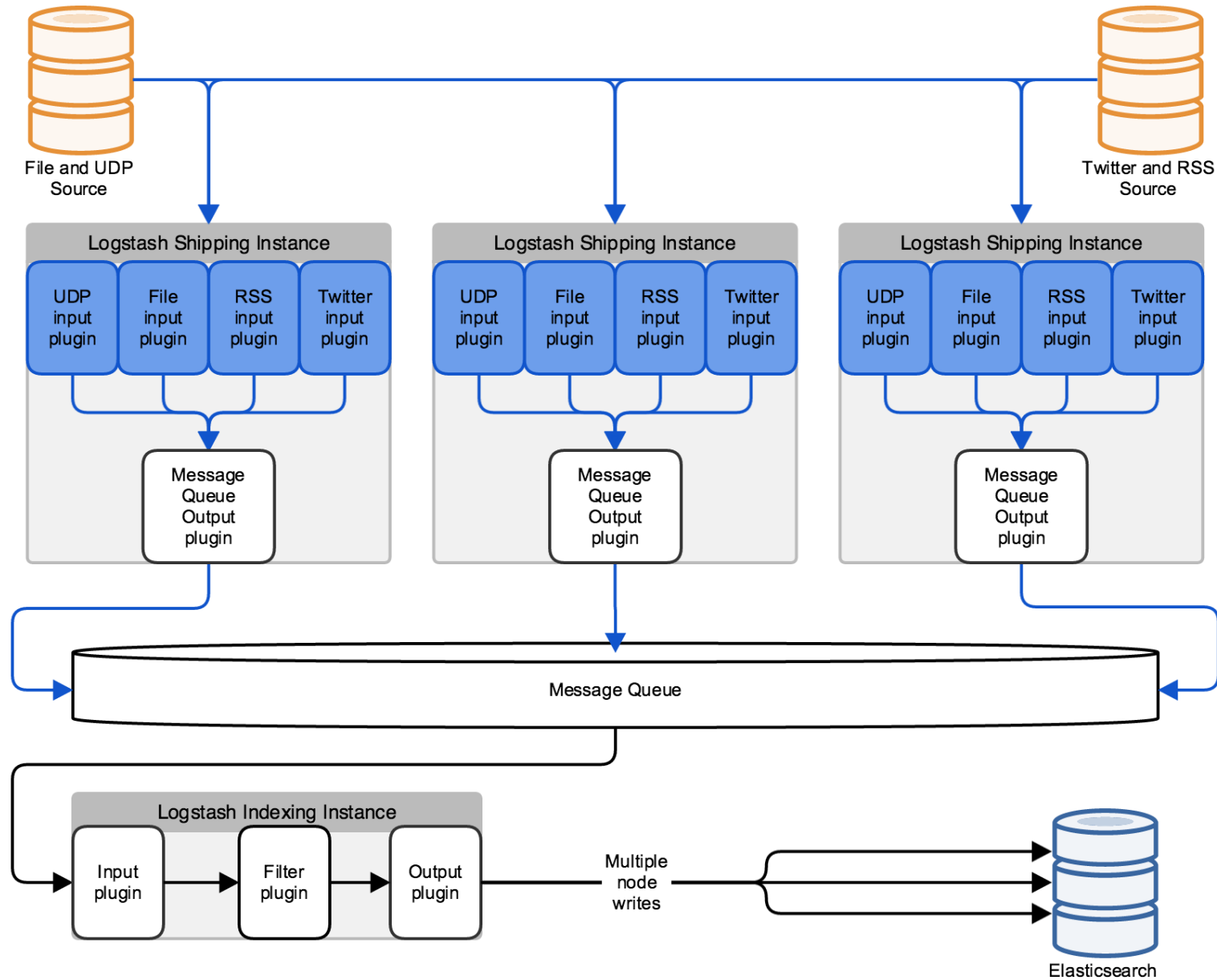
# Répartir la charge sur plusieurs Logstash



# 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

# Monitoring logstash

# 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 formatés 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/pipeline**

- 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

# Example 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

# Architecture Indexation/Recherche

# 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 coeur plutôt que la rapidité du CPU
- **Disques** : Si possible rapides, SSDs ? Éviter NAS (network-attached storage)

# JVM

- Dernière version d'Oracle ou OpenJDK
- 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 définit 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.

