



UNIVERSITÉ
DE LORRAINE

IUT
Charlemagne
nancy

Lothaire

LICENCE ASRALL

Analyse de logs avec ELK

Auteur :

François DUPONT

Tuteur :

Vincent DELOVE



elastic



Compilé le 18 juin 2015

Année Scolaire **2014-2015**

Remerciements

Je tiens tout d'abord à remercier Vincent DELOVE mon maître de stage pour m'avoir proposé le sujet, mais également pour son support, ses conseils, sa patience et sa disponibilité.

Je remercie également Emanuel NATAF mon tuteur pour l'intérêt qu'il a porté au sujet ainsi que les marges de manœuvres qu'il a su me laisser.

Merci à toute l'équipe du réseau Lothaire pour son accueil chaleureux et son soutien. Je remercie plus particulièrement Luc DIDRY et Stéphane FETTER pour leur aide, en général, et notamment en système.

Table des matières

I	Introduction	4
1	Sujet du stage	5
1.1	Problématique	5
1.2	Objectif	5
1.3	Organisation du rapport	6
2	Environnement	7
2.1	Réseau Lothaire	7
2.2	Environnement de travail	8
II	Les logiciels	9
3	Logstash	10
3.1	Présentation de Logstash	10
3.2	Installation	12
3.3	Grammaire et conjugaison	13
3.4	Utilisation	18
4	Elasticsearch	21
4.1	Introduction	21
4.2	Présentation de Elasticsearch	21
4.3	Installation	22
4.4	La théorie	22
4.5	L'infrastructure	22
4.6	Les API Elasticsearch	27
4.7	Le mapping et l'analyse	31
4.8	Conclusion	33
5	Kibana	34
5.1	Qu'est ce que Kibana	34
5.2	Installation de Kibana	34
5.3	Utilisation de Kibana	35

III Synthèse et conclusion	41
6 Synthèse	42
6.1 Analyse des performances	42
6.2 Améliorations	42
7 Conclusion	44
 IV Annexes	 45
8 Sources/Webographie/Bibliographie	46
9 Images	48
9.1 Cartes Lothaire	48
9.2 Images Kibana	50
9.3 Statistiques Munin	53
10 Code Source et scripts	55
10.1 Elasticsearch	55
10.2 Code en production	57
11 Parties non essentielles	61

Première partie

Introduction

Chapitre 1

Sujet du stage

1.1 Problématique

La gestion de logs est une problématique commune à presque tous les services informatiques, tous les fournisseurs de services et plus globalement, toute personne gérant une infrastructure informatisée.

Les logs sont précieux car ils sont un des éléments indispensables au dépannage, au monitoring. Ils permettent d'anticiper des problèmes inhérents à l'exploitation des infrastructures, et de constater de potentielles tentatives d'intrusions. À ces aspects pratiques relatifs au bon fonctionnement des systèmes s'ajoute également une problématique judiciaire. Il est souvent nécessaire de garder des logs pour un temps défini dans la loi. Ces logs doivent évidemment être consultables de manière diligente.

Avec des milliers de lignes de logs par seconde, il est devenu impossible de naviguer "au hasard" dans les fichiers pour regarder si tout se passe bien.

Cependant, *de nombreuses structures continuent encore aujourd'hui d'analyser plus ou moins "manuellement" leurs logs*. Cette méthode reste envisageable si on cherche quelque chose de précis, comme une IP dans un intervalle de quelques secondes, mais impossible pour des recherches moins ciblées.

1.2 Objectif

L'objectif de ce projet est de fournir un outil supplémentaire aux administrateurs pour vérifier le bon fonctionnement de leur infrastructure sans avoir à attendre l'alerte Nagios (qui reste indispensable). L'idée est de permettre un suivi quotidien, permettant de mettre en évidence des indicateurs de bonne ou de mauvaise santé du système, sans pour autant que ces problèmes soient critiques (mieux le système est entretenu, moins on risque la panne). Pour réaliser ce projet il a été décidé d'utiliser la pile ELK¹.

1. Elasticsearch Logstash Kibana

1.2.1 Limites

Pour des raisons de disponibilité matérielle mais, aussi, d'intérêt par rapport au sujet, je n'ai pas travaillé sur les clusters Elasticsearch de façon pratique (une seule instance du logiciel tourne actuellement dans la cluster de salle machine). Elasticsearch est un logiciel hautement parallélisable, et qui donc *scale* très bien à l'horizontal.

Lors de notre étude de Logstash nous nous concentrerons sur l'exploitation de syslog. Il est possible de faire d'autres choses avec ce logiciel, notamment concernant, l'exploitation des logs d'apache. Ce qui sera présenté fournira tout de même une base solide des possibilités offertes par Logstash.

1.3 Organisation du rapport

La première partie du rapport sera consacrée à l'environnement de travail et à la présentation des objectifs.

La seconde partie présente l'utilisation et le fonctionnement des principaux logiciels utilisés dans le projet. Pour des raisons de lisibilité certains passages moins essentiels ont été déplacés en Annexe.

Enfin la troisième partie, qui conclura ce rapport, présente la solution retenue pour la mise en production. Elle sert également à faire le bilan du projet ainsi qu'à proposer quelques pistes d'amélioration.

Chapitre 2

Environnement

J'ai réalisé mon stage de Licence professionnelle ASRALL, dans l'équipe réseau Lothaire. Cette équipe dépend de la Sous Direction des Infrastructures, membre de la Direction du Numérique de l'Université de Lorraine. Son rôle est d'opérer le réseau des établissements scolaires et de recherche de la région Lorraine, le réseau Lothaire.

2.1 Réseau Lothaire



Le réseau de télécommunications à haut débit sur le territoire de la région Lorraine, dénommé Lothaire, est réservé aux secteurs de l'Enseignement Supérieur, de la Recherche et du Transfert de Technologie.

L'équipe Lothaire assure l'interconnexion des réseaux de Nancy (*StanNet*), Metz (*AmpereNet*), Epinal (*EpiNet*) ainsi que d'autres sites en région (IUFR, CROUS...) avec le réseau *RENATER* (REseau NATIONAL de télécommunication pour la Technologie, l'Enseignement et la Recherche). Ce lien est réalisé au nœud régional situé sur le site du Montet (mon lieu de stage) à Vandœuvre-lès-Nancy.

(cf schéma 10.1 et schéma 10.2 en annexes)

2.1.1 Historique, rôle et organigramme

Avant la création de l'Université de Lorraine au 1er Janvier 2012 l'équipe réseau Lothaire était attachée au CIRIL (Centre Interuniversitaire de Ressources Informatiques de Lorraine). Depuis, l'organisme a été absorbé par la Sous Direction des Infrastructures de l'université.

En plus d'opérer les infrastructures du réseau Régional, l' équipe réseau Lothaire assure un ensemble de services auprès de la Communauté de l'Enseignement Supérieur et de la Recherche de Lorraine :

- interface technique et financière avec les établissements partenaires, les opérateurs et les prestataires retenus ;
- configuration du routage (IPv4 et IPv6) et des règles de sécurité (ACL , Firewall) ;
- service de gestion des noms de domaines (DNS) ;
- développement, maintenance et diffusion nationale d'outils de métrologie ;
- administration d'un portail captif (YaCaP) pour les accès Wifi ;
- filtrage des accès web sur certains réseaux ;
- gestion d'une plateforme pour les accès sécurisés (VPN) ;
- collecte et archivage des traces (logs) des activités réseaux ;
- mise à disposition de logiciels libres pour la communauté ;
- service de synchronisation d'horloge (NTP) ;
- routage d'appel de visioconférences (gatekeeper H323) ;
- gestion des réseaux de campus de l'Université de Lorraine.

L'équipe est constituée de 10 ingénieurs, dont 1 responsable, Alexandre Simon, ainsi que d'un enseignant chercheur détaché.

2.2 Environnement de travail

J'ai physiquement réalisé mon stage dans un bureau, en compagnie d'un autre stagiaire. Ma machine de travail était un ordinateur fixe Dell. Il m'a été laissé une grande liberté quant à l'installation de mon système d'exploitation, j'ai donc choisi une classique Debian GNU/Linux.

Cette machine était bien pratique pour réaliser des tests (machines virtuelles, installation de logiciels...), mais n'était évidemment pas capable d'ingurgiter tous les logs réseau des établissements d'enseignement en Lorraine. On a donc mis à ma disposition (dans le Datacenter du bâtiment) 2 serveurs qui serviront, après la fin du stage, de machines de production. Un Dell R520 (12 cœurs, 12Go de RAM), la machine qui fera tourner Elasticsearch et un Dell 2950 (4 coeurs 4 Go de RAM) la machine qui accueillera le Logstash qui traitant tous les logs.

Par cohérence avec le reste de l'infrastructure il a été décidé d'installer une Debian Jessie, nouvellement stable au moment de l'installation.

Deuxième partie

Les logiciels

Chapitre 3

Logstash



3.1 Présentation de Logstash

Logstash permet de traiter et/ou de transférer des données en masse. Dans notre projet, nous l'utiliserons essentiellement en conjonction avec Elasticsearch en utilitaire de sortie. Cependant, et comme le montre la liste des ses abondants plugins d'output, Logstash est capable de fonctionner avec de nombreux autres logiciels.

3.1.1 Pourquoi Logstash ?

Bien qu'un administrateur système compétent soit capable d'analyser de façon rapide et efficace les logs d'une machine (à l'aide de perl,awk,sed,tail,grep...), cette méthode est fastidieuse. De plus, face à des dizaines/milliers de machines (virtuelles aussi), cette méthode de travail n'est plus applicable, elle ne peut pas passer à l'échelle. De plus Il devient de plus en plus fréquent (cloud, applications multicouches...) que les logs d'une seule machine ne suffisent pas à diagnostiquer convenablement un problème.

C'est pour centraliser tout types de logs que logstash a été créé, nous l'utilisons surtout pour sa vélocité et sa grande synergie avec Elasticsearch.

3.1.2 Fonctionnement interne

L'utilisation de Logstash s'articule autour de 3 *blocs* également appelés *stages* (phase).

- Le bloc : *Inputs* génère des événements à partir des informations reçues par logstash en entrée.
- Le bloc : *Filters* modifie, manipule, ces événements dans logstash
- Le bloc : *Outputs* envoie les événements de logstash vers leur prochaine destination.

Logstash est un logiciel développé en Jruby¹ (Jruby est en fait une implémentation de ruby 1.8 en Java qui était à l'époque plus rapide que ruby "seul"). Le passage d'une phase à l'autre est implémenté via les *SizedQueue* de ruby. Elles sont dimensionnées pour contenir 20 éléments², ce n'est pas paramétrable sans modifier le code source, c'est un choix délibéré. Ces queues ne sont pas conçues pour stocker des données à long terme. On verra plus tard que cela justifie l'utilisation d'un *buffer*, comme *Redis*³.

Chaque bloc est composé d'une multitude de plugins. Ce sont des modules indépendants qui peuvent également fonctionner en conjonction les uns des autres.

Il est, par exemple, possible de configurer le bloc *inputs* pour utiliser plusieurs fois le plugin "file" (on imagine pour des cas d'utilisations différentes) et de se servir dans le même temps d'un autre plugin du bloc *inputs* : *stdin* qui prendra typiquement en entrée le clavier.

Il est possible d'implémenter de nouveaux plugins en ruby afin de les ajouter à notre logstash.

Il existe également un *pseudo-bloc* qui peut s'insérer dans les autres, ce bloc, *codec* permet de gérer la *représentation des données* c'est à dire qu'un codec est capable de lire ou d'écrire dans une syntaxe particulière comme par exemple *ruby-debug*, *collected*, ou, bien plus intéressant pour nous, *netflow*.

Tolérance de pannes

Les logs sont **importants**, les traiter efficacement est la raison d'être de Logstash. Il ne souhaite pas que vous en perdiez le moindre à cause d'un problème réseau ou d'une défaillance quelconque rendant la destination indisponible. Lors d'une indisponibilité, les plugins *outputs* tentent de renvoyer les événements vers leur destination. Si ce n'est pas possible le plugin arrête de lire sa queue tant que le message n'a pas pu être envoyé. Par effet domino, une fois la queue *filtre* => *output* remplie, le bloc *filtre*, ne pouvant plus envoyer de nouveaux messages à la queue *FO*. Le plugin du bloc *filtre* va également retenter d'envoyer ses messages, et refuser en attendant de lire les nouveaux arrivant dans la queue *input* => *filtre*. Si cette dernière venait à se remplir c'est le Logstash tout entier qui refuserait de traiter de nouvelles informations. Dans le meilleur des mondes, les expéditeurs de données se comporteraient comme logstash et attendraient patiemment que le problème se

1. sauf mention contraire, on supposera dans ce rapport que logstash est développé en ruby

2. appelés messages lorsqu'on parle de queues, on parle bien ici des **événements** logstash

3. voir la sous partie sur la tolérance de pannes ainsi que le chapitre consacré à Redis

résolve. Malheureusement cela n'est pas toujours possible d'où l'importance d'un Redis en amont (ou en aval) afin de faire tampon.⁴

Consulter la partie sur le **multithread** p61

3.2 Installation

Sur Debian Jessie, il n'existe pas de paquet officiel logstash maintenu. Il existe, en revanche, un paquet tiers⁵ fourni par l'éditeur du logiciel. Attention, le paquet nécessite l'ajout de dépendances manuelles⁶ ainsi qu'un rechargement de la configuration des services : *systemctl daemon-reload*.

Les dépendances nécessaires sont *jruby* et *openjdk-7-jre*, les mêmes que pour elasticsearch.

Une autre manière de réaliser l'installation est d'ajouter les dépôts logstash à **/etc/apt/source.list.d/logstash.list.d/logstash.list**

```
deb https://packages.elasticsearch.org/logstash/1.4/debian stable  
main
```

et évidemment la clef qui va bien⁷

```
wget -qO - https://packages.elasticsearch.org/GPG-KEY-elasticsearch |  
sudo apt-key add -
```

3.2.1 Configurations

Il est d'usage dans une installation *propre* de créer un utilisateur spécifique à notre utilisation. C'est également le cas dans notre paquet. L'administrateur système doit donner à cet utilisateur, nouvellement créé les droits nécessaires pour qu'il puisse faire son travail correctement. Dans le cadre de l'analyse de logs, faire de l'utilisateur logstash un membre du groupe *adm*. Le groupe d'administration de Debian lui permettra, par exemple, d'avoir accès en lecture à la plupart des fichiers de **/var/log/**.

Set Capabilities

Cependant, cette façon de faire peut ne pas être suffisante. Notre utilisation de logstash consiste, entre autre, à centraliser les logs. Ces derniers sont généralement envoyés par l'intermédiaire de syslog (RFC 5244). Par défaut, syslog utilise le port 514. Ce port, inférieur à 1024, est donc *privilegié*. Aussi, seul *root* a le droit d'écouter ces ports. Ajouter notre utilisateur logstash au groupe root enlèverait le

4. Il existe d'autres outils que Redis, (dont ce n'est pas la fonction principale) pour réaliser ce travail, ils sont plus adaptés mais aussi moins documentés dans leur utilisation avec Logstash.

5. https://download.elastic.co/logstash/logstash/packages/debian/logstash_1.4.2-1-2c0f5a1_all.deb

6. `wget paquet, dpkg -i paquet, apt-get install -f`

7. permet d'authentifier les paquets téléchargés (cf :SecureApt)

bénéfice de sécurité obtenu en isolant les utilisateurs en fonction de leurs besoins. Nous allons plutôt utiliser les *capabilities*⁸ et la commande *setcap*⁹

```
1 setcap cap_net_bind_service=+epi /usr/lib/jvm/java-7-openjdk-amd64/  
   jre/bin/java  
2 setcap cap_net_bind_service=+epi /usr/lib/jvm/java-1.7.0-openjdk-  
   amd64/jre/bin/java
```

qui permettent à un processus (thread en faite) de ne pas se soumettre à certaines vérifications de sécurité du noyau.

cap_net_bind_service permet à un utilisateur non privilégié (non root) d'écouter sur les ports privilégiés.

Les informations concernant les autres *capabilities* existantes sont disponibles dans la page de manuel correspondante : *man capabilities*.

=+epi signifie que l'on ajoute une *capability* à un fichier. Plus précisément **=+** signifie que l'on écrase les droits précédents pour les remplacer par les nouveaux. **e**, **p** et **i** sont la gradation de droits que l'on peut attribuer à un fichier avec les *capabilities*¹⁰.

- **effective** : indique si la *capability* est utilisée actuellement
- **permitted** : définit quelles *capabilities* sont autorisées pour un processus donné
- **inherited** : permet de transmettre la *capability* à un autre programme

Note : par défaut, changer le propriétaire d'un fichier de root, à non root enlève les *capabilities* associées à ce dernier¹¹.

Pour vérifier les *capabilities* d'un binaire, on utilise la commande *getcap*.

```
1 getcap /bin/ping  
2 /bin/ping = cap_net_raw+ep
```

Pour enlever les *capabilities*, il faut utiliser la commande suivante.

```
1 setcap cap_net_bind_service=-epi /usr/lib/jvm/java-1.7.0-openjdk-  
   amd64/jre/bin/java
```

Enfin, il est possible, de donner le droit à un utilisateur non root, l'autorisation de modifier certaines *capabilities*. Pour ce faire, il faut modifier le fichier */etc/security/capability.conf*.

3.3 Grammaire et conjugaison

3.3.1 Généralités

Dans cette section, nous allons brièvement expliquer le fonctionnement de la syntaxe du fichier de configuration de Logstash.¹²

8. <http://man7.org/linux/man-pages/man7/capabilities.7.html>

9. <https://github.com/elastic/logstash/issues/1587>

10. <https://www.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.2/capfaq-0.2.txt>

11. <http://stackoverflow.com/questions/17743062/changing-user-ids-for-assigning-additional-capabilities>

12. <http://logstash.net/docs/1.4.2/configuration>

Le ou les fichiers de configuration comportent deux ou trois parties, représentant les blocs dont nous avons parlé au préalable. Le bloc *filter* est optionnel et il est important de noter que, s'il est possible de répartir sa configuration dans plusieurs fichiers, cette dernière sera concaténée, attention aux boucles infinies dans ce cas là.

```
1 input {
2     stdin{}#ceci est un plugin
3 }
4
5 #filter {
6     #}#on peut également commenter en fin de ligne
7
8 output {
9     stdout{}#ceci est un autre plugin
10 }
```

Listing 3.1 – Configuration minimale

Ici, on constate l'utilisation des plugins `stdin` et `stdout`. Comme leur nom et leur emplacement dans le fichier de configuration peuvent le laisser supposer, Logstash collecte en entrée tout ce qui vient de l'entrée standard, typiquement le clavier, et l'envoie sur la sortie standard, typiquement, un terminal. Pour pouvoir recevoir l'output, il est d'ailleurs conseillé d'utiliser, le flag `-f` et de ne pas lancer logstash en mode `dæmon`.

