

Projet Scala

Réalisé par :

ZIANE Hicham

DAHOUCHE Zainab

Encadré par :

Pr. Zyati

Mouhssin

Année académique :2021/2022

Filière : Data Engineer

Remerciements

Nous voudrions transmettre nos remerciements les plus sincères à notre professeur **Zyati Mouhssin**, notre encadreur dans ce projet pour sa précieuse contribution pédagogique et ses encouragements envers l'innovation et la persévérance.

Table des matières

I. Générateur de données logs.....	6
II. Test du Script python modifié.....	8
III. L'application Scala avec SBT qui joue le rôle de médiateur	9
IV. Ajouter les dépendances nécessaires.....	9
V. Le code scala.....	11
VI. Définir la structure des données	13
VII. Création d'une session spark	15
VIII. Lire le répertoire contenant les logs	16
IX. Enregistrer le flux de données en tant qu'un view	17
X. Enregistrement des Logs dans Elasticsearch	18
XI. Exécution de l'application	19
XII. Visualisation des logs avec Kibana	26
XIII. Définition de notre modèle d'index.....	26
XIV. Découvrir les données.....	27
XV. Visualiser les données.....	28
XVI. Affichage des visualisations dans un tableau de bord.....	31

Liste des figures

Figure 1:la partie originale du script	6
Figure 2:la partie modifiée du script.....	7
Figure 3:Fichiers Logs créés en temps réel	8
Figure 4:Contenu d'un fichier Log	8
Figure 5:fichier log	13
Figure 6:etat initial de la base de données	19
Figure 7:lancement de l'application scala qui joue le rôle de médiateur	20
Figure 8:lancement du script python	21
Figure 9:lignes log générées	21
Figure 10:generation des fichiers Logs	22
Figure 11:contenu d'un fichier log	23
Figure 12: à t = 0 secondes.....	24
Figure 13:enregistrement des logs dans elasticsearch	25
Figure 14:diagramme à barres de nombre de différents sites générés	28
Figure 15:diagramme en secteur de nombre de différents sites générés.....	29
Figure 16:diagramme linéaire de nombre de différents sites générés	29
Figure 17:visualisation de métriques des différents sites génères.....	30
Figure 18:visualisation de nombre de différents services	30
Figure 19:Dashboard.....	31

Générateur de données logs

Afin de générer les fichiers Log On a utilisé le script Python donné par notre professeur « **Zyati Mouhssin** » mais on a appliqué quelques modifications afin que notre script puisse nous générer à chaque 10 nouvelles lignes générées, un nouveau fichier log, cela nous va permettre par la suite de lire ces fichiers log générés par le script python modifié comme des batch venant en temps réel qu'on va les recevoir à travers Spark Streaming puis faire les traitements nécessaires.

```
33
34     log_File = open('/tmp/log-generator.Log', 'w')
35
36     count = 0
37     # infinite daemon
38     while True:
39         if(count > 1000):
40             count = 0;
41             time.sleep(5); #sleep 5 seconds between writes
42         else:
43             #Http = buildHTTP()
44             Http = getHTTP()
45             count = count + 1
46             Http = str(Http)
47             Url = buildURL()
48             Path = buildPath()
49             Ip = str(buildIP())
50
51             line = "HTTP " + Http + " " + Url + " " + Path + " " + Ip + "\n"
52             print line
53             log_File.write(line)
54
55     log_File.close()
56
```

Figure 1: la partie originale du script

```

35  j=0
36  ▼ while True:
37      count=1
38  ▼  while(count<=10):
39      #Http = buildHTTP()
40      Http = getHTTP()
41      Http = str(Http)
42      Url = buildURL()
43      Path = buildPath()
44      Ip = str(buildIP())
45      line = "HTTP " + Http + " " + Url + " " + Path + " " + Ip + "\n"
46      lines.append(line)
47      count=count+1
48      print(line)
49      time.sleep(1)
50  ▼  with open("Logs/Log_generator"+str(j)+".Log", 'w') as Log:
51  ▼      for linee in lines:
52          Log.write(linee)
53      lines=[]
54      j=j+1
55

```

Figure 2: la partie modifiée du script

Cette modification consiste simplement à ajouter un compteur qui compte jusqu'à 10 et une liste qu'on lui ajoute à chaque itération une ligne durant les 10 itérations, entre deux itérations successives en attend une seconde.

Lorsque le compteur atteint 10 itérations, on décharge la liste qui contient les 10 lignes dans un nouveau fichier log, puis on vide la liste et on recommence à nouveau.

Test du Script

Après l'exécution du script, voilà ce que ça donne après quelques minutes, A chaque 10 lignes nouvellement créées, un fichier log est généré :

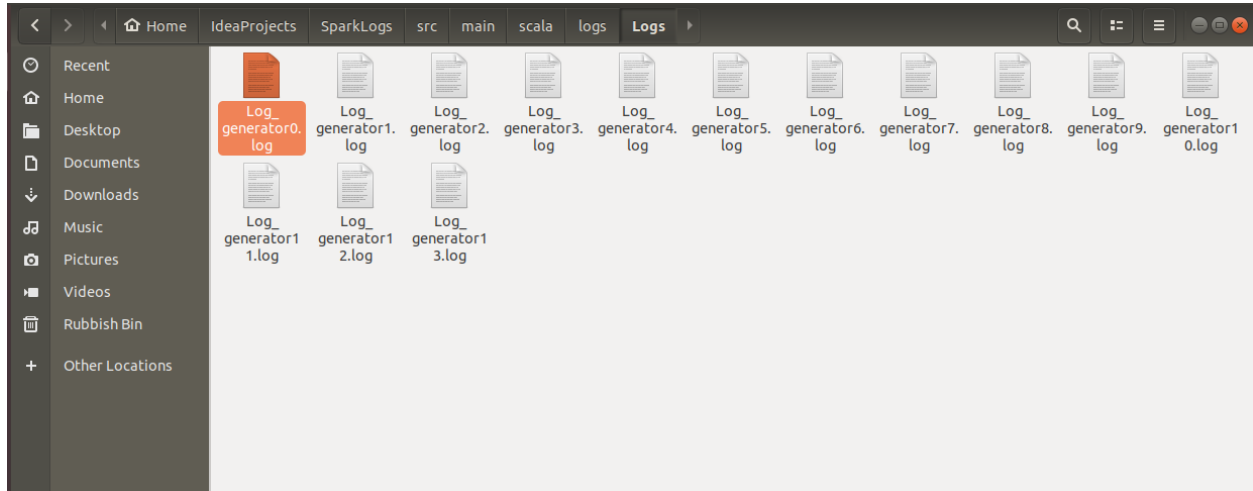


Figure 3: Fichiers Logs créés en temps réel

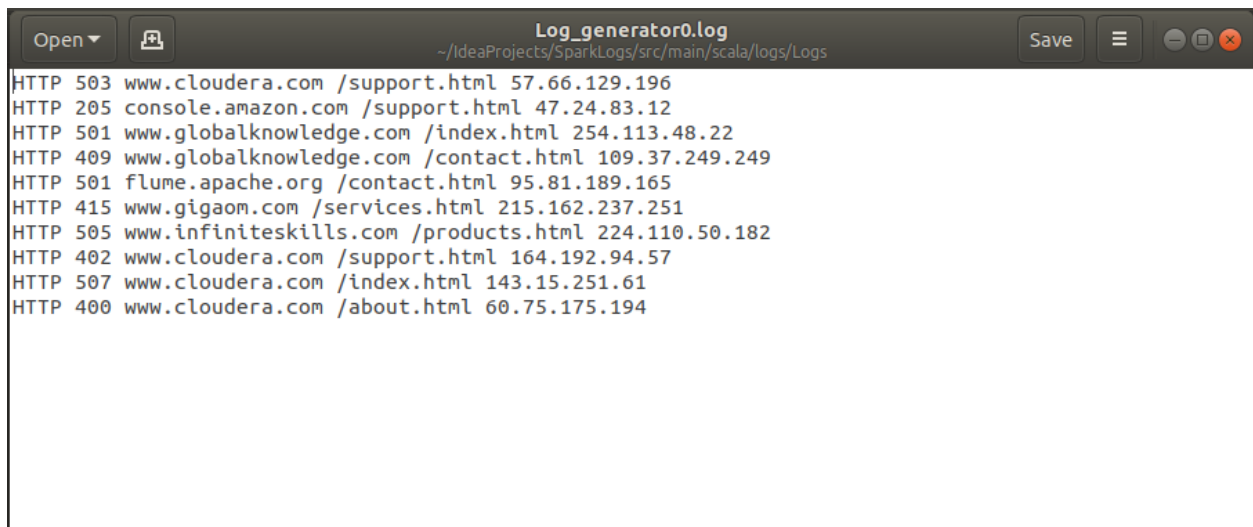


Figure 4: Contenu d'un fichier Log

L'application Scala avec SBT qui joue le rôle de médiateur

1-Ajouter les dépendances nécessaires

Avant de découvrir la partie code, on va savoir quelles sont les bibliothèques nécessaires pour l'exécution de notre projet scala.

Pour ce projet on aura besoin de 2 dépendances qu'on va les importer grâce à **sbt**.

Sbt est un outil populaire pour compiler, exécuter et tester des projets Scala de toute taille. **sbt** devient essentielle une fois qu'on crée des projets avec des dépendances ou plus d'un fichier de code.

Les 2 bibliothèques qu'on aura besoins dans le projet sont les suivantes :

a- spark-sql

Spark SQL apporte une prise en charge native de SQL à Spark et rationalise le processus d'interrogation des données stockées à la fois dans RDD.

