

APACHE NiFi 1.0

Introduction

Nifi est un outil d'ingestion de données. Il permet de collecter et traiter de larges quantités de données grâce à une interface web simple d'utilisation.

Eléments clés dans NiFi

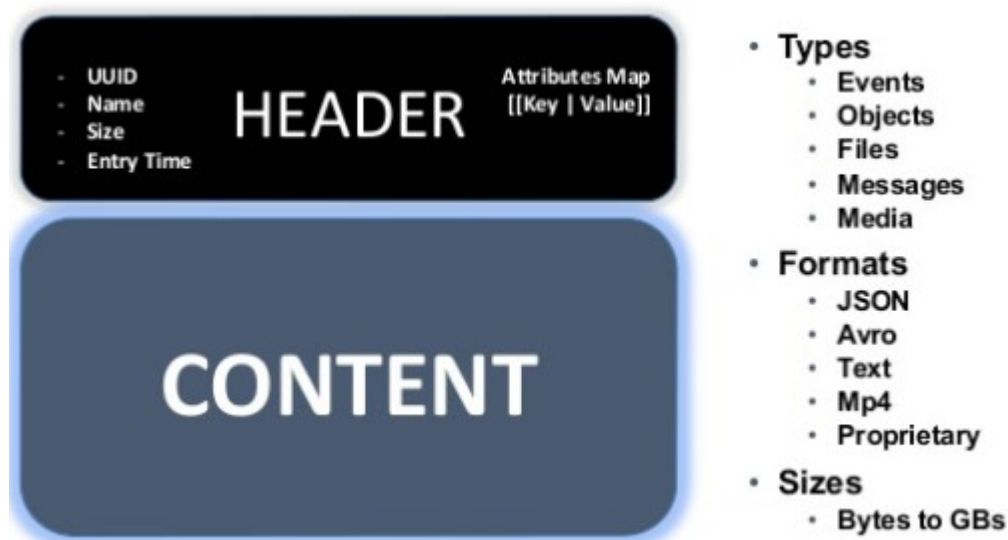
« FlowFile » : l'unité de donnée. Il s'agit du nom attribué aux données en mouvement au sein du workflow. Ce fichier dispose d'un contenu (« content ») ainsi que d'attributs clés.

Processeur : C'est l'objet qui va permettre de lire les données. Il reçoit en **entrée** un ou plusieurs flowfiles puis va effectuer en **sortie** des opérations sur ces données (vérification, ajout d'attributs, modification du contenu etc...).

Connexion : lien entre deux processeurs. Ce lien est représenté par

Architecture de Nifi :

Unité de base : le « flowfile ». Fichier caractérisant la donnée en mouvement dans NiFi. Ce fichier est constitué d'attributs ainsi que d'un contenu.



Processeurs utiles

- **AttributesToJSON** :

Ce processeur construit un fichier JSON à partir d'attributs. Une fois que le fichier JSON est construit, il peut être stocké dans un entrepôt de données comme HDFS ou converti dans un autre format (XML, CSV, AVRO etc...)

Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
----------	------------	------------	----------

Required field +

Property	Value
Attributes List	? direction, typeTrain, region, heureArrivee
Destination	? flowfile-content
Include Core Attributes	? true
Null Value	? false

Le fichier JSON (dont la liste d'attributs est spécifiée par l'utilisateur) peut être écrit de deux manières différentes : on écrase le contenu du *flowfile* en écrivant le fichier JSON à cet endroit ou on rajoute les attributs du fichier JSON aux attributs déjà présents du *flowfile* (option **Destination**).

- ConvertJSONtoAvro :

Conversion d'un fichier de type JSON en un fichier Apache AVRO.