Dans ce second exemple, plus complexe, nous allons présenter une utilisation plus vaste des plugins et la logique sous-jacente à leur utilisation.

```
1 #mail.conf
2 input {
3     file {
4         path => ["/home/fdupont/testmail"]
5     }
6 }
7
8 filter{
9     throttle{
10         before_count => 2
11         after_count => 4
12         period => 120
13         key => "%{message}"
14         add_tag => "contenu"
15     }
16 }
17
18 output {
19     stdout{ }
20     if "contenu" not in [tags]{
21         email{
22             to => "fdupont@localhost"
23             from => "logstash@%{host}"
24             subject => "%{message}"
25             body => "Ceci est un test sur l'hôte %{host} \n avec pour message : %{message}"
26         }
27     }
28 }
29
30 #shipping.conf
31 input {
32     file {
```

```
33     path => ["/var/log/secure", "/var/log/messages", "/var/log/*.  
34         log", "/var/mail/fdupont"]  
35     exclude => ["*.gz"]  
36 }  
37 file {  
38     path => ["/var/mail/ldidry"]  
39 }  
40  
41  
42 output {  
43     stdout { }  
44     redis {  
45         host => "100.127.255.1"  
46         data_type => "list"  
47         key => "logstash"  
48     }  
49 }
```

Listing 3.2 – Infinite loop

Cet exemple de code ne doit pas être utilisé : c'est une boucle infinie. Il permet en revanche, d'expliquer de nombreux points de fonctionnement de la syntaxe du fichier de configuration de Logstash. Ce code représente, en fait, **2 fichiers**, mail.conf et shipping.conf, tous deux situés dans le répertoire par défaut des fichiers de configuration de Logstash `/etc/logstash/conf.d/`. Ils sont lus par ordre alphabétique et concaténés (comprendre que les inputs ainsi que les outputs sont fusionnés, il faut donc être rigoureux dans leur écriture afin de ne pas créer d'effets de bord indésirables). Une bonne pratique consiste, à utiliser la convention de nommage suivante : *00-description*, *01-description* etc ...

Chacun possède un bloc input, output et l'un d'entre eux : filter. Dans chacun des blocs, on trouve un ou plusieurs **plugins** comme email, file, redis ou d'autres. Chacun de ces plugins possède son propre paramétrage, réglable au travers de directives, mais avec une syntaxe commune.

```
1 directive => int  
2 directive => "string"  
3 directive => ["membre", "de", "l'array"]
```

Listing 3.3 – Syntaxe

La plupart des directives de plugins se comportent ainsi, pas toutes. Nous verrons quelques exemples dans des configurations plus avancées. Dans tout les cas, il est indispensable de se référer à la documentation pour toute première utilisation d'un plugin.

Comme montré dans le code 3.3 il est également possible d'utiliser des structures conditionnelles (les if, else, else if, en faite). De nombreux opérateurs sont également supportés : `==`, `!=`, `<`, `>`, `<=`, `>=`, mais aussi les expressions régulières (syntaxe ruby) `=~`, `~!` et enfin : `in`, `not in`, `and`, `nand`, `or`, `xor` et `!`.

Dans notre exemple de code 3.2, la structure conditionnelle est utilisé pour faire le trie en utilisant les *tags*.

Il est à noter, également, que les directives **path**, disponibles par exemple dans le plugin file, prennent en compte le globbing et le globbing récursif (`le/chemin/*/*log`).

Autopsie d'une la boucle infinie

La configuration 3.2, qui ne fonctionne pas correctement, n'a pas été mise là par hasard. Cette erreur, permet de bien comprendre le fonctionnement des fichiers de configuration de Logstash.

Rappels : Il y a dans cette configuration, 2 fichiers de configuration. Les fichiers, sont concaténés au lancement de Logstash de telle sorte que virtuellement cela donnerait un résultat similaire à :

```
1 #mail.conf
2 input {
3   file {
4     path => ["/home/fdupont/testmail"]
5   }
6   file {
7     path => ["/var/log/secure", "/var/log/messages", "/var/log/*.
8       log", "/var/mail/fdupont"]
9     exclude => ["*.gz"]
10   }
11   file {
12     path => ["/var/mail/ldidry"]
13   }
14 }
15 filter{
16   throttle{
17     before_count => 2
18     after_count => 4
19     period => 120
20     key => "%{message}"
21     add_tag => "contenu"
22   }
23 }
24
25 output {
26   stdout{ }
27   if "contenu" not in [tags]{
28     email{
29       to => "fdupont@localhost"
30       from => "logstash@%{host}"
31       subject => "%{message}"
32       body => "Ceci_est_un_test_sur_l'hote_%{host}\n_avec_pour_
33         message_:_%{message}"
34     }
35   }
36   redis {
37     host => "100.127.255.1"
38     data_type => "list"
39     key => "logstash"
40   }
41 }
```

Listing 3.4 – Infinite loop concaténé

Ici, le problème semble plus évident l'output et l'input sont "liés". Des messages peuvent être détectés et envoyés par mail à l'adresse **fdupont@localhost** (soit l'équivalent de **/var/mail/fdupont**). Ce fichier est également écouté, ainsi que la plupart de fichiers de logs. Voilà la source des boucles infinies, assez facilement reproductible si l'on n'est pas suffisamment rigoureux, et surtout si l'on travail sur plusieurs fichiers à la fois.

L'autre erreur majeure de ce fichier se situe à la ligne 27. Si le terme contenu

n'est pas présent dans "tags" alors on envoie un mail. Le fonctionnement est l'inverse de celui suggérer par throttle. Ici, un mail est envoyé dès qu'un événement se produit dans un fichier de log. Ce mail envoyé dans /var/mail/fdupont sera renvoyé dans cette même adresse. Le contenu de l'événement se modifiera en permanence puisque le message sera encapsulé dans son prédécesseur ...

3.3.2 Expressions régulières et patterns

Les expressions régulières de Grok

Le moteur d'expression régulière de ruby¹³ est très puissant. Nous allons ici présenter les *named groups*, une fonctionnalité assez avancée présente dans ce moteur.

Ces derniers permettent avantageusement de ne pas utiliser les patterns, ou plutôt d'en créer de nouveaux sans modifier les fichiers de Logstash. (Grok est le plugin dont on se servira le plus avec Logstash, c'est pour cela que nous prenons le temps de présenter son système d'expression régulière)

```
1 (?<nom_du_champ>Pattern)
2 (?<username> [a-zA-Z0-9._-]+)
```

Listing 3.5 – Named group

Grok créera rangera automatiquement une chaîne de caractères, satisfaisant la contrainte /[a-zA-Z0-9._-]+/ dans un champ username. Ce champ pourra être réutiliser ultérieurement par Logstash ou même Elasticsearch.

Les patterns de Grok

Logstash¹⁴ met en place un système très utile et très facile à utiliser de patterns (motifs). Ces patterns correspondent à des mots clefs représentant des expressions régulières ou bien des concaténations d'expressions régulières et/ou de patterns.

Pour mieux comprendre en voici quelques exemples.

```
1 CISCOMAC (?:(?:[A-Fa-f0-9]{4}\.){2}[A-Fa-f0-9]{4})
2 WINDOWSMAC (?:(?:[A-Fa-f0-9]{2}-){5}[A-Fa-f0-9]{2})
3 COMMONMAC (?:(?:[A-Fa-f0-9]{2}:){5}[A-Fa-f0-9]{2})
4
5 MAC (?:%{CISCOMAC}|%{WINDOWSMAC}|%{COMMONMAC})
```

Listing 3.6 – Exemple de définition et d'utilisation de Patterns

On comprend bien en lisant le code précédent le fonctionnement des patterns : il est très similaire à celui des *named groups* : définition d'un nom, expression régulière associée.

On les utilise en invoquant leur nom entouré par %

Nous n'avons pas créé de nouveaux patterns pour notre projet et préféré l'utilisation des expressions régulières (named groups). Cela reste cependant possible en modifiant les fichiers paramétrant ces patterns. Ils sont par défaut situés dans /opt/logstash/patterns.

13. Oniguruma, présenté p61

14. certains patterns sont bien utilisables en dehors de grok

3.3.3 Les flags

Les *flags* sont les noms donnés aux paramètres que l'on peut donner au binaire de Logstash.

- `-f` : file, désigne le fichier de configuration à utiliser ;
- `-e` : permet d'utiliser une chaîne de caractères (depuis la console) pour configurer Logstash, à utiliser pour faire des tests de configuration simples ;
- `-w` : filterworkers, comme expliqué plus haut, permet d'affecter plusieurs threads à la gestion des filtres. (par défaut 1) ;
- `-configtest` : associé à `-f path/to/conffile` vérifie la syntaxe du fichier de configuration.

Par défaut Logstash utilisera les fichiers de configuration présents dans `/etc/logstash/conf.d/`, ils seront ouverts par ordre alphabétique.

3.4 Utilisation

Logstash peut s'intégrer dans une architecture simple ou très compliquée. En fonction du cas il peut être utilisé dans plusieurs modes.

3.4.1 La base

Pour fonctionner Logstash doit recevoir des données en input, et les envoyer sur une destination d'output. La configuration la plus simple consiste à lire l'input d'un clavier et à diriger son output sur la sortie standard. Cela a été montré dans une partie précédente. Dans ce cas, une seule machine est impliquée, cela n'a d'utilité que pour une démonstration et éventuellement des tests.

Logstash est parfois utilisé pour centraliser les logs en un seul point en attendant une sauvegarde du fichier nouvellement créé.

```
1 input {
2   file {
3     path => ["/var/log/mail/fdupont", "/var/log/mail/apache", "/
4       var/log/mail/...", ]
5   }
6 }
7 filter {
8   #On peut imaginer faire de l'assainissement/standardisation de
9     logs ici...
10 }
11 output {
12   file {
13     path => ["/var/log/logoftheday-%{+YYYY-MM-dd}.log", ]
14   }
15 }
```

Listing 3.7 – Un autre exemple de configuration minimale

Ce genre de configuration bien qu'un peu exotique peut s'envisager dans le cas de petites infrastructures avec une simple réplication des logs pour la sauvegarde.

Ici on tirera principalement avantage de la capacité de traitement de Logstash grace à son bloc filtre.

Mais le plus souvent on utilise Logstash comme un serveur central, parfois pour d'autres Logstash.

3.4.2 Mon serveur central

Notre configuration sera présentée dans la troisième partie, mais c'est ce genre de montage que nous utilisons pour le projet.

Ici l'idée est de faire pointer les syslogs configurés sur tous les équipements vers un serveur, faisant tourné une instance de Logstash. Cette instance renverra ensuite le flux *utile* vers sa destination, ici un Elasticsearch (après une préalable mise en tampon dans un redis).

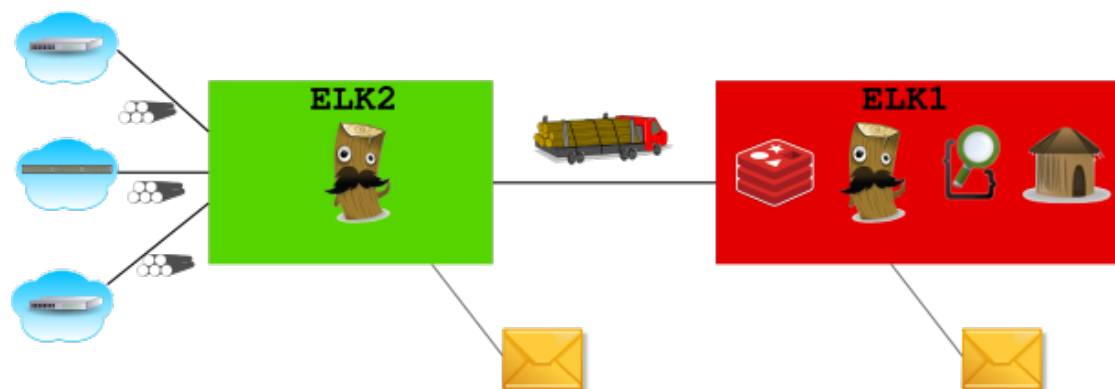


FIGURE 3.1 – Pile ELK

Ici logstash sert donc à trié et centralisé le flux il n'est strictosensus pas indispensable on prend cependant de gros risque à envoyer tous nos logs d'un coup sur une machine dont le but est d'analyser des logs, pas d'encaisser le flux. Ne pas utiliser ce genre d'infrastructure poserait probablement des problèmes I/O, et empecherait la scalabilité à plus long terme. si la machine intermédiaire, le serveur central, ne peut plus encaiser la charge, il suffit d'en déployer un autre et de diviser le flux des syslogs (et autres) entre ces deux serveurs *centraux*.

3.4.3 Monter à l'échelle

Dans de très grosses structures, il est possible qu'il faille des infrastructures encore plus complexes !

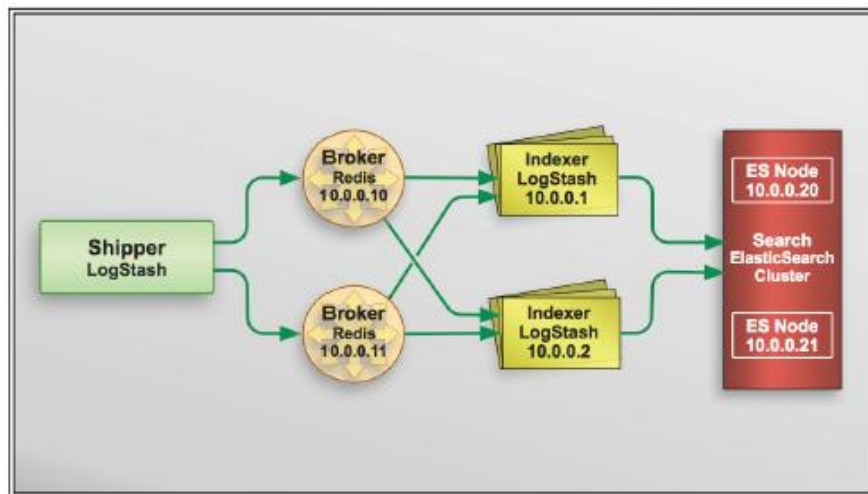


FIGURE 3.2 – Exemple de montage plus complexe

Image de : The Logstash Book

L'idée reste la même que dans l'infrastructure précédente, mais, on est passé en multicouche. C'est à dire qu'on a séparé physiquement chaque partie de l'infrastructure.

Une couche pour le *cluster Elasticsearch*, une couche pour les *tampons Redis*. . . Chaque couche utilise plusieurs instances de leur *logiciel métier*, cette redondance empêche les points de défaillance unique (single point of failure). Elle permet également une plus grande *scalabilité* horizontale.

Autrement dit, si le besoin s'en fait sentir, il est possible avec des changements de configuration minimes de d'ajouter des machines dans nos couches. Il devient possible d'augmenter sans jamais l'éteindre complètement la capacité de traitement de notre infrastructure. C'est la base de la haute disponibilité.

Chapitre 4

Elasticsearch



4.1 Introduction

4.2 Présentation de Elasticsearch

Elasticsearch est moteur de recherche et d'analyse en temps réel. Il permet l'exploration de données de façon très rapide par des recherches, que ce soit par des recherches fulltext, ou bien des recherches structurées.

Elasticsearch est depuis peu développé par la société *Elastic*. C'était, à la base, un projet du développeur Shay Banon, qui souhaitait concevoir une API simplifiée pour la bibliothèque de moteur de recherche : Apache Lucene¹.

Elasticsearch et la stack ELK est en passe de devenir un standard dans l'industrie. En effet le logiciel à une utilisation assez versatile, de l'aide à la prise de décision en temps réelle à l'analyse de code source. Et à l'heure actuelle, la capacité à exploiter les masses de données et de méta données accumulées chaque jour devient un enjeu économique majeur (mais pas seulement ...). Enfin l'un des principaux atouts d'Elasticsearch réside dans son aptitude à pouvoir passer à l'échelle simplement.

Attention, ce chapitre ne sera qu'une présentation très succincte des possibilités du logiciel Elasticsearch, tout détailler nécessiterait sans doute des milliers de pages, le résumé officiel² en fait déjà plus de 950 ...

1. Page du projet Lucene <https://lucenenet.apache.org/>

2. Elasticsearch : The Definitive Guide, disponible chez Pearson et Github

4.3 Installation

Avant même de regarder les différentes dépendances nécessaires à l'utilisation de Elasticsearch il est recommandé de vérifier que l'on utilise bien la même version que son compère Logstash (vérifier la compatibilité dans la documentation si les versions de Logstash et Elasticsearch ont un numéro différent).

Son installation est sensiblement la même que son comparse logstash puisque ce logiciel nécessite également l'installation des dépendances *jruby* et *openjdk-7-jre* à noter qu'il fonctionne également sur *openjdk-8-jre*.

Là aussi les paquets debian officiels n'existant pas on utilisera celui fourni par elastic.co. Et ici aussi le paquet est un peu approximatif puisqu'il faut rajouter certains chemins, et ajouter des droits.

4.3.1 Configuration de base

Acompleter et revoir p24-26 test

4.4 La théorie

La comparaison entre Elasticsearch et une base de données n'est pas forcément heureuse, c'est un moteur de recherche (d'indexation), pas une base de données.

Elasticsearch s'organise autour d'index que l'on peut comparer aux bases (de données) dans le modèle relationnel, un index peut utiliser plusieurs types³, ils s'apparentent aux tables dans le modèle relationnel.

Chaque enregistrement d'Elasticsearch est effectué sous forme document. Les documents sont nécessairement rangés dans un type, éventuellement type par défaut. Le document peut être comparé à une ligne (row).

Ces lignes sont constituées de colonnes, appelées champs (ou fields) dans Elasticsearch.

BDD Relationnelle ⇒ Base de données ⇒ Tables ⇒ Ligne ⇒ Colonnes
Elasticsearch ⇒ Index ⇒ Types ⇒ Documents ⇒ Champs

4.5 L'infrastructure

Dans cette partie nous allons expliquer comment fonctionne Elasticsearch à plus bas niveau. Nous allons parler de noeuds, de shards, d'instances...

Comprendre cette partie est nécessaire pour pouvoir paramétrer une installation simple nous expliquerons brièvement les enjeux pour un cluster plus imposant.

4.5.1 La base

Grappes et nœuds

Chaque instance d'Elasticsearch fonctionne nécessairement dans une entité abstraite appelé cluster (grappe), c'est son groupe. On peut avoir un cluster d'une seule instance. Les clusters sont identifiés par des noms.

Une instance est appelée *node* (nœuds). Il est possible de lancer plusieurs nodes par machines mais c'est déconseillé (cf partie tuning).

3. Dans notre projet **nous n'avons pas fait ce choix** pour pouvoir plus facilement modifier les mappings sans supprimer toutes les données. Il aurait été envisageable par exemple de différencier les logs firewall des logs *states* en utilisant les types. Cf partie 3

Les nodes partagent leurs données et la charge de travail au sein du cluster. Le cluster est capable automatiquement de répartir équitablement ses données entre les nodes, et ce même lors de l'ajout ou la suppression d'un node.

