# NIFI

**Apache NiFi** es una herramienta de software para la automatización del flujo de datos entre sistemas de software. Se utiliza ampliamente para la ingesta, procesamiento y distribución de datos en entornos de big data y en aplicaciones que requieren el procesamiento de flujos de datos en tiempo real o por lotes. NiFi fue desarrollado originalmente por la Agencia de Seguridad Nacional de Estados Unidos (NSA) y luego donado a la Apache Software Foundation, donde se ha desarrollado y mantenido como un proyecto de código abierto.

NiFi se caracteriza por su interfaz gráfica de usuario basada en web, que permite a los usuarios diseñar, configurar y monitorear flujos de datos de manera fácil y visual. Ofrece una amplia gama de procesadores preconstruidos para diversas tareas de manipulación de datos, como la obtención de datos de diversas fuentes, transformación de datos, enrutamiento basado en condiciones, y más.

Las principales características y usos de NiFi incluyen:

1. **Facilidad de uso**: gracias a su interfaz gráfica, los usuarios pueden arrastrar y soltar componentes para construir flujos de datos sin necesidad de escribir código.
2. **Flexibilidad**: NiFi puede integrarse con una amplia variedad de sistemas de origen y destino, incluyendo bases de datos, sistemas de archivos, servicios web, colas de mensajes, y más.
3. **Escalabilidad**: NiFi puede escalar para manejar grandes volúmenes de datos y complejidad en los flujos de datos.
4. **Fiabilidad**: ofrece características robustas de manejo de errores y reintentos, asegurando que los datos se procesen de manera confiable.
5. **Seguridad**: incluye características de seguridad como la autenticación, autorización, y cifrado de datos en tránsito.
6. **Procesamiento en tiempo real y por lotes**: NiFi es adecuado tanto para el procesamiento de flujos de datos en tiempo real como para el procesamiento de grandes lotes de datos.
7. **Proveniencia de datos**: NiFi registra detalladamente el camino que los datos toman a través del sistema, lo que facilita la auditoría y el diagnóstico de problemas.

Sus casos de uso son:

* transferencias de datos entre sistemas, por ejemplo, de JSON a una base de datos, de un FTP a Hadoop, etc….
* preparar y enriquecer los datos.
* enrutar datos en función de características, con diferentes prioridades.
* conversión de datos entre formatos.

En cambio, no es apropiado para:

* procesamiento y operaciones de cómputo distribuido.
* operaciones de streaming complejas con joins o agregaciones.
* procesamiento complejo de eventos.

## PUESTA EN MARCHA

Antes de instalar Nifi hay que instalar java si no lo tenemos instalado y crear la variable de entorno JAVA\_HOME

***sudo apt install -y openjdk-21-jdk***

***nano /home/iabd/.bashrc***

***/home/iabd/.bashrc***

export JAVA\_HOME=/usr/lib/jvm/java-21-openjdk-amd64

***source ./.bashrc***

Para instalar Nifi, sólo hemos de descargar la última versión desde [https://nifi.apache.org](https://nifi.apache.org/) (el binario standard).

Descomprimimos el fichero y lo guardamos en la carpeta apropiada:

*unzip nifi-2.0.0-M2-bin.zip*

*sudo mv nifi-2.0.0-M2 /opt/nifi*

***sudo chown -R iabd:iabd /opt/nifi***

Ejecutamos el siguiente comando indicando el usuario y contraseña que queramos (por ejemplo, nifi/nifinifinifi):

*/opt/nifi/bin/nifi.sh set-single-user-credentials nifi nifinifinifi*

Podemos arrancar Nifi con el siguiente comando (se ejecutará en segundo plano):

*/opt/nifi/bin/nifi.sh start*

Si queremos detenerlo ejecutaremos: */opt/nifi/bin/nifi.sh stop*

Esperaremos un par de minutos tras arrancar Nifi para acceder al entorno de trabajo. Para ello, introduciremos en el navegador la URL *https://localhost:8443/nifi* y tras aceptar la alerta de seguridad respecto al certificado veremos un interfaz similar a la siguiente imagen:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Respecto al interfaz de usuario, cabe destacar cuatro zonas:

* Menú superior: con los iconos grandes (procesadores, puertos de entrada y salida, etc..)
* Barra de debajo con iconos: indican el estado de la ejecución (hilos, procesadores en marcha, detenidos, etc.)
* Cuadro Navigate para hacer zoom
* Cuadro Operate con las opciones del flujo de trabajo o del recurso seleccionado
* Zona de trabajo drag&drop.