- EvaluateJSONPath :

Extraction des valeurs/attributs du contenu du flowfile en JSON (on suppose donc que les fichiers en entrée sont de type **JSON**)

Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
----------	------------	------------	----------

Required field +

Property	Value
Destination	? flowfile-attribute
Return Type	? auto-detect
Path Not Found Behavior	? warn
Null Value Representation	? empty string
direction	? \$.display_informations.direction 🗑
heureArrivee	? \$.stop_date_time.arrival_date_time 🗑
region	? \$.display_informations.label 🗑
typeTrain	? \$.display_informations.physical_mode 🗑

Dans le cas où le résultat est écrit sous formes d'attributs du fichier (*flowfile-attribute*), il faut alors préciser les propriétés des noms des attributs (chemin dans le fichier JSON des attributs à extraire). Ainsi dans

l'exemple ci-dessus, l'attribut *direction* nous indique que le/les fichiers JSON en entrée sont sous la forme suivante :


```
"display_informations":{
  "direction":"Mantes-la-Jolie (Mantes-la-Jolie)",
  ...
},
```








- ExecuteStreamCommand

Lancement d'une commande. Le résultat de cette commande apparaît dans le fichier (*flow-file*). Ce processeur est utile pour ajouter des attributs supplémentaires au fichier existant.

Configure Processor

SETTINGS SCHEDULING **PROPERTIES** COMMENTS

Required field 

Property		Value
Command Arguments		\${filename:substringBefore('_')}
Command Path		D:\Profiles\user\Documents\Nifi\demo\Scripts\...
Ignore STDIN		false
Working Directory		\${absolute.path}
Argument Delimiter		;
Output Destination Attribute		No value set
Max Attribute Length		256

- \${filename} : attribut correspondant au nom du fichier, comprenant l'extension du fichier (.txt, .json , .xml ,etc...)
- \${absolute.path} : attribut correspondant au chemin d'accès du fichier

La commande lancée correspond à un fichier script, dont le chemin d'accès est : **D:\Profiles\user\Documents\Nifi\demo\Scripts\scriptTTS.bat**

On utilise ici le langage d'expression de Nifi afin d'obtenir une sous-chaine de caractère du nom du fichier. Dans le schéma ci-dessus, si le nom du fichier est slide_12.txt, \${filename:substringBefore('_')} donnera « slide » comme résultat.

- ExtractMediaMetadata

- GetFile

Lecture de tous les fichiers d'un répertoire afin de les ajouter comme fichiers au flow. Lorsque ce processeur est exécuté, il continue de lire les fichiers ajoutés au répertoire : il est donc coûteux en mémoire.









Property		Value
Input Directory	?	D:\Profiles\user\Documents\Nifi\demo\send
File Filter	?	[\].*
Path Filter	?	No value set
Batch Size	?	10
Keep Source File	?	false
Recurse Subdirectories	?	true
Polling Interval	?	0 sec
Ignore Hidden Files	?	true
Minimum File Age	?	0 sec
Maximum File Age	?	No value set
Minimum File Size	?	0 B
Maximum File Size	?	No value set

On peut également filtrer les fichiers lus à l'aide des **expressions régulières** dans le champ « File filter ».

Exemple : lire et obtenir uniquement les fichiers commençant par « log ». La valeur de « File Filter » sera alors (log)*

- GetHTTP

Extrait le contenu/ les fichiers de sites http ou HTTPS.

Property	Value	
URL		https://api.sncf.com/v1/coverage/sncf/stop_area...
Filename		departsMontparnasse.json
SSL Context Service		connexion https
Username		b63fd422-985a-48c5-8045-fa1094d30962
Password		No value set
Connection Timeout		30 sec
Data Timeout		30 sec
User Agent		No value set
Accept Content-Type		No value set
Follow Redirects		false
Redirect Cookie Policy		default
Proxy Host		ntes.proxy.corp.sopra
Proxy Port		8080




Dans le cas de sites HTTPS, il faut établir une connexion sécurisée (*StandardSSLContextService*) au moyen du *SSL Context Service* (voir partie **Controller services**).

L'utilisation d'un proxy est quasi systématique dans une entreprise. Il faut alors renseigner l'hôte (*Proxy Host*) et le port associé (*Proxy Port*).

Le nom donné au fichier récupéré est ici `departsMontparnasse.json` (on récupère le fichier grâce à l'API de SNCF). Un nom d'utilisateur est requis pour accéder à l'API : on le précise dans le champ *Username*.

- **RouteOnAttribute**

Permet la sélection de fichiers en se basant sur leurs attributs afin de les classer/router vers une destination précise. Cette sélection se fait grâce au langage d'expression des attributs de NiFi.

Property	Value	
Routing Strategy		Route to Property name
NiFi		<code>\${title:matches('NiFi')}</code> 

Dans cet exemple, on sélectionne tous les documents dont l'attribut « title » coïncide avec « NiFi ».