Il existe un/footnotedans de grosses infrastructure il est possible d'en avoir plus pour aider à la gestion du cluster master node qui est élu au sein du cluster. Ce node est chargé de la gestion du cluster ajout ou suppression des des nodes, des index. Si le trafic dans le cluster augmente suffisamment le master node peut n'être dédié qu'à cette tâche.

Shards

On a vu précédemment que les index sont l'endroit où sont stocké les données. Un index est un *namespace logique* qui pointe vers un ou plusieurs shards.

Un shard (éclat) contient une fraction de toutes les données de son index. Dans le cas ou index n'aurait qu'un seul shard (il en a 5 par défaut), ce shard contiendrait l'index entier. Chaque shard est une instance de Apache lucene. Chaque shard est un moteur de recherche complet, mais ne possédant qu'une fraction des données d'un index entier.

Les applications utilisant Elasticsearch (comme Kibana) ne dialogue qu'avec les index la couche d'abstraction au dessus des shards, justement parce que les informations d'un shard sont parcellaire.

Il existe deux types de shards les shards primaires, et les shards répliqués. Les shards répliqués sont des copies de shards primaires, ils servent à accélérer le traitement des requêtes Elasticsearch, ils peuvent également servir de *failover* en cas de défaillance d'une machine.

Le nombre de shards **primaire** attribué à chaque index est fixe et ne peut pas être changé au cours de la vie de l'index. Il est en revanche tout à fait possible d'ajouter ou de supprimer des shards répliqués.

Redondances et montées à l'échelle

Dans cette partie en plus du fonctionnement du cluster et de la répartition des shards nous évoquerons sa santé. Un cluster est considéré en bonne santé lorsque tous ses shards sont répliqués au moins une fois (*failover*).

Nous prendrons ici l'exemple d'un index *blog* possédant 3 shards primaires.⁴ Nous souhaitons que chaque shard primaire soit répliqué une fois (donc trois shards répliqués). Cette figure de style n'est pas innocente, le réglage de ce paramètre⁵ est présenté de la même façon : "number_of_shards" :3, "number_of_replica" :1.

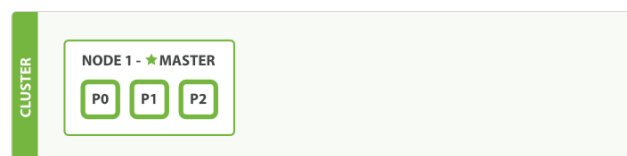


FIGURE 4.1 – Cluster minimal, 1 node, 1 index

La santé de notre serveur est ici considéré comme moyenne, il peut parfaitement fonctionner, mais tous ses shards répliqués ne sont pas actifs. En fait aucun shard répliqué n'est actif. Cela n'aurait pas d'intérêt d'ajouter les shards répliqués sur ce node, car cela n'améliorera pas le traitement des données et ne protégera pas notre cluster contre la perte de données, en cas de défaillance de son unique node.

Nous allons maintenant rajouter un node, sans changer notre configuration.

4. Exemple inspiré et images tirées de *A definitive guide to Elasticsearch*

5. via l'API présenté après la partie sur le tuning

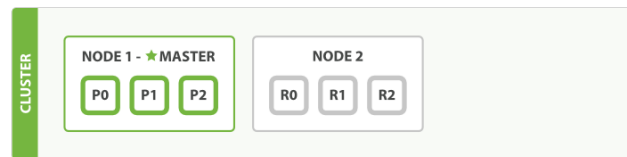


FIGURE 4.2 – Cluster, 2 node, 1 index

On constate que notre cluster est maintenant redondé, le node master possède les shards primaires, le second node les shards répliqués. Ces actions sont effectuées automatiquement à l'ajout du node.

La santé de notre cluster est maintenant considéré comme bonne, il peut supporter une panne. Nous allons maintenant rajouter encore un node et ainsi répartir mieux la charge

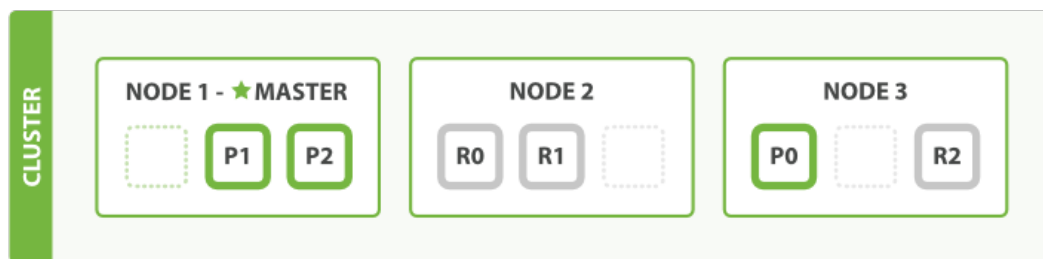


FIGURE 4.3 – Cluster, 3 node, 1 index

Cette configuration fonctionne plus efficacement que la précédente puisqu'elle a accès à plus de ressource et que Elasticsearch répartie intelligemment la ses ressources et donc la charge de travail sur les différents nodes.

Remarque, si l'on perdait un node maintenant, nous nous retrouverions dans la configuration numéro 2 (avec le failover), quitte à devoir réélir un master node. Notre état de santé, serait toujours bon.

Pour voir des choses plus intéressantes, nous allons ajouter un shard répliqué (donc 3).

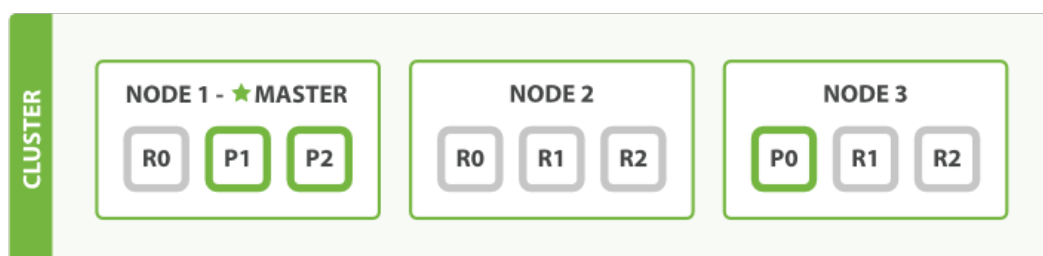


FIGURE 4.4 – Cluster, 3 node, 1 index, mais plus de shards répliqués !

Rien de particulièrement notable.

Mais si nous enlevons le node 1...



FIGURE 4.5 – Cluster, 2 node, 1 index encore... ou pas

Au moment précis de la coupure notre cluster est passé en mauvaise santé, car il n'avait plus de node master, et surtout des shards primaire étaient manquants.

Si une recherche avait été effectuée sur cet index à ce moment précis, des résultats partiels auraient été obtenus, ainsi qu'un avertissement du fait que toutes les données n'étaient pas disponibles.

On constate qu'une élection de master node a eu lieu. En constatant que deux shards primaires sont manquants il a immédiatement promu les shards répliqués au rang de shard primaires.

La santé de notre cluster est maintenant considérée comme moyenne, puisque tous ses shards répliqués ne peuvent pas être affectés.

Cette partie permet de résoudre les problèmes qui pourraient survenir (même sur un cluster d'une seule machine) et permet surtout de mieux comprendre le fonctionnement interne de Elasticsearch.

4.5.2 Le tuning

Voici quelques conseils applicables dans pratiquement toutes les situations et permettant d'optimiser le fonctionnement d'Elasticsearch.

Ne pas utiliser la swap

Il est très fréquent (et souvent nécessaire) de formater une partition de swap pour que le système puisse optimiser son fonctionnement. C'est très pratique pour ne pas saturer la mémoire RAM de nos machines de bureau, notamment quand on voit la consommation pantagruélique de certains navigateurs internet récents. Les fichiers qui ne sont susceptibles de ne pas être utilisés avant un long moment sont parfois stockés sur le disque dur afin de libérer de l'espace pour d'autres applications, la swap est également utilisée lorsque l'on met un ordinateur en veille...

La situation n'est pas la même sur un serveur. Ici nous sommes relativement maître de notre environnement, et de plus nous voulons que le service Elasticsearch soit le plus vélocité possible, pour se faire il faut au contraire l'empêcher au maximum d'utiliser la partition de swap (forcément plus lente).

Il existe plusieurs façons de faire (y compris ne pas utiliser de partition de swap). Celle que j'ai choisie d'utiliser est de modifier la variable **vm.swapiness** dans le fichier **/etc/sysctl.conf**. La **swapiness** représente le pourcentage de mémoire RAM restant à partir duquel on commence à envoyer de l'information en swap.

```
1 vm.swapiness = 2
2 swapoff -a
3 swapon -a
```

Listing 4.1 – Configuration swapiness

Il existe deux méthodes pour que le changement de swapiness soit pris en compte : redémarrer la machine ou bien désactiver puis réactiver la swap, via les commandes `swapon`, `swapoff`. Utiliser la commande `swapoff` est également une mesure radicale à notre problème.

Utiliser une quantité de RAM raisonné

Elasticsearch est limité par Java. Avant de parler de ces limites, il faut rappeler que Elasticsearch est un front-end à Apache Lucene. Lucene est conçu pour tirer parti très efficacement du cache des système de fichiers (probablement ext4 si vous utilisez Debian), qui sont en définitif, gérés par le noyau⁶. Il est conseillé par la documentation de donner au maximum 50% de la mémoire RAM disponible à Elasticsearch, le reste étant dévoué à Lucene et au bon fonctionnement du système. Dans notre installation dont nous parlerons plus en détail ultérieurement, nous avons choisi de nous réserver une bonne marge de manœuvre puisqu'un Redis et un Logstash minimalistes tournent en sus sur la machine hébergeant Elasticsearch.

Cependant cette valeur de 50% n'a de sens que si Elasticsearch consomme moins de 32Go de RAM, en effet, en allouer plus devient contreproductif puisque jusqu'à 32Go la machine virtuelle Java (JVM) *compresse* les adresses des pointeurs, (tant qu'on reste sous la limite des 32Go de RAM on peut continuer à utiliser les adresses mémoires sur 4 octets), après, la consommation de mémoire explose et le *garbage collector* devient bien moins efficace.

Pour les machines possédant une énorme quantité de mémoire RAM (128Go à 1To) il conseille pour optimiser le fonctionnement de la machine de faire tourner plusieurs nœuds d'Elasticsearch dessus (attention aux entrées/sorties et à l'utilisation processeur).

4.5.3 Faire du nettoyage régulièrement

Plus Elasticsearch agrège des données plus il s'empâte, ce phénomène est inévitable sur le long terme. Les contre-mesures sont de *scale-out* (monter à l'échelle horizontalement) en rajoutant de nouveaux nœuds à notre cluster Elastic Search. On s'appliquera à manager les shards de façon optimisée bref à faire du *fine tuning*.

Pour éviter ce ralentissement la solution la plus radicale reste de supprimer les données. Justement, dans le cadre de notre projet, nous n'avons pas besoin de conserver toutes les données sur le long terme. Nous avons donc décidé de supprimer les logs de changement d'état des pare-feu tous les 4 jours. Ces logs peuvent représenter jusqu'à 30Go de données journalières. Pour ce faire nous utiliserons un tâche **cron** couplé à un script bash 10.3 tirant parti de l'API d'Elasticsearch dont nous parlerons plus bas. Pour indication, les index de firewall sont supprimés tous les 4 jours, (une requête judiciaire/alerte sécurité arrive en général en moins de 3 jours).

Les logs sont de toute façon également conservés ailleurs pour se soumettre aux impératifs judiciaires. Les index d'état (intéressants pour établir des statistiques) sont conservés 1 mois.

Remarque sur la commande `curl` et son utilisation dans `cron`

```
1 curl -sS -XDELETE $blade:$port/$target-$d
```

Listing 4.2 – Extrait de notre script 10.3

On remarque ici l'utilisation de l'option `-sS` pour *silent*, *Show errors*. Ainsi `cron` n'est pas importuné par les retours sur la sortie standard cela permet également, en ajoutant la directive **MAILTO : mail@example.fr** de renvoyer la sortie d'erreur vers mail@example.fr. Cela nécessite évidemment d'avoir configuré un serveur mail sur notre machine, par défaut sur Debian, Exim (`dpkg-reconfigure exim-config`).

Concernant `-XDELETE` c'est une utilisation de l'API REST décrite plus bas.

6. https://www.elastic.co/guide/en/elasticsearch/guide/current/_limiting_memory_usage.html

Une des stratégies préconisée dans le cas où l'on souhaite conserver des données sur le long terme alors qu'on en a seulement un besoin d'accès et d'analyse ponctuel ; consiste à les enregistrer dans des fichiers séparés (output Logstash) et de les faire ingérer à un cluster Elasticsearch dimesionné en conséquence le moment venu.

4.6 Les API Elasticsearch

Elasticsearch est énorme, mais peut être assez facilement utilisable par le plus grand nombre grâce aux API *optimisées* pour chaque tâches qui ont été implémentées.

Pour faciliter l'utilisation des différentes APIs il est conseillé d'utiliser **Sense**, cet outil est présenté p61

4.6.1 API REST

Cette API sert à la communication avec Elasticsearch, comme toutes les API REST elle utilise les méthodes HTTP, ici : GET, POST, PUT et DELETE et s'appuie ensuite sur l'architecture d'Elasticsearch.

http `://host :port/[index]/[type]/[_action/id]`

Imaginons que dans un projet fictif, nous souhaitons référencer des tweets. On pourrait procéder comme suit pour ajouter le premier.

PUT **http** `://100.127.255.1 :9200/twitter/tweet/1`

```
1 curl -XPUT http://100.127.255.1:9200/twitter/tweet/1
```

Listing 4.3 – Avec curl

```
1 PUT /twitter/tweet/1
```

Listing 4.4 – Avec Sense

Voici la syntaxe que nous obtiendrions en utilisant Sense, puisque notre serveur et notre port sont déjà renseignés. Nous utiliserons par convention cette syntaxe plus courte.

La directive **PUT** est utilisé pour renseigner de nouvelles informations dans Elasticsearch.

Dans notre projet d'analyse de logs, il est fréquent que nous dussions supprimer des index, notamment dans le cas du paramétrage des *mappings*, mais également pour soulager notre infrastructure du poids des logs de firewall (20-30Go/jours).

DELETE **http** `://100.127.255.1 :9200/twitter/`

Si nous avons un index twitter journalier il serait possible d'utiliser une commande très similaire pour supprimer tous les index d'un seul coup :

DELETE **http** `://100.127.255.1 :9200/twitter*`

La directive **DELETE** est utilisé pour supprimer des informations d'Elasticsearch.

La directive **GET** sert à récupérer des information à tout niveaux : index, type, mapping... Il faut par contre avoir l'identifiant exact de la ressource à laquelle on souhaite accéder pour utiliser cette directive. Ce n'est pas un outils de recherche⁷.

7. Outils de recherche abordé dans la section suivante

GET <http://100.127.255.1:9200/twitter/tweet/1>

Les réponses d'Elasticsearch sont en **JSON**.

4.6.2 Les API de recherche

Il existe plusieurs API (ou méthodes) de recherche dans Elasticsearch nous les détaillerons plus en détail l'API searchLite après avoir expliqué le fonctionnement général d'une recherche dans Elasticsearch.

La recherche dans Elasticsearch est particulièrement efficace, car Elasticsearch indexe tout le contenu de ses documents (il indexe chaque champ, en fonction d'un mapping particulier⁸).

Empty Search

C'est la forme la plus basique de l'API de recherche, on ne spécifie pas de requête spécifique. Comprendre le fonctionnement et les possibilités de cette API basique permettra d'utiliser plus efficacement SearchLite, qui en est l'évolution directe.

```
1 GET /_search
```

Listing 4.5 – le "Hello World" de la recherche

Cette requête retourne tous les documents, de tous les index du cluster duquel nous sommes membre.

```
1 {
2   "took": 798,
3   "timed_out": false,
4   "_shards": {
5     "total": 201,
6     "successful": 201,
7     "failed": 0
8   },
9   "hits": {
10    "total": 353585048,
11    "max_score": 1,
12    "hits": [
13      {
14        "_index": ".kibana",
15        "_type": "visualization",
16        "_id": "Top5-nat-firewall-router",
17        "_score": 1,
18        "_source": {
19          "title": "Top5 nat-firewall router",
20          "visState": "{\"type\":\"histogram\",\"params\":{\"shareYAxis\":true,\"addTooltip\":true,\"addLegend\":true,\"mode\":\"stacked\",\"defaultYExtents\":false},\"aggs\":[{\"id\":\"1\",\"type\":\"count\",\"schema\":\"metric\",\"params\":{}},{\"id\":\"2\",\"type\":\"terms\",\"schema\":\"group\",\"params\":{\"field\":\"IPnat\",\"size\":5,\"order\":\"desc\",\"orderBy\":\"1\"}}],\"listeners\":{}}",
21          "description": "",
22          "version": 1,
23          "kibanaSavedObjectMeta": {
24            "searchSourceJSON": "{\"index\":\"firewall-*\",\"query\":{\"query_string\":{\"query\":\"*\"},\"analyze_wildcard\":true}},\"filter\":[]}"
25        }
26      }
27    ]
28  }
29 }
```

8. explication sur le mapping p31

```

26         },
27         .... 9 Autres ...
28     }
29 }
30 }

```

Listing 4.6 – Réponse type à notre requête précédente

Quelques explications :

hits L'élément `hits` contient des informations sur la requête et les réponse à la requête. *total* représente le nombre total de documents retournés pour cette requête, ici le nombre total de documents indexés : 353585048. Cela signifie qu'il y avait environ 353 millions de lignes de logs indexés dans notre Elasticsearch au moment de la requête.

Les réponses à la requête nous sont renvoyées dans l'array *hits*, par défaut seul les 10 premiers documents satisfaisant notre requête nous sont renvoyés (comme il n'y avait pas de requête, simplement les 10 premiers documents).

Les documents renvoyés dans l'array sont classés en fonction de leur `_score`, une fois encore, puisque nous n'avons rien précisé dans notre requête, tous les résultats nous sont renvoyés avec le score 1.

On notera que le **max_score**, score maximum obtenu lors de la requête.

