

# PRÁCTICA SESIÓN 8 - NIFI

En este documento se expone cómo trabajar con apache nifi: atributos, contenido, expresiones, conexiones entre procesadores, control de versiones y un proyecto final.

## Contenido

1. Ejercicio 1 - Proceso para la creación de atributos.....	2
2. Ejercicio 2 - Trabajar con conexiones, colas y funnels.....	11
2.2. Conexiones y grupos.....	13
3. Proyecto con Apache Nifi.....	16
3.1. Crear un servicio Kafka de producción de mensajes.....	16
3.2. Unimos nuestro productor de mensajes con una base de datos como Elasticsearch.....	18

## 1. Ejercicio 1 - Proceso para la creación de atributos

- En NIFI solo hay un canvas de nivel superior, pero podemos construir tantos flujos lógicos como deseemos. Normalmente, para organizar las cosas, se utilizan grupos de procesos, por lo que el canvas de nivel superior puede tener varios grupos de procesos, cada uno de los cuales representa un flujo lógico, pero no necesariamente conectados entre sí.
  - Cada vez que se generan Flowfiles (representa un registro de datos que viaja por el flujo) estos van a tener asignados ciertos atributos por defecto. Entre estos atributos están el UUID o identificador único, su timestamp y el tamaño del fichero, mediante el uso de procesadores podremos modificar estos o añadir nuevos atributos.
  - Vamos a ver cómo hacerlo realizando los siguientes pasos:
- I. Vamos a añadir un procesador del tipo GenerateFlowFile: crea flowfiles con datos aleatorios o contenidos personalizados: útil para testear y debugear.



- II. Clicamos sobre el procesador y vamos a la pestaña propiedades completando los campos Flow Size: 10 bytes – Batch Size: 1 para que nos genere uno por cada ejecución, Data Format: texto e indicamos que los FlowFiles van a ser únicos con la opción Unique Flowfiles: true

### Configure Processor

Invalid

SETTINGS SCHEDULING PROPERTIES COMMENTS

Required field +

Property	Value
File Size	10B
Batch Size	1
Data Format	Text
Unique FlowFiles	true
Custom Text	No value set
Character Set	UTF-8
Mime Type	No value set

CANCEL APPLY

III. A continuación, en la configuración de ejecución de este procesador vamos a indicar que se ejecute cada 3 segundos

### Configure Processor

Invalid

SETTINGS SCHEDULING PROPERTIES COMMENTS

Scheduling Strategy  
Timer driven

Concurrent Tasks  
1

Execution  
All nodes

Run Duration  
0ms 25ms 50ms 100ms 250ms 500ms 1s 2s  
Lower latency Higher throughput

Run Schedule  
3 sec

CANCEL APPLY

IV. Ahora que ya tenemos listo nuestro generador de datos vamos a añadir un procesador con el que cambiar el texto. Para ello, añadimos un procesador del tipo ReplaceText.

### Add Processor

Source: all groups | Displaying 2 of 283 | replace

Type	Version	Tags
ReplaceText	1.13.2	Regular Expression, Replace, R...
ReplaceTextWithMapping	1.13.2	Regular Expression, Replace, R...

**amazon attributes**  
**avro aws azure**  
**consume csv**  
**database delete**  
**fetch get hadoop**  
**ingest insert json**  
**listen logs**  
**message pubsub**  
**put record**  
**restricted source**  
**text update**

**ReplaceText 1.13.2** org.apache.nifi - nifi-standard-nar

Updates the content of a FlowFile by evaluating a Regular Expression (regex) against it and replacing the section of the content that matches the Regular Expression with some alternate value.

CANCEL ADD

V. Una vez que lo hemos añadido conectamos ambos procesadores

### Create Connection

DETAILS SETTINGS

From Processor: GenerateFlowFile, GenerateFlowFile

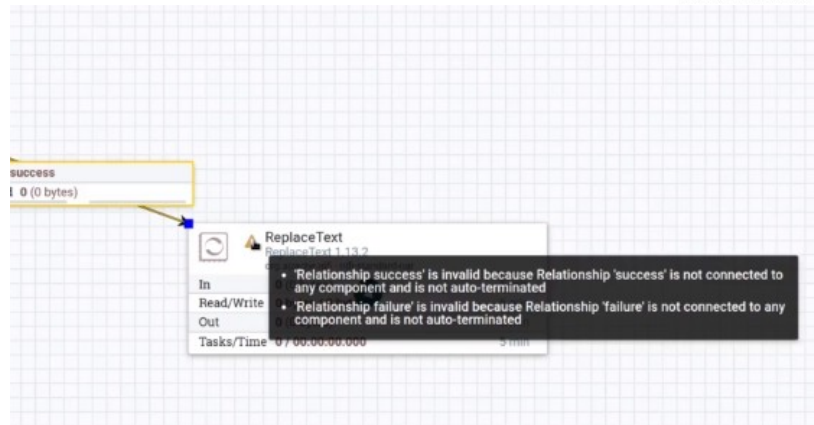
To Processor: ReplaceText, ReplaceText

Within Group: NiFi Flow, NiFi Flow

For Relationships: ☒ success

CANCEL ADD

VI. Aquí vemos que NIFI nos indica que nos hace falta configurar nuestro nuevo procesador, además de indicarnos que ambas relaciones no están conectadas o que faltan por autocompletar.



VII. Vamos a configurarlo indicando la estrategia de reemplazo. En este caso vamos a indicarle que lo reemplace siempre (al hacer esto hace invalido Search Value)

VIII. En Replacement Value vamos a indicar simplemente "test"

### Configure Processor

Invalid

SETTINGS SCHEDULING PROPERTIES COMMENTS

Required field

Property	Value
Search Value	
Replacement Value	
Character Set	
Maximum Buffer Size	
Replacement Strategy	
Evaluation Mode	
Line-by-Line Evaluation Mode	