On lui aura besoin dans ce projet pour effectuer du streaming structuré ainsi d'exécuter des requêtes SQL sur le flux des data frames.

b- elasticsearch-hadoop

Pour permettre l'indexation vers la base de données NoSql **Elasticsearch** et l'envoi de requêtes depuis **Elasticsearch**.

Voilà le fichier **build.sbt** :

```
ThisBuild / version := "0.1.0-SNAPSHOT"

ThisBuild / scalaVersion := "2.11.12"

lazy val root = (project in file("."))
  .settings(
    name := "SparkLogs"
  )

// https://mvnrepository.com/artifact/org.apache.spark/spark-sql
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.3"

// https://mvnrepository.com/artifact/org.elasticsearch/elasticsearch-hadoop
libraryDependencies += "org.elasticsearch" % "elasticsearch-hadoop" %
"6.4.1"
```

Le code Scala

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.{StringType, StructField,
StructType}
import org.elasticsearch.hadoop.cfg.ConfigurationOptions

object Stream2 extends App {

  val schemaT= StructType(List(

    StructField("protocole",StringType,true),
    StructField("status",StringType,true),
    StructField("url",StringType,true),
    StructField("path",StringType,true),
    StructField("ip",StringType,true)
  ))

  val sparkSession = SparkSession.builder()
    .config(ConfigurationOptions.ES_NODES, "localhost")
    .config(ConfigurationOptions.ES_PORT, "9200")
    .master("local[*]")
    .appName("Logs-Streaming")
    .getOrCreate()
```

```
val streamDF = sparkSession.readStream.option("delimiter"
, " ")

.schema(schemaT).csv("/home/dba/IdeaProjects/SparkLogs/src/main/scala/l
ogs/Logs")

    streamDF.createOrReplaceTempView("Logs")

val outDF = sparkSession.sql("select * from Logs")

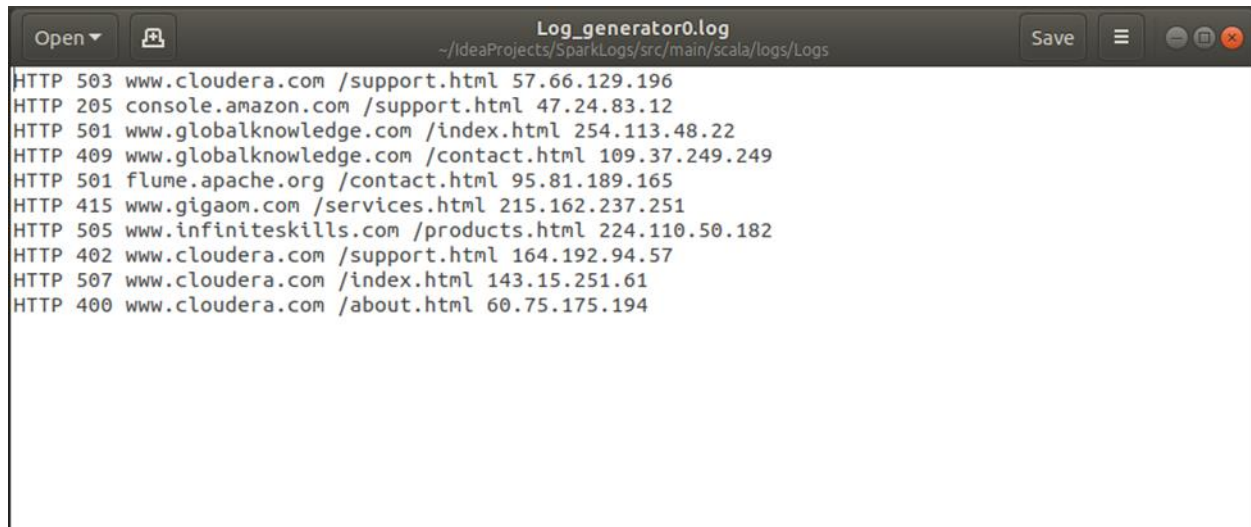
outDF
    .writeStream
    .outputMode("append")
    .format("org.elasticsearch.spark.sql")
    .option("checkpointLocation",
"/home/dba/IdeaProjects/SparkLogs/src/main/scala/temp")
    .start("sparklogs/logs").awaitTermination()

}
```

1-définir la structure des données

La première question qu'on doit se poser c'est quelle est la structure de données qu'on attend de les recevoir à travers Spark streaming puisqu'il s'agit d'un streaming structuré.

Prenons un parmi nos fichier logs pour examiner sa structure.



```
Log_generator0.log
~/IdeaProjects/SparkLogs/src/main/scala/logs/Logs

HTTP 503 www.cloudera.com /support.html 57.66.129.196
HTTP 205 console.amazon.com /support.html 47.24.83.12
HTTP 501 www.globalknowledge.com /index.html 254.113.48.22
HTTP 409 www.globalknowledge.com /contact.html 109.37.249.249
HTTP 501 flume.apache.org /contact.html 95.81.189.165
HTTP 415 www.gigaom.com /services.html 215.162.237.251
HTTP 505 www.infinitemskills.com /products.html 224.110.50.182
HTTP 402 www.cloudera.com /support.html 164.192.94.57
HTTP 507 www.cloudera.com /index.html 143.15.251.61
HTTP 400 www.cloudera.com /about.html 60.75.175.194
```

Figure 5:fichier log

On voit bien qu'une ligne de données est formée par 6 champs séparés par des espaces.

1 er champ : représente le protocole utilisé

2 -ème champ : représente le statut

3 -ème champ : représente l'url

4 -ème champ : représente le path

5 -ème champ : représente l'adresse ip

Pour définir cette structure dans notre code scala on aura recourt à un type de données appelé **StructType**.

StructType est un type de données intégré qui est une collection de **champs structurés**, ça nous permet de définir un schéma pour nos données d'entrées en temps réels.

Maintenant on ajout nos champs (protocol, status,url,path et ip) à l'intérieur du **StructType** en créant pour chaque champ un nouveau **StructField** qui prend comme paramètres le nom du champ et son type de données.

```
val schemaT= StructType(List(  
    StructField("protocole",StringType,true),  
    StructField("status",StringType,true),  
    StructField("url",StringType,true),  
    StructField("path",StringType,true),  
    StructField("ip",StringType,true)  
))
```

Une valeur booléen **true** est aussi ajouté comme paramètre, car si une donnée est **null** on aura quand même un flux de sortie.

Notre structure de données est maintenant créée.

2-Créer une session spark

La deuxième étape consiste à créer une session spark ou ce qui est connue par SparkSession

SparkSession crée un nouveau **SparkContext** qui est le point d'entrée de toute application Spark et utilisé pour accéder à toutes les fonctionnalités de Spark et a besoin d'un sparkConf qui avait toutes les configurations et tous les paramètres du cluster pour créer un objet Spark Context.

On crée tout d'abord une session spark en définissant :

- 1) un nom pour l'application, qui sera affiché dans l'interface utilisateur Web Spark → `appName("Logs-Streaming")`
- 2) Définir le maître auquel se connecter, comme "local" pour s'exécuter localement, "local[*]" pour s'exécuter localement avec autant de threads que de processeurs disponibles.
- 3) Et pour ajouter **elasticsearch** comme destination pour **Spark Structured Streaming**, nous devons ajouter la configuration d'elasticsearch dans l'objet de SparkSession.

Comme vu dans le code, `config(ConfigurationOptions.ES_NODES, "localhost")` et `config(ConfigurationOptions.ES_PORT, "9200")` représente la configuration d'elasticsearch à savoir l'adresse du nœud Elasticsearch auquel se connecter, dans notre cas c'est localhost ainsi que le port HTTP/REST par défaut utilisé pour la connexion à Elasticsearch qui est **9200**.

- 4) Finalement `getOrCreate()` pour Obtenir une SparkSession existante ou, s'il n'y en a pas, en crée une nouvelle basée sur les options déjà définies .

3-lire le répertoire contenant les logs

Maintenant qu'on a défini le schéma des données ainsi que la session Spark, l'étape suivante consiste à lire les logs venant en temps réel qui sont à l'intérieure du répertoire qu'on a nommé Logs

Pour Traduire cette lecture en temps réel des Logs dans notre code, on utilise la fonction **readstream** de l'objet **spark** qu'on a créé et qui sert à récupérer le flux de données structuré en temps réel en donnant comme paramètres :

1)le délimiteur qui sépare entre les champs des enregistrements, dans notre cas c'est espace (" ").

Ceci est traduit dans le code par **option("delimiter"," ")**.

2)le schéma de données qu'on a défini précédemment **“schemaT”**

Ceci est traduit dans le code par **schema(schemaT)**.

3)le chemin absolu du répertoire qui contient les logs

Ceci est traduit dans le code par :

csv("/home/dba/IdeaProjects/SparkLogs/src/main/scala/logs/Logs")

4-Enregistrer le flux de données en tant qu'un view

Après la récupération du flux de données vue dans la section 3.

On doit maintenant enregistrer ce flux sous la forme d'un **view** afin d'effectuer des requêtes SQL.

Ce **view** est pratiquement une table, mais elle doit être évaluée car elle n'est pas matérialisée dans des fichiers.

Pour faire cela on a recourt à la fonction **createOrReplaceTempView()**

Appliqué à notre flux de données représenté dans le code par **streamDF**