- **PutFile**

Écriture des fichiers reçus dans un dossier (écriture en local)

Property		Value
Directory	?	D:\Profiles\user\Documents\Nifi
Conflict Resolution Strategy	?	fail
Create Missing Directories	?	true
Maximum File Count	?	No value set
Last Modified Time	?	No value set
Permissions	?	No value set
Owner	?	No value set
Group	?	No value set

- MergeContent

Fusionne plusieurs documents en un seul document. Le format de sortie est multiple :

Property		Value
Merge Strategy	?	Bin-Packing Algorithm
Merge Format	?	Binary Concatenation
Attribute Strategy	?	Keep Only Common Attributes
Correlation Attribute Name	?	No value set
Minimum Number of Entries	?	100
Maximum Number of Entries	?	500
Minimum Group Size	?	0 B
Maximum Group Size	?	No value set
Max Bin Age	?	60 seconds
Maximum number of Bins	?	100
Delimiter Strategy	?	Text
Header	?	[
Footer	?]
Demarcator	?	,
Compression Level	?	1
Keep Path	?	false

Connexions dans NiFi : « Controller services »

Afin de configurer les connexions entre

- StandardSSLContextService
- DBCPConnectionPool :

Sert pour se connecter aux bases de données relationnelles (JDBC Driver), Apache Phoenix.

- HBase_1_1_2_ClientService :
 - o Connexion avec HBase
 - o Informations à remplir relatives à HBase et à Zookeeper
- HiveConnectionPool :
 - o Connexion avec Hadoop Hive
- JMSConnectionFactoryProvider

Exemples d'applications

- 1) Transformation de données
- 2) Extraction d'informations
- 3) Indexation de documents (Elasticsearch)

1) 1^{er} cas d'usage simple de transformation des données

L'extensibilité de NiFi permet lorsqu'aucun processeur ne répond au problème posé de développer son propre processeur afin de l'utiliser. Mais il existe également des processeurs permettant d'exécuter du code :

ExecuteScript et **ExecuteStreamCommand** en sont deux exemples.

Dans cet exemple, nous allons détailler l'utilisation d'ExecuteStreamCommand en récupérant les fichiers dans un dossier puis en exécutant une application/commande pour transformer ces fichiers :



En combinant ce processeur avec un autre processeur, **ListFiles**, on obtient un exemple simple de flow effectuant l'action décrite.

Configurons le processeur ListFiles :

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field

Property		Value
Input Directory	?	D:\Profiles\user\Documents\Nifi\demo\LectureF...
Recurse Subdirectories	?	false
Input Directory Location	?	Local
File Filter	?	[^\\].*
Path Filter	?	No value set
Minimum File Age	?	0 sec
Maximum File Age	?	No value set
Minimum File Size	?	0 B
Maximum File Size	?	No value set
Ignore Hidden Files	?	true

Pour chaque fichier du dossier source « D:\Profiles\user\Documents\Nifi\demo\LectureFichiers », ce processeur va générer un fichier (*FlowFile*) avec des attributs utiles et sans contenu.

Ensuite, la configuration du processeur « ExecuteStreamCommand » se fait de la façon suivante :

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field

Property		Value
Command Arguments	?	\${absolute.path}\${filename}
Command Path	?	D:\Profiles\user\Documents\Nifi\demo\Scripts\...
Ignore STDIN	?	true
Working Directory	?	No value set
Argument Delimiter	?	;
Output Destination Attribute	?	résultat
Max Attribute Length	?	256

Afin d'extraire des informations relatives aux fichiers (nom de fichier, chemin de dossier etc...), on utilise le langage d'expression de NiFi :

```
${absolute.path}${filename}
```

On récupère ainsi le chemin d'accès et le nom de chaque fichier présents dans le répertoire où on liste les fichiers avec le processeur Lisfiles. On concatène les deux attributs (**chemin d'accès** et **nom de fichier**)

Le processeur permet de garder ou non le contenu de nos fichiers lorsqu'on exécute la commande. Dans notre cas, les fichiers récupérés ne possèdent pas de contenu (fichiers texte vides) et on veut les déplacer : on choisit ainsi de référencer l'option Ignore STDIN à true (on ignore le contenu des fichiers).