## COMPONENTES

### FLOWFILE

Flowfile, fichero de flujo o FF es básicamente el dato, el cual se persiste en disco tras su creación. Realmente es un puntero al dato en su almacenamiento local.

El Flowfile a su vez se compone de dos partes:

* contenido: el dato en sí
* atributos: metadatos en pares de clave/valor

### PROCESADOR

El procesador es el encargado de ejecutar alguna transformación o regla sobre los datos o el flujo para generar un nuevo Flowfile. La salida de un procesador es un Flowfile que será la entrada de otro procesador. Así pues, para implementar un flujo de datos en NiFi, crearemos una secuencia de procesadores que reproduzcan las acciones y transformaciones que queremos realizar sobre sobre los datos.

Todos los procesadores se ejecutan en paralelo (mediante diferentes hilos de ejecución), abstrayendo la complejidad de la programación concurrente y además se pueden ejecutar en varios nodos de forma simultánea o bien en el nodo primario de un clúster.

Si bien es posible diseñar a mano un procesador, por defecto, NiFi ofrece un amplio catálogo con más de 300 procesadores, que cubre ampliamente las operaciones más frecuentes que se van a necesitar en un flujo de datos: añadir o modificar los atributos del FF, capturar cambios en una base de datos, cambiar de formato el contenido del FF (JSON, CSV, Avro…), extraer el contenido textual de un fichero, extraer el valor de un campo de un fichero JSON a partir de su ruta, extraer cabeceras de un email, consultar a ElasticSearch, geolocalizar una IP, obtener un fichero de una cola Kafka, escribir en un log, unir el contenido de varios FFs, realizar una petición HTTP, transformar un fichero XML, validar un fichero CSV, enviar mensajes a un web socket, etc.

Los más utilizados son:

* Transformación de datos: ReplaceText, JoltTransformJSON, CompressContent.
* Enrutado y mediación: RouteOnAttribute, RouteOnContent
* Acceso a base de datos: ExecuteSQL, ConvertJSONToSQL, PutSQL
* Extracción de atributos: EvaluateJsonPath, ExtractText, UpdateAttribute
* Interacción con el sistema: ExecuteProcess
* Ingestión de datos: GetFile, GetFTP, GetHTTP, GetHDFS
* Envío de datos: PutEmail, PutFile, PutFTP, PutKafka, PutMongo
* División y agregación: SplitText, SplitJson, SplitXml, MergeContent
* HTTP: GetHTTP, ListenHTTP, PostHTTP
* AWS: FetchS3Object, PutS3Object, PutSNS, PutSQS

### CONECTOR

Así pues, los conectores van a unir la salida de un procesador con la entrada de otro (o un procesador consigo mismo, por ejemplo, para realizar reintentos sobre una operación).

Las conexiones se caracterizan y nombran por el tipo de puerto de salida del procesador del que nacen. En la mayoría de los casos nos enfrentaremos a conexiones de tipo success, que recogen el FF que devuelve un procesador cuando ha terminado satisfactoriamente su tarea, o de tipo failure, que conducen el FF en los casos en los que la tarea ha fallado.

Existe la posibilidad de configurar algunos aspectos de la conexión, como el número de FF que pueden viajar de forma simultánea por ella, la prioridad de salida de los FF que hay en la conexión, o el tiempo que los FF deben permanecer a la espera para ser recogidos por el procesador de destino.

## EJEMPLO 1 - MOVIENDO DATOS

Vamos a hacer un pequeño ejercicio con Nifi para familiarizarnos con el entorno desarrollando un flujo de datos sencillo que mueva un fichero de un directorio a otro:

1. Seleccionamos un procesador (primer icono grande) y lo arrastramos en nuestra área de trabajo.
2. Nos aparece un dialogo con tres partes diferenciadas:
   * A la izquierda una nube de etiquetas para poder filtrar los procesadores.
   * Arriba a la derecha tenemos un buscador para buscar procesadores por su nombre
   * La parte central con el listado de procesadores, desde donde lo podemos seleccionar.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Así pues, buscamos el procesador GetFile y lo añadimos al flujo, el cual permite recuperar un fichero desde una carpeta.