1 test

☐ Set empty string

CANCEL OK

CANCEL APPLY

IX. Ahora vamos a añadir un procesador de tipo LogAttribute para logear los atributos de un flowfile:

### Add Processor

Source

all groups

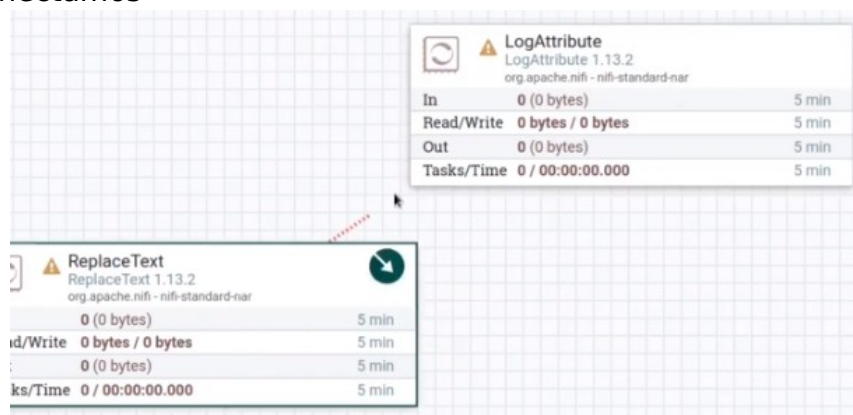
Displaying 1 of 283

log

Type	Version	Tags
LogAttribute	1.13.2	logging, attributes

amazon attributes  
avro aws azure  
consume csv  
database delete  
fetch get hadoop  
ingest insert json  
listen logs  
message pubsub  
put record  
restricted source  
text update

X. Los conectamos



XI. Chequeamos para la relación success

Create Connection

DETAILS SETTINGS

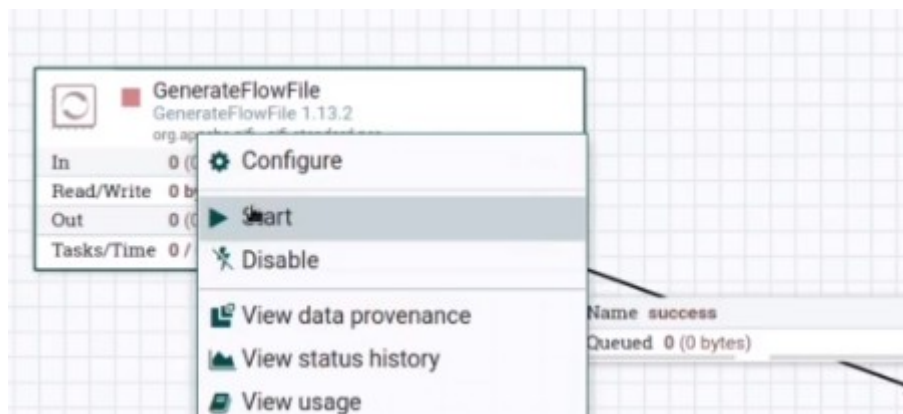
From Processor  
ReplaceText  
ReplaceText  
Within Group  
NIFI Flow

To Processor  
LogAttribute  
LogAttribute  
Within Group  
NIFI Flow

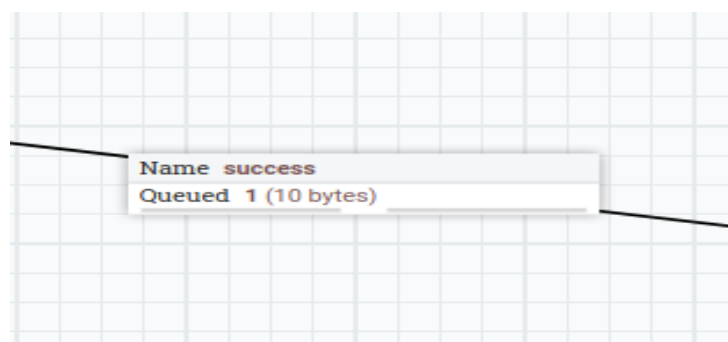
For Relationships  
☐ failure  
☒ success

CANCEL ADD

XII. Ahora arrancamos el primer procesador y paramos cuando nos haya generado el primer flowfile

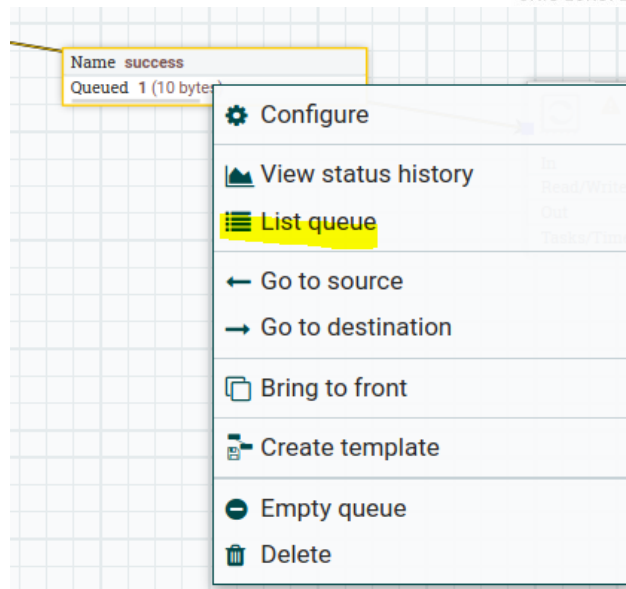


XIII. Vemos que tenemos un flowfile encolado.



XIV. Elegimos la opción list queue para ver su contenido.





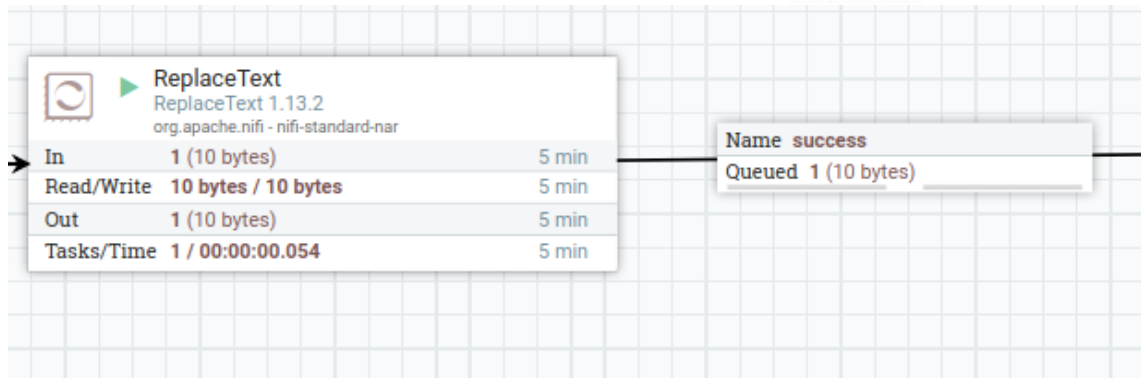
XV. Damos a la opción visualizar el flowfile