```
streamDF.createOrReplaceTempView("Logs")
val outDF = sparkSession.sql("select * from Logs")
```

"Logs" est le nom qu'on a donné à ce view.

outDF représente un streaming data frame équivalent à une table dans une base de données relationnelle qui possède comme colonnes (protocol, status,url,path et ip) et comme enregistrements les lignes des logs générées.

5-Enregistrement des Logs dans Elasticsearch

Nous arrivons maintenant à la dernière étape dans notre application médiatrice où nous allons enregistrer les logs dans la base de données NoSql **elasticsearch**

Maintenant qu'on a le flux de **dataframes** qu'on a récupéré dans la partie 3 il nous reste qu'à l'enregistrer dans **elasticsearch**, pour cela nous utiliserons la fonction **writeStream** en passant certains options/arguments.

1) *outputMode("append")*

Pour ajouter que les nouvelles lignes dans le document **logs** de l'index **sparklogs** dans elasticsearch.

2) *format("org.elasticsearch.spark.sql")*

Pour indiquer que la sortie pour le flux de **dataframes** est la base de données **elasticsearch**

3) *option("checkpointLocation", "/home/dba/IdeaProjects/SparkLogs/src/main/scala/temp")*

Lors de l'exécution d'une requête de diffusion en continu, la plupart des sources et des récepteurs nécessitent qu'on spécifie un **"checkpointLocation"** afin de conserver l'état du travail. En cas d'interruption, le lancement d'une nouvelle requête de streaming avec le même emplacement de point de contrôle récupérera l'état du travail et reprendra là où il s'était arrêté.

4) *start("sparklogs/logs")*

Pour commencer l'écriture en indexons en permanence l'ensemble de données dans Elasticsearch sous l'index **Sparklogs** et le document **logs**.

5) *awaitTermination()*

Pour attendre juste le signal de terminaison de l'utilisateur, Lorsque ce dernier est reçu, le contexte de diffusion en continu sera arrêté.

6-Execution de l'application

L'index **sparklogs** dans **elasticsearch** est initialement vide, rien n'est encore généré.

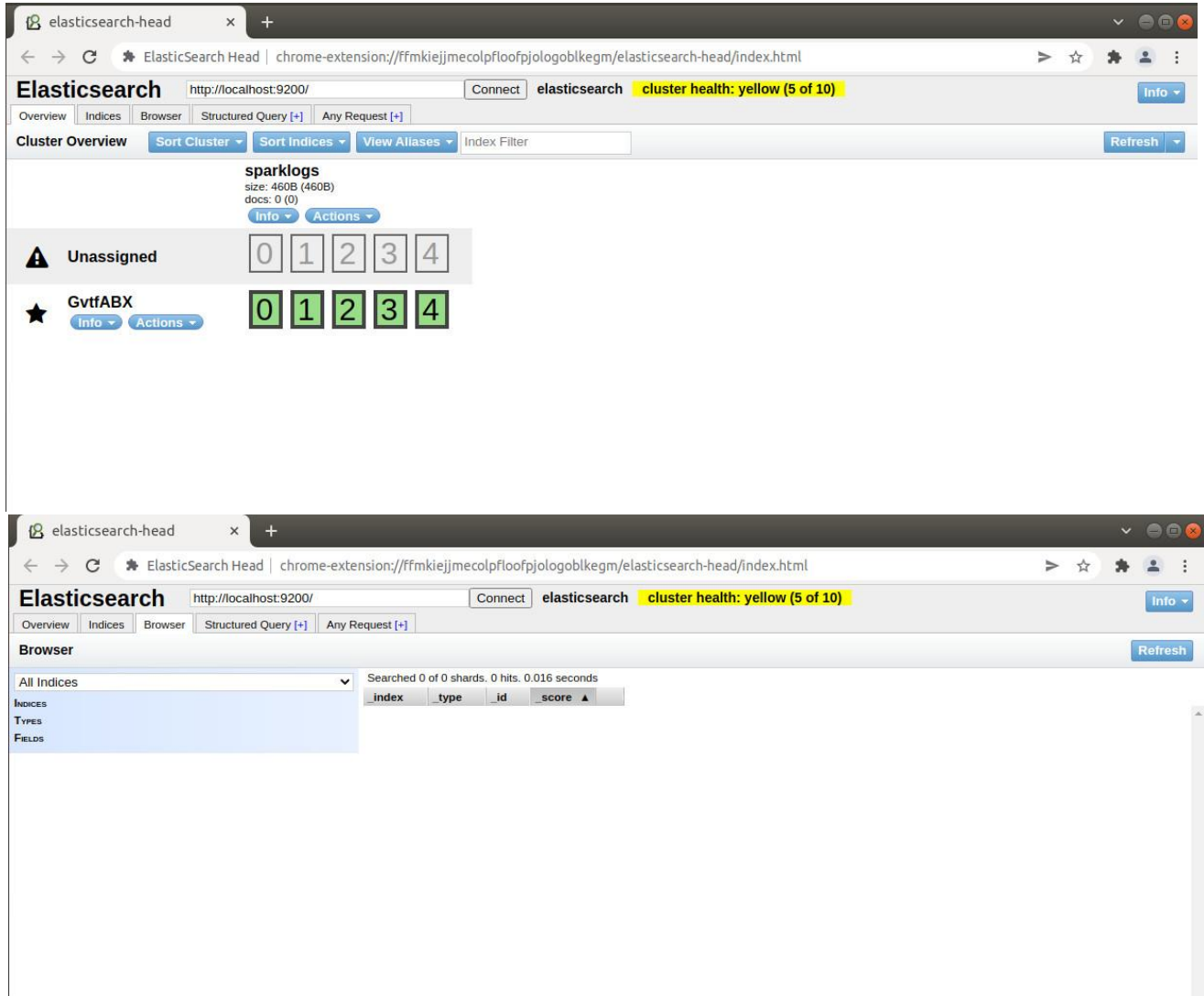


Figure 6:etat initial de la base de données

Premièrement on lance l'application Scala avec SBT qui joue le rôle de médiateur pour attendre les logs qui vont être créés après qu'on lance le script python qu'on a modifié.

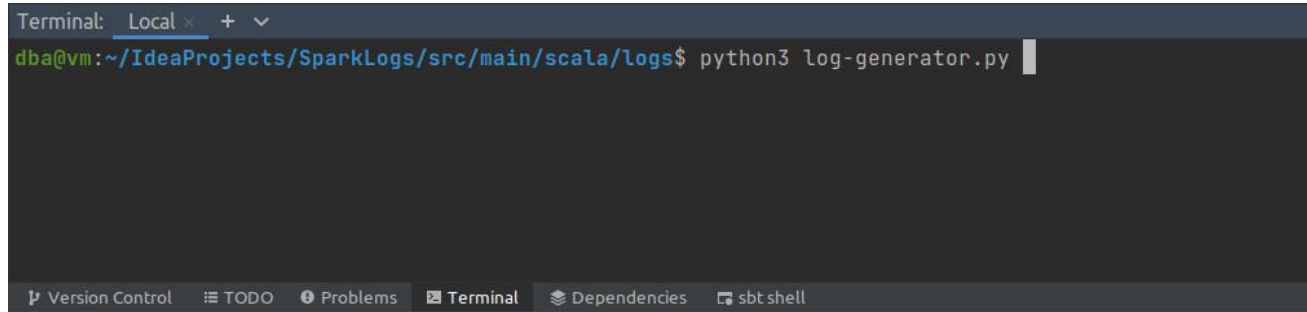
```

2  import org.apache.spark.sql.SparkSession
3  import org.apache.spark.sql.types.{StringType, StructField, StructType}
4  import org.elasticsearch.hadoop.cfg.ConfigurationOptions
5
6  object Stream2 extends App {
7
8
9      val schemaT= StructType(List(
10
11          StructField("protocole",StringType,true),
12          StructField("status",StringType,true),
13          StructField("url",StringType,true),
14          StructField("path",StringType,true),

```

Figure 7:lancement de l'application scala qui joue le rôle de médiateur

Ensuite on lance le script python qu'on a modifié