On peut également créer un nouveau fichier (flowFile) correspondant au résultat de notre commande (dans le cas où notre commande nous rend un résultat sous la forme d'un autre fichier modifié par exemple). Ici, on déplace des fichiers : on n'a donc pas besoin de créer un nouveau fichier pour cela. C'est pourquoi on référence la valeur **Output Destination Attribute** : le résultat de la commande sera utilisé/écrit sous la forme d'un attribut du fichier source.

On va déplacer les fichiers du dossier source au dossier « receive ». La commande est la suivante :

```
@echo off
MOVE %1 "D:\Profiles\user\Documents\Nifi\demo\receive" >nul
```

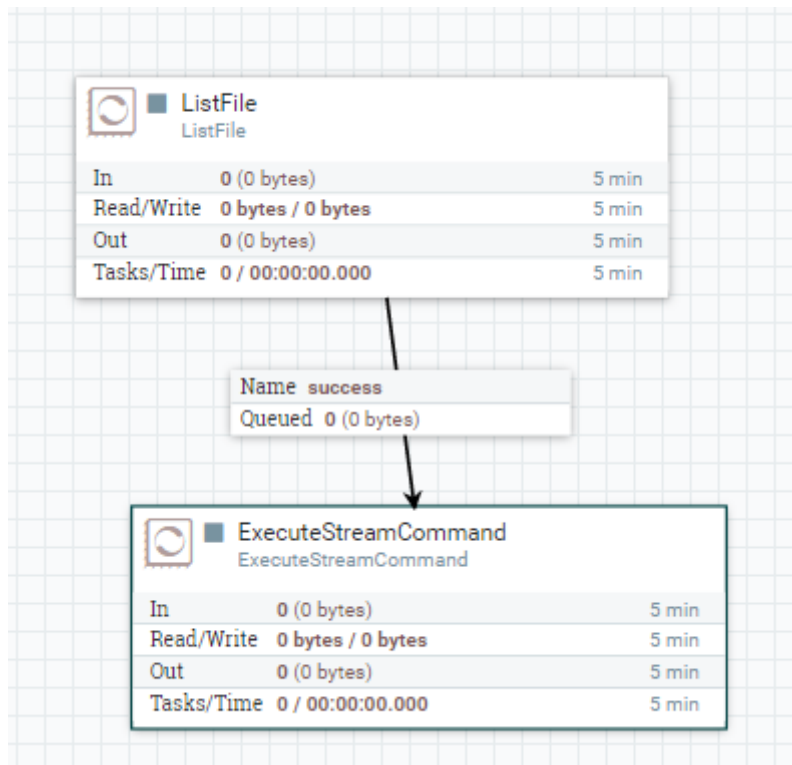


Schéma du flow

On peut également poursuivre en exécutant un autre processeur à la suite afin de traiter les fichiers créés.

Tutoriel complet (en anglais) :

<https://pierrevillard.com/2016/03/09/transform-data-with-apache-nifi/>

2) Extraire des informations au sein d'un document : EXEMPLE API SNCF

Le but de cet exemple est de requêter l'API SNCF afin d'extraire des informations d'une recherche précise.

Dans cet exemple, on requête l'API afin d'obtenir l'ensemble des gare SNCF. Nous voulons obtenir le nombre total de pages de recherches contenant les gares. Cette information se trouve au sein de l'attribut « links » du document JSON retourné par la requête :

```
{
  "href": "https://api.sncf.com/v1/coverage/sncf/stop_areas?start_page=120.24",
  "type": "last",
  "templated": false
},
```

Pour ce faire, listons les étapes nécessaires à la construction du flow :

a) Processeur GetHTTP

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field

Property		Value
URL	?	https://api.sncf.com/v1/coverage/sncf/stop_area...
Filename	?	departsMontparnasse.json
SSL Context Service	?	connexion https
Username	?	b63fd422-985a-48c5-8045-fa1094d30962
Password	?	No value set
Connection Timeout	?	30 sec
Data Timeout	?	30 sec
User Agent	?	No value set
Accept Content-Type	?	No value set
Follow Redirects	?	false
Redirect Cookie Policy	?	default
Proxy Host	?	ntes.proxy.corp.sopra
Proxy Port	?	8080

Ce processeur permet de récupérer les documents voulus en requêtant l'API SNCF :