Lineage Duration	Penalized	
00:06:41.719	No	  

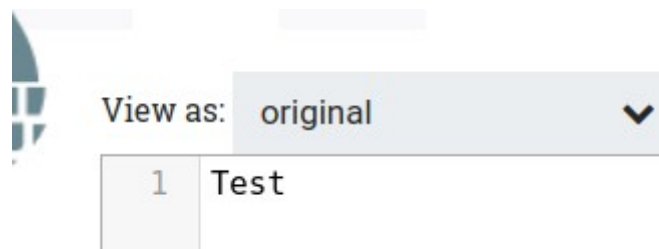
XVI. Y vemos que ha generado datos aleatorios



XVII. Ahora ejecutamos el siguiente procesador y vemos que deja en la cola el flowfile



XVIII. Si le echamos un vistazo, vemos que ha sustituido el texto aleatorio original por test.



XIX. Ahora vamos a extraer el contenido del flowfile a un atributo mediante un nuevo procesador del tipo ExtractText.

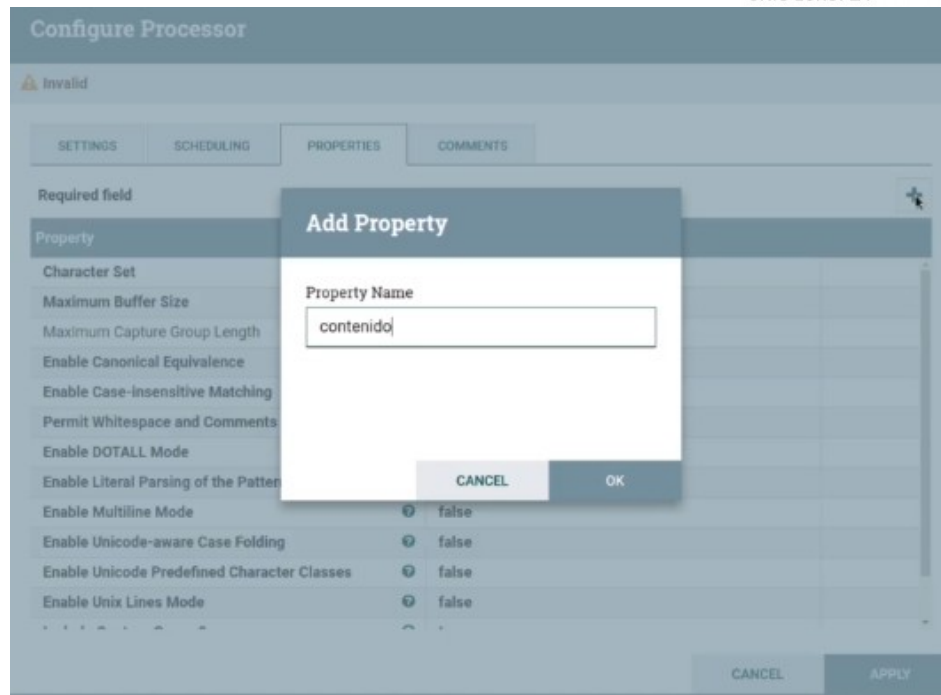
**Add Processor**

Source: all groups ▼ Displaying 8 of 283

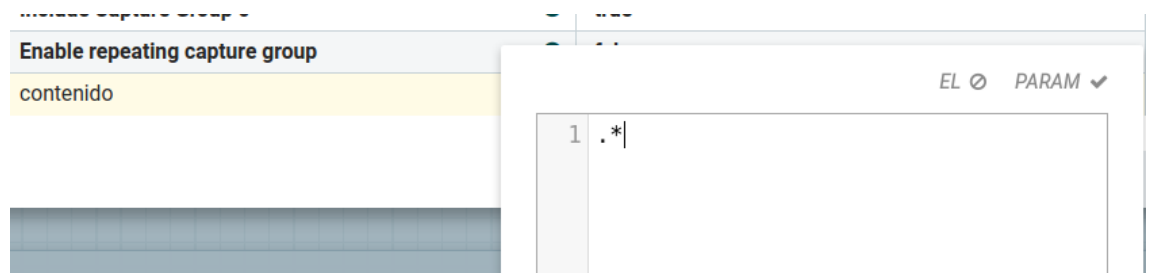
Type	Version	Tags
ExtractAvroMetadata	1.13.2	schema, metadata, avro
ExtractCCDAAttributes	1.13.2	CCDA, extract, attributes, health...
ExtractEmailAttachments	1.13.2	split, email
ExtractEmailHeaders	1.13.2	split, email
ExtractGrok	1.13.2	delimit, extract, log, grok, text, p...
ExtractHL7Attributes	1.13.2	HL7, extract, attributes, health I...
ExtractTNEFAttachments	1.13.2	split, email
ExtractText	1.13.2	Regular Expression, regex, extra...

amazon attributes  
avro aws azure  
consume csv  
database delete  
fetch get hadoop  
ingest insert json  
listen logs  
message pubsub  
put record  
restricted source

XX. Añadimos una nueva propiedad que llamaremos "contenido"



XXI. Indicamos la expresión regular que queremos que extraiga (en este caso vamos a poner una expresión que coincida con todo)



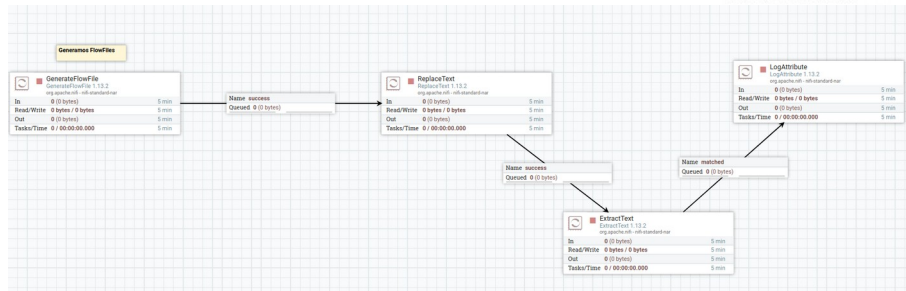
XXII. El siguiente paso es colocar el nuevo procesador entre ReplaceText y LogAttribute e indicamos el caso de matched (el caso en que ha coincidido)

For Relationships

☒ matched

☐ unmatched

XXIII. En el caso de unmatched en el propio procesador la autoterminamos. Y ya vemos que ningún procesador está con advertencia.



XXIV. Ahora solo tenemos que ir ejecutando y revisando cómo va pasando la información hasta la última cola donde veremos que se ha añadido un nuevo atributo.