# Nœuds 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
```

# Personnalisation d'une configuration (2)

- ***minimum\_master\_nodes*** : Le nombre minimal de nœuds éligible comme master pour qu'une élection ait lieu. Le fixer à un quorum des nœuds de type master  
`discovery.zen.minimum_master_nodes: 2`
- **Attributs pour le redémarrage**  
Exemple attendre le démarrage de huit nœuds puis 5 minutes ou les 10 nœuds avant d'entamer le processus de recovery  
`gateway.recover_after_nodes: 8`  
`gateway.expected_nodes: 10`  
`gateway.recover_after_time: 5m`

# 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

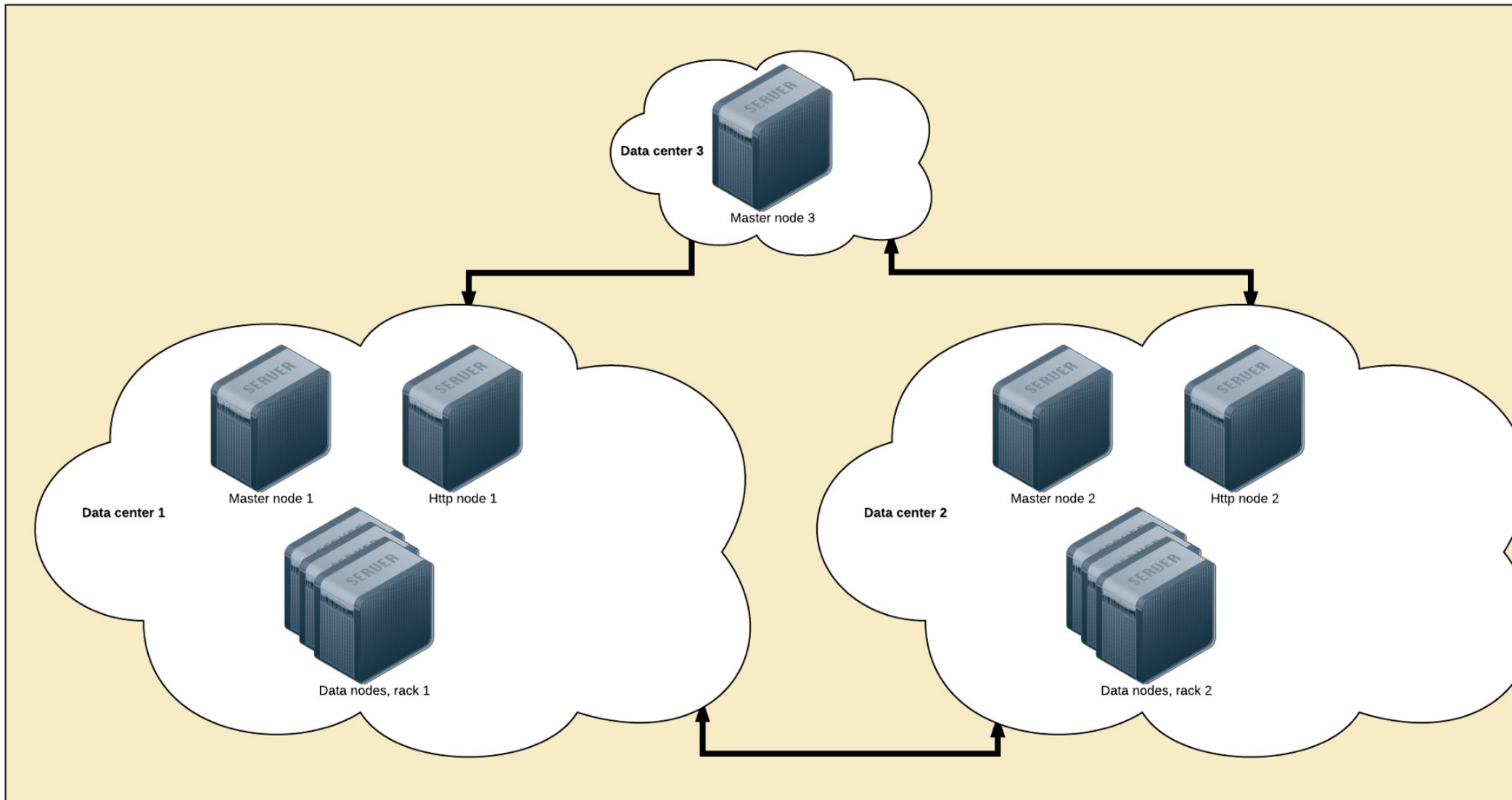
- Lucene utilise énormément de fichiers. ELS énormément de sockets
- La plupart des distributions Linux limite le nombre de descripteurs e fichiers à 1024.
- Cette valeur doit être augmenté à 64,000

# 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



# 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 effectuée 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 : 5)
- 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
```

# Débit de stockage

- ELS a 2 propriétés qui protègent contre des situation d'étranglement lors d'écriture
  - ***indices.store.throttle.max\_bytes\_per\_sec*** : limite le débit d'écriture. Par défaut il est de 10mb/s ; ce qui est peu. Elle peut éventuellement être augmentée :  
`indices.store.throttle.max_bytes_per_sec: 2g`
  - ***indices.store.throttle.type*** protège de trop nombreuses opérations de fusion. Elle peut être désactivée si on utilise l'API\_bulk  
`indices.store.throttle.type: "none"`



# Monitoring Elasticsearch

# X-Pack

- ELS offre une API permettant d'obtenir certains métriques d'un cluster
- La version X-Pack permet de disposer de tableaux de bord Kibana 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 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, ...