```
Terminal: Local x + v
dba@vm:~/IdeaProjects/SparkLogs/src/main/scala/logs$ python3 log-generator.py
```

Figure 8:lancement du script python

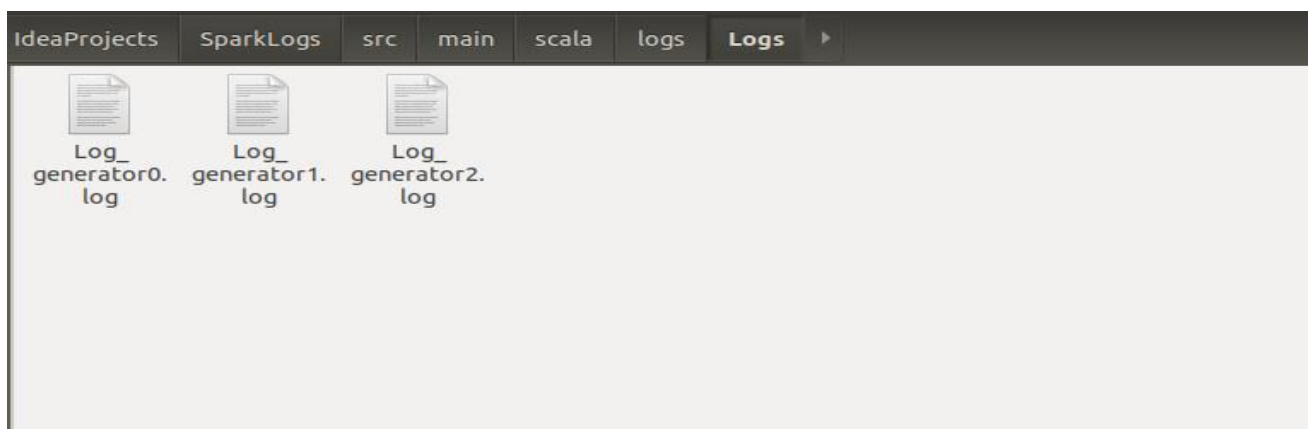
Par la suite les lignes du fichier log commencent à être générées, on peut voir ça dans le terminal



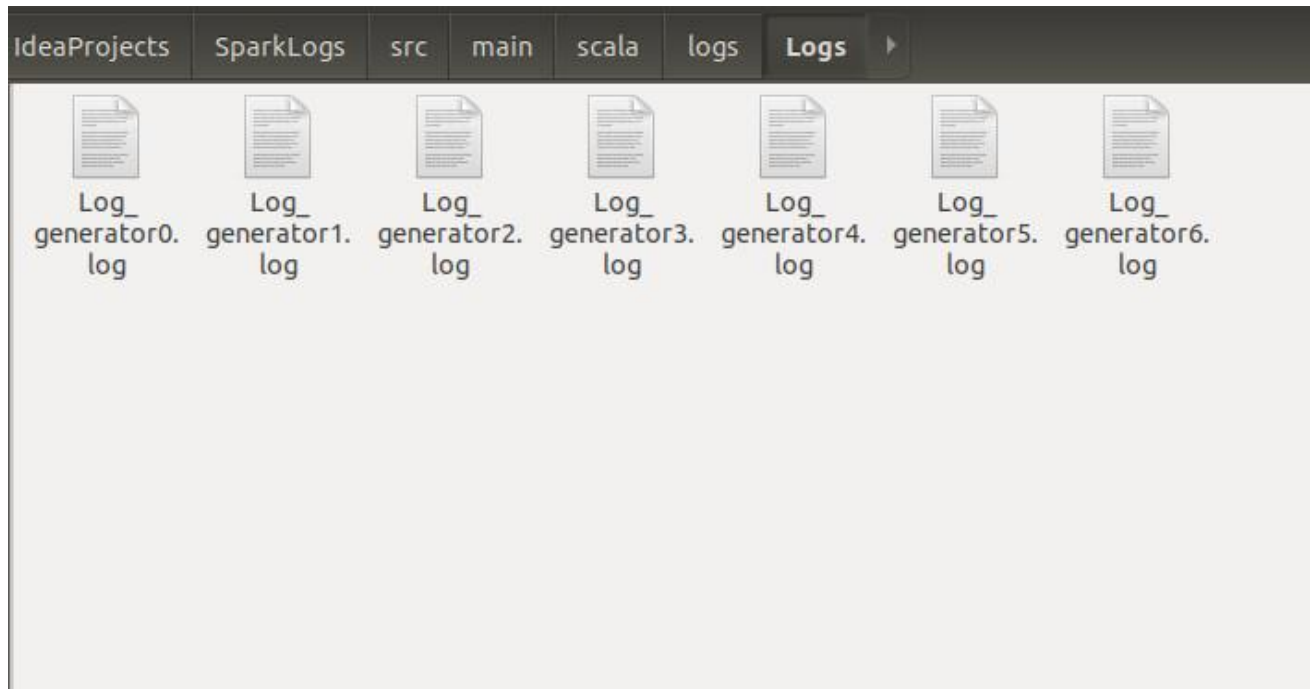
```
Terminal: Local x + v
dba@vm:~/IdeaProjects/SparkLogs/src/main/scala/logs$ python3 log-generator.py
HTTP 424 www.infinitemkills.com /about.html 83.209.185.160
HTTP 504 www.globalknowledge.com /contact/submit.html 188.98.93.61
HTTP 505 console.amazon.com /contact.html 99.202.30.250
```

Figure 9:lignes log générées

Voilà le contenu du répertoire Logs après lancement du script



Après quelques secondes



•

•

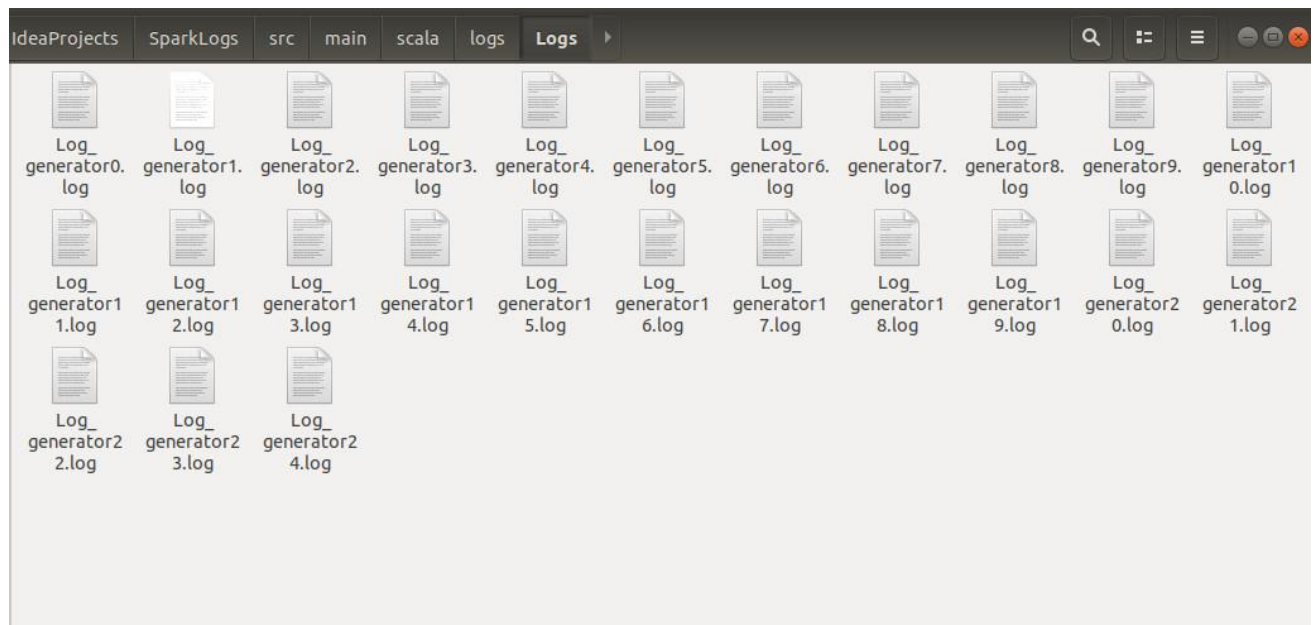
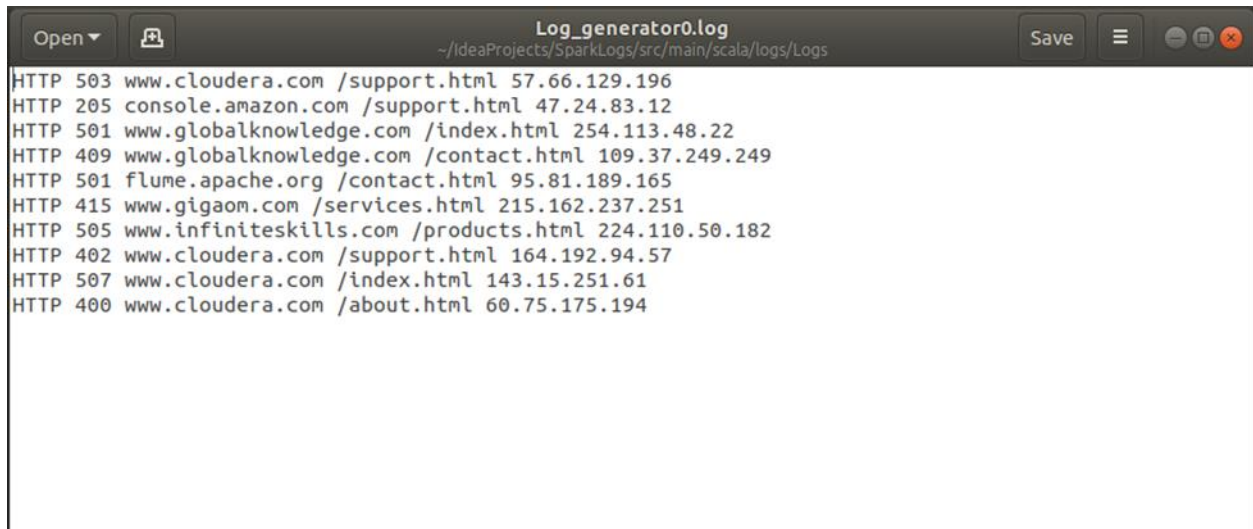


Figure 10:generation des fichiers Logs

Et ainsi de suite....

Voilà le contenu d'un fichier Log :



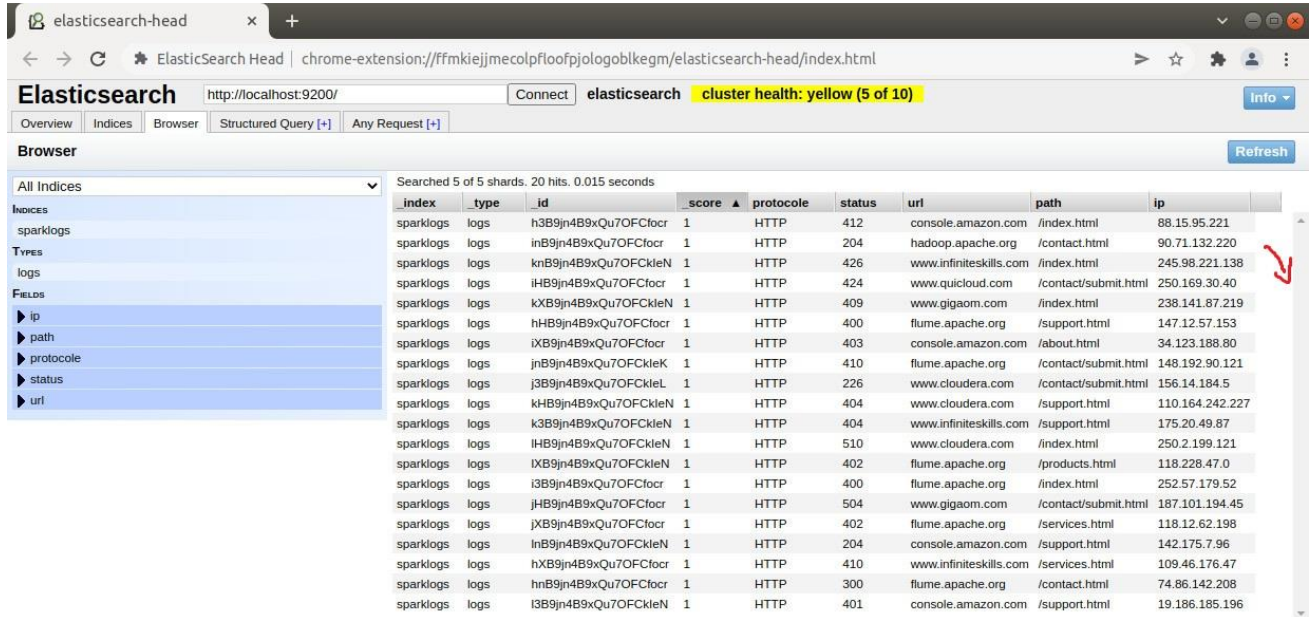
The screenshot shows a text editor window with the title 'Log_generator0.log' and a file path '~/.IdeaProjects/SparkLogs/src/main/scala/logs/Logs'. The window contains a list of HTTP log entries, each on a new line. The entries are as follows:

```
HTTP 503 www.cloudera.com /support.html 57.66.129.196
HTTP 205 console.amazon.com /support.html 47.24.83.12
HTTP 501 www.globalknowledge.com /index.html 254.113.48.22
HTTP 409 www.globalknowledge.com /contact.html 109.37.249.249
HTTP 501 flume.apache.org /contact.html 95.81.189.165
HTTP 415 www.gigaom.com /services.html 215.162.237.251
HTTP 505 www.infinitieskills.com /products.html 224.110.50.182
HTTP 402 www.cloudera.com /support.html 164.192.94.57
HTTP 507 www.cloudera.com /index.html 143.15.251.61
HTTP 400 www.cloudera.com /about.html 60.75.175.194
```

Figure 11: contenu d'un fichier log

Passant maintenant à **elasticsearch** pour savoir ce qui se passe dans l'index **sparklogs**

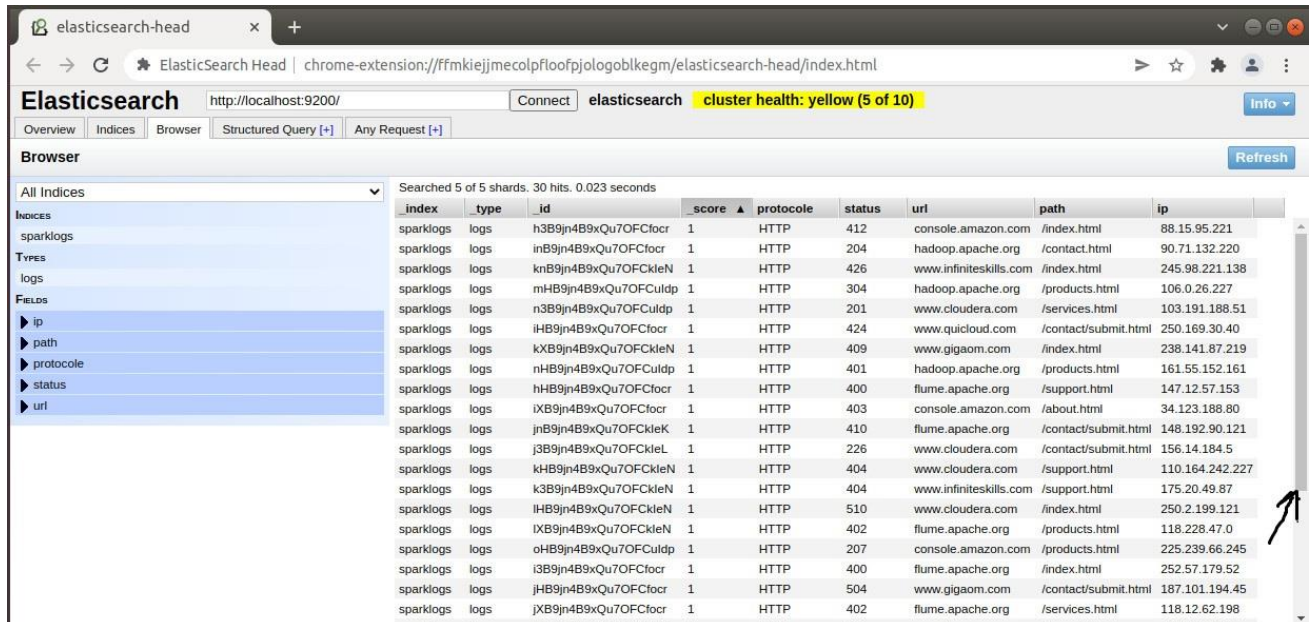
Voilà le contenu de l'index **sparklogs** après lancement du script



index	type	id	score	protocole	status	url	path	ip
sparklogs	logs	h3B9jn4B9xQu7OFCfocr	1	HTTP	412	console.amazon.com	/index.html	88.15.95.221
sparklogs	logs	inB9jn4B9xQu7OFCfocr	1	HTTP	204	hadoop.apache.org	/contact.html	90.71.132.220
sparklogs	logs	knB9jn4B9xQu7OFCkleN	1	HTTP	426	www.infinitemskills.com	/index.html	245.98.221.138
sparklogs	logs	iHB9jn4B9xQu7OFCfocr	1	HTTP	424	www.quicloud.com	/contact/submit.html	250.169.30.40
sparklogs	logs	kXB9jn4B9xQu7OFCkleN	1	HTTP	409	www.gigaom.com	/index.html	238.141.87.219
sparklogs	logs	hXB9jn4B9xQu7OFCfocr	1	HTTP	400	flume.apache.org	/support.html	147.12.57.153
sparklogs	logs	iXB9jn4B9xQu7OFCfocr	1	HTTP	403	console.amazon.com	/about.html	34.123.188.80
sparklogs	logs	jnB9jn4B9xQu7OFCkleK	1	HTTP	410	flume.apache.org	/contact/submit.html	148.192.90.121
sparklogs	logs	j3B9jn4B9xQu7OFCkleL	1	HTTP	226	www.cloudera.com	/contact/submit.html	156.14.184.5
sparklogs	logs	kHB9jn4B9xQu7OFCkleN	1	HTTP	404	www.cloudera.com	/support.html	110.164.242.227
sparklogs	logs	k3B9jn4B9xQu7OFCkleN	1	HTTP	404	www.infinitemskills.com	/support.html	175.20.49.87
sparklogs	logs	iHB9jn4B9xQu7OFCkleN	1	HTTP	510	www.cloudera.com	/index.html	250.2.199.121
sparklogs	logs	iXB9jn4B9xQu7OFCkleN	1	HTTP	402	flume.apache.org	/products.html	118.228.47.0
sparklogs	logs	i3B9jn4B9xQu7OFCfocr	1	HTTP	400	flume.apache.org	/index.html	252.57.179.52
sparklogs	logs	jHB9jn4B9xQu7OFCfocr	1	HTTP	504	www.gigaom.com	/contact/submit.html	187.101.194.45
sparklogs	logs	jXB9jn4B9xQu7OFCfocr	1	HTTP	402	flume.apache.org	/services.html	118.12.62.198
sparklogs	logs	lnB9jn4B9xQu7OFCkleN	1	HTTP	204	console.amazon.com	/support.html	142.175.7.96
sparklogs	logs	hXB9jn4B9xQu7OFCfocr	1	HTTP	410	www.infinitemskills.com	/services.html	109.46.176.47
sparklogs	logs	hnB9jn4B9xQu7OFCfocr	1	HTTP	300	flume.apache.org	/contact.html	74.86.142.208
sparklogs	logs	i3B9jn4B9xQu7OFCkleN	1	HTTP	401	console.amazon.com	/support.html	19.186.185.196

Figure 12: à $t = 0$ secondes

Si on attend quelques secondes puis on rafraîchit voilà ce qu'on trouve



index	type	id	score	protocole	status	url	path	ip