**matched**

Displaying 2 of 2 (8.00 bytes)

	Position	UUID
1	1	15312c29-e7cc-455a-96f3-11
2	2	c7f8904e-b536-4c9e-bbec-01

**FlowFile**

DETAILS ATTRIBUTES

**Attribute Values**

contenido.0  
Test

filename  
15312c29-e7cc-455a-96f3-1961d66070b0

path  
./

uuid  
15312c29-e7cc-455a-96f3-1961d66070b0

## 2. Ejercicio 2 – Trabajar con conexiones, colas y funnels

- En este ejercicio vamos a explorar lo que son las conexiones entre los procesadores de nifi, la gestión de colas y lo que son los funnels.

- Empezamos añadiendo un procesador de tipo GetFile indicándole en las propiedades el directorio de entrada.

Required field

Property Value

Input Directory

File Filter

Path Filter

Batch Size

Keep Source File

Recurse Subdirectories

Polling Interval

Ignore Hidden Files

1 /home/hadoop/Documentos/in

☐ Set empty string

CANCEL OK

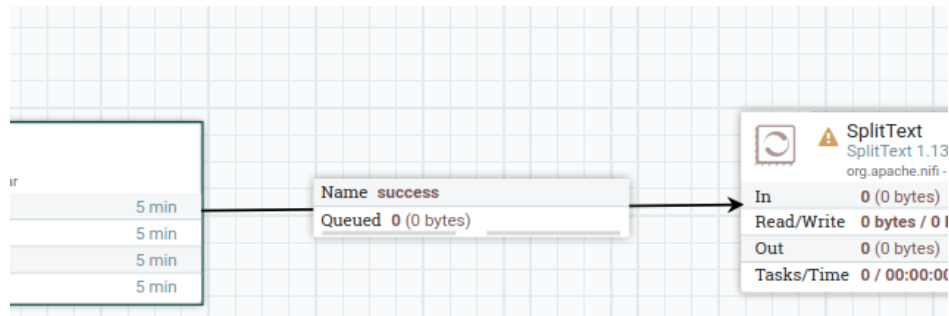
- Añadimos un procesador del tipo SplitText

Displaying 1 of 283

SplitT

Type ▲	Version	Tags
SplitText	1.13.2	split, text

III. Conectamos los dos procesadores con la relación success.



IV. Configuramos la conexión en la pestaña settings

a. Añadimos un nombre personalizado.

Name  
Mueve Fichero

b. Flow File Expiration nos indica cuándo nuestros datos dejan de tener valor. Con 0 indica que nunca expiran.

FlowFile Expiration ⓘ  
0 sec

c. La cantidad de flowfiles permitidos en la cola o tamaño.

Back Pressure Object Threshold ⓘ Size Threshold ⓘ  
10000 1 GB

d. Estrategia de balanceo de la cola (aplicará cuando tengamos un cluster de nifi)

Load Balance Strategy ⓘ

- Do not load balance ✓
- Do not load balance ?
- Partition by attribute ?
- Round robin ?
- Single node ?

e. A la derecha tenemos la opción de añadir las prioridades de la cola.

#### Available Prioritizers ?

NewestFlowFileFirstPrioritizer

OldestFlowFileFirstPrioritizer

PriorityAttributePrioritizer

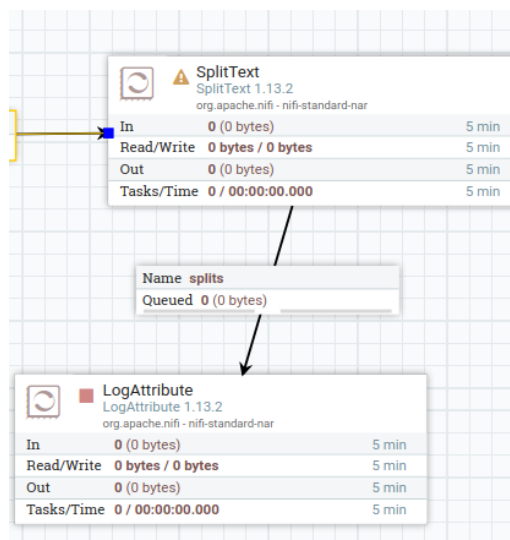
#### Selected Prioritizers ?

FirstInFirstOutPrioritizer

- V. Un procesador puede tener varias relaciones de salida. Por ejemplo, si miramos las relaciones SplitText, tenemos:
- Failure: para dirigir aquellos ficheros que no hayan podido ser divididos.
  - Original: la entrada del fichero original, la marcamos para que no avance el fichero original y termine en este procesador.

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
<p>Name SplitText <input checked="" type="checkbox"/> Enabled</p> <p>Id 859ee0d1-0179-1000-1df4-3bf0f0be4504a</p> <p>Type SplitText 1.13.2</p> <p>Bundle org.apache.nifi - nifi-standard-nar</p> <p>Penalty Duration ? 30 sec</p> <p>Yield Duration ? 1 sec</p>			
<p>Automatically Terminate Relationships ?</p> <p><input type="checkbox"/> failure If a file cannot be split for some reason, the original file will be routed to this destination and nothing will be routed elsewhere</p> <p><input checked="" type="checkbox"/> original The original input file will be routed to this destination when it has been successfully split into 1 or more files</p> <p><input type="checkbox"/> splits The split files will be routed to this destination when an input file is successfully split into 1 or more split files</p>			

- Splits: se moverán los flowfiles con las particiones hacia el siguiente destino.
- VI. Añadimos un procesador debug como LogAttribute para comprobar que vamos por el buen camino.
- VII. Conectamos con la salida splits.



- VIII. Nos indica que nos falta la opción failure y una opción de configuración.
- a. La opción de configuración que nos falta es Line Split Count para indicar en cuántas partes vamos a dividir el fichero.

SETTINGS
SCHEDULING
PROPERTIES
COMMENTS

Required field

Property	Value
Line Split Count	1

- b. La conexión failure la enlazamos con el propio procesador para permitir el relanzado de Split en caso de fallo.