Enfin on remarque que le source (tous les champs indexés) de chaque document est présent dans la réponse à la requête.

took Took est le temps en millisecondes pris pour effectué cette requête, ici 798ms

shard L'élément `_shards`, nous indique le nombre de shards utilisé lors de la requête. Dans combien de shards tout c'est bien passé, dans combien il y a eu des erreurs. Les erreurs sont très peu probables, cela se produit généralement sur un cluster de plusieurs machines, alors que plusieurs d'entre elles sont inaccessibles (il faut que les primary et replica d'un shard soient inaccessible en même temps). Cela peut aussi produire sur un serveur surchargé (les temps d'accès trop long feront croire à Elasticsearch que le shard est inaccessible).

timeout Il est possible d'effectuer une recherche *par timeout*, cela signifie que Elasticsearch va effectuer sa recherche de façon classique, mais que lorsque le temps autorisé pour la recherche est dépassé, il renverra les informations qu'il a eu le temps de rassembler. Toujours dans l'ordre qui lui semble le plus pertinent (ordonné par score). Attention cependant, cela ne nous absouds pas pour autant du coût de la recherche. Si les shards renvoient bien les résultats pertinents au moment souhaité, cela n'arrête pas pour autant la requête qui finira de s'exécuter en arrière plan même si les résultats ne sont pas pris en compte.

```
1 GET /_search?timeout=10ms
```

Listing 4.7 – Une recherche "vide" avec timeout

Recherche multi-index et multitype La recherche multi-index et multitype est toujours possible comme dans l'API REST

```

1 GET /gb,us/_search
2 GET /g*,u*/_search
3 GET /gb,us/user,tweet/_search

```

Listing 4.8 – Une recherche multi-index ...

Pagination Par défaut lors d'une requête Elasticsearch renvoie seulement les 10^{ers} résultats. Il est bien sûr possible de paramétrer cela, pour afficher plus de résultats, et pas forcément les plus pertinents (ceux qui présentent le score le plus élevé).

size permet d'indiquer le nombre de résultats souhaités

from permet d'indiquer le nombre de résultats que l'on souhaite sauter.

```
1 GET /_search?size=5&from=10
```

Listing 4.9 – Pagination

Search Lite

Comme expliqué avant il existe deux formes d'API de recherche. L'API lite avec requête en chaîne de caractères (*string query*) et l'API "full body request", qui nécessite comme corps de la requête, du JSON⁹.

Utiliser la recherche SearchLite, nécessite simplement d'ajouter à notre empty search `?q=` puis la requête. *Attention, si vous utilisez cette API avec curl*, il pourrait être nécessaire d'utiliser la syntaxe `http (%2B pour + par exemple)`.

```
1 GET /_search?q=epinal
2 GET /state-2015.05.22/_search?q=epinal
```

Listing 4.10 – Exemples simples

Ici on cherche le terme *epinal* dans tous les champs, c'est parfois utile, mais dans de nombreux cas nous connaissons déjà le nom du champ (cf Logstash) dans lequel nous souhaitons avoir le terme.

Pour se faire il suffit d'utiliser la syntaxe suivante :

```
1 GET /_search?q=logsource:epinal
```

Listing 4.11 – Choix du champ

Bien plus intéressant, il est possible d'imposer des conditions à remplir : doit contenir le terme suivant, ou au contraire ne doit **pas** contenir le terme suivant.

```
1 GET /_search?q=+logsource:*epinal* -logsource:sw*
```

Listing 4.12 – Conditions must (not) match

Dans l'exemple précédant on souhaite chercher les équipements réseaux situés à epinal mais pas les switches. L'utilisation des `+` et `-` change vraiment le sens de la requête, dans la précédente sans les symboles, la présence ou non augmentait la pertinence des réponses contenant le terme recherché, mais n'était pas discriminante. Avec ces symboles, on exclut les réponses qui ne se conforment pas à nos exigences.

La présence des `*` est une sorte de globbing, nécessaire lorsque l'on fait une recherche fulltext

Il existe un autre moyen d'influencer la pertinence des résultats (de faire remonter en haut de liste ce qui nous intéresse le plus). Cela est particulièrement pratique dans les longues requêtes. Il s'agit de l'opérateur `^` qu'on n'utilise en général en conjonction de parenthèses (qui servent simplement à faire des groupes).

```
1 GET /_search?q=+logsource:*epinal* -logsource:sw* (timestamp:May)^10
   Power
```

Listing 4.13 – Modifier la pertinence

9. Voir la documentation

Dans cette requête nous cherchons toujours des équipements non switch situés à Epinal. Nous sommes cette fois ci très intéressés par ce qui s'est passé en Mai et d'autant plus si cela concerne un problème d'alimentation électrique.

Concernant la gestion du temps, et des timestamp, nous allons voir une propriété intéressante d'Elasticsearch lorsque le mapping de l'index est bien réalisé. Il est capable de *s'orienter* dans le temps à partir d'un champ texte. Cela n'est pas forcément très utile si l'on utilise Kibana, puisque son interface nous permet très facilement d'être plus précis, mais cela laisse imaginer les possibilités d'Elasticsearch.

```
1 GET /_search?q=+logsource:*epinal* -logsource:sw* (timestamp:>Jun)^10
   Power
2 GET /_search?q=+logsource:*epinal* -logsource:sw* (timestamp:=<Jun)
   ^10 Power
```

Listing 4.14 – Le temps dans SearchLite

SearchLite accepte les opérateur <>=<=> et Elasticsearch est capable de les interpréter pour peu que le champ soit *mappé comme date*(cf Mapping).

Il est possible d'utiliser les opérateurs logiques **OR** et **AND** dont l'utilisation est assez évidente

...

```
1 +IPoutside:"8.8.8.8" +IPnat:"8.8.4.4" +(portnat:"37657" OR portnat
   : "19474")
```

Listing 4.15 – Opérateurs logiques

Il faut faire attention à l'utilisation des symboles -, :, /," qui on une signification.

Remarque générale, les syntaxes présentées jusqu'à présent son utilisables telles quelles dans Sense, et en retirant **GET /_search ?q=** dans Kibana.

4.7 Le mapping et l'analyse

4.7.1 Mapping

Le mapping est **comparable au schéma d'une base, dans le modèle relationnel**. Le mapping définit donc le "type" de nos "champs". Attention on fait rentrer dans le mapping plus, que dans le schéma d'une base (analyse, indexation, ...).

Il y'a un mapping différent par index (il existe aussi des templates, applicable à plusieurs index). Pour accéder au mapping d'un index il suffit dans Sense de lancer la requête suivante.

```
1 GET /firewall-2015.05.07/_mapping
```

Listing 4.16 – Obtenir un mapping

Voir un **exemple de mapping** : p55

On remarque que beaucoup de champs ont pour type **string**, c'est le type choisi par défaut.

Bien qu'ils ne soient pas utilisés ici il existe d'autres types de base, il est aussi possible d'en définir soi-même, cela ne sera pas abordé dans ce rapport.

Types par défaut :

- boolean : true/false
- long : nombres entiers
- double : nombres à virgule
- string : chaîne de caractères
- date : une date valide, (voir la doc pour les normes acceptées)

Pour créer un nouveau mapping, il faut au préalable supprimer le précédent, (supprimer l'index auquel est attaché le mapping).


```
1 DELETE /firewall-2015.05.07
2 PUT /firewall-2015.05.07/
3 {
4   "mappings": {
5     ....
6     "properties": {
7       "@timestamp": {
8         "type": "date",
9         "format": "dateOptionalTime"
10      },
11      "@version": {
12        "type": "string",
13        "index": "not_analyzed"
14      },
15      "IPinside": {
16        "type": "ip",
17        "index": "analyzed",
18        "store": "yes"
19      },
20      "IPnat": {
21        "type": "string",
22        "index": "not_analyzed",
23        "norms": {
24          "enabled": false
25        },
26      }
27      ....
28    }
29  }
30 }
```

Listing 4.17 – Changer le mapping d'un index

La requête entière est trop longue pour être affichée, donne une idée de la forme. Une méthode efficace pour générer la sienne est d'utiliser le mapping créé par Elasticsearch par défaut puis modeler notre requête à notre convenance à partir de ce squelette.

Cette méthode est pratique pour faire des tests mais elle ne concerne qu'un seul index. Dans le projet nous travaillons avec de multiples index, il sera plus pratique d'utiliser les templates pour que le même modèle s'applique partout où nous le souhaitons.

Autre remarque, **si vous faites la manipulation ci-dessus dans un environnement de production, il y a toutes les chances pour qu'Elasticsearch vous renvoie une erreur**. Par défaut Logstash crée de nouveaux index si rien n'existe pour accueillir les logs qu'il envoie à Elasticsearch. Ainsi, si on supprime l'index du jour (on perd les données), mais un autre index du même nom sera recréé avec le template par défaut presque instantanément puisque Logstash reçoit (dans notre cas) environ **500logs/s**. Pendant les tests il peut être judicieux d'éteindre Logstash.

Un template sera ici le mapping par défaut utilisé par une certaine forme d'index, une sorte de modèle. Pour créer un template, il faut simplement supprimer l'ancien et écrire le nouveau. Il est également conseillé de supprimer les anciens index utilisant un mapping différent. Ils seront **toujours utilisables dans Sense**, mais seront **incohérents aux yeux de Kibana** et empêcheront de faire des recherches¹⁰.

Voir un exemple de **création de template** p55

Du point de vue mapping on peut noter que integer est également un type valable (liste exhaustive dans la documentation), on constate aussi que le type date (un type particulier de string), qui **ne**

10. Faire des recherches avec des mappings différents, c'est comme faire des opérations dans des bases différentes

peut pas être analysé, supporte plusieurs formats, définis d'après la norme JAVA ¹¹. Avec quelques types prédéfinis ici ¹².

4.7.2 Analyse

On remarque que certains champs ont également un **index** : "**not_analyzed**". Tous n'en n'ont pas car il est souvent plus intéressant d'avoir des champs analysés. Et par défaut tous les champs sont analysés.

Il existe trois niveaux d'indexation :

- **Analyzed**, signifie que la chaîne de caractère est analysé puis indexé, donc cherchable en *full text*.
- **not_analyzed**, le champ est toujours cherchable, mais puisqu'il n'a pas été analysé, on ne peut le trouver qu'avec sa valeur exact. *Cette propriété peut se révéler essentielle pour réaliser des graphiques efficaces*. Il faut parfois savoir comment on veut exploiter ses données pour pouvoir les traiter en conséquence.
- **no**, n'index pas ce champ, il n'est pas cherchable. Je ne me suis jamais servi de cette propriété, peut sans doute avoir une utilité pour l'optimisation de performances.

Fonctionnement de l'analyse

L'analyse consiste en deux grandes étapes, découper une chaîne de caractère en termes recherchable, puis assainissement de ces termes en enlevant par exemple les majuscules ou en retirant les termes non pertinents (le, la, les, des ...), voir même en remplaçant des synonymes.

En fait le processus compte trois étapes. Tout d'abord le *filtrage de caractères* (faire disparaître les / ou transformer & en et. La tokenization (*termisation* en traduction approximative), créer un terme pour chaque chaîne de caractère séparer par un espace, une virgule, un point, ... (paramétrable cf doc). Enfin le filtre de termes, (décrit dans l'assainissement au dessus).

Le fonctionnement des **recherches full text** est expliqué plus en détail p63

4.8 Conclusion

Nous savons maintenant ranger nos données de façon efficace dans Elasticsearch cela va nous permettre de créer des index faciles à analyser et à *requêter* avec Kibana. Cette partie est probablement la plus complexe et la plus importante mais aussi la moins user-friendly. Construire un mapping n'est pas facile et nécessite de bien comprendre anticiper l'exploitation que l'on souhaite faire de ses données.

11. <http://joda-time.sourceforge.net/api-release/org/joda/time/format/DateTimeFormat.html>

12. <https://www.elastic.co/guide/en/elasticsearch/reference/1.5/mapping-date-format.html>

Chapitre 5

Kibana



5.1 Qu'est ce que Kibana

Kibana est à la fois un front-end à Elasticsearch permettant de présenter les données.

C'est l'outil de visualisation officiel d'Elasticsearch, il est assez ergonomique et donc plutôt facile d'utilisation. Sa barre de recherche utilise la syntaxe search Lite présenté plus haut dans le chapitre traitant d'Elasticsearch¹. Son utilisation pour réaliser des tableaux, dashboard (tableaux de bord) et autre camemberts est assez aisé une fois les principes de bases assimilés.

5.2 Installation de Kibana

L'installation de Kibana est relativement simple, il suffit de télécharger l'application compressé à l'adresse sur le site de <https://download.elastic.co/kibana/kibana/kibana-4.0.2-linux-x64.tar.gz>. On décompresse, on lance (**path/bin/kibana**), et c'est parti ! Évidemment, si on exécute Kibana d'une telle manière il est préférable de le lancer depuis un tmux ou un screen (ou un quelconque multiplexeur de terminal). Il est également facile de réaliser son propre service systemd.

5.2.1 Paramétrage

Afin de pouvoir se connecter à Elasticsearch il faut tout de même renseigner **path/config/kibana.yml** notamment : *port* (par défaut 5601), *host* (ip d'écoute) et surtout *elasticsearch_url*. Dans des circonstances normales c'est tout ce que vous aurez à paramétrer (hors SSL ...).

Il peut arriver dans de rares cas (lorsque Elasticsearch est saturé ou bien lors d'une requête particulièrement gourmande) que Kibana se ferme car Elasticsearch met trop de temps à répondre. Il est dans ces cas là, il est fortement conseillé de redémarrer l'instance en question, d'étendre sa *HEAP_SIZE*, ou bien si possible, rajouter de la RAM.

1. voir p30

Il est possible de modifier le `requestTimeout` dans `path/src/lib/waitForEs.js`. C'est pour l'instant la seule façon de faire, d'après une issue github, la prochaine mouture de Kibana devrait intégrer ce paramètre dans son fichier de configuration.

5.3 Utilisation de Kibana

Dans cette partie nous allons faire une brève présentation (illustré) de Kibana et de son fonctionnement.

5.3.1 Recherche

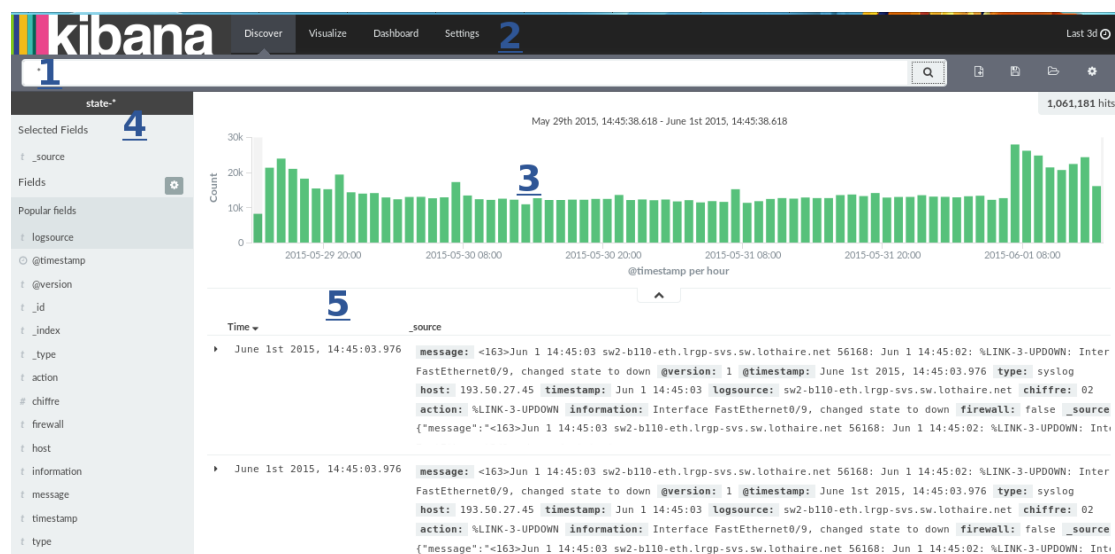


FIGURE 5.1 – Présentation générale de Kibana

Voici la page principale sur laquelle on arrive lorsque l'on se connecte à Kibana depuis son navigateur.

1. La barre de recherche (syntaxe SearchLite)
2. Barre de sélection des *modes* (Discover : Recherche, Visualize : Création de Tableaux et autre représentations graphiques, Dashboard : Rassemblement de ces présentations, Settings : Paramétrage de certaines options)
3. Tableau graphique des résultats de recherche
4. Tableau des champs de l'index
5. Tableaux des lignes correspondant à la recherche

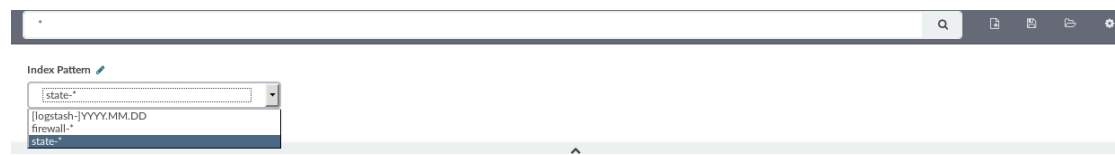


FIGURE 5.2 – Choisir son index pour une recherche

Pour effectuer une recherche, on doit choisir dans quels index, si il est possible de réaliser une recherche dans plusieurs index à la fois, il faut dans Kibana que ce soit des index disposant du même mapping. Typiquement rangés par **nom-***. Pour choisir son index il faut *sélectionner la roue dentée à droite de la barre de recherche*.

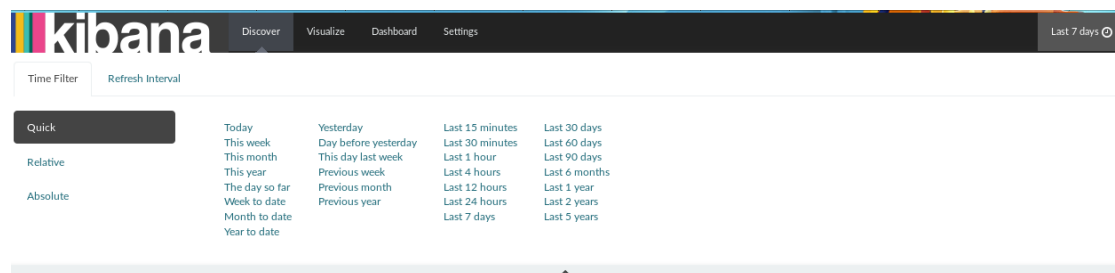


FIGURE 5.3 – Choisir son intervalle de temps

Lorsque l'on effectue une recherche avec Kibana il est possible de choisir de façon très flexible son intervalle de temps (bouton de temps en haut à droite). Il est tout d'abord possible de choisir les possibilités du menu rapide, affichées dans l'image ci-dessus. Le choix est déjà assez exhaustif, mais il est également possible de se positionner relativement par rapport à maintenant. Par exemple rechercher parmi les 30 dernières secondes, ou bien les deux derniers mois (de la seconde à l'année). Il est également possible de choisir un intervalle de temps absolu, à la seconde près.

Enfin il est possible de rafraîchir ces données en choisissant un intervalle allant de 5 secondes à une journée (également désactivable).

Voir tableau des champs p50

Le tableau des champs disponibles dans l'index est très pratique pour rendre plus lisible les informations que l'on recherche ainsi que pour obtenir des statistiques rapides sur les occurrences des termes les plus communs.

