BD71

TP1 : Introduction à la collecte et au traitement des données

Nous allons voir comment récupérer des données, les analyser, et les envoyer à plusieurs destinations avec Logstash.

I. Logstash

Logstash(https://www.elastic.co/fr/logstash) est un outil multifonction, de type ETL(https://fr.wikipedia.org/wiki/Extract-transform-load. Commercialisé par Elastic(www.elastic.co), il est historiquement l'outil de prédilection pour récupérer et transformer des données pour Elasticsearch(https://www.elastic.co/fr/elasticsearch).

En tant qu'ETL, il est capable de:

- lire et écrire depuis de nombreuses sources de données: fichier, tcp, http, sql, kafka, redis, elasticsearch, ...

- transformer les données, pour qu'elles soient interprétables et utilisables dans Elasticsearch (et donc visualisable dans Kibana)

I.1. Installation, test et configuration de Logstash

Dans un premier temps, nous allons installer Logstash, et dans un second temps, le tester, pour voir si tout fonctionne bien ; et le configurer dans un dernier temps.

I.1.1. Installation de Logstash

Il est extrêmement recommandé d'exécuter les commandes suivantes dans un répertoire local, et non pas sur un disque réseau.

****Linux:****

Ouvrez un shell et tapez les commandes suivantes:

```bash
# Télécharger l'archive contenant logstash
wget https://artifacts.elastic.co/downloads/logstash/logstash-8.13.4-linux-x86_64.tar.gz -O logstash.tar.gz

# Décompresser l'archive
tar -zxvf logstash.tar.gz

# Suppression de l'archive
rm logstash.tar.gz

# Entez dans le répertoire de logstash
cd logstash-8.13.4
```

Le lancement du **binaire logstash** se fait avec la commande, et vous l'utiliserez dans toutes les commandes:

```bash
JAVA_HOME='' ./bin/logstash
```

Windows:

Préalable : Installer un JDK dont la version figure dans la matrice de support (https://www.elastic.co/support/matrix#matrix_jvm)

Télécharger le contenu de cette archive (https://artifacts.elastic.co/downloads/logstash/logstash-8.13.4-windows-x86_64.zip) ou allez le télécharger sur le site de Elastico (https://www.elastic.co/downloads/past-releases/logstash-8-13-4), et dézippez la dans un répertoire local, et non pas sur un disque réseau:

Puis ouvrez un shell, pour aller dans le répertoire que vous venez de dézipper.

Ouvrez une invite de commande. Allez dans le répertoire Logstash. À l'intérieur du dossier bin, exécutez

> cd logstash-8.13.4/bin

Le lancement du binaire logstash se fait avec la commande, et vous l'utiliserez dans toutes les commandes:

``` <commande_logstash>

**logstash.bat** ou **logstash**

```

****Linux + Windows:****

Maintenant que Logstash est téléchargé, effectuons un test sommaire pour voir si il est correctement configuré (remplacer la commande par la bonne commande, dans les blocs précédents):

Vérification de la version de logstash

<commande_logstash> -v

Vous devez obtenir un message comme celui-ci (le chemin du jdk utilisé n'est pas important):

Using bundled JDK: /my/path/to/logstash/directory/logstash-8.13.4/jdk

Logstash- 8.13.4

Si ce n'est pas le cas, il y a probablement un problème.

I.1.2. Test de Logstash

Lancez la commande suivante:

Entrez dans le répertoire de logstash

cd logstash-8.13.4

Lancement de logstash avec une configuration en ligne de commande

<commande_logstash> --log.level=error -e "input { stdin { type =>stdin } } output { stdout { codec =>rubydebug } }"

Exemple sous Windows on aura :

logstash --log.level=error -e "input { stdin { type =>stdin } } output { stdout { codec =>rubydebug } }"

Une fois la ligne suivante affichée:

> The stdin plugin is now waiting for input:

Taper une chaine de caractère, suivis de la touche entrée : cette première devrait être répétée, avec des métadonnées supplémentaires, comme dans l'exemple suivant:

Ceci est un simple test

On obtient comme réponse les lignes ci-dessous qui s'affichent à l'écran

{

"message" => "Ceci est un simple test",

"@timestamp" => 2022-02-20T13:40:32.529389Z,

"event" => {

"original" => "Ceci est un simple test"