Create Connection

DETAILS
SETTINGS

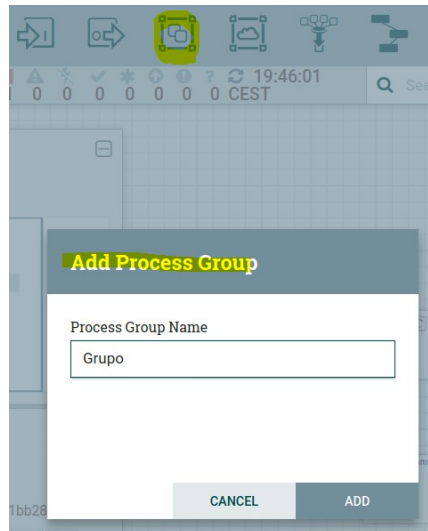
From Processor	To Processor
SplitText	SplitText
SplitText	SplitText
Within Group	Within Group
NiFi Flow	NiFi Flow
For Relationships	
<input checked="" type="checkbox"/> failure	
<input type="checkbox"/> original	
<input type="checkbox"/> splits	

- IX. Ahora copiamos un fichero por ejemplo el README que tenemos en el directorio de instalación de NiFi sobre nuestro directorio origen del flujo /home/hadoop/Documentos/in y damos a ejecutar para comprobar cómo viaja la información.

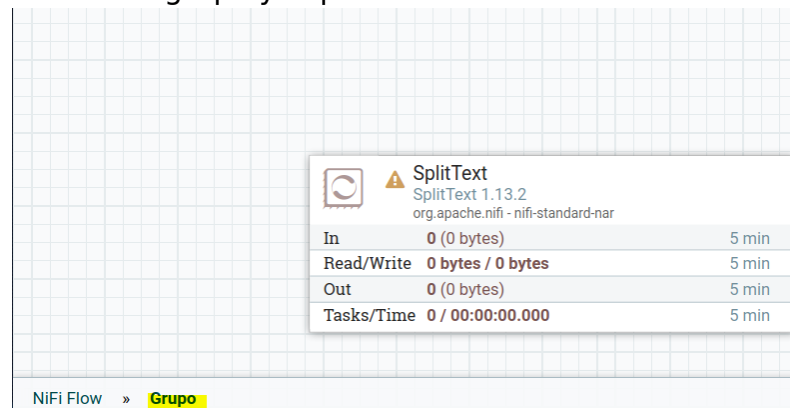
## 2.2. Conexiones y grupos

Ahora vamos a ver cómo crear conexiones entre diferentes grupos de procesos configurando lo que serían puertos de entrada y de salida.

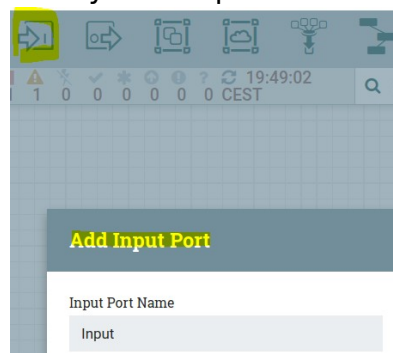
- I. Añadimos un Process Group que llamaremos grupo.



II. Seleccionamos el procesador SplitText y damos a ctrl+c para copiar. Abrimos el grupo y copiamos.



III. Añadimos un Input Port y un Output Port

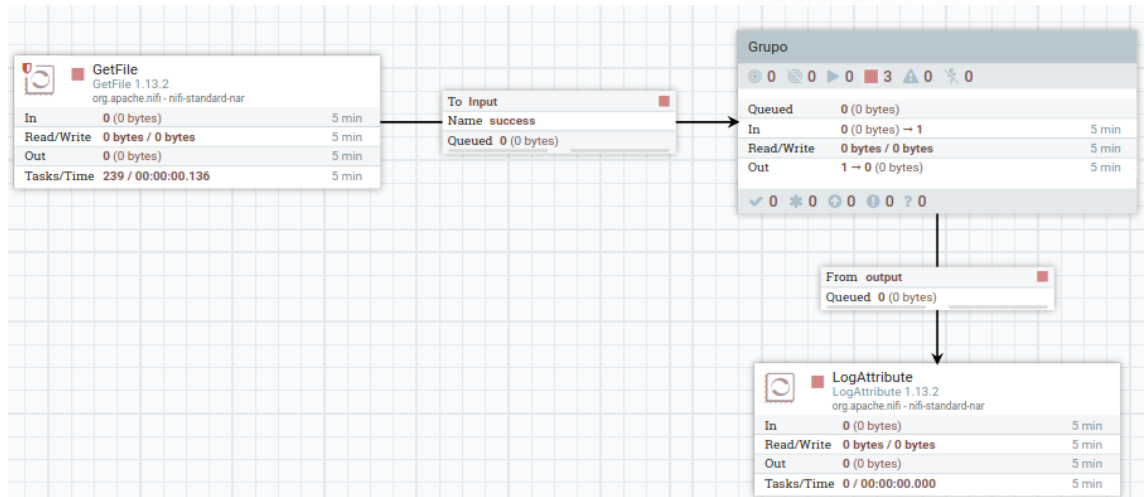


IV. Y unimos las conexiones:

- o Input → SplitText
- o SplitText:
  - Splits → Output
  - Original → Autoterminada
  - Failure → consigo mismo

V. Suprimimos las conexiones anteriores y establecemos las conexiones con el grupo.

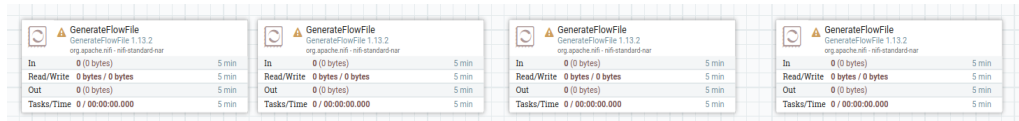




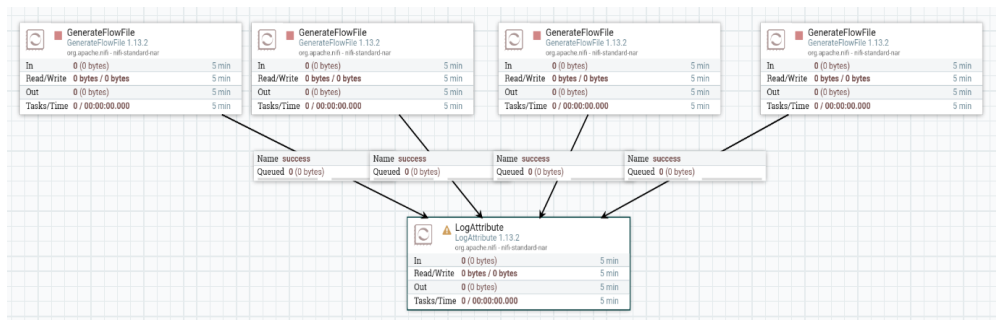
VI. Comprobamos que el funcionamiento es el mismo.