Ce tableau contient tous les champs disponibles dans l'index. Il est possible de faire de la ségrégation dans les résultats de nos recherches en incluant ou en excluant un ou plusieurs termes proposés (*on peut par exemple exclure le résultat le plus courant car on sait un switch défectueux, afin de se concentrer sur les autres pannes pas encore détectées et donc plus pertinentes*). Il suffit pour cela d'appuyer sur les loupes situées à droite des termes suggérés.

Il est également possible de choisir de ne montrer que la ou les parties que l'on estime pertinente dans le log (en appuyant sur le bouton add à droite de chaque champ) et bien sure de combiner les deux dernières "techniques".



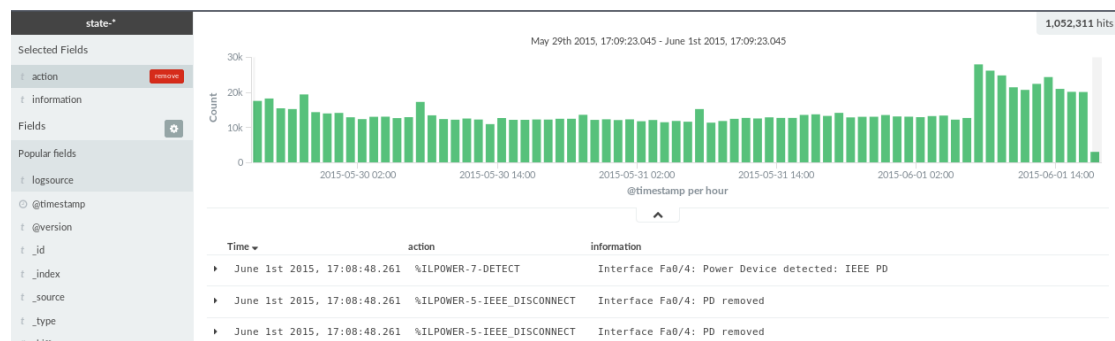


FIGURE 5.4 – Résultat

5.3.2 Visualization

Nous allons maintenant parler de la visualisation (en français) des données, il s'agit là encore que de donner un très bref aperçu des possibilités.

Voir la page de garde des visualisations p51

Voilà les différents types de visualisations parmi lesquelles nous pouvons faire notre choix. Toute ne conviennent pas, le meilleur choix est affaire d'expérience.

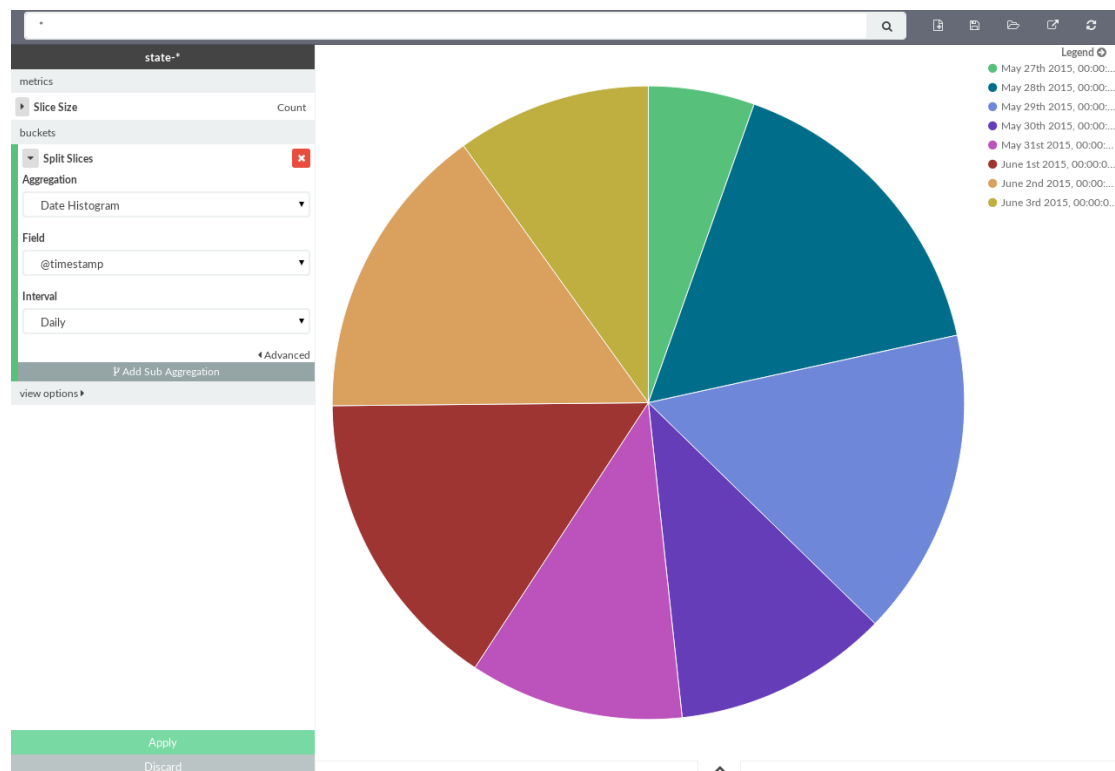


FIGURE 5.5 – Camembert de répartition des logs sur la dernière semaine

Voici un camembert très simple, réalisable en moins d'une minute. Il agrège les logs sur une semaine en les classant par date d'émission, groupés par jour. Il est évidemment possible de faire des choses plus compliquées avec plus ou moins données/de traitement.

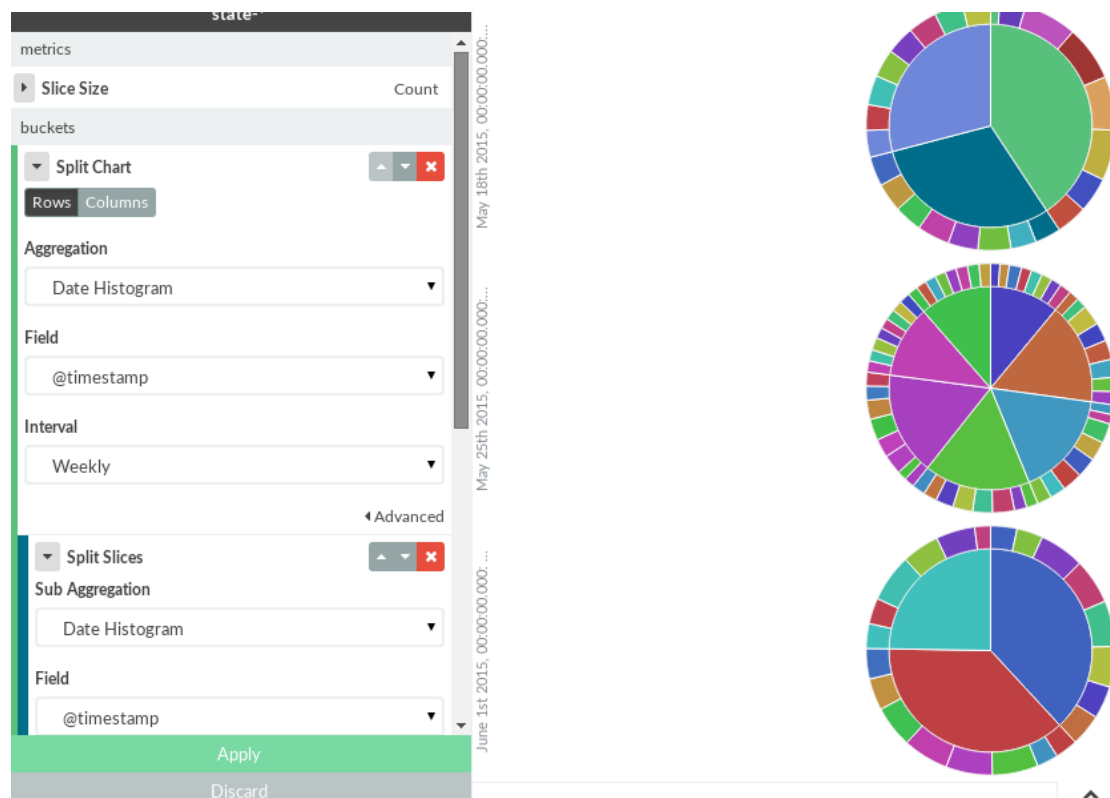


FIGURE 5.6 – Camembert de répartition des logs sur sur 3 semaines

Il s'agit grosso modo des mêmes informations mais cette fois ci groupé par semaine par camembert, par jour dans le camembert interne, et par tranche de 3h sur l'anneau externe. Ces camemberts sont interactifs, on le verra dans la partie suivante sur les dashboards.

5.3.3 Dashboard

Présentons maintenant les dashboards. Sorte de tableaux virtuels, ils servent à rassembler les informations. Ces informations ont normalement plus de sens une fois mises cote à cote ou permette une analyse plus rapide d'une situation donnée.

La création de ces dashboards est également très simple, on en créer un nouveau en appuyant sur le plus à droite.

Il suffit ensuite de choisir les visualisation et ou les recherches enregistrées que l'on souhaite voir affiché.

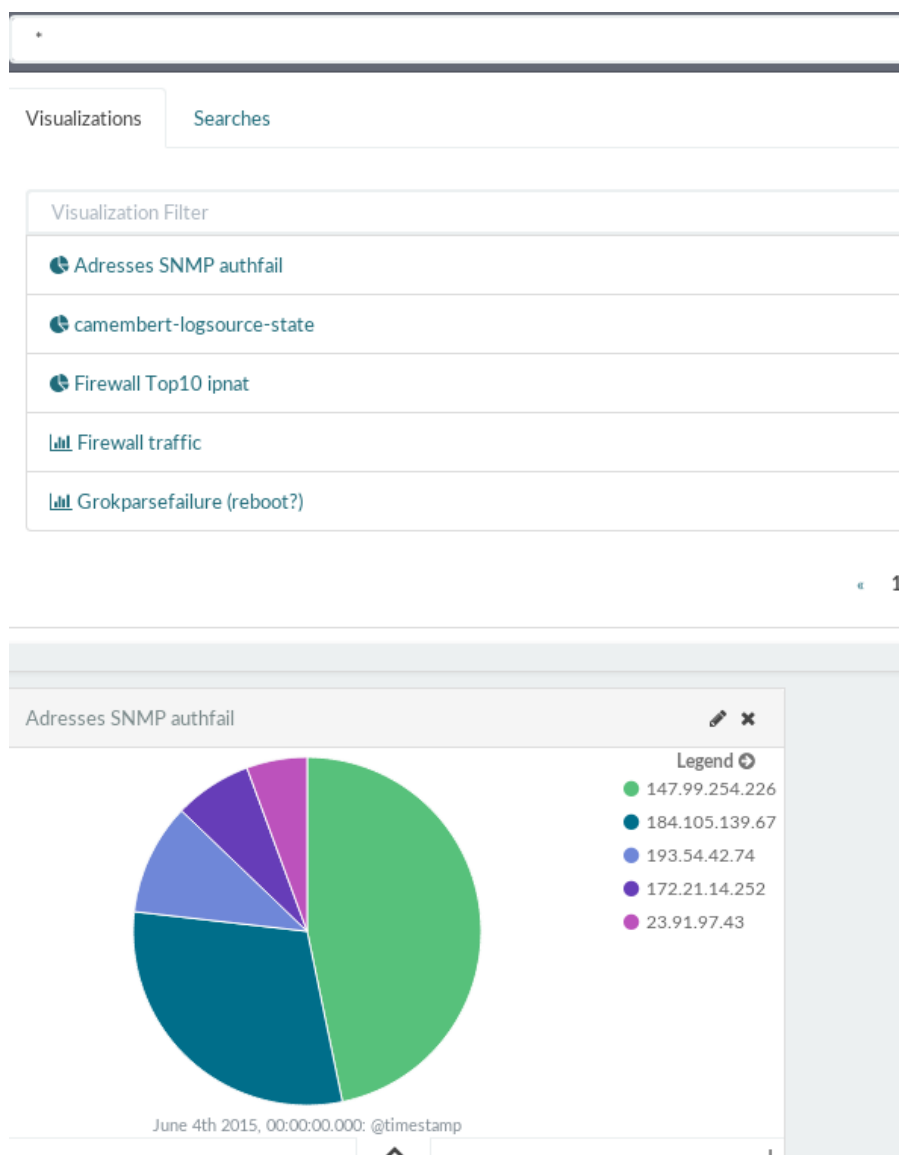


FIGURE 5.7 – Choix des visualisations

Comme expliqué plus haut les visualisations ainsi que les tableaux sont interactifs. Il est par exemple possible de façon assez intuitive faire une recherche limitée ségréguée. Ici il suffit de cliquer sur le quartier désiré de la visualisation. Cela aura pour effet de faire apparaître la barre verte (contextuelle) et de ne faire apparaître dans le tableau de résultat à droite que les logs correspondants. Ici les logs ayant pour ip source 147.99.x.x affiché sur l'image ci-dessous.

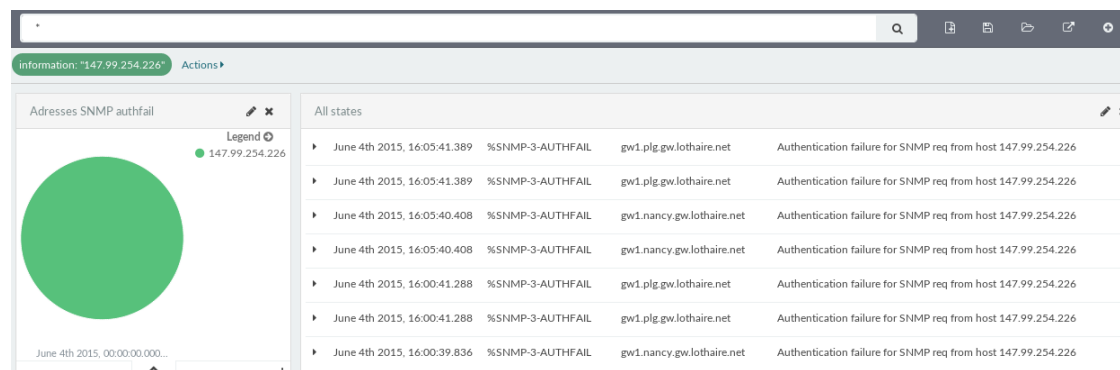


FIGURE 5.8 – Dashboard simple

Il est possible de partager facilement ces dashboard, via des liens avec iframes. On peut ainsi facilement les ajouter dans une page de monitoring par exemple.

Voir l'image d'un Dashboard plus avancé p52

Partie non essentielle sur les *settings* p64

Troisième partie

Synthèse et conclusion

Chapitre 6

Synthèse

6.1 Analyse des performances

Globalement en fin de stage, l'infrastructure ELK fonctionne comme demandé. Cependant, la marge de manœuvre n'est pas très importante.

ELK2 fonctionne très bien, nous pourrions même ajouter plus de trafic et/ou faire un tri plus drastique (consommateur en CPU) sur les logs. La ressource la plus exploitée est le réseau mais là aussi il y a une **très** grande marge.¹ Voir les graphiques Munin associés : p53

Ça va un peu moins bien du côté ELK1. Elasticsearch est un logiciel, gourmand, particulièrement pour de gros volumes de données.

Les 12Go de RAM ne sont pas de trop. 8 sont en permanence utilisés avec des variations importantes, notamment en fonction de l'utilisation de Redis. Voir p54

Le disque dur malgré, ses 15000tr/min est également un goulot d'étranglement. Lors d'une grosse recherche avec Kibana, les temps d'accès sont souvent la cause de ralentissements. Ces effets étant très ponctuels Munin ne les détecte pas.

On pourrait penser en voyant le graphique processeur p54, que l'on dispose d'une très large marge, mais il s'agit d'un effet d'optique. Là aussi Munin n'est pas assez précis pour capter les variations ponctuelles, lors d'une recherche, il est fréquent que les 12 cœurs soient entièrement utilisés. On voit un tout petit peu l'utilisation augmenter vers la fin, lors d'un stress test.

Attention, l'infrastructure n'est pas pour autant au bord du gouffre, elle est juste bien dimensionnée pour pouvoir fonctionner avec quelques utilisateurs concurrents et quelques dizaines de millions de logs envoyés par jours, comme c'est le cas actuellement.

6.2 Améliorations

6.2.1 Sécurité et déploiement

Je n'ai pas particulièrement mis en place de dispositifs pour sécuriser l'accès à l'infrastructure. Les machines sont à jour et isolées sur un VLAN.

Mais il est possible de faire mieux au niveau des applications comme par exemple ajouter un proxy SSL devant Kibana.

Il est possible de communiquer de syslog-ng vers Logstash par SSL.

1. carte réseau Gigabit

Enfin concernant la sécurisation de Redis un bon début est de commencer par écouter une seule IP et non pas 0.0.0.0 (toutes).

Concernant Elasticsearch pas de solutions possibles² il faut l'isoler au maximum du reste du réseau.

L'installation n'a pas été automatisé, il faudra le faire, avec Saltstack, logiciel utilisé par l'équipe Lothaire.

6.2.2 Tuning Elasticsearch

Les possibilités d'optimisations avec Elasticsearch sont nombreuses tout n'a pas été étudié. Voici une liste non exhaustive des pistes envisageables : Limiter le nombre de shards (ce qui pourrait augmenter les performances, notamment les accès), diminuer le nombre de champs et/ou d'informations enregistrées afin de diminuer la taille de nos index.

On peut sans doute améliorer le mapping actuel pour mieux le faire correspondre aux besoins de l'équipe Lothaire (analyseurs particuliers, ajouter des champs non analysés,etc)

2. la société semble avoir récemment développer le produit payant : Shield, qui apporterait une couche de sécurité

Chapitre 7

Conclusion

Ce stage a été une très bonne expérience. L'équipe, accueillante et disponible, m'a permis de bien m'intégrer dès le début du stage.

Les limites et les objectifs du projet avaient été bien fixés par mon maitre de stage, Vincent DELOVE, ce qui m'a empêché de me disperser. Un écueil facile, lorsque l'on travail avec un outil aussi versatile qu'Elasticsearch.

L'objectif de fournir *un outil facilitant l'exploration de données* dans les logs des équipements réseau de Lorraine a été rempli. Plusieurs membres de l'équipe s'en sont déjà servi avec succès. Il a également permis de mettre en évidence des défaillances sur des équipements qui avait été jusqu'alors sous-estimées.

Réaliser ce projet m'a permis de mettre en pratique de nombreuses connaissances vues en licence ASRALL, d'approfondir des notions de système, parfois assez pointues, comme les capacités. Mais aussi d'en apprendre plus sur un outil dont nous avons juste gratter la surface en cours : les expressions régulières. Ce stage m'a surtout permis d'appréhender et de mettre en place deux outils que je ne connaissais pas Elasticsearch et Logstash. Ces logiciels sont en ce moment à la mode et ils semblent, au moins concernant Elasticsearch, être partis pour rester. C'est donc pour moi une chance d'avoir pu acquérir de l'expérience sur ces technologies, me seront utile dans mon prochain emploi.