- URL : https://api.sncf.com/v1/coverage/sncf/stop_areas?start_page=1
 - Pour obtenir des explications de L'API SNCF, cf. *Fonctionnement API SNCF*
- Filename : choix du nom du fichier obtenu de l'API (c'est ici un fichier JSON)
- SSL Context Service : Afin de pouvoir requêter des sites référencés sur **https**, il faut établir une connexion de type SSL Context Service (Voir : <https://community.hortonworks.com/questions/9509/connecting-to-datasift-https-api-using-nifi.html>)
- Username : cf. *Fonctionnement API SNCF*
- Proxy Host/Port : nom de domaine et port du proxy dans le cas où est un proxy est défini dans l'entreprise.

b) Processeur SplitJSON

Property	Value
JsonPath Expression	? \$.links[*]
Null Value Representation	? empty string

On découpe le fichier JSON en une multitude de fichiers selon un attribut du fichier JSON d'origine spécifié. Ici la partie du fichier JSON qui nous intéresse se trouve dans le tableau suivant :

```
"links": [{...}, {...}, {...}, ...]
```

On obtiendra donc une multitude de fichiers JSON de ce type :

```
{
  "href": "https://api.sncf.com/v1/coverage/sncf/stop_areas/{stop_areas.id}",
  "type": "stop_areas",
  "rel": "stop_areas",
  "templated": true
},
```

c) Processeur EvaluateJSONPath


Property	Value	
Destination	? flowfile-attribute	
Return Type	? auto-detect	
Path Not Found Behavior	? warn	
Null Value Representation	? empty string	
href	? \$.href	🗑
type	? \$.type	🗑

Pour tous les fichiers JSON en entrée dont la forme est similaire au schéma au-dessus, on extrait les champs « type » et « href » que l'on ajoute aux attributs de nos fichiers.

d) Processeur RouteOnAttribute

Le but est d'obtenir un document ayant un champ « type » et un champ « href » spécifique :

```
{
  "href": "https://api.sncf.com/v1/coverage/sncf/stop_areas?start_page=120.24",
  "type": "last",
  "templated": false
},
```

Property	Value		
Routing Strategy	?	Route to Property name	
link_last_page	?	<code>\${type:equals("last")}</code>	

On choisit donc parmi ces fichiers de ne retenir que ceux dont le *type* correspond à « last » (référence à la dernière page de la recherche).

e) Processeur UpdateAttribute

Property	Value		
Delete Attributes Expression	?	No value set	
last_page	?	<code>\${href.substringAfter('='):substringBefore('.')}</code>	

Le but est d'obtenir le numéro suivant « start_page » dans la référence http suivante :

```
"href": "https://api.sncf.com/v1/coverage/sncf/stop_areas?start_page=120.24"
```

Pour cela, on crée la propriété (qui constituera l'attribut résultat que l'on veut obtenir) *last_page*.

`${href}` désigne l'attribut créé précédemment avec le processeur EvaluateJSONPath. On utilise le langage d'expression de NiFi afin d'obtenir la chaîne de caractère correspondant uniquement à la partie entière de 120.24

f) Processeur LogAttribute

Enfin, on affiche notre attribut *last_page*.

Property	Value		
Log Level	?	info	
Log Payload	?	false	
Attributes to Log	?	last_page	
Attributes to Ignore	?	No value set	
Log prefix	?	No value set	

Résultat :