2.3. Ahora vamos a ver cómo trabajar con Funnels, los funnels te permite trabajar en paralelo y después unirlos en un único flujo de ejecución.

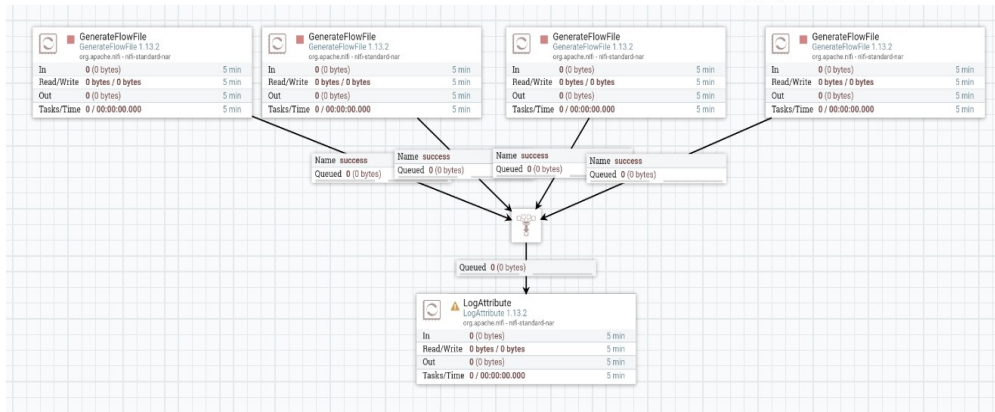
I. Para ello vamos a poner varios GenerateFlowFile (4 en este caso).



II. Añadimos un nuevo procesador LogAttribute y conectamos



III. Como podemos comprobar si quisiéramos hacer un cambio tendríamos que volver a conectar todas las salidas, para evitar esto añadimos un funnel que va a centralizar todas las conexiones en un único punto.



### 3. Proyecto con Apache Nifi.

#### 3.1. Crear un servicio Kafka de producción de mensajes.

- I. Nos vamos a: <https://kafka.apache.org/downloads>
- II. Y descargamos la versión 2.6.0 con scala 2.13

- Released Aug 3, 2020
- [Release Notes](#)
- Source download: [kafka-2.6.0-src.tgz](#) (asc, sha512)
- Binary downloads:
  - Scala 2.12 - [kafka\\_2.12-2.6.0.tgz](#) (asc, sha512)
  - **Scala 2.13 - [kafka\\_2.13-2.6.0.tgz](#) (asc, sha512)**

We build for multiple versions of Scala. This only matters if you are using Scala and you want a version built for the same Scala version you use. Otherwise any version should work (2.13 is recommended).

- III. Descomprimos:

```
tar xvf kafka_2.13-2.6.0.tgz
```

- IV. Entramos en el directorio de Kafka /bin y ejecutamos el proceso `zookeeper-server-start.sh` al que tendremos que pasarle el fichero como parámetro el fichero de configuración `zookeeper.properties` que se encuentra en el directorio `config` de Kafka donde estarán todas las propiedades por defecto del servidor de zookeeper desplegando el puerto 2181.

```
/kafka_2.13-2.6.0/bin$ ./zookeeper-server-start.sh ../config/zookeeper.properties
```

- V. Ahora abrimos otra pestaña para levantar el servidor de Kafka

```
kafka_2.13-2.6.0/bin$ ./kafka-server-start.sh ../config/server.properties
```

- VI. Si todo ha ido bien nos abrimos una nueva pestaña para crear nuestro topic de Kafka. Para en el propio directorio bin de Kafka tecleamos:

```
./kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic nifi
```

VII. Ahora creamos un productor de Kafka con el comando

```
./kafka-console-producer.sh --broker-list localhost:9092 --topic nifi
```

VIII. Si hemos ejecutado correctamente, nos devolverá el control pero en este punto vamos a dejarlo aquí y nos vamos a ir a Nifi.

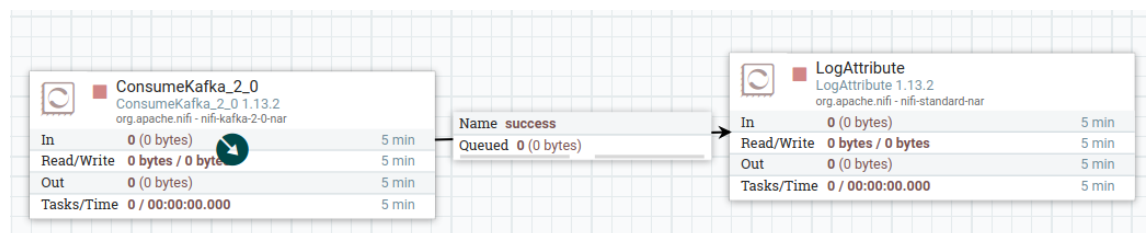
IX. Añadimos un procesador del tipo ConsumeKafka\_2\_0.

Displaying 6 of 283

Type ▲	Version	Tags
ConsumeKafkaRecord_1_0	1.13.2	PubSub, Consume, 1.0, Ingest, ...
ConsumeKafkaRecord_2_0	1.13.2	PubSub, Consume, Ingest, 2.0, ...
ConsumeKafkaRecord_2_6	1.13.2	PubSub, Consume, Ingest, Get, ...
ConsumeKafka_1_0	1.13.2	PubSub, Consume, 1.0, Ingest, ...
ConsumeKafka_2_0	1.13.2	PubSub, Consume, Ingest, 2.0, ...
ConsumeKafka_2_6	1.13.2	PubSub, Consume, Ingest, Get, ...

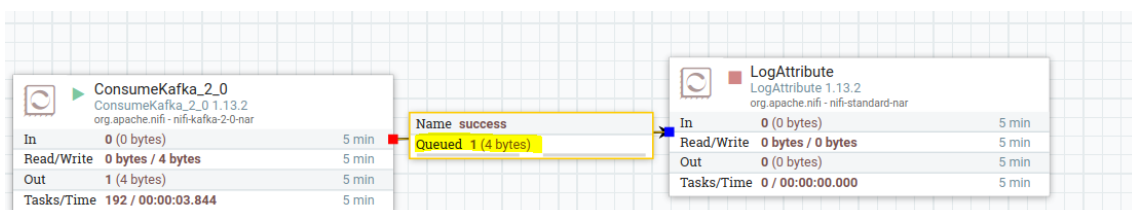
X. En properties le indicamos el nombre del **topic**: nifi y el **Group ID**:nifi

XI. Una vez completado, añadimos un procesador de tipo logAttribute para probar.



XII. Le damos a ejecutar y nos vamos a nuestro terminal donde habíamos levantado el productor de Kafka y tecleamos cualquier palabra.