sparklogs	logs	h3B9jn4B9xQu7OFCfocr	1	HTTP	412	console.amazon.com	/index.html	88.15.95.221
sparklogs	logs	inB9jn4B9xQu7OFCfocr	1	HTTP	204	hadoop.apache.org	/contact.html	90.71.132.220
sparklogs	logs	knB9jn4B9xQu7OFCkleN	1	HTTP	426	www.infinitemskills.com	/index.html	245.98.221.138
sparklogs	logs	mHB9jn4B9xQu7OFCculdp	1	HTTP	304	hadoop.apache.org	/products.html	106.0.26.227
sparklogs	logs	n3B9jn4B9xQu7OFCculdp	1	HTTP	201	www.cloudera.com	/services.html	103.191.188.51
sparklogs	logs	iHB9jn4B9xQu7OFCfocr	1	HTTP	424	www.quicloud.com	/contact/submit.html	250.169.30.40
sparklogs	logs	kXB9jn4B9xQu7OFCkleN	1	HTTP	409	www.gigaom.com	/index.html	238.141.87.219
sparklogs	logs	nHB9jn4B9xQu7OFCculdp	1	HTTP	401	hadoop.apache.org	/products.html	161.55.152.161
sparklogs	logs	hHB9jn4B9xQu7OFCfocr	1	HTTP	400	flume.apache.org	/support.html	147.12.57.153
sparklogs	logs	iXB9jn4B9xQu7OFCfocr	1	HTTP	403	console.amazon.com	/about.html	34.123.188.80
sparklogs	logs	jnB9jn4B9xQu7OFCkleK	1	HTTP	410	flume.apache.org	/contact/submit.html	148.192.90.121
sparklogs	logs	j3B9jn4B9xQu7OFCkleL	1	HTTP	226	www.cloudera.com	/contact/submit.html	156.14.184.5
sparklogs	logs	kHB9jn4B9xQu7OFCkleN	1	HTTP	404	www.cloudera.com	/support.html	110.164.242.227
sparklogs	logs	k3B9jn4B9xQu7OFCkleN	1	HTTP	404	www.infinitemskills.com	/support.html	175.20.49.87
sparklogs	logs	iHB9jn4B9xQu7OFCkleN	1	HTTP	510	www.cloudera.com	/index.html	250.2.199.121
sparklogs	logs	iXB9jn4B9xQu7OFCkleN	1	HTTP	402	flume.apache.org	/products.html	118.228.47.0
sparklogs	logs	oHB9jn4B9xQu7OFCculdp	1	HTTP	207	console.amazon.com	/products.html	225.239.66.245
sparklogs	logs	i3B9jn4B9xQu7OFCfocr	1	HTTP	400	flume.apache.org	/index.html	252.57.179.52
sparklogs	logs	jHB9jn4B9xQu7OFCfocr	1	HTTP	504	www.gigaom.com	/contact/submit.html	187.101.194.45
sparklogs	logs	jXB9jn4B9xQu7OFCfocr	1	HTTP	402	flume.apache.org	/services.html	118.12.62.198

index	type	_id	score	protocole	status	url	path	ip
sparklogs	logs	h3B9jn4B9xQu7OFCfocr	1	HTTP	412	console.amazon.com	/index.html	88.15.95.221
sparklogs	logs	inB9jn4B9xQu7OFCfocr	1	HTTP	204	hadoop.apache.org	/contact.html	90.71.132.220
sparklogs	logs	knB9jn4B9xQu7OFCkleN	1	HTTP	426	www.infinitieskills.com	/index.html	245.98.221.138
sparklogs	logs	mHB9jn4B9xQu7OFCuldP	1	HTTP	304	hadoop.apache.org	/products.html	106.0.26.227
sparklogs	logs	n3B9jn4B9xQu7OFCuldP	1	HTTP	201	www.cloudera.com	/services.html	103.191.188.51
sparklogs	logs	o3B9jn4B9xQu7OFC3oe_	1	HTTP	200	www.infinitieskills.com	/contact.html	163.196.193.136
sparklogs	logs	pHB9jn4B9xQu7OFC3oe_	1	HTTP	302	www.gigamon.com	/products.html	83.156.7.244
sparklogs	logs	qHB9jn4B9xQu7OFC3oe_	1	HTTP	307	www.quickcloud.com	/services.html	236.144.97.75
sparklogs	logs	qnB9jn4B9xQu7OFC3oe_	1	HTTP	504	flume.apache.org	/index.html	93.94.159.233
sparklogs	logs	mB-jn4B9xQu7OFCBYfp	1	HTTP	204	flume.apache.org	/about.html	235.32.125.203
sparklogs	logs	r3B-jn4B9xQu7OFCBYfp	1	HTTP	100	www.globalknowledge.com	/products.html	176.119.1.4