1. Hacemos doble click sobre en el procesador, se abre una ventana para configurarlo y vamos a la pestaña properties. Indicamos el directorio de entrada de donde se recogerá el fichero en la propiedad Input Directory:

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

En la pestaña Settings, en el campo Name ponemos ObtenerFichero.

1. Ahora añadimos un nuevo procesador de tipo [PutFile](https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/latest/org.apache.nifi.processors.standard.PutFile/index.html), y en las propiedades indicamos el directorio de salida en la propiedad Directory a /home/iabd/Documentos/out.
2. En la pestaña Settings, le ponemos el nombre PonerFichero.
3. En la pestaña Relationships podemos configurar el comportamiento a seguir si el procesador se ejecuta correctamente (success) o falla (failure).

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Si nos olvidamos de autoterminar las relaciones, o tenemos conexiones sin conectar, no podremos iniciar los procesadores implicados.

1. Unimos ambos procesadores creando una conexión. Para ello, tras pulsar sobre el icono de la flecha que aparece al dejar el ratón sobre el primer procesador lo arrastramos hasta el segundo.

Imagen que contiene Calendario

Descripción generada automáticamente

1. Antes de arrancar el primer procesador, creamos un pequeño fichero en el directorio que hemos puesto como entrada:

*echo "Hola IABD!" > hola-iabd.txt*

1. Arrancamos el procesador mediante el botón derecho y la opción start, y comprobamos que el fichero ya no está en la carpeta in, y que sí aparece en la cola (queued 1), también podemos comprobar como tampoco está en la carpeta out.

Diagrama

Descripción generada automáticamente

1. Finalmente, arrancamos el procesador PonerFichero, y veremos como la cola se vacía y el archivo aparece en la carpeta out.

### GESTIONANDO LOS ERRORES

¿Qué sucede si leemos dos veces un archivo con el mismo nombre? Tal como lo hemos definido en nuestro flujo, sólo se guardará la primera copia.

Si vamos a la pestaña Properties del procesador PonerFichero, podemos cambiar este comportamiento en la propiedad Conflict Resolution Strategy a replace, de esta manera, se guardará el último archivo.

Interfaz de usuario gráfica

Descripción generada automáticamente

Realmente, en vez de decidir si lo ignora o lo sobrescribe, lo ideal es definir un nuevo flujo que dependa del estado de finalización del procesador. De esta manera, podremos almacenar todos los archivos que han llegado con el mismo nombre para su posterior estudio.

Así pues, vamos a quitar la autoterminación que antes habíamos puesto al procesador de PonerFichero, para que cuando falle, redirija el flujo a un nuevo procesador PutFile que coloque el archivo en una nueva carpeta (en nuestro caso en /home/iabd/Documentos/conflictos):

Diagrama

Descripción generada automáticamente

Aunque ahora tenemos un mecanismo para almacenar los ficheros que coinciden en nombre, sólo nos guardará uno (nos sucede lo mismo que antes, pero ahora sólo con los repetidos).

Así pues, necesitamos renombrar los ficheros que vayamos a colocar en la carpeta conflictos para guardar el histórico. Para ello, necesitamos introducir un procesador previo que le cambie el nombre al archivo.

Nifi añade la propiedad filename a todos los FF. Esta propiedad la podemos consultar mediante el [Nifi Expression Language (Nifi EL)](https://nifi.apache.org/docs/nifi-docs/html/expression-language-guide.html) y haciendo uso del procesador UpdateAttribute modificar su valor. Así pues, vamos a colocar el procesador UpdateAttribute antes de colocar los archivos en la carpeta de conflictos:

Diagrama

Descripción generada automáticamente

Hemos decidido añadir como prefijo al nombre del archivo la fecha del sistema en formato de milisegundos, de manera que obtendremos archivos similares a 1637151536113-fichero.txt. Para ello, añadimos un nuevo atributo que llamaremos filename haciendo clic sobre el icono de + y en su valor ponemos la expresión ${now():toNumber()}-${filename}:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## EJEMPLO 2 - TRABAJANDO CON ATRIBUTOS

Cada vez que se generan FF 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. Como ya hemos visto, mediante el uso de procesadores podremos modificar estos o añadir nuevos atributos.

### GENERANDO CONTENIDO

Vamos a añadir un procesador del tipo GenerateFlowFile (este procesador crea FF con datos aleatorios o contenidos personalizados, lo cual es muy útil para testear y depurar nuestros flujos de datos).