XIII. En el flujo deberíamos ver encolado el mensaje.



### 3.2. Unimos nuestro productor de mensajes con una base de datos como Elasticsearch.

I. Para comenzar nos descargamos la bbdd de:  
<https://elastic.co/es/elasticsearch>

## El corazón del Elastic Stack, gratuito y abierto

Elasticsearch es un motor de búsqueda y analítica de RESTful distribuido capaz de abordar un número creciente de casos de uso. Como núcleo del Elastic Stack, almacena de forma central tus datos para una búsqueda a la velocidad de la luz, relevancia refinada y analíticas poderosas que escalan con facilidad.

Pruébalo ahora

↓ Descarga Elasticsearch

II. La versión para el SO con el que estéis trabajando.

Version: 7.12.1  
Release date: April 27, 2021  
License: [Elastic License 2.0](#)  
Downloads: [WINDOWS](#) [sha](#) [asc](#) [MACOS](#) [sha](#) [asc](#)  
[LINUX X86\\_64](#) [sha](#) [asc](#) [LINUX AARCH64](#) [sha](#) [asc](#)  
[DEB X86\\_64](#) [sha](#) [asc](#) [DEB AARCH64](#) [sha](#) [asc](#)  
[RPM X86\\_64](#) [sha](#) [asc](#) [RPM AARCH64](#) [sha](#) [asc](#)  
[MSI \(BETA\)](#) [sha](#) [asc](#)

III. La descomprimos desde el terminal y nos movemos dentro del directorio, ejecutando el comando: `./bin/elasticsearch`

IV. Para comprobar que ha ido bien, nos abrimos otra pestaña y tecleamos:

```
curl -X GET "localhost:9200/_cat/health?v=true&pretty"
```

V. Con todo ya listo nos vamos a nifi y seleccionamos añadir un procesador de tipo PutElasticSearchHttp indicándole la dirección con el protocolo:

SETTINGS SCHEDULING PROPERTIES COMMENTS

Required field

Property	Value
Elasticsearch URL	1 http://localhost:9200
SSL Context Service	
Character Set	
Username	
Password	
Connection Timeout	
Response Timeout	
Proxy Configuration Service	

☐ Set empty string

CANCEL OK

VI. en la misma configuración, indicamos en el índice "ficheros"

Proxy Username

Proxy Password

Identifier Attribute

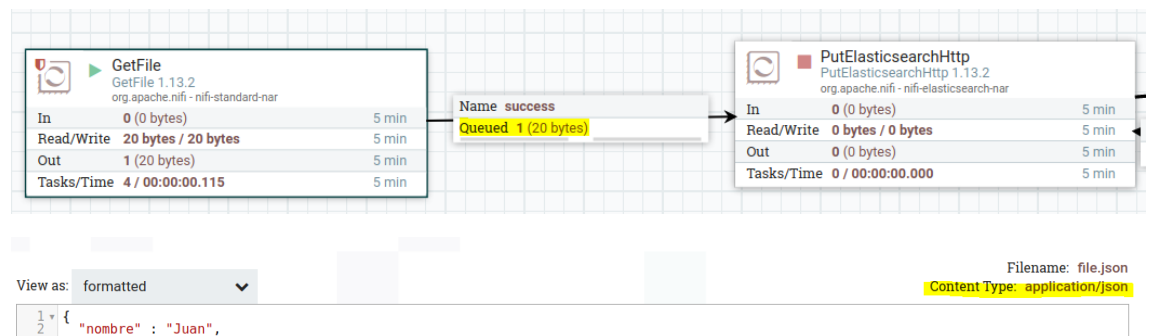
Index

Type

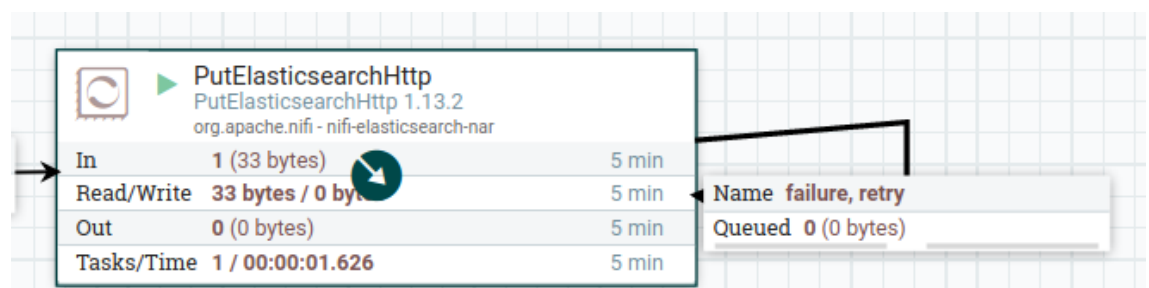
1 ficheros

VII. Las conexiones retry y failure le damos a autoterminar.

VIII. Ahora añadimos un procesador de tipo GetFile y los unimos. Solamente para realizar la prueba generamos un fichero json sencillo en el directorio input:



IX. Comprobamos que el procesador procesa correctamente el fichero sin encolarlo en las salidas de error.



```
hadoop@hadoop-VirtualBox:~/Descargas/elasticsearch-7.12.1/bin$ curl -X GET "localhost:9200/ficheros/_search?pretty"
{
  "took" : 377,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "ficheros",
        "_type" : "_doc",
        "_id" : "ArAphnkBKvwtS96TjKql",
        "score" : 1.0,
        "_source" : {
          "nombre" : "Juan",
          "edad" : 30
        }
      }
    ]
  }
}
```

- X. Ahora vamos a utilizar un procesador que nos va a permitir enrutar el flujo en función de un determinado campo de json. Para ello usamos EvaluateJsonPath.

Displaying 4 of 283

Type	Version	Tags
EvaluateJsonPath	1.13.2	JSON, JsonPath, evaluate

- XI. Indicamos que el campo que vamos a utilizar para enrutar se almacene como un atributo.