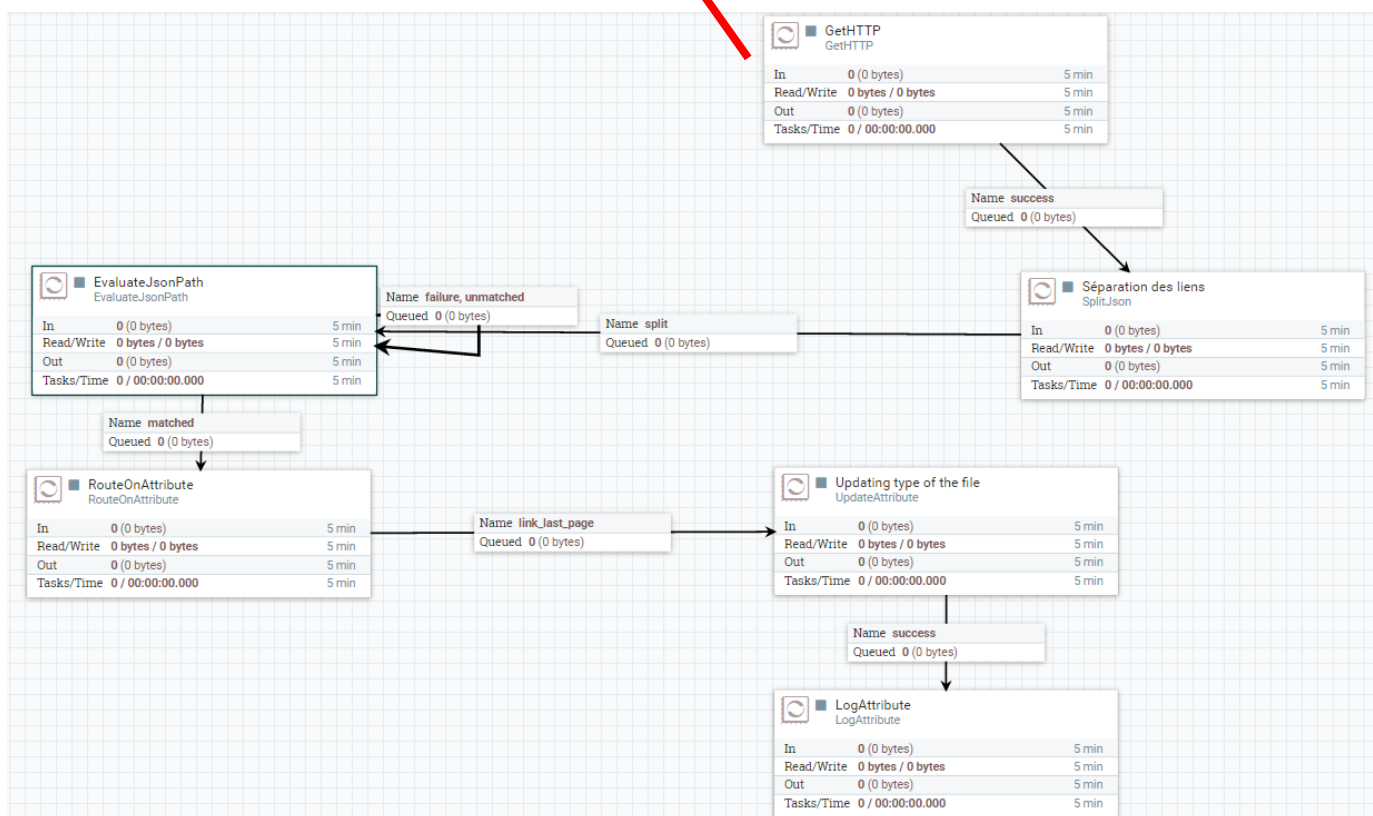
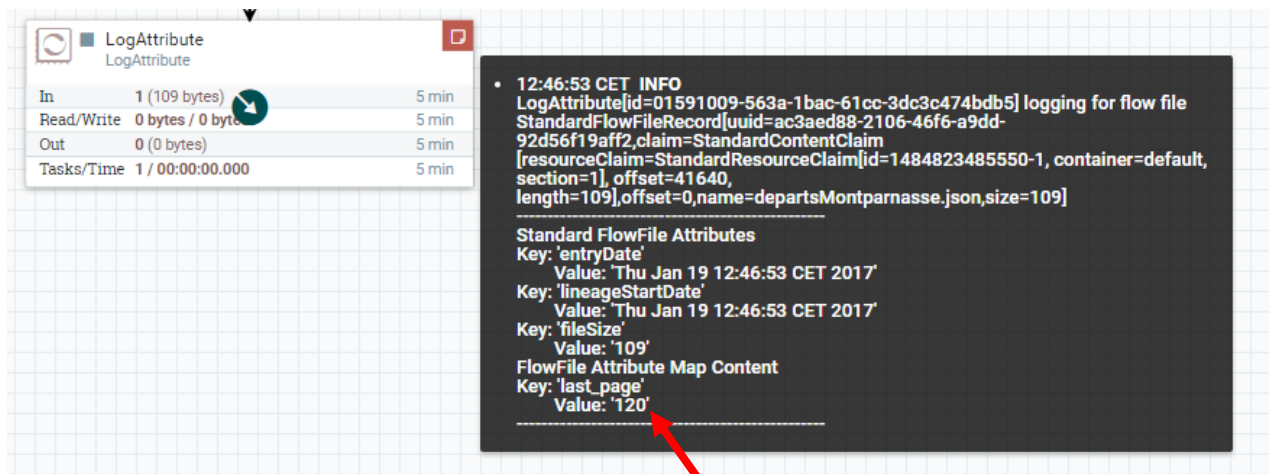