Quatrième partie

Annexes

Chapitre 8

Sources/Webographie/Bibliographie

Web

Introduction généraliste à ELK

<https://wooster.checkmy.ws/2014/04/elk-elasticsearch-logstash-kibana/>

Explications sur les moteurs d'indexation/elasticsearch

<https://zestedesavoir.com/articles/120/elasticsearch-maintenant-en-version-14/>

Articles sur les performances des SGBD vs moteurs d'indexation

<http://jolicode.com/blog/elasticsearch-comme-nouveau-standard-de-la-recherche-applicative>

Documentation de logstash générale

<http://logstash.net/docs/1.4.2/>

<http://logstash.net/docs/1.4.2/flags>

Installation de logstash

<http://www.elastic.co/guide/en/elasticsearch/reference/1.4/setup-repositories.html>

<https://github.com/elastic/logstash>

Debugger pour le filtre grok

<http://grokdebug.herokuapp.com/>

Documentation d'Elasticsearch

<http://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>

<http://www.elastic.co/guide/en/elasticsearch/guide/current/index.html>

<http://www.elastic.co/guide/en/elasticsearch/reference/1.x/glossary.html>

Clustering Elasticsearch

<http://blog.trifork.com/2014/01/07/elasticsearch-how-many-shards/>

<http://stackoverflow.com/questions/12409438/when-do-you-start-additional-elasticsearch-nodes>

Documentation de Kibana

<http://www.elastic.co/guide/en/kibana/current/>

<https://github.com/elastic/kibana/tree/master/docs>

Livres

The logstash book

<http://www.logstashbook.com/>

Elasticsearch : The Definitive Guide

<https://github.com/elastic/elasticsearch-definitive-guide>

Vidéos

Conférence de Jean Baptiste Favre, architecte réseau blablacar, PSES 2014

<http://numaparis.ubicast.tv/videos/kibana/>

Chapitre 9

Images

9.1 Cartes Lothaire

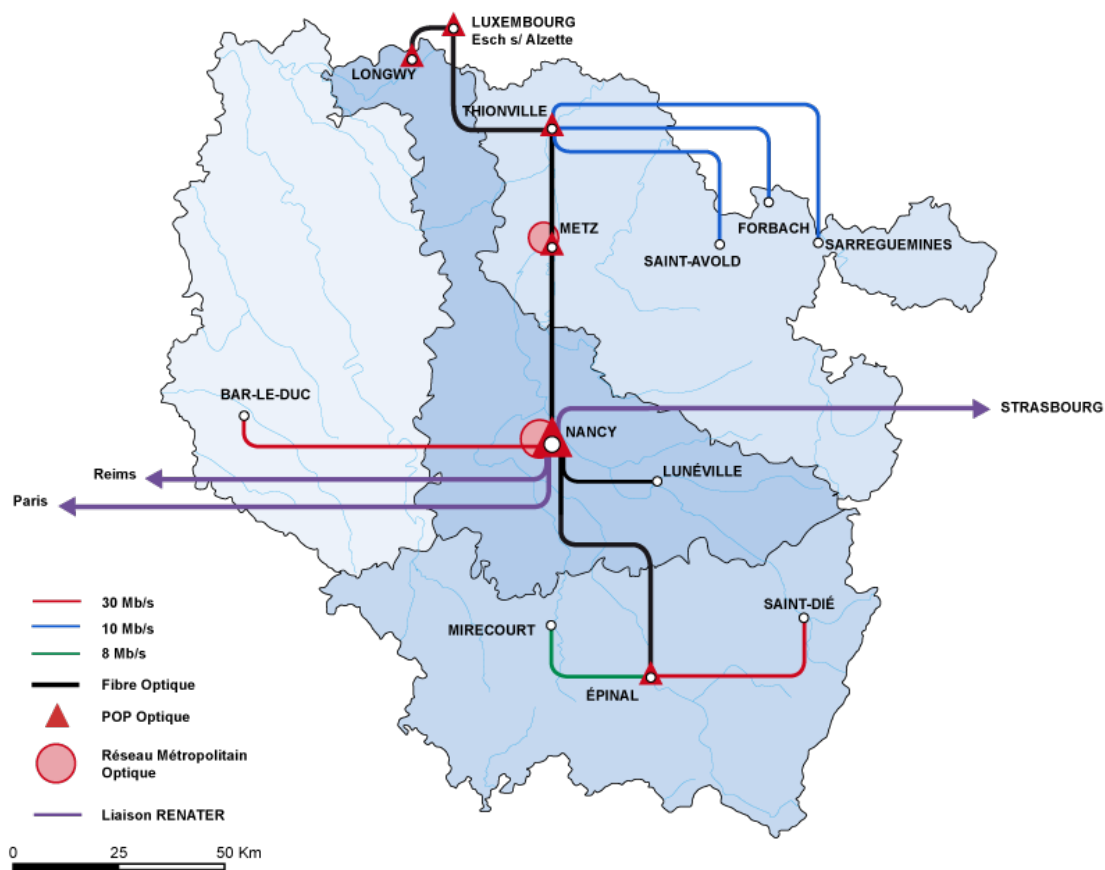


FIGURE 9.1 – Cartographie des interconnexions

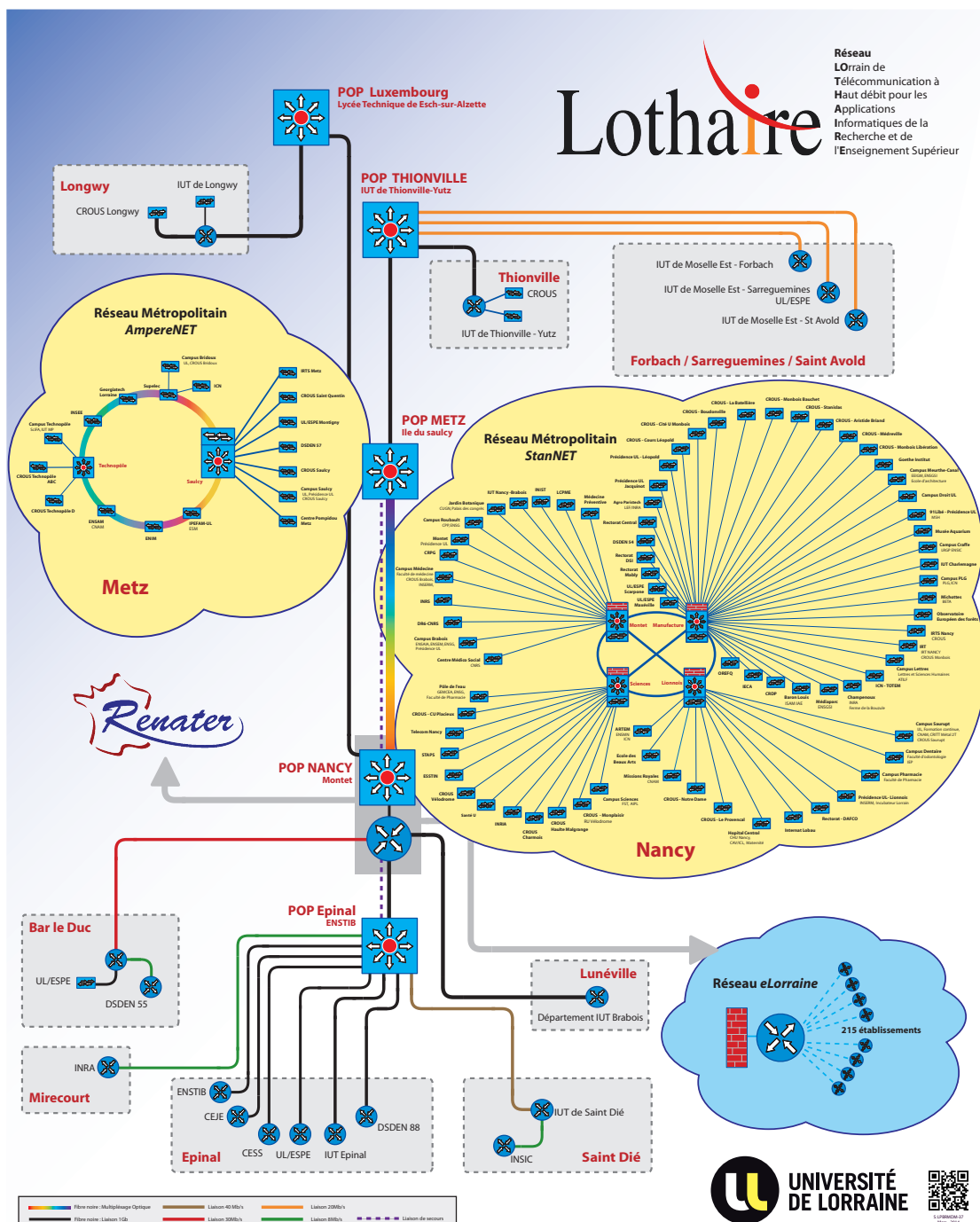


FIGURE 9.2 – Représentation du réseau Lothaire

9.2 Images Kibana

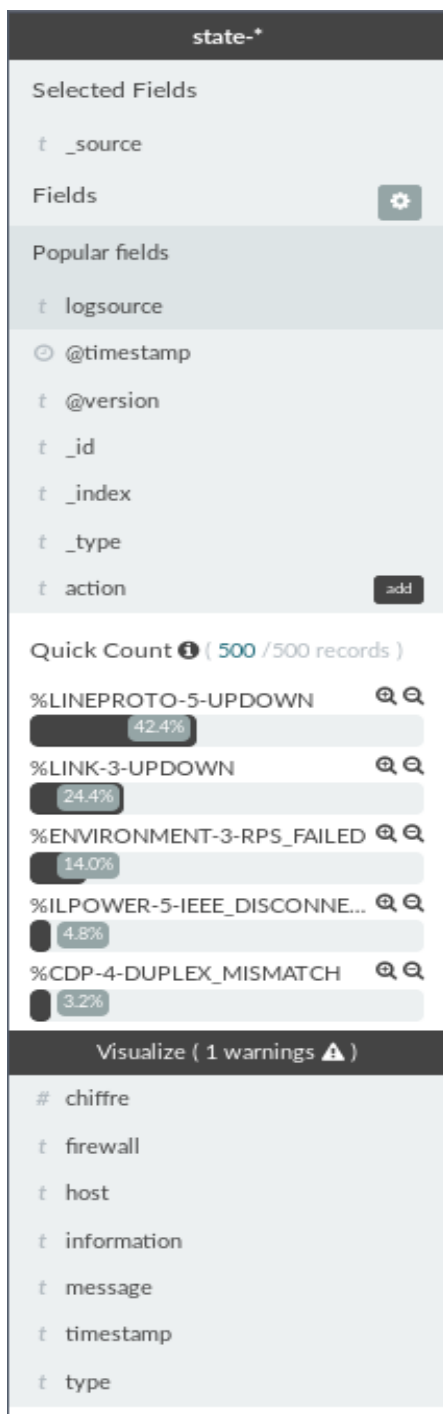
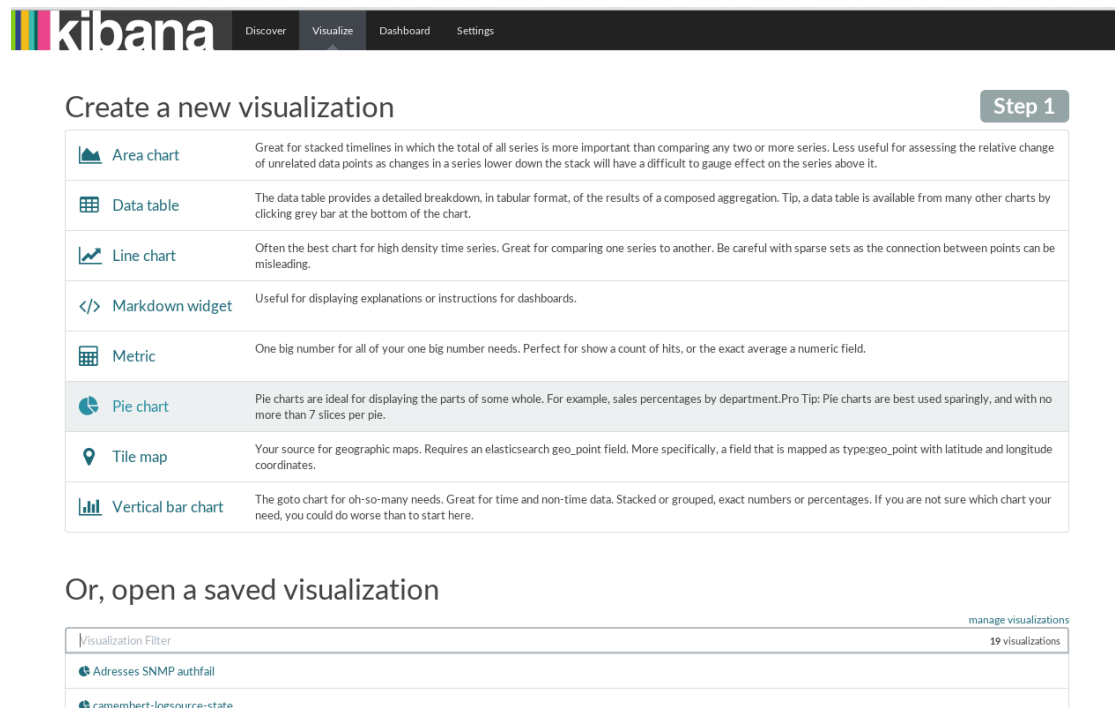


FIGURE 9.3 – Le tableau des champs



Create a new visualization Step 1

Area chart	Great for stacked timelines in which the total of all series is more important than comparing any two or more series. Less useful for assessing the relative change of unrelated data points as changes in a series lower down the stack will have a difficult to gauge effect on the series above it.
Data table	The data table provides a detailed breakdown, in tabular format, of the results of a composed aggregation. Tip, a data table is available from many other charts by clicking grey bar at the bottom of the chart.
Line chart	Often the best chart for high density time series. Great for comparing one series to another. Be careful with sparse sets as the connection between points can be misleading.
Markdown widget	Useful for displaying explanations or instructions for dashboards.
Metric	One big number for all of your one big number needs. Perfect for show a count of hits, or the exact average a numeric field.
Pie chart	Pie charts are ideal for displaying the parts of some whole. For example, sales percentages by department. Pro Tip: Pie charts are best used sparingly, and with no more than 7 slices per pie.
Tile map	Your source for geographic maps. Requires an elasticsearch geo_point field. More specifically, a field that is mapped as type:geo_point with latitude and longitude coordinates.
Vertical bar chart	The goto chart for oh-so-many needs. Great for time and non-time data. Stacked or grouped, exact numbers or percentages. If you are not sure which chart your need, you could do worse than to start here.

Or, open a saved visualization

[manage visualizations](#)

Visualization Filter	19 visualizations
Adresses SNMP authfail	
camembert-lnservice-state	

FIGURE 9.4 – Visualisations possibles

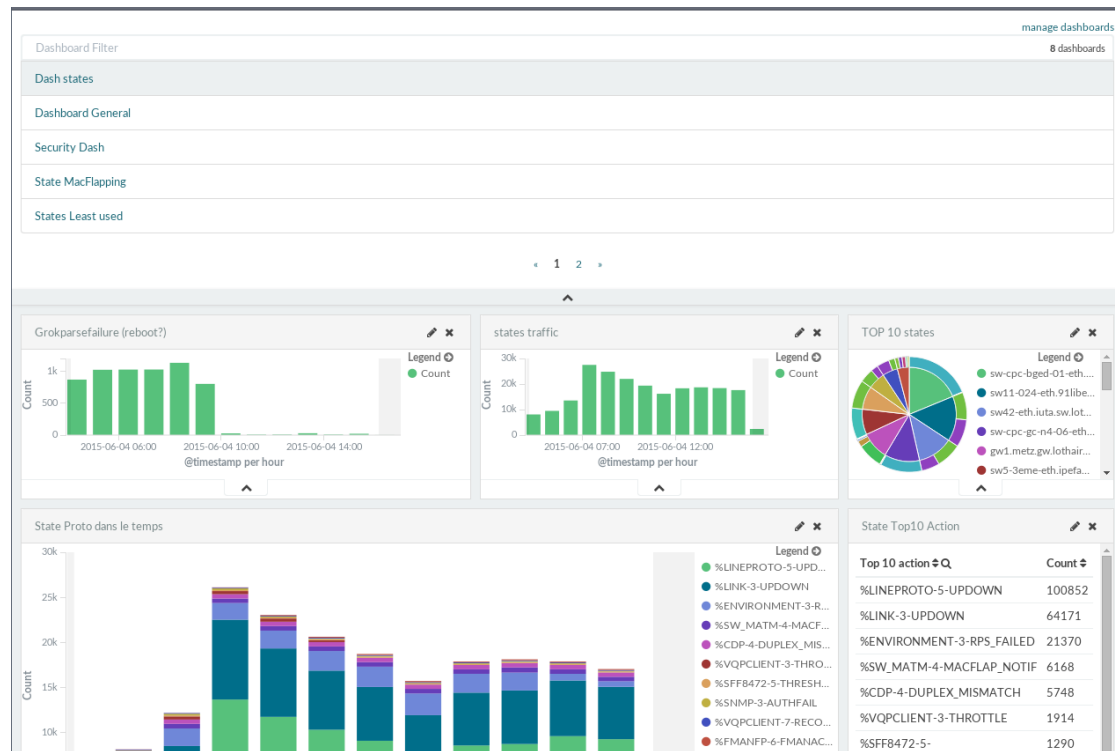


FIGURE 9.5 – Dashboard plus avancé

9.3 Statistiques Munin

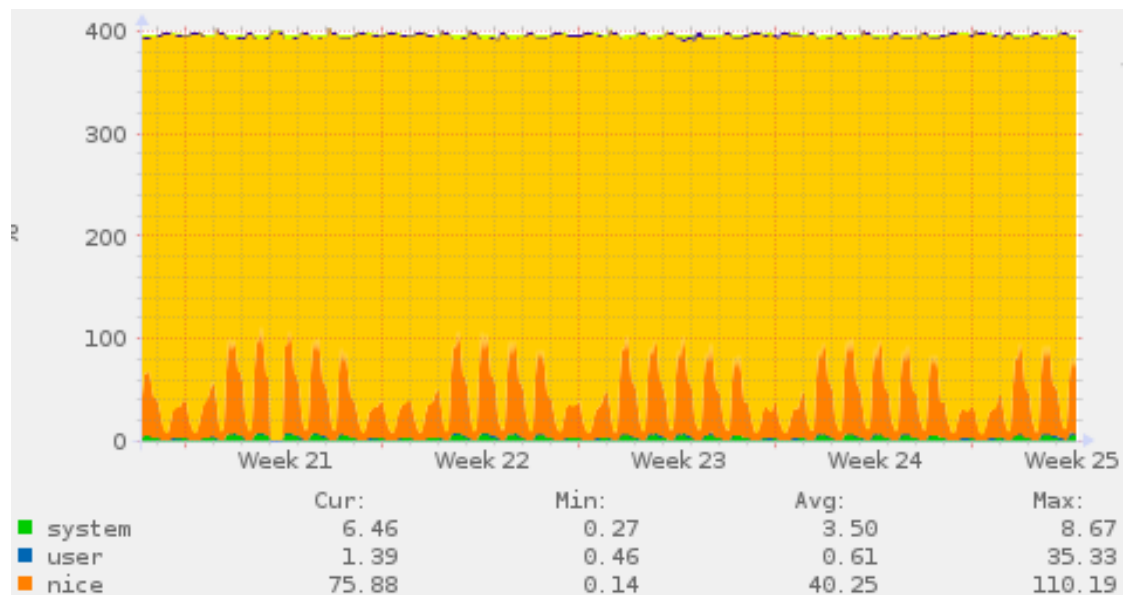


FIGURE 9.6 – Consommation CPU effective mensuel d'Elk2, en bleu...