sparklogs	logs	sHB-jn4B9xQu7OFCBYfp	1	HTTP	304	flume.apache.org	/index.html	90.91.220.224
sparklogs	logs	sXB-jn4B9xQu7OFCBYfp	1	HTTP	416	www.globalknowledge.com	/contact/submit.html	86.252.155.201
sparklogs	logs	uXB-jn4B9xQu7OFCCLIF	1	HTTP	414	www.globalknowledge.com	/products.html	182.231.198.119
sparklogs	logs	unB-jn4B9xQu7OFCCLIF	1	HTTP	200	www.globalknowledge.com	/services.html	76.51.180.56
sparklogs	logs	u3B-jn4B9xQu7OFCCLIF	1	HTTP	201	www.globalknowledge.com	/index.html	71.96.215.190
sparklogs	logs	v3B-jn4B9xQu7OFCCLIF	1	HTTP	424	hadoop.apache.org	/support.html	16.19.117.129
sparklogs	logs	wXB-jn4B9xQu7OFCU4fn	1	HTTP	409	hadoop.apache.org	/products.html	82.181.214.153
sparklogs	logs	wnB-jn4B9xQu7OFCU4fn	1	HTTP	507	www.globalknowledge.com	/index.html	38.55.29.155
sparklogs	logs	x3B-jn4B9xQu7OFCU4fn	1	HTTP	510	www.quickcloud.com	/index.html	33.98.58.118

Figure 13:enregistrement des logs dans elasticsearch

On remarque bien que à chaque fois qu'on clique sur Refresh de nouveaux enregistrements apparaissent.

On conclut que notre application Scala avec SBT joue bien le rôle de médiateur.

Visualisation des logs avec Kibana

Le but final est de créer un tableau de bord qui contient l'ensemble de visualisation des données logs, alors on va suivre les étapes suivantes :

- Définir un modèle d'index
- Découvrez et explorez les données
- Visualisez les données
- Ajouter des visualisations à un tableau de bord

1-Définition de notre modèle d'index

Les modèles d'index indiquent à Kibana quels index Elasticsearch vous souhaitez explorer.

Nous allons créer un modèle pour les données Logs générées par le script log-generator.py et qui ont un index nommé : sparklogs

The screenshot shows the Kibana Management interface for creating an index pattern. The left sidebar contains the Kibana logo and navigation links: Discover, Visualize, Dashboard, Timelion, APM, Dev Tools, Monitoring, and Management (which is highlighted). The main content area has a breadcrumb 'Management / Kibana' and links for 'Index Patterns', 'Saved Objects', 'Reporting', and 'Advanced Settings'. A warning box states: 'Warning: No default index pattern. You must select or create one to continue.' The title is 'Create index pattern' with a subtext: 'Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.' There is a toggle for 'Include system indices' which is currently turned off. The main section is 'Step 1 of 2: Define index pattern'. It features an 'Index pattern' input field containing 'sparklogs*'. Below the input, it says: 'You can use a * as a wildcard in your index pattern. You can't use spaces or the characters \, /, ?, ", <, >, |.' A 'Next step' button is on the right. A success message reads: 'Success! Your Index pattern matches 1 index.' Below this, the index 'sparklogs' is listed. At the bottom, there is a 'Rows per page: 10' dropdown menu.

kibana