Schéma final du flow

3) Récupération et indexation de documents avec Elasticsearch

L'objectif de cet exemple est d'obtenir l'ensemble des départs de trains depuis la gare Montparnasse à un horaire fixé (on a choisi ici l'heure actuelle).

A partir des horaires et des directions des trains, on va créer des fichiers JSON que l'on va indexer dans Elasticsearch (moteur de recherche indexé).

Liste des processeurs utilisés :

a) Processeur GetHTTP

Sur le même modèle que 2) a), on récupère l'ensemble des départs depuis la gare Montparnasse. Seul l'URL de la requête change :

URL :

[https://api.sncf.com/v1/coverage/sncf/stop_areas/stop_area:OCE:SA:87391003/departures?datetime=\\${now\(\):format\("yyyyMMdd"\):append\('T'\)}\\${now\(\):format\('HH:mm:ss'\)}](https://api.sncf.com/v1/coverage/sncf/stop_areas/stop_area:OCE:SA:87391003/departures?datetime=${now():format()

On choisit ici de prendre la date actuelle. Pour cela l'API supporte une date du type :

yyyyMMddTHH:mm:ss où T représente un séparateur (les autres champs sont intuitifs).

Afin d'obtenir un format de date similaire à celui-ci-dessus, on fait appelle à la fonction `now()` qui donne la date actuelle puis on reproduit le format ci-dessus à l'aide de fonctions de concaténation (*append*).

b) Processeur SplitJSON

On choisit de découper le document JSON reçu en plusieurs documents JSON selon l'expression :

`$.departures[*]`

c) Processeur EvaluateJSONPath

On définit 4 propriétés/attributs définis dans les documents JSON :

- direction, heureArrivee, region, typeTrain

d) Processeur PutFile

Property		Value
Directory	?	D:\Profiles\user\Documents\Nifi\matched
Conflict Resolution Strategy	?	replace
Create Missing Directories	?	true
Maximum File Count	?	No value set
Last Modified Time	?	No value set
Permissions	?	No value set
Owner	?	No value set
Group	?	No value set

On place le fichier **obtenu depuis l'API** dans un dossier. On choisit (au sein du processeur GetHTTP) de requêter l'API toutes les 5 minutes par exemple :

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Scheduling Strategy ?

Timer driven

▼

Concurrent Tasks ?

1

Run Schedule ?

300 sec

Ainsi, 5 minutes après avoir récupéré le premier fichier, ce dernier sera écrasé par le nouveau fichier : on choisit de le remplacer dans le paramètre *Conflict Resolution Strategy*

e) Processeur AttributesToJSON

Property		Value
Attributes List	?	direction, typeTrain, region, heureArrivee
Destination	?	flowfile-content
Include Core Attributes	?	true
Null Value	?	false

On choisit avec ce processeur de générer de nouveaux fichiers JSON à partir d'une liste d'attributs des fichiers en entrée. En sortie, le contenu des fichiers (*flowfile-content*) est modifié : ce contenu représente le fichier JSON dont les attributs sont ceux spécifiés dans *Attributes List*.

f) Processeur LogAttribute

On choisit d'afficher la *direction* par exemple.

g) Processeur PutElasticsearch (cf. *Guide pratique Elasticsearch*)

Property		Value
Cluster Name	?	elasticsearch
ElasticSearch Hosts	?	localhost:9300
SSL Context Service	?	No value set
Shield Plugin Filename	?	No value set
Username	?	No value set
Password	?	No value set
ElasticSearch Ping Timeout	?	5s
Sampler Interval	?	5s
Identifier Attribute	?	direction
Index	?	departs
Type	?	\${now():format('yyyy-MM-dd')}_departs
Character Set	?	UTF-8
Batch Size	?	100
Index Operation	?	index

On va ensuite pouvoir indexer ces nouveaux documents JSON au sein d'Elasticsearch.