En las opciones del procesador vamos a la pestaña de Properties y completamos los campos:

* Flow Size: 10B (bytes)
* Batch Size: 1 para que nos genere un FF por cada ejecución
* Data Format: Text
* Unique FlowFiles: true (para indicar que los FF van a ser únicos)

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

A continuación, en la pestaña de planificación (Scheduling) de este procesador vamos a indicar que se ejecute cada 3 segundos (Run Schedule: 3s).

Después vamos a añadir el procesador ReplaceText con el que cambiaremos el texto. Tras ello, conectamos ambos procesadores.

Calendario

Descripción generada automáticamente con confianza baja

Si nos fijamos, a la izquierda del nombre del procesador, aparece un icono de aviso, el cual nos indica que necesitamos configurar el nuevo procesador, además de indicarnos que ambas relaciones no están conectadas o que faltan por autocompletar.

Para ello, configuramos la estrategia de reemplazo para que lo haga siempre (en el campo Replacement Value seleccionamos Always Replace). Además, en el campo Replacement Value vamos a indicar simplemente prueba. Finalmente, marcamos para que autotermine la conexión failure.

Añadimos un procesador de tipo [LogAttribute](https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/latest/org.apache.nifi.processors.standard.LogAttribute/index.html) para mostrar en el log los atributos del FF, y conectamos el procesador anterior (ReplaceText) a éste mediante la relación success. No nos podemos olvidar de indicar que autotermine.

Diagrama

Descripción generada automáticamente

Arrancamos el primer procesador y visualizamos la cola para comprobar qué ha generado. Para ello, sobre la cola elegimos la opción list queue y visualizamos alguno de los elementos de la cola:

Tabla

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Si ejecutamos el siguiente procesador, vemos que saca el FF de la cola anterior y aparecerá en la siguiente. Si comprobamos su valor, veremos que ha cambiado el valor original por prueba.

### AÑADIENDO UN ATRIBUTO

Ahora vamos a extraer el contenido del FF a un atributo mediante el procesador ExtractText.

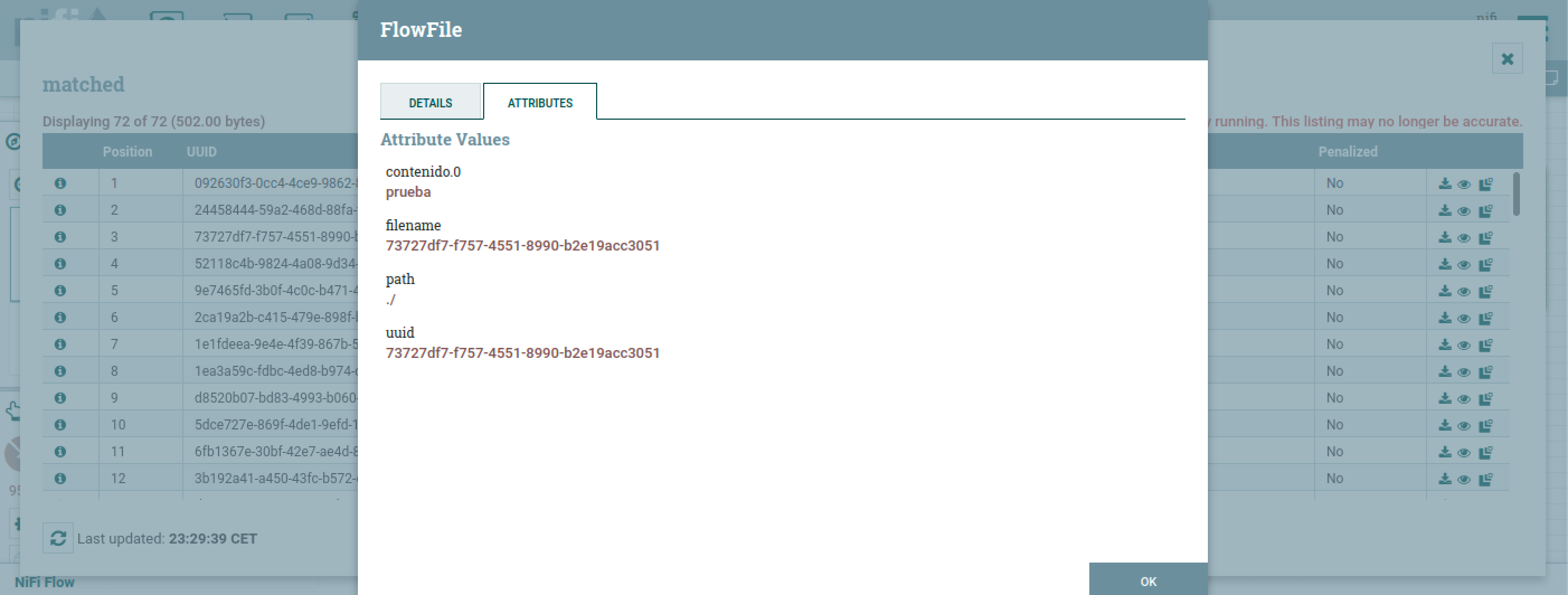
En las propiedades, creamos una nueva propiedad que llamaremos contenido, y en cuyo valor vamos a poner la expresión .\* que indica que queremos que coincida con todo.