},

"host" => {

"hostname" => "x970670"

},

"@version" => "1",

"type" => "stdin"

}

N.B.: Vous pouvez faire CTRL+C pour quitter à tout moment

I.1.3. Configuration de Logstash

Comme nous n'allons pas à chaque fois entrer la configuration en ligne de commande, nous allons configurer Logstash afin qu'il prenne en compte des fichiers de configuration

Recréons le fichier de configuration logstash `**logstash.yml**` situé dans le dossier `config` à partir de la configuration suivante:

```yml
# Chemin vers le dossier de data
path.data: "./data"

# Si les évènements en sortie doivent-être ordonné ou non
pipeline.ordered: true
```

Desactivation de la compatibilité ECS

pipeline.ecs_compatibility: disabled

Log level

log.level: info

Nombre de worker pour pour chaque pipeline

pipeline.workers: 1

Chemin vers les fichiers de logs

path.logs: "./logs"

```

Puis créez le dossier qui va servir à contenir nos fichiers de configurations:

***mkdir "conf" "conf/my-first-test"***

Ainsi que deux autres dossiers, que nous utiliserons plus tard:

***mkdir output input***

Puis nous allons créer dans ce dossier notre fichier de configuration logstash, au chemin ./conf/my-first-test/logstash.conf` (se référer au schéma de l'architecture des dossiers dans la partie suivante si vous êtes perdu): cf. I.2.1

```ruby

**Contenu du fichier logstash.conf

input {

  stdin {

    type => stdin

```
   }
}

output {
stdout {
codec =>rubydebug
   }
}
```

Enfin, lançons Logstash, en prenant en compte ces fichiers de configuration:

<commande_logstash> -f "conf/my-first-test"

Exemple sous Windows : > logstash –f "conf/my-first-test"

Si tout fonctionne bien, vous devriez avoir le même résultat qu'au premier test que nous avons effectué (avec plus de logs), et un format de sortie légèrement différent:

```
> Ceci est un simple test
{
   "@timestamp" => 2022-02-20T13:46:55.165996Z,
      "message" => "Ceci est un simple test",
"type" => "stdin",
     "@version" => "1",
        "host" => "x970670"
}
```

I.2. Fonctionnement de Logstash

I.2.1. Arborescence des fichiers

Même s'il y a beaucoup plus de dossiers que ça, voici les fichiers et dossiers qui vont nous intéresser:

logstash-8.0.0

├— bin

├— conf

| └— my-first-test

| └— logstash.conf

├— config

| ├—jvm.options

| ├— log4j2.properties

| ├—logstash.yml

| ├—pipelines.yml

├— data2

| ├—dead_letter_queue

| ├— queue

| └— uuid

├— logs

| ├— logstash-deprecation.log

| ├— logstash-plain.log

| └— logstash-slowlog-plain.log

├— input

├── output

Le dossier:

- `bin` va contenir les fichiers binaires de Logstash, nous permettant, entre autres, de l'exécuter

- **`conf`** va contenir, dans ses sous-dossiers, les configurations logstash que nous allons réaliser

- **`config`** est le dossier contenant toute la configuration. On peut noter, entre autres:

  - `jvm.options`: la configuration de Java

  - `log4j.properties`: la gestion des fichiers de logs de Logstash

  - `logstash.yml`: le fichier de configuration principal de Logstash

  - `pipelines.yml`: le fichier de configuration de pipelines Logstash (détaillé plus tard)

- **`data2`**: Le dossier contenant les données temporaires internes Logstash

- **`input`**: Le dossier qui va contenir nos fichiers de données initiaux

- **`output`**: Le dossier contenant nos données, une fois transformé

I.2.2. Pipelines Logstash

Les pipelines sont une notion très importante: elles sont le cœur de Logstash, et ce sont elles qui vont permettre de recevoir des données, les transformer, et les transférer ailleurs.