Ne vous esquentez pas les yeux, le bleu est pratiquement indiscernable.

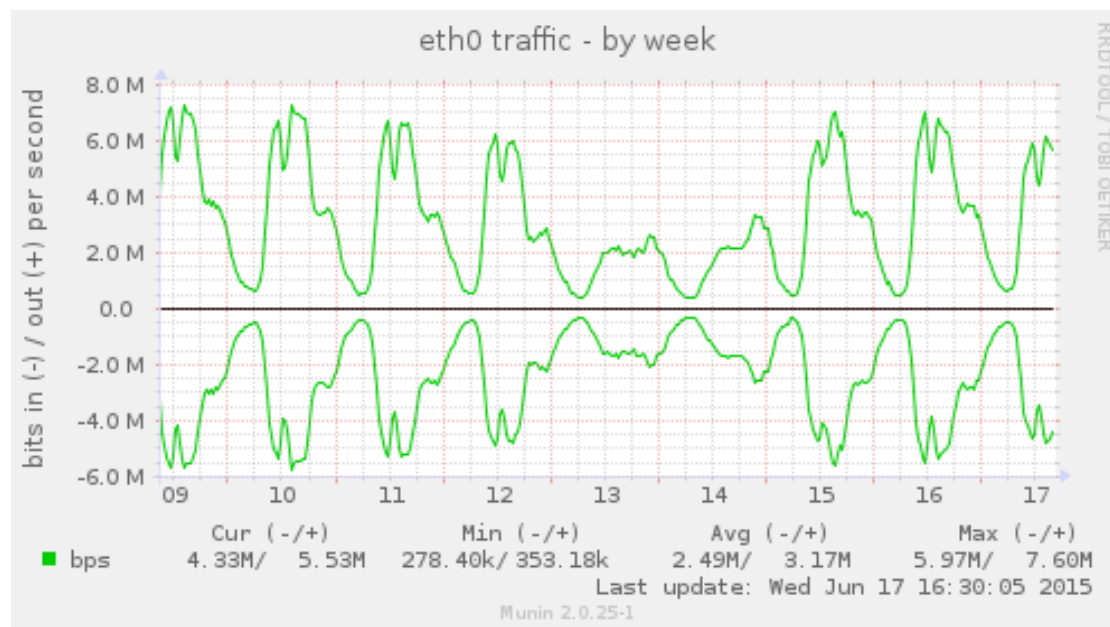


FIGURE 9.7 – Bande passante Up et Down de Elk2

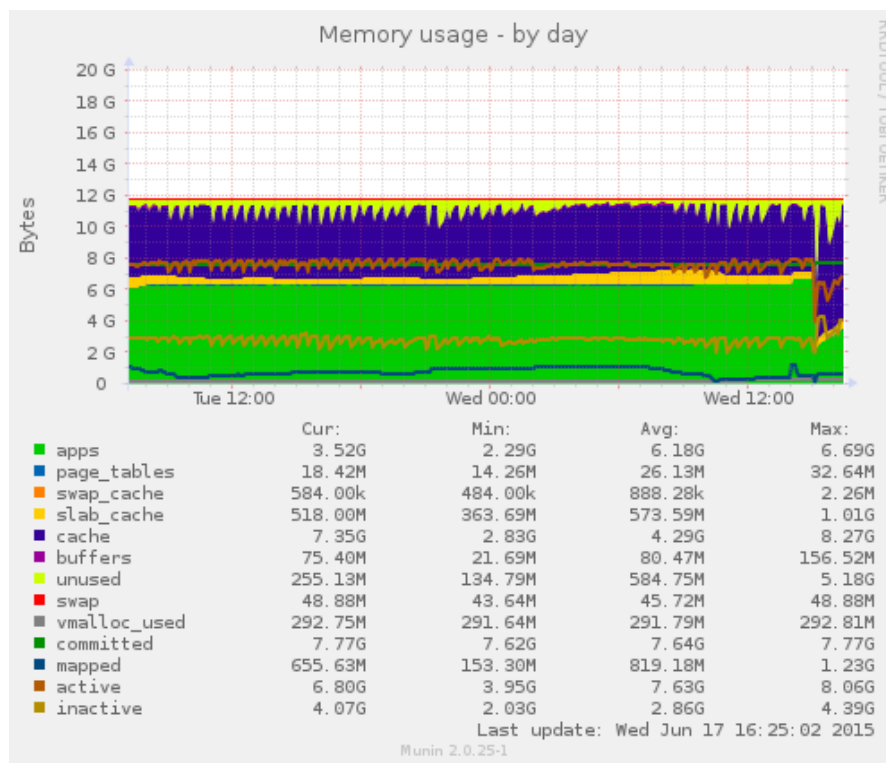


FIGURE 9.8 – Utilisation RAM Elk1

La ligne orange représente la RAM effectivement utilisée, la zone bleue la mémoire virtuelle.

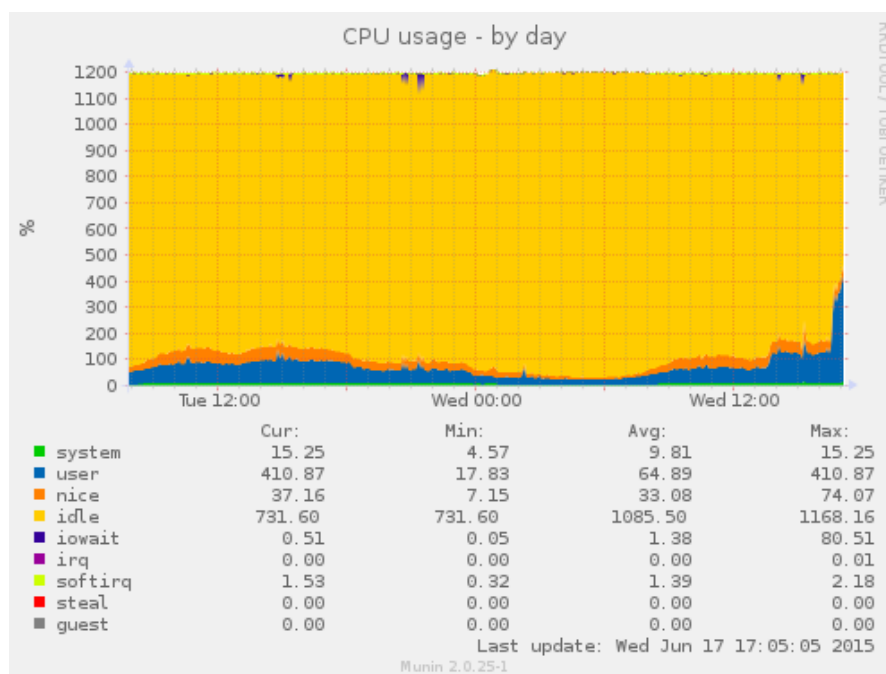


FIGURE 9.9 – Utilisation CPU Elk1

Chapitre 10

Code Source et scripts

10.1 Elasticsearch

```
1 {
2   "firewall-2015.06.10": {
3     "mappings": { "syslog": { "properties": {
4       "@timestamp": {
5         "type": "date",
6         "format": "dateOptionalTime"
7       },
8       "@version": {
9         "type": "string"
10      },
11      "IPinside": {"type": "string"},
12      "IPnat": {"type": "string"},
13      "IPoutside": {"type": "string"},
14      "action": {"type": "string"},
15      "chiffre": {"type": "integer"},
16      "datetest": {
17        "type": "string",
18        "index": "not_analyzed"
19      },
20      "firewall": {"type": "string"},
21      "host": {"type": "string"},
22      "information": {"type": "string"},
23      "logsource": {
24        "type": "string",
25        "index": "not_analyzed"
26      },
27      "message": {"type": "string"},
28      "numero": {"type": "string"},
29      "operation": {"type": "string"},
30      "portin": {"type": "string"},
31      "portnat": {"type": "string"},
32      "portout": {"type": "string"},
33      "program": {"type": "string"},
34      "reste": {"type": "string"},
35      "tags": {"type": "string"},
36      "timestamp": {"type": "string"},
37      "type": {"type": "string"}
38    } } }
39 }
40 }
```

Listing 10.1 – Exemple de mapping


```
1 (DELETE /state-2015.05*)
2 DELETE /_template/state-log
3
4 PUT /_template/state-log
5 {
6   "template": "state-*",
7   "order": 1,
8   "settings": {"number_of_shards": 5},
9   "mappings": {
10     "syslog": {
11       "properties": {
12         "@timestamp": {
13           "type": "date",
14           "format": "dateOptionalTime"
15         },
16         "@version": {
17           "type": "string"
18         },
19         "action": {
20           "type": "string",
21           "index": "not_analyzed"
22         },
23         "chiffre": {
24           "type": "integer"
25         },
26         "datetest": {
27           "type": "date",
28           "format": "MMM d H:m:s"
29         },
30         "firewall": {
31           "type": "string"
32         },
33         "host": {
34           "type": "string"
35         },
36         "information": {
37           "type": "string"
38         },
39         "logsource": {
40           "type": "string",
41           "index": "not_analyzed"
42         },
43         "message": {
44           "type": "string"
45         },
46         "reste": {
47           "type": "string"
48         },
49         "tags": {
50           "type": "string"
51         },
52         "timestamp": {
53           "type": "string"
54         },
55         "type": {
56           "type": "string"
57         }
58       }
59     }
60   }
61 }
```

Listing 10.2 – Ajouter un template de mapping

lst :mappingput2

10.2 Code en production

```
1 #!/bin/bash
2 # -*- coding: utf-8 -*-
3 # Copyright (c) 2013, Frank Rosquin <frank@rosquin.net>
4 # ISC license
5 # Permission to use, copy, modify, and/or distribute this software
6 #   for any
7 #   purpose with or without fee is hereby granted, provided that the
8 #   above
9 #   copyright notice and this permission notice appear in all copies.
10 # Last Modified François Dupont 2015-05-28
11 blade="100.127.255.1"
12 daysback=31
13 port=9200
14 start_from=4
15 target="firewall"
16
17 while getopts b:d:p:s:t: option
18 do
19   case $option in
20     b)
21       blade=$OPTARG
22       ;;
23     d)
24       if [ $OPTARG -ge 0 ]; then
25         daysback=$OPTARG
26       else
27         echo "-d should be a number, 0 or more"
28         echo ""
29         print_usage
30       fi
31       ;;
32     p)
33       if [ $OPTARG -ge 0 ]; then
34         port=$OPTARG
35       else
36         echo "-p should be a number, 0 or more"
37         echo ""
38         print_usage
39       fi
40       ;;
41     s)
42       if [ $OPTARG -ge 0 ]; then
43         start_from=$OPTARG
44       else
45         echo "-s should be a number, 0 or more"
46         echo ""
47         print_usage
48       fi
49       ;;
50     t)
51       target=$OPTARG
52       ;;
53     \?)
54       print_usage
55       ;;
56   esac
57 done
58
59 end=$(expr $start_from \+ $daysback)
60 if [ $? == 0 ]; then
61   for i in $(seq $start_from $end); do
62     d=$(date --date "$i_days_ago" +"%Y.%m.%d")
63     curl -sS -XDELETE $blade:$port/$target-$d
```

```

62 done
63 else
64     echo "Invalid_number_of_days_specified,_aborting"
65 fi
66 curl -XPOST "$blade:$port/_optimize"
67
68 print_usage() {
69     echo "
70 Usage: _$_[OPTIONS]
71 Options:
72 -b blade: name_or_ip_of_the_elasticsearch_server
73           default: 100.127.255.1
74 -d daysback: number_of_days/indexes_deleted_(from_start_day)
75           default: 31
76 -p port: number
77           default: 9200
78 -s start_number, _starting_delete_day_(-s_4=_if_we_are_the_30/04_
       it_starts_deleting_from_26/04_included)
79           default: 4
80 -t target_name_of_the_indices
81           default: firewall
82 Example:
83
84 "
85     exit 0
86 }

```

Listing 10.3 – Script de suppression d'index

```

1 [Unit]
2 Description=Kibana
3
4 [Service]
5 ExecStart=/home/fdupont/kibana-4.0.2-linux-x64/bin/kibana
6
7 [Install]
8 WantedBy=multi-user.target

```

Listing 10.4 – Service simple pour kibana

Le chemin de l'application est là à titre indicatif, **/usr/local/bin** est plus indiqué, par exemple.

```

1 input{
2     tcp{
3         port => 514
4         type => syslog
5     }
6
7     udp{
8         port => 514
9         type => syslog
10    }
11 }
12
13 filter{
14     grok{
15         # match => [ "message", "%{SYSLOGBASE}" ]
16         match => [ "message", "%{SYSLOGTIMESTAMP:timestamp}_(?:%{
           SYSLOGFACILITY:facility}_)?%{SYSLOGHOST:logsource}_%{
           GREEDYDATA:reste}" ]
17     }
18
19     #suppression des informations pas interessantes pour firewall (
       TCP/UDP teardrdown)
20     if "-302014:" in [reste] or "-302016:" in [reste]{
21         drop{}

```

```
22 }
23
24 #eclatement pour firewall
25 if "FWSM" in [reste]{
26     grok{
27         match => [ "reste", "%{PROG:program}:_(?<operation>[a-z,A
-Z,_,+])_(?<numero>\d+)_%{DATA}_inside:%{IP:IPinside
}/(?<portin>\d+)_\(%{IP:IPnat}/(?<portnat>\d+)\)_%{
DATA}_outside:%{IP:IPoutside}/(?<portout>\d+)\)_%{DATA}"
        ]
28     }
29     mutate{
30         add_field => {
31             "firewall" => true
32         }
33         remove_field => ["reste", "timestamp"]
34     }
35     grok{
36         match => [ "@timestamp", "%{TIMESTAMP_ISO8601:timestamp}"
        ]
37     }
38 }
39 #traitement log DNS, inexploité pour le moment
40 # else if "queries: info: client" in [reste]{
41 #     grok{
42 #         match => [ "reste", "(?<ipclient>%{IP})#([0-9]{1,5})\
+\\((.*)?)\\).*?query:\\ (?<url>.*?)\\ (?<type>IN|HS|CH|ANY)\\ (?<
champ>.*?)\\ (?<option>.*?)\\ " ]
43 #     }
44 #     mutate{
45 #         add_field => {"dns" => true}
46 #         remove_field => ["reste"]
47 #     }
48 # }
49
50 else{
51     #traitement des states
52     grok{
53         match => [ "reste", "(?<chiffre>\d+):\\_(?<uptime
>([1-9][0-9]?([d|w|y))([1-9][0-9]?([d|w|y))?:\\_)?((?<
datetest>(?: Jan(?:uary)?|Feb(?:ruary)?|Mar(?:ch)?|Apr
(?:il)?|May|Jun(?:e)?|Jul(?:y)?|Aug(?:ust)?|Sep(?:
tember)?|Oct(?:ober)?|Nov(?:ember)?|Dec(?:ember)?)_
(?::(?:0[1-9])|(?:[12][0-9])|(?:3[01])|[1-9])_
(?::2[0123]|[01]?[0-9]):(?:[0-5][0-9])
:((?:[0-5]?[0-9]|60)(?:[:.,][0-9]+)?)):\\_)?(?<action
>[\\w._\\/%-]+):\\_%{GREEDYDATA:information}" ] #
        attention aux espaces
54     }
55
56     #si ce n'est pas un state bien formé
57     if "_grokparsefailure" in [tags]{
58         #mise en throttle pour mail
59         throttle{
60             before_count => 2
61             after_count => 4
62             period => 600
63             key => "%{message}"
64             #ajout du tag pour envoi d'email
65             add_tag => "mail"
66         }
67     }
68     mutate{
69         add_field => {"firewall" => false}
70         remove_field => ["reste"]
71     }
}
```

```
72 }
73 }
74
75 output{
76   #envoi de firewall
77   if [firewall]=="true"{
78     redis {
79       host => "100.127.255.1"
80       data_type => "list"
81       key => "firewall"
82     }
83   }
84   #non utilisé pour le moment
85   #   else if [dns]=="true"{
86   #     stdout{codec => rubydebug}
87   #     #   redis {
88   #     #     host => "100.127.255.1"
89   #     #     data_type => "list"
90   #     #     key => "firewall"
91   #     #   }
92   #   }
93   #envoi de states
94   else{
95     redis {
96       host => "100.127.255.1"
97       data_type => "list"
98       key => "state"
99     }
100   }
101   #envoi de mail
102   if "mail" in [tags]{
103     email{
104       to => "fdupont@localhost"
105       from => "logstashfail@%{host}"
106       subject => "%{@timestamp}_logstashfailure"
107       body => "Logstash n'a pas pu parser un log contenant : \n
108               %{message}\n
109               Ce message requiert votre attention, car
110               il est souvent synonyme d'un probleme reseau
111               quelconque"
112     }
113   }
114 }
```

Listing 10.5 – Script de configuration général coté server central

Chapitre 11

Parties non essentielles

Multithread

Attention ces informations sont pour le moment, *Jeudi 16 Avril 2015*, correctes mais sont amenées à changer, notamment concernant les outputs.

Chaque plugin input utilise un thread. Cela permet d'éviter les engorgements si certaines entrées sont plus longues à traiter que d'autres.

Le bloc filtre entier utilise, par défaut, un seul thread. Mais, il est possible d'augmenter le nombre de threads affectés au traitement des filtres avec le flag `-w` lors du démarrage de Logstash.

À l'heure actuelle, le bloc output de logstash ne peut utiliser qu'un seul thread. Il lit donc sa queue de façon séquentielle.

11.0.1 Moteur d'expression régulière

Les expressions régulières utilisées par Logstash dans le plugin grok utilisent le moteur Oniguruma dont les spécifications sont disponibles à cette adresse ¹. Ruby n'utilise plus Oniguruma depuis la version 2.0 mais son fork Onigmo ². Mais cette évolution ne s'applique pas à Logstash puisque ce dernier est codé en Jruby.

11.0.2 Sense

Ce module chrome a été développé par Boaz Leskes. Il sert de client à Elasticsearch pour éviter d'avoir à le manipuler directement via curl.