Una vez creado, vamos a colocar este procesador entre los dos anteriores (para el segundo con el caso matched, que es cuando ha coincidido con la expresión regular). En la conexión unmatched la marcamos para que autotermine.

Diagrama

Descripción generada automáticamente

Finalmente, ejecutamos los procesadores y podemos acceder a la última cola, y en algún flujo hacemos clic en el icono i y hacemos las comprobaciones en la pestaña Attributes.



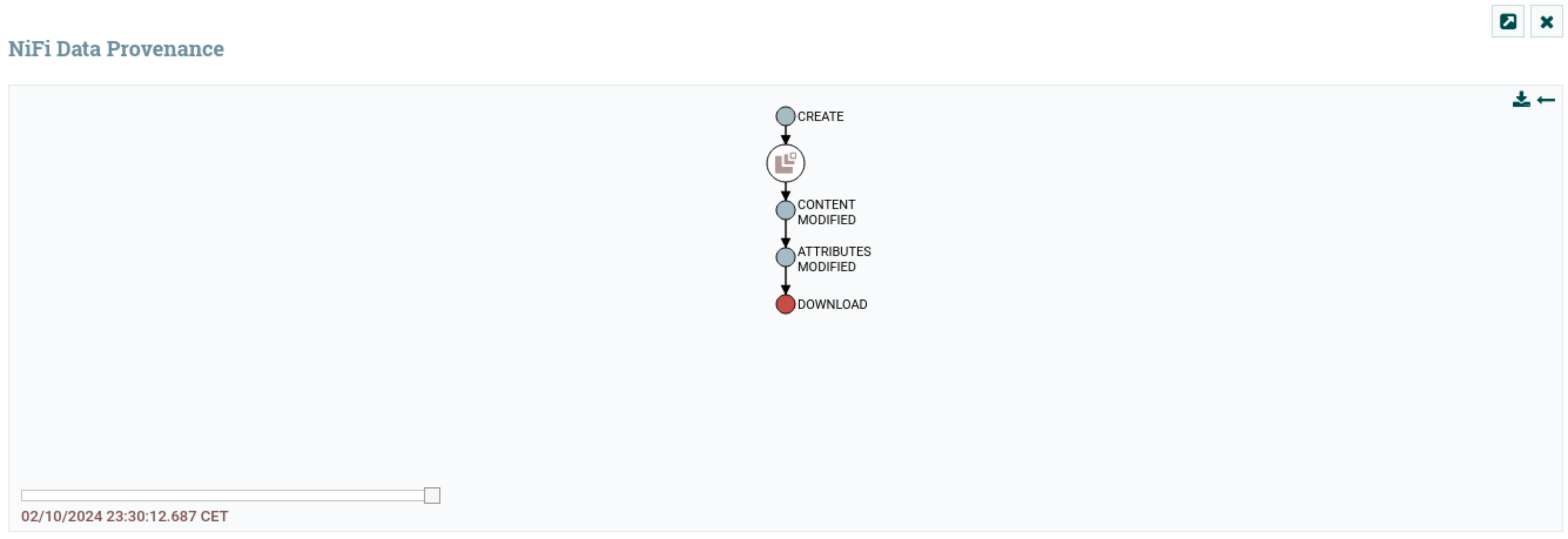
### LINAJE DE LOS DATOS

Para comprobar el dato final, es muy útil utilizar la opción de Data provenance, la cual nos ofrece un linaje de los datos (de donde viene el dato desde su origen).