Un pipeline peut contenir quatre types de modules différents:

- Les ****input****, permettant de lire les données. Dans l'exemple précédant, nous avons utilisé un [input stdin](https://www.elastic.co/guide/en/logstash/current/plugins-inputs-stdin.html) indique l'entrée utilisateur

- Les ****filters****, non présents dans l'exemple précédent, qui vont permettre la transformationde la donnée

- Les **output**, qui vont nous permettre de transférer les données ailleurs. Dans l'exemple précédent, nous avons fait écrire Logstash sur l'[output

stdout](https://www.elastic.co/guide/en/logstash/current/plugins-outputs-stdout.html) -> ~ la console

- Les ****codecs****, qui vont permettre des transformations légères de données, utilisable seulement dans les **input** et **output**

Vous pouvez trouver, dans la [documentation Logstash](https://www.elastic.co/guide/en/logstash/current/index.html), la liste:

- des [inputs](https://www.elastic.co/guide/en/logstash/current/input-plugins.html)

- des [filters](https://www.elastic.co/guide/en/logstash/current/filter-plugins.html)

- des [outputs](https://www.elastic.co/guide/en/logstash/current/output-plugins.html)

- des [codecs](https://www.elastic.co/guide/en/logstash/current/codec-plugins.html)

Dans un environnement complexe, nous pouvons lancer de multiples pipelines en parallèle, d'où le fichier de configuration ***pipelines.yml***, mais dans notre cas, nous utiliserons uniquement la ligne de commande, pour indiquer l'emplacement des configurations à traiter

Un pipeline peut contenir plusieurs fichiers de configuration. Si c'est le cas, les fichiers de configurations seront chargés séquentiellement, par ordre alphabétique (leur ordre est important).

I.2.3. Protocole de création d'un nouveau pipeline

Pour chaque nouvel exercice, il va vous être demandé, pour créer un nouveau pipeline:

- De créer le, les fichiers de données initiaux, à faire dans le dossier `input` (si nécessaire)

- De créer un nouveau dossier, qui correspondra au nom de votre pipeline, dans `conf` (et de mettre les fichiers de configuration Logstash dedans)

- De modifier la ligne de commande, utilisée lors du dernier test, pour pointer vers le bon dossier (`<commande_logstash> -f "conf/<mon-dossier-pipeline>"`)

II. Exercices

II.1. Exercice guidé

Nous allons nous intéresser au parsing d'un fichier de log nommé **auth.log**, présent dans presque toutes les distributions Linux, et qui correspond à la surveillance des utilisateurs, de leur authentification, et de leurs usages de privilèges.

Créons dans un premier temps notre jeu de donnée : le fichier `input/auth.log`, avec le contenu suivant: cf. fichier resources/auth_log.md

À partir de cet extrait de données, nous allons essayer de déterminer les "patterns", ou "motifs", qui se répètent.

En effet, la structure, au moins au début, est très similaire entre les différentes lignes (prenons exemple sur la première ligne):

> Apr 23 18:01:16 valentin-test sshd[13824]: Accepted publickey for ubuntu from 172.16.180.99 port 53332 ssh2: RSA ...

- Le premier motif est `Apr  3 18:01:16`, et correspond à une date, avec une heure

- Le second motif est `valentin-test`, une chaine de caractère

- Le troisième motif est `sshd`, une chaine de caractère

- Le quatrième motif est `13824`, un nombre, entouré de crochets

- La fin du message, après le caractère `:`, varie et est ici `Accepted publickey for ubuntufrom 172.16.180.99 port 53332 ssh2: RSA SHA256:3rzOXXM+dv3rtFQqjmyLUz2y0OjLbcYWrFeIEt2if+c`

Si nous créons un tableau (pour l'illustration), ces patterns sont très visibles:

| Motif 1 (date) | Motif 2 | Motif 3 | Motif 4 (nombre) | Motif 5 (texte)
|

|----------|----------|---|----|----------|

| Apr 3 18:01:16 | valentin-test | sshd | 13824 | Accepted publickey for ubuntu from 172.16.180.99 port 53332 ssh2: RSA ... |

| Apr 23 18:01:16 | valentin-test | sshd | 13824 | pam_unix(sshd:session): session opened for user ubuntu by (uid=0) |

| Apr 23 18:02:25 | valentin-test | su | 14053 | Successful su for root by root |

| Apr 23 18:02:25 | valentin-test | su | 14053 | + /dev/pts/0 root:root |

| Apr 23 18:02:25 | valentin-test | su | 14053 | pam_unix(su:session): session opened for user root by ubuntu(uid=0) |

| Apr 23 18:02:25 | valentin-test | su | 14053 | pam_systemd(su:session): Cannot create session: Already running in a session |

| Apr 23 17:05:01 | valentin-test | su | 13781 | pam_unix(cron:session): session opened for user root by (uid=0) |