Cela permettra d'effectuer des recherches sur ces documents ou de faire de la visualisation avec **KIBANA**.

Remarques:

- Le port précisé est ici **9300** et non pas **9200** !
- Il faut modifier le fichier elasticsearch.yml : remplacer la ligne où est défini le nom du cluster par : *cluster.name: elasticsearch*
- L'identifiant (*Identifier Attribute*) doit être unique. Ici on choisit l'attribut *direction*.

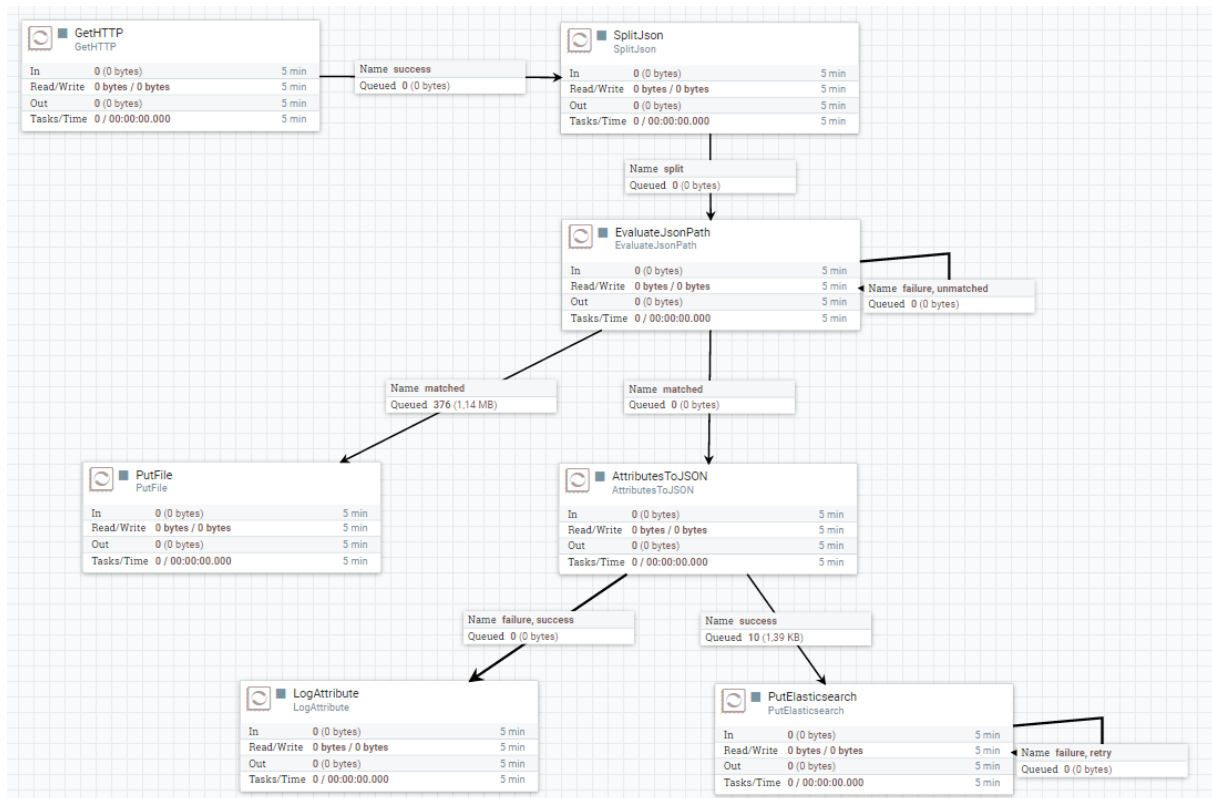


Schéma final du flow

Liens utiles :

- Exemples et tests d'expressions régulières : <http://regexr.com/>
- Ensembles de tutoriels (NiFi) : <http://www.dataflowdeveloper.com/2016/09/21/apache-nifi-tutorials/>
- Ensemble d'applications à NiFi (tutos, blogs, templates, vidéos etc..) : <https://github.com/jfrazee/awesome-nifi>
- Exemples de Templates NiFi : <https://cwiki.apache.org/confluence/display/NIFI/Example+Dataflow+Templates>
- Blog sur le processeur « **ExecuteScript** » : <http://funnifi.blogspot.fr/>