Le développement (public) de ce module est arrêté depuis son intégration dans *Marvel* le logiciel de monitoring et d'optimisation, vendu ³ par la société *elastic*.

Installation

Installer Sense est simple comme installer une extension Chrome ⁴ tierce. Tout d'abord : récupérer le module à l'adresse <https://github.com/bleskes/sense> en utilisant par exemple :

```
1 git clone https://github.com/bleskes/sense.git
```

Il suffit ensuite de l'activer dans chromium :
chrome ://extension => Developer mode => Load unpacked extension.

1. <http://www.geocities.jp/kosako3/oniguruma/doc/RE.txt>, si des symboles ¥ apparaissent vous avez probablement un problème d'UTF8 sur votre navigateur, ils correspondent à des \

2. dicit Wikipédia <https://en.wikipedia.org/wiki/Oniguruma>

3. voir basde page : <https://www.elastic.co/products/marvel/signup.html>

4. testé sur chromium

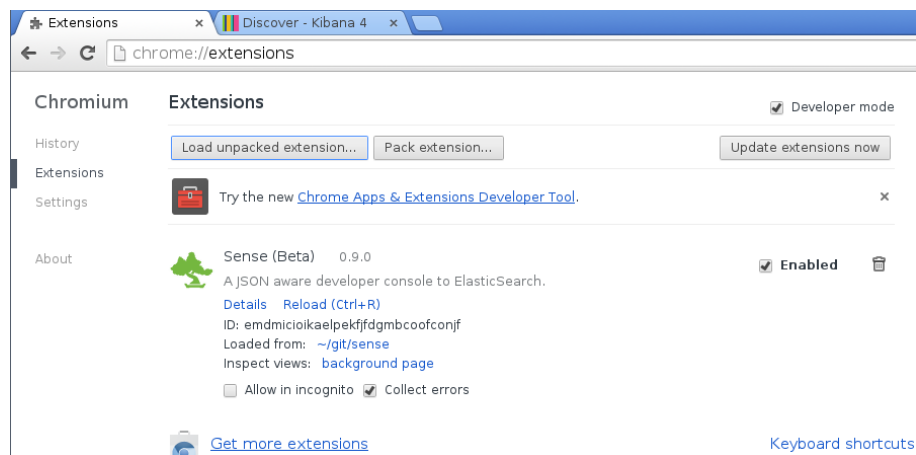


FIGURE 11.1 – Plugin Sense installé dans chromium

Et voilà ! (avec un accent anglais)

Utilisation de Sense

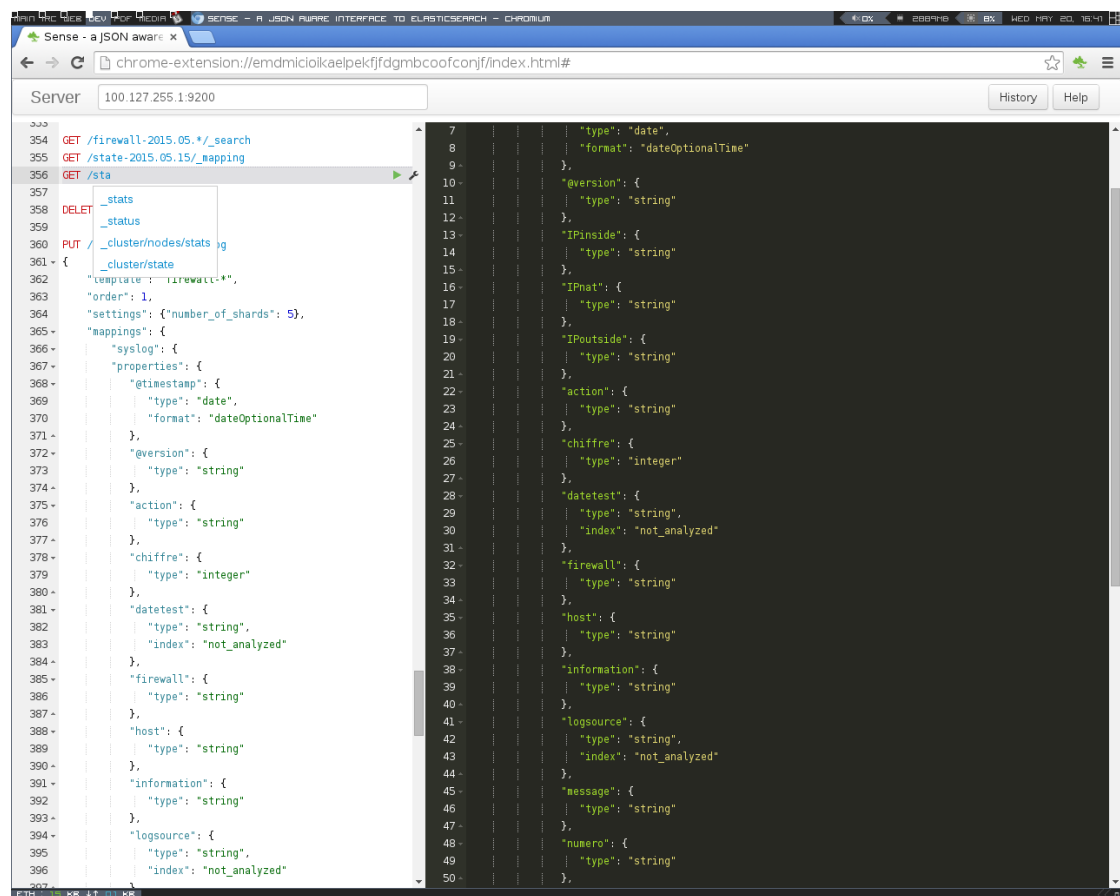


FIGURE 11.2 – Vue générale de Sense

Nous allons à l'aide de l'image ci-dessous brièvement expliquer le fonctionnement de Sense

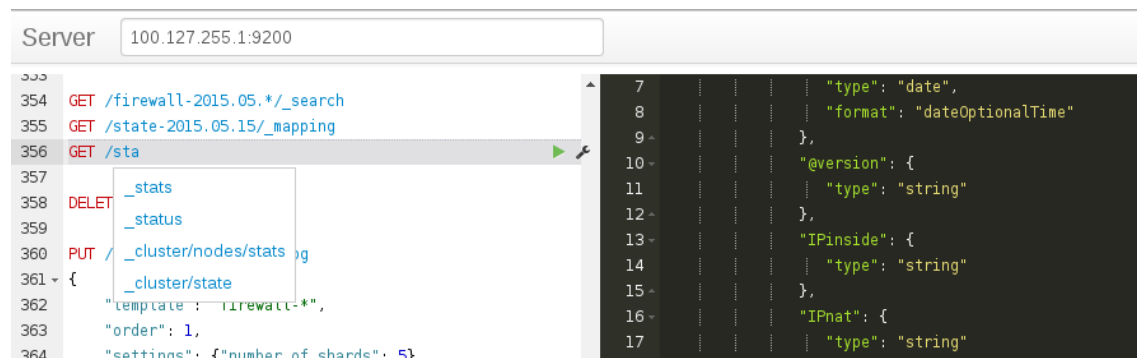


FIGURE 11.3 – Zoom sur les fonctionnalités de Sense

Le formulaire **Server** situé en haut correspond à l'adresse et au port d'écoute de l'instance Elasticsearch sur laquelle on souhaite travailler.

Le panneau de gauche correspond au **panneau de requête**. On utilise l'API REST d'Elasticsearch pour envoyer des requêtes (recherches, modifications... une présentation de l'API est réalisée ultérieurement). Il est à noter que le panneau de gauche est doté d'une autocomplétion pour les fonctions et les éléments standards d'Elasticsearch.

Le panneau de droite est le **panneau de réponse** aux requêtes. Les informations nous parviennent en JSON.

11.0.3 Full text

Le full text s'oppose aux valeurs exacts. Les champs non analysés sont traités comme des valeurs exacts. Comme expliqué plus haut, les chiffres, les booléens, les dates, sont des valeurs exacts. Il est possible qu'une chaîne de caractère en soit une aussi.

"Le chien" est différent de **"le chien"** ou encore de **"Lechien"**.

Pour le full text la notion d'index et d'analyseur est primordiale. L'analyseur va décider de comment les différents membres de la chaîne de caractère vont être considérés. Après cette étape ces membres vont être indexés et une recherche les renverra par popularité décroissante.

Puisqu'un exemple vaut mieux qu'un long discours, je vais m'inspirer de ceux donnés dans *"The Definitive guide to Elasticsearch"*.

Considérons 2 phrases :

- The quick brown fox jumped over the lazy dog
- Quick brown foxes leap over lazy dogs in summer

Termes	P1	P2
Quick		X
The	X	
brown	X	X
dog	X	
dogs		X
fox	X	
foxes		X
in		X
jumped	X	
lazy	X	X
leap		X
over	X	X
quick	X	
summer		X
the	X	X

FIGURE 11.4 – Index des phrase

Lors d'une recherche full text, Elasticsearch va créer ce genre de listes.
Si je cherche *quick brown*, le tableau ressemblerait à cela

Termes	P1	P2
brown	X	X
quick	X	
Total	2	1

FIGURE 11.5 – Index correspondant à une recherche

On voit donc bien que la phrase la plus populaire est la phrase 1. On constate également que Elasticsearch *fait une différence entre Quick et quick*. Cela est réglable (on peut, ne pas tenir compte de la casse). Ce qu'il faut retenir c'est que pour discriminer efficacement, il est aussi possible d'utiliser des syntaxe comme `+fox`, de même on peut faire une recherche avec un bloc de deux termes pour être plus efficace : `"over the"` qui n'existe que dans P1.

11.0.4 Settings

Si un doute s'était imissé dans notre esprit, les paramètres de Kibana, concernent uniquement Kibana, et pas Elasticsearch, certaines informations sont cependant affichées pour aider l'utilisation de Kibana mais pas de possibilité de les modifier.

En revanche, il faut bien avouer que leurs consultations est plus agréable depuis Kibana que depuis l'export json d'Elasticsearch.

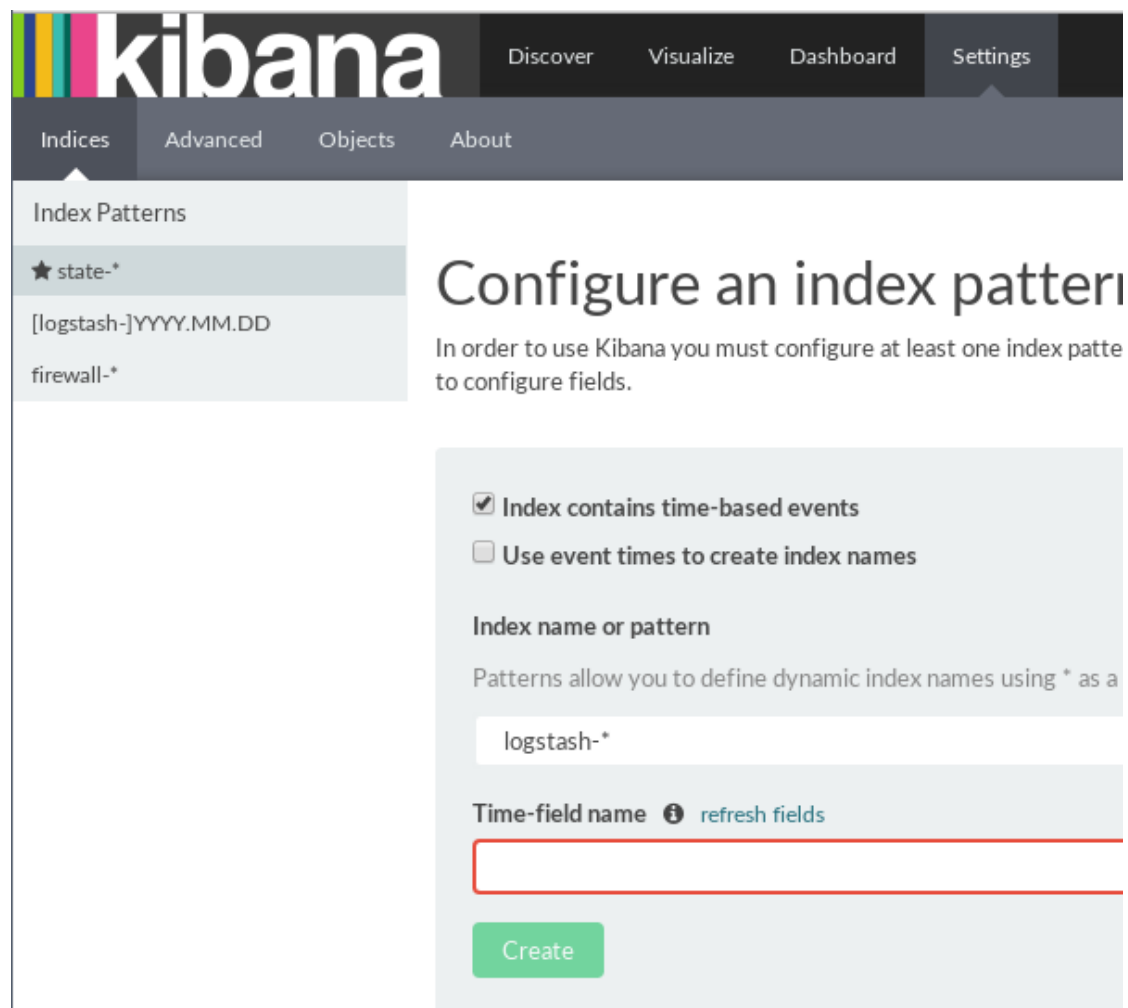
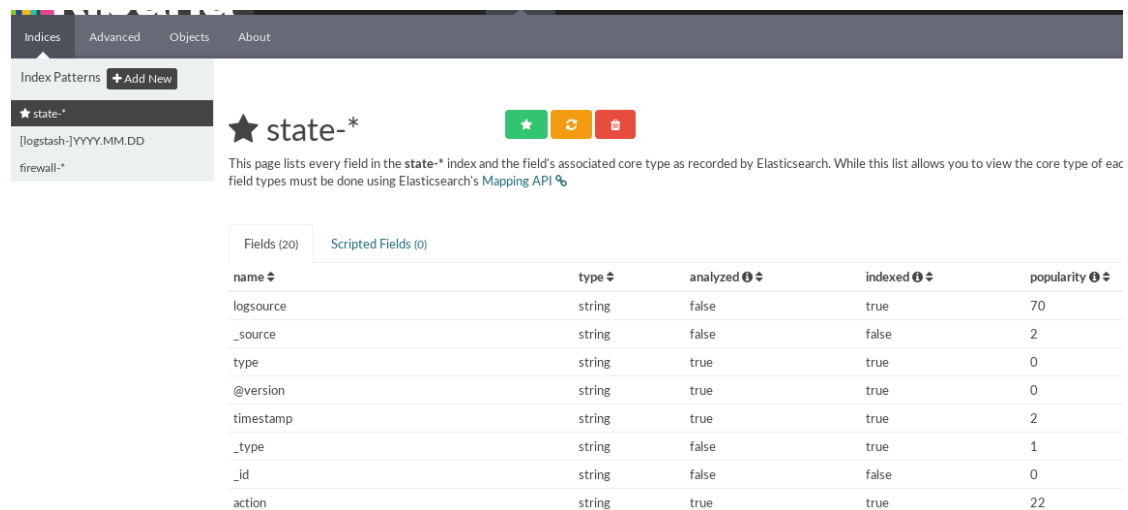


FIGURE 11.6 – Accueil de settings

Dans cette page vous pouvez créer de nouveaux ensembles d'index, ou en choisir un déjà existant.

En choisissant un index nous obtenons des informations sur son mapping, ce qui peut être utile pour effectuer des recherches plus pertinentes.



Indices | Advanced | Objects | About

Index Patterns **+ Add New**

- ★ state-*
- [logstash-*]YYYY.MM.DD
- firewall-*

★ state-*

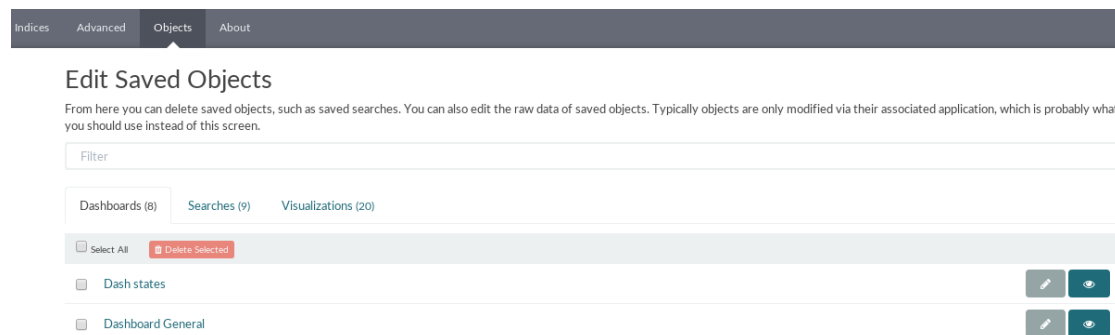
This page lists every field in the `state-*` index and the field's associated core type as recorded by Elasticsearch. While this list allows you to view the core type of each field, types must be done using Elasticsearch's [Mapping API](#).

Fields (20) | Scripted Fields (0)

name	type	analyzed	indexed	popularity
logsource	string	false	true	70
_source	string	false	false	2
type	string	true	true	0
@version	string	true	true	0
timestamp	string	true	true	2
_type	string	false	true	1
_id	string	false	false	0
action	string	true	true	22

FIGURE 11.7 – Mapping d'un index

Enfin, c'est aussi dans cette section que l'on a une liste et que l'on peut supprimer les objets enregistrés (recherche, visualisations, dashboard)



Indices | Advanced | Objects | About

Edit Saved Objects

From here you can delete saved objects, such as saved searches. You can also edit the raw data of saved objects. Typically objects are only modified via their associated application, which is probably what you should use instead of this screen.

Filter

Dashboards (8) | Searches (9) | Visualizations (20)

☐ Select All ☐ Delete Selected





<input type="checkbox"/> Dash states	 
<input type="checkbox"/> Dashboard General	 

FIGURE 11.8 – Liste des objets

Glossary

API Une API, Application Programming Interface, est une interface permettant d'interagir avec un programme par l'intermédiaire d'"opérations simples". Elles sont en générale la pour faciliter la réutilisation d'un programme plus complexe. C'est parce que openstreetmap à une API que l'on a vu autant de logiciels utilisant ses cartes/tuiles.. 27

flag Un drapeau, dans la dénomination logstash, le flag représente une option passé en argument exemple : -w.. 61

fulltext Un drapeau, dans la dénomination logstash, le flag représente une option passé en argument exemple : -w.. 21

logs Trace (informations) provenant d'un programme ou d'un équipement, souvent sous forme de texte. 5, 10, 12, 42, 44

Munin Munin est un logiciel de monitoring populaire et simple d'installation. 42

thread explication thread processus etc... 61