Discover

Visualize

Dashboard

Timelion

APM

Dev Tools

Monitoring

Management

Collapse

Management / Kibana

Index Patterns Saved Objects Reporting Advanced Settings

Warning

No default index pattern. You must select or create one to continue.

Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

☐ Include system indices

Step 2 of 2: Configure settings

You've defined **sparklogs*** as your index pattern. Now you can specify some settings before we create it. The indices which match this index pattern don't contain any time fields.

[Show advanced options](#)

[< Back](#) [Create Index pattern](#)

Management / Kibana

Index Patterns Saved Objects Reporting Advanced Settings

+ Create Index Pattern

★ sparklogs*

★ sparklogs*

★

↺

🗑

This page lists every field in the **sparklogs*** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch [Mapping API](#)

Fields (15)

Scripted fields (0)

Source filters (0)

🔍 Filter

All field types ▼

Name	Type	Format	Searchable	Aggregatable	Excluded
_id	string		●	●	
_index	string		●	●	
_score	number				
_source	_source				
_type	string		●	●	

2-Découvrir les données

Dans notre cas on va s'intéresser aux nombres de logs en les catégorisant suivant les différents **url** et **path** , on va utiliser ces deux champs pour la partie de visualisation qu'on va voir par la suite

3-Visualiser vos données

Dans la partie Visualize, nous pouvons visualiser nos données à l'aide de divers graphiques, tableaux et cartes, etc.

Nous allons créer cinq visualisations : un graphique à secteurs, trois graphiques à barres, une partie dédiée aux métriques, on suit les étapes suivantes :

- Ouvrir Visualize.
- cliquer sur new visualisation
- choisir le type de visualisation
- définir les champs à agréger
- enregistrer la visualisation

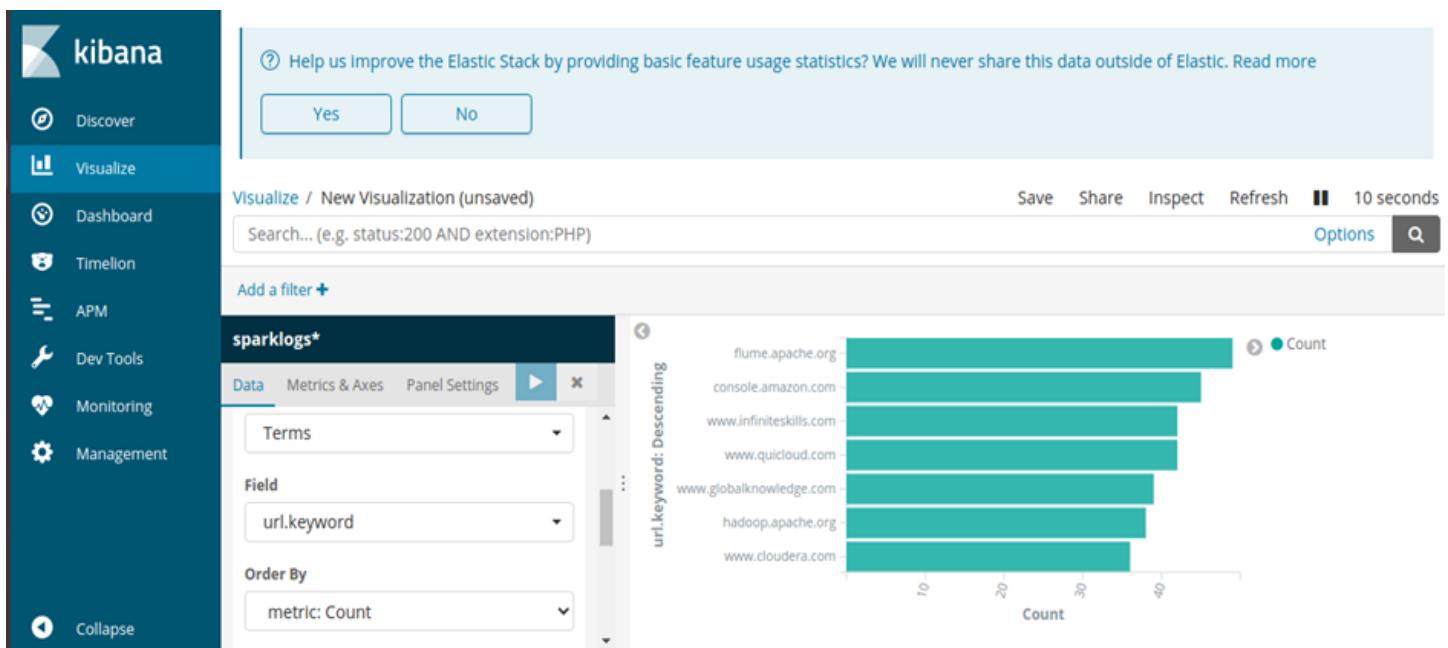


Figure 14:diagramme à barres de nombre de différents sites générés

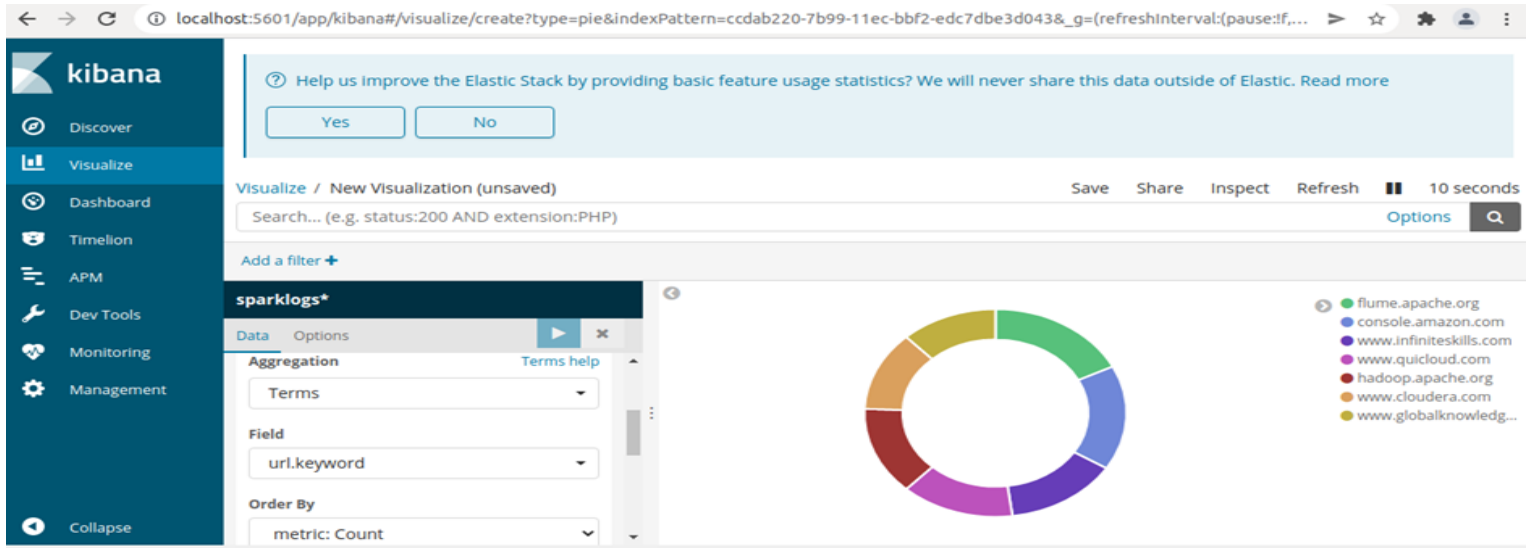


Figure 15: diagramme en secteur de nombre de différents sites générés

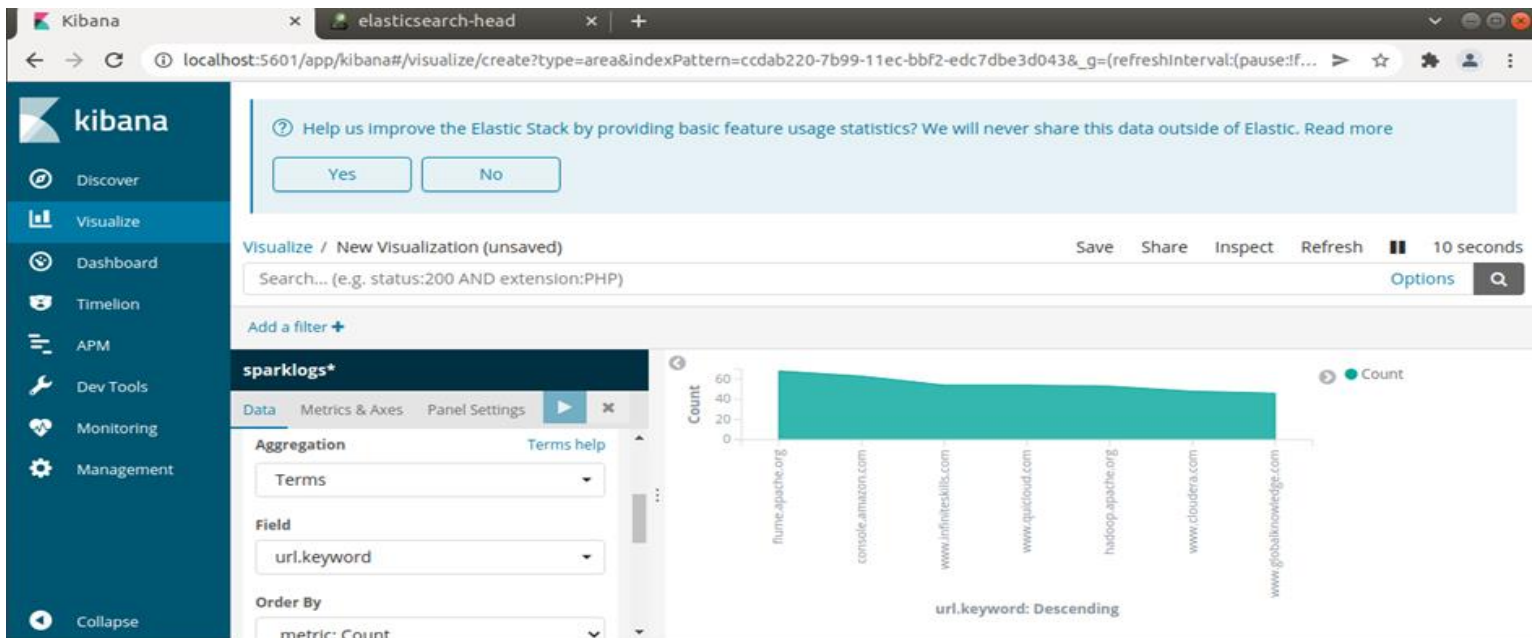


Figure 16: diagramme linéaire de nombre de différents sites générés

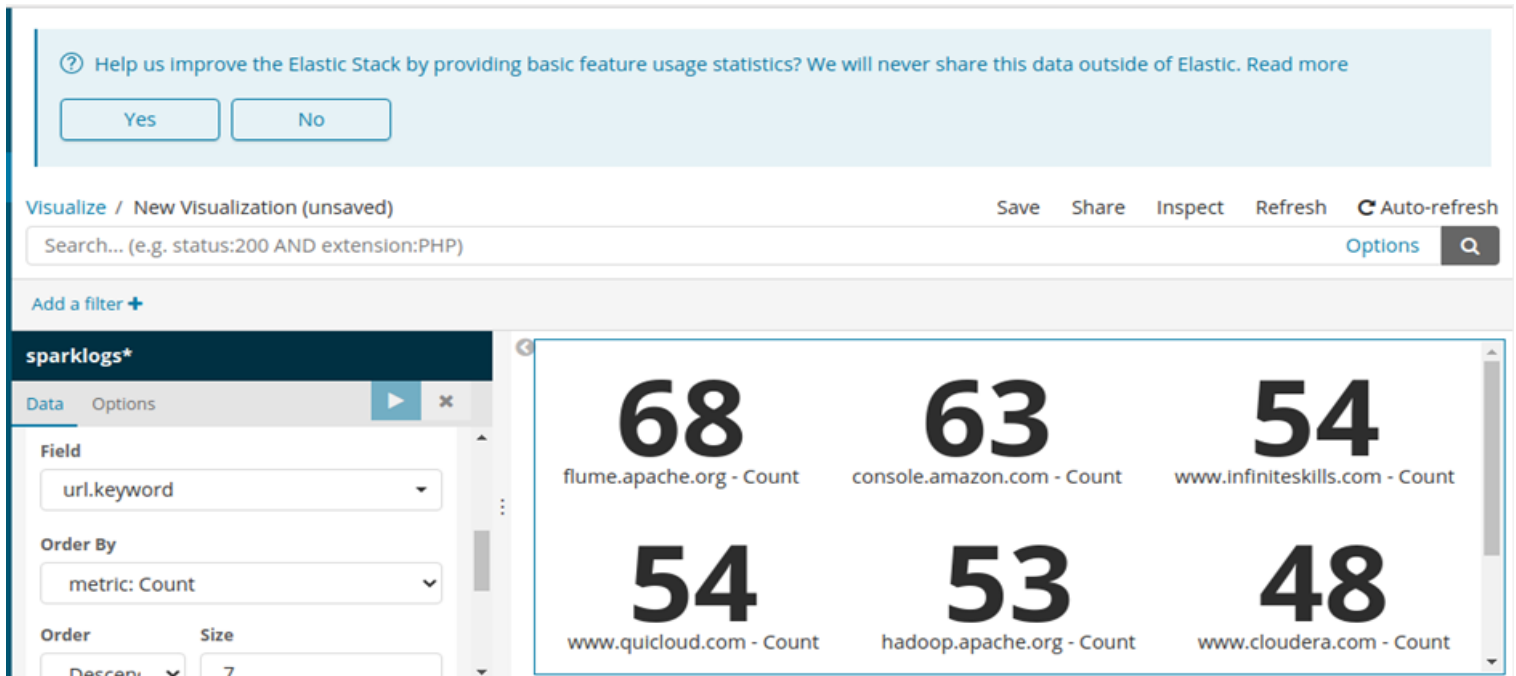


Figure 17: visualisation de métriques des différents sites génères

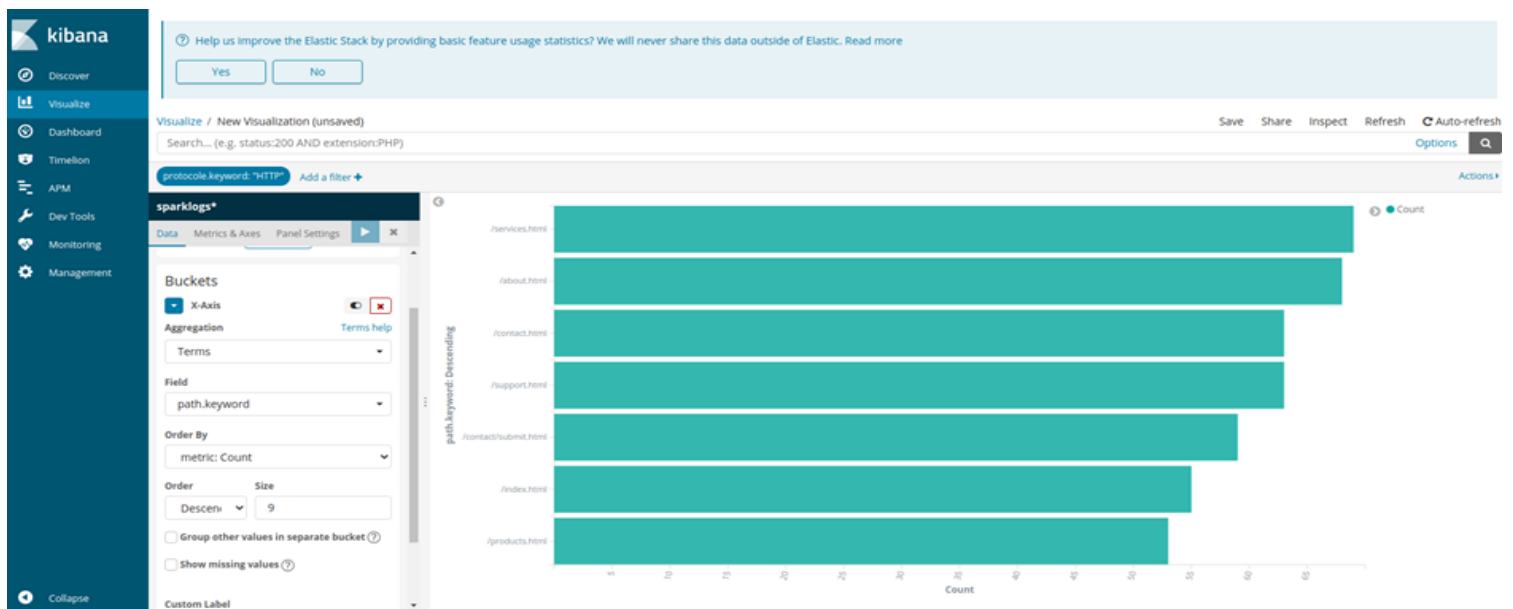


Figure 18: visualisation de nombre de différents services

4-Affichage de nos visualisations dans un tableau de bord

Un tableau de bord est une collection de visualisations que vous pouvez organiser et partager.

Nous allons créer un tableau de bord qui contient les visualisations que nous avons enregistré au cours de ce cette partie.

- Ouvrir le Dashboard
- Cliquez sur
- Create new dashboard
- Cliquez sur Add
- Ajouter metrique, diagramme en colonnes, diagramme en lignes, diagramme circulaire et types de services

Notre exemple de tableau de bord ressemble à ceci :

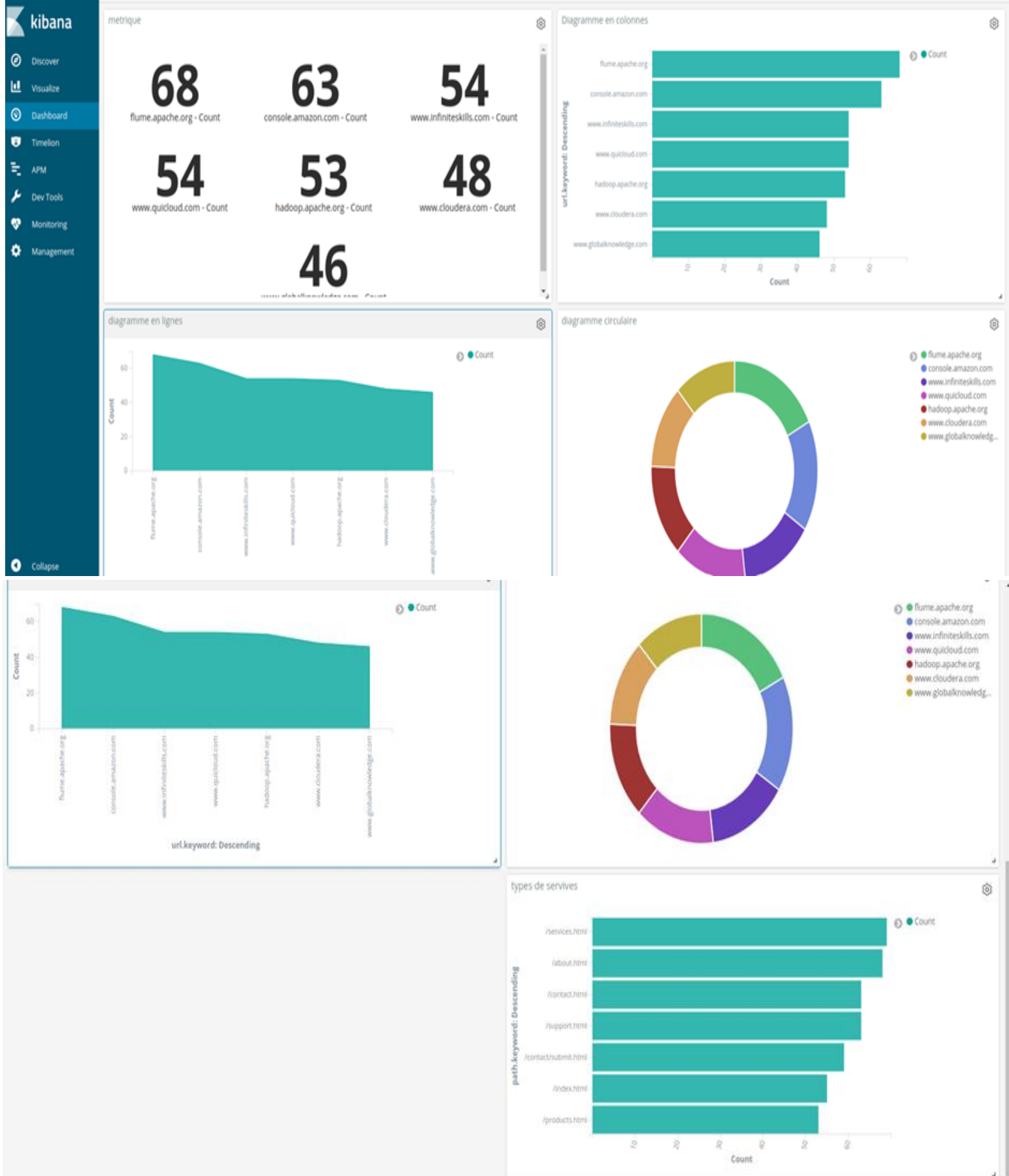


Figure 19:Dashboard