# 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

# Example

```
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 un 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"
  }
}
```

Ensuite

```
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

# Liens intéressant

- <https://thoughts.t37.net/designing-the-perfect-elasticsearch-cluster-the-almost-definitive-guide-e614eabc1a87>
- <https://fr.slideshare.net/VadimSolovey/s-your-elastic-cluster-stable-and-production-ready>

# Annexes

# 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 la détection d'anomalies dans des données temporelles

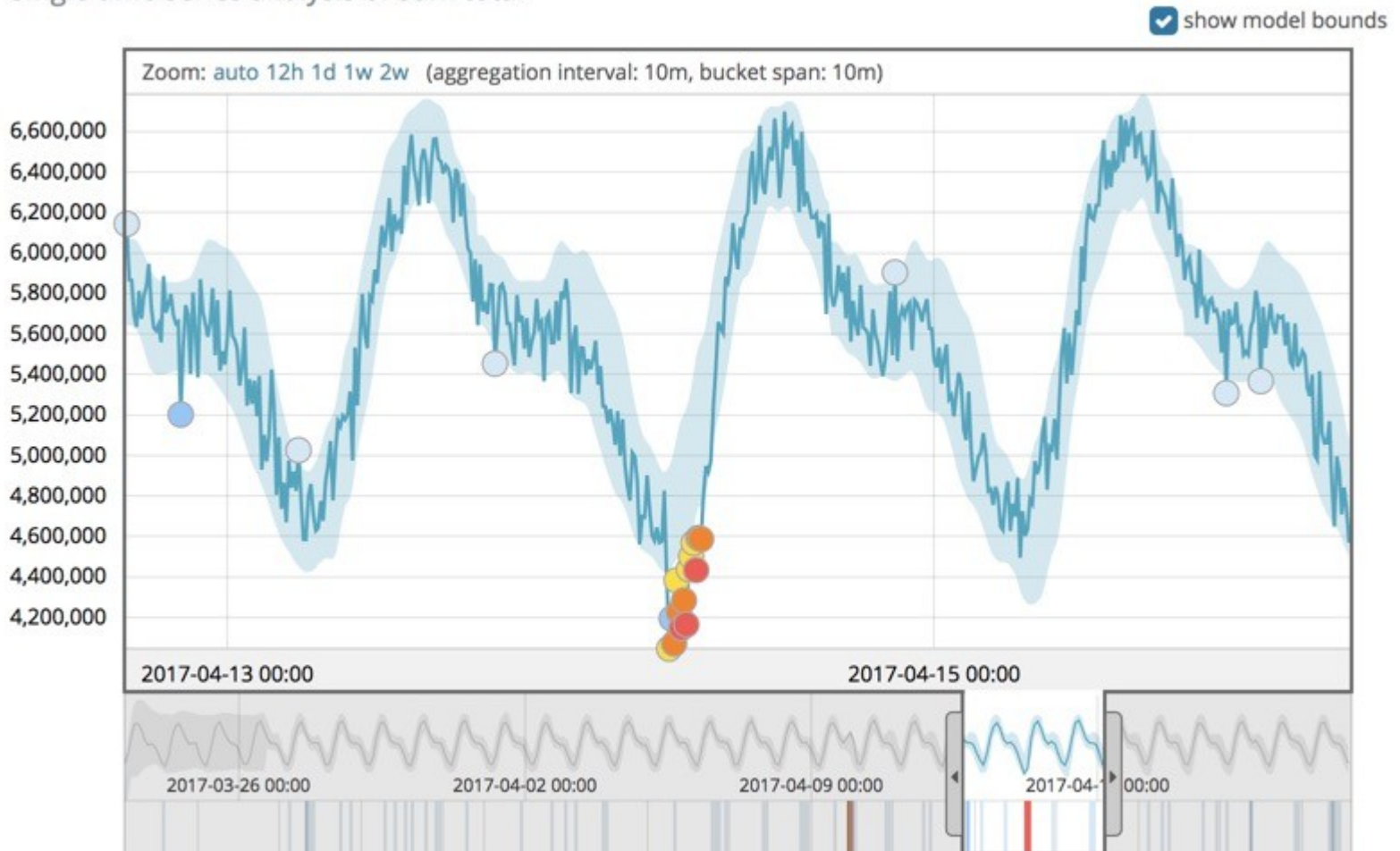