En l'occurrence, il s'agit d'un format, dérivé du format de log [syslog](https://fr.wikipedia.org/wiki/Syslog#Le_format_Syslog), qui à le schéma suivant:

> date machine programme[process_id]: message

À partir de là, nous savons comment transformer la donnée : il suffit de répliquer ce concept, en langage compréhensible par Logstash!

Commençons déjà par créer le pipeline de Logstash:

mkdir "conf/auth"

La partie suivante est une explication du contenu du fichier. Il sera explicitement écrit quand vous devrez créer le fichier en question.

****Input****

Dans un premier temps, nous allons dire à Logstash que nous voulons lire le fichier en question. Pour cela, nous allons utiliser l'[**input file**](https://www.elastic.co/guide/en/logstash/current/plugins-inputs-file.html), qui permet de lire un fichier sur disque.

Avec quelques modifications, cela va nous donner:

```ruby
input {
  file {
    path => "<chemin complet>/input/auth.log"
sincedb_path => "/dev/null"
start_position => "beginning"
}
}
```

!> Sur Windows, vous devrez remplacer `/dev/null` par `NUL`

****Filter****

La partie filtre, est très souvent le plus compliquée à faire. Dans ce cas-là, cela va se traduire par ça:

```ruby
```

```
filter {


grok {

match => {

    "message" => ["%{SYSLOGTIMESTAMP:date} %{PROG:machine} %{WORD:programme}\[%{INT:pid}\]: %{GREEDYDATA:contenu}"]

}

 }


 mutate {

  convert => {

    "pid" => "integer"

  }

 }


 date {

    match => [ "[date]", "MMM  dHH:mm:ss", "MMM dd HH:mm:ss" ]

target => "date"

 }


}
```

Les 3 filtres utilisés sont les plus courants, et sont utilisés dans la majorité des configurations:

- le **[grok]**(https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html) va nous permettre de **découper** le message en morceaux (contenu, par défaut, dans le champ `message`). Il s'agit de ce que nous avons fait tout à l'heure. La liste des Grok patterns

(cf. documentation du filter) est disponible [ici](https://github.com/logstash-plugins/logstash-patterns-core/blob/main/patterns/legacy/grok-patterns)

- le **[mutate]**(https://www.elastic.co/guide/en/logstash/current/plugins-filters-mutate.html) va nous permettre de convertir le **pid** du processus en nombre. En effet, le grok précédent n'interprète pas les données : pour lui, toutes les parties qu'il récupèrera seront considérées comme du texte.

- le **[date]**(https://www.elastic.co/guide/en/logstash/current/plugins-filters-date.html) va permettre de standardiser le format de la date. Ce format de date est particulier, car il existe deux versions différentes, mais ils sont construits à partir de la définition dans la documentation du module. La date résultante (qui va écraser le format initial) sera sous format [ISO8601](https://fr.wikipedia.org/wiki/ISO_8601), un format nativement reconnu par Elasticsearch. Si la timezone est présente (par défaut, la date est considérée en tant qu'UTC), n'oubliez pas de le préciser!