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Required field			
Property		flowfile-attribute	
Destination			
Return Type			
Path Not Found Behavior			
Null Value Representation		empty string	

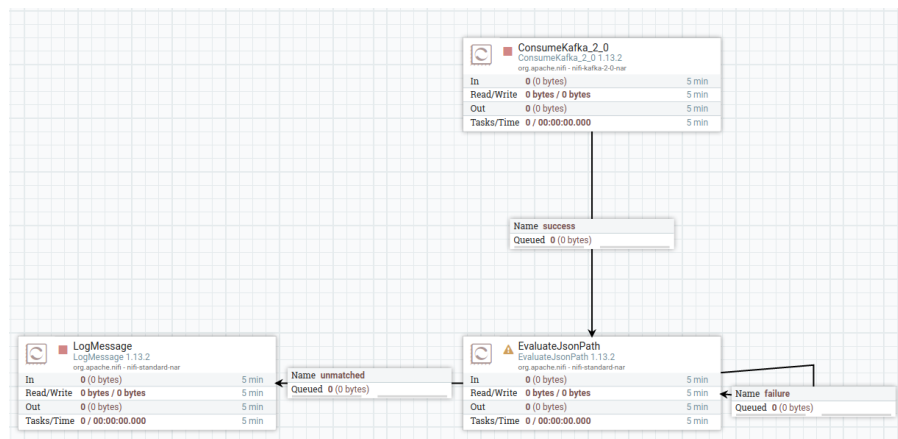
- XII. Añadimos un atributo llamado experiencia que tendrá como value \$.experiencia

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Required field			
Property	Value		
Destination	flowfile-attribute		
Return Type	auto-detect		
Path Not Found Behavior	ignore		
Null Value Representation	empty string		
experiencia	\$.experiencia		

- XIII. Añadimos un nuevo procesador del tipo LogMessage que conectamos con EvaluateJsonPath con la conexión unmatched. Una vez hecho esto damos a configurar para que nos muestre el mensaje.

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Required field			
Property	Value		
Log Level	Info		
Log prefix	No value set		
Log message	JSON No Valido		

- XIV. Ahora, limpiamos el área de trabajo y unimos ConsumeKakfa\_2\_0 con EvaluateJson. El aspecto de nuestro flujo debe ser el siguiente:



- XV. Para enrutar en función de un atributo añadimos un nuevo procesador:

Displaying 2 of 283

Type	Version	Tags
RouteOnAttribute	1.13.2	regex, string, Attribute Express...

- XVI. Configuramos el properties para que enrute según un determinado atributo

EL ✓ PARAM ✓

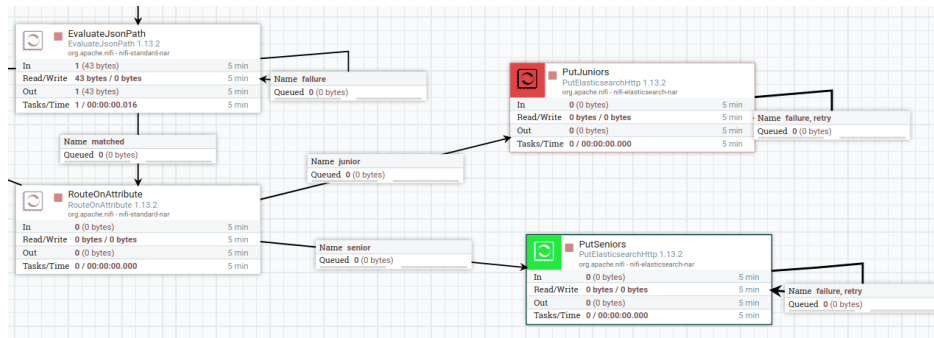
1 `${experiencia:ge(10)}`

☐ Set empty string

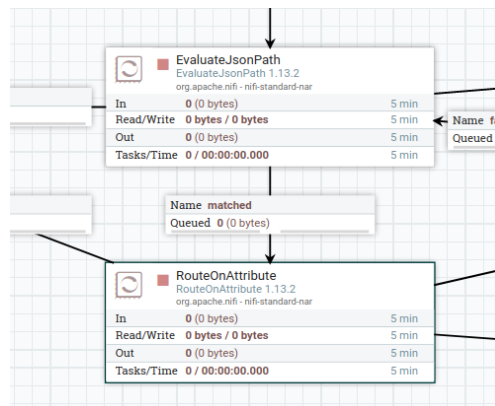
CANCEL OK

Property	Value
Routing Strategy	Route to Property name
senior	\${experiencia:ge(10)}
junior	\${experiencia:lt(10)}

- XVII. Ahora copiamos los procesadores PutElasticSearch y cambiamos los índices en cada caso:
- juniors
  - seniors
- XVIII. En ambos casos las conexiones: retry y failure irán sobre el propio procesador. Success autocompletada.
- XIX. Unimos RouteOnAttribute con cada caso de PutElasticSearch como se muestra a continuación:



- XX. Por último, unimos EvaluateJsonPath con RouteOnAttribute



Si se han seguido todos los pasos correctamente, deberíamos crear un json desde el productor de Kafka con tu nombre y años de experiencia y enrutarse por la rama correcta.

```
hadoop@hadoop-VirtualBox: ~/Descargas/kafka_2.13-2.6.0/bin$ ./kafka-console-producer.sh --broker-list localhost:9092 --topic nifi {"nombre":"Carlos López","experiencia":10}
```



Adjuntar captura de pantalla del comando curl -X GET en función del índice donde se haya enrutado tu json.

### He introduït un senior i un junior

```
alvar@ultrarapid:~$ curl -X GET "localhost:9200/_search?pretty"
curl: (3) Failed to convert -X to ACE; string contains a disallowed character

curl: (6) Could not resolve host: GET
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 3,
    "successful" : 3,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "fitxers",
        "_type" : "_doc",
        "_id" : "6ekK8HkBprD0rqq2mYWH",
        "_score" : 1.0,
        "_source" : {
          "nom" : "Joan",
          "edat" : 20
        }
      },
      {
        "_index" : "junior",
        "_type" : "_doc",
        "_id" : "6-kh8HkBprD0rqq2QIV_",
        "_score" : 1.0,
        "_source" : {
          "nom" : "Joan",
          "experiencia" : 5
        }
      },
      {
        "_index" : "senior",
        "_type" : "_doc",
        "_id" : "6ukh8HkBprD0rqq20YUs",
        "_score" : 1.0,
        "_source" : {
          "nom" : "Alvar",
          "experiencia" : 35
        }
      }
    ]
  }
}
```

Adjuntar captura final del flujo.

Enviar memoria a [clopez@teralco.com](mailto:clopez@teralco.com) para poder hacer seguimiento.