# Principe

- 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

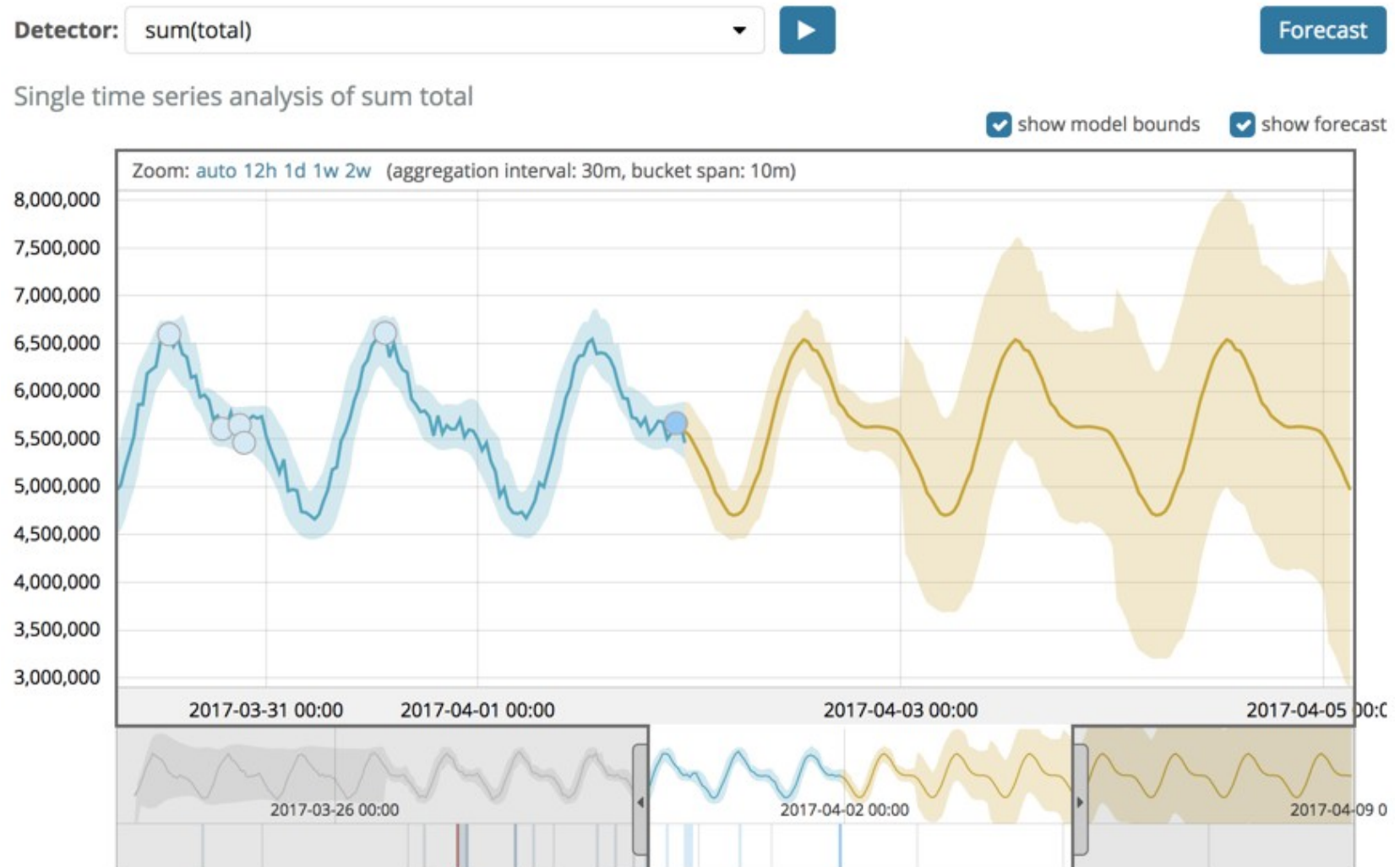
Single time series analysis of sum total



# 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** : Détecte des anomalies lorsque le nombre d'événements dans un groupement est anormal
  - **Géographiques** : Détecte les anomalies dans l'emplacement géographique d'une donnée
  - Sur le **contenu** : Détecte 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** : Détecte 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 la 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