****Output****

La sortie sera tout aussi simple, nous allons écrire dans un fichier, avec un [codec](https://www.elastic.co/guide/en/logstash/current/codec-plugins.html) personnalisé ([rubydebug](https://www.elastic.co/guide/en/logstash/current/plugins-codecs-rubydebug.html)), afin que l'affichage soit un peu plus lisible:

```ruby
output {
   file {
      path => "<chemin complet>/output/auth-transforme.log"
codec =>rubydebug
   }
}
```

****Résultats****

Une fois ces parties faites (vous pourrez réutiliser l'input & l'output pour les exercices prochains, en modifiant les chemins), nous allons pouvoir voir le résultat:

Créons le fichier `conf/auth/logstash.conf` avec pour contenu la concaténation de ces trois parties:

```ruby
# Chemin complet doit uniquement contenir des '/' (et non pas des '\'), même sous Windows!

input {
  file {
    path => "<chemin complet>/input/auth.log"
start_position => "beginning"

    # Supprimer celui des deux qui ne s'applique pas
sincedb_path => "/dev/null" # Linux
sincedb_path => NUL # Windows
  }
}

filter {

 grok {
  match => {
   "message" => ["%{SYSLOGTIMESTAMP:date} %{PROG:machine} %{WORD:programme}\[%{INT:pid}\]: %{GREEDYDATA:contenu}"]
```

```
  }
 }

 mutate {
  convert => {
   "pid" => "integer"
  }
 }

 date {
    match => [ "[date]", "MMM  dHH:mm:ss", "MMM dd HH:mm:ss" ]
    target => "date"
 }

}

output {
  file {
     path => "<chemin complet>/auth-transforme.log"
codec =>rubydebug
  }
}
```

Et lançons Logstash:

```bash
**<commande_logstash> -f  "conf/auth"**
```

```

Le résultat du fichier de destination, `output/auth-transforme.log`, devrait-être tel que [celui-ci](resources/tp-1/output_auth.md)

Le **@timestamp** et l'**host** seront différents, car ils correspondent respectivement à:

- la date d'ingestion par Logstash de la donnée

- la machine sur laquelle l'ingestion a été effectuée

On peut également noté l'**absence de guillemets** autour des valeurs des champs *date* et *pid*: cela signifie qu'ils ont bien été convertis, respectivement en date et en nombre, dans un format compréhensible par une base de données

## II.2. Logs multilines

Partons de l'exercice précédent : quelqu'un a créé un nouveau format de logs, basé sur celui de l'exercice précédent, mais avec un contenu pouvant s'étendre sur plusieurs lignes :

cf. fichier  resources/log_multiline.md

Vous pouvez partir du filtre précédent, qui ne demande pas de modification. Néanmoins, il faudra faire une modification ailleurs ..

Si vous le testez sans modification, certains messages auront un tags `_grokparsefailure`, signifiant qu'il y a eu un problème lors du **grok**.

>Aide: Le mot clé est dans le titre.. **multiline**, à chercher dans la documentation

S’inspirer du fichier resources/answer/2.md

## II.3. Apache access logs

Nouveau format de logs, il faudra donc construire le filtre depuis le début!
```

cf. fichier resources/apache_combined_logs.md

En vous servant de la [documentation loggingapache](https://httpd.apache.org/docs/2.4/fr/mod/mod_log_config.html), essayer de trouver la structure, et la signification de tous ces champs, puis réaliser le filter, et tester-le.

> Ne PAS utiliser le [grokpattern](https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html#_description_129) `COMBINEDAPACHELOG`

Contraintes:

- le champ date doit-être bien formaté

- les champs sous format nombre doivent-être convertis en nombre

S’inspirer du fichier resources/answer/3.md

II.4. Data venant d'un CSV

Nous pouvons traiter de nombreux types de données avec Logstash, dont des [CSV](https://fr.wikipedia.org/wiki/Comma-separated_values)

cf. fichier resources/ csv_example.md

Réaliser le filter, et tester

S’inspirer du fichier resources/answer/4.md

II.5. JSON logs

Nous avons maintenant des logs web, dans un format [JSON](https://fr.wikipedia.org/wiki/JavaScript_Object_Notation).

cf. fichier resources/ web_log_json.md

Même processus.

S’inspirer du fichier resources/answer/5.md

III. Questions ouvertes

Qui seront discutés au début du tp suivant:

- Est-ce que la transformation des données est simple ?

- Les données de l'exercice II.3 et II.5 sont les mêmes : lesquelles sont plus simples à traiter ? Et pourquoi ?

- Dans le cadre d'exploitation de logs vaut-il mieux formater ses logs dans une application elle-même, ou à postériori, dans une phase de transformation ?

- Autres question à venir