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Podemos hacer clic en Show Linaje para ver el flujo del dato:



## EJEMPLO 3 - FILTRADO DE DATOS

En este caso, vamos a coger los datos de ventas pdi\_sales\_small.csv que está junto al enunciado del ejercicio.

Utilizando Nifi, vamos a crear un nuevo fichero CSV que contenga únicamente los datos de Francia que han realizado más de una venta.

Para ello, tendremos que leer el fichero haciendo uso del procesador GetFile, separar cada fila en un FF mediante SplitRecord, filtrar los datos usando el procesador QueryRecord y finalmente los almacenaremos en disco gracias al procesador PutFile.

### LECTURA Y DIVISIÓN

Así pues, comenzamos leyendo el fichero con el procesador GetFile (en Directory ponemos el nombre del directorio y File Filter el nombre del fichero). En este caso vamos a dejar la opción keep source file a true para que no lo elimine.

Mediante el procesador SplitRecord, vamos a separar cada fila del CSV a un FF:

* Record Reader: Create new service 🡪 CSVReader
* RecordWriter: Create new service 🡪 CVSRecordSetWriter

Para configurar estos servicios, pulsaremos sobre la flecha, y veremos la pantalla de configuración.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para cada uno de ellos, tendremos otros tres iconos: la rueda para configurar, el rayo para activar/desactivar el servicio y la papelera para eliminarlo.

En la configuración indicamos el ; como separador de campos tanto para la lectura como la escritura en el campo Value Separator y marcamos como true que el archivo contiene la primera fila con encabezados (Treat First Line as Header))y pulsamos sobre el rayo para activar ambos servicios.

Finalmente, en el campo Records Per Split le indicamos 1 para que coloque cada fila en un FF.

Interfaz de usuario gráfica, Texto, Aplicación, Word, Correo electrónico

Descripción generada automáticamente

### FILTRADO DE FF

En este paso, mediante el procesador QueryRecord vamos a ejecutar una consulta SQL contra el FF. El resultado del nuevo FF será el resultado de la consulta. En este caso, vamos a quedarnos con las ventas de más de una unidad realizadas en Francia.

Configuramos los mismos Record Reader y Record Writer. Además, hemos de poner la propiedad Include Zero Record FlowFiles a false para que no vuelva a enrutar los FF que no cumplan la consulta. Finalmente, añadimos una nueva propiedad para indicar la consulta. En nuestro caso la hemos denominado FranciaMayor1 y en el contenido ponemos la consulta: "select \* from Flowfile where Country = 'France' and Units > 1"

Finalmente, igual que hicimos en el ejemplo 1, vamos a cambiarle el nombre a cada FF para generar un archivo por cada resultado mediante UpdateAttribute y persistimos los datos con PutFile.

El resultado del flujo de datos será similar a:

Diagrama

Descripción generada automáticamente

## EJEMPLO 4 - FUSIONAR DATOS

Vamos a realizar los siguientes pasos:

* Escuchar datos que recibimos de un servidor web.
* Reconocer si el mensaje contiene la cadena error.
* Fusionar los mensajes en varios ficheros, dependiendo de si contienen un error.
* Almacenar el fichero resultante en MongoDB.

### RECIBIENDO DATOS VIA HTTP

Vamos a utilizar el procesador [ListenHTTP](https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/latest/org.apache.nifi.processors.standard.ListenHTTP/index.html) para escuchar peticiones HTTP. Para ello, lo añadimos a nuestro flujo de trabajo y configuramos:

* Listening port (puerto de escucha): 8081
* Base Path (endpoint de la petición): iabd

A continuación, para distinguir entre los diferentes datos de entrada, utilizaremos el procesador [RouteOnContent](https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/latest/org.apache.nifi.processors.standard.RouteOnContent/index.html), con el objetivo de separar en dos flujos de datos, los que contienen la cadena error y los que no. Para ello, tras añadir el procesador, le conectamos al flujo success que viene de ListenHTTP, y configuramos:

* Match Requirement: content must contain match (no tienen que coincidir exactamente, sino contener el valor).
* textoError: ERROR (hay que añadir la propiedad)

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para poder probarlo, arrancamos el primer procesador, y desde un terminal, hacemos una petición a:

*curl --data "texto de prueba" http://localhost:8081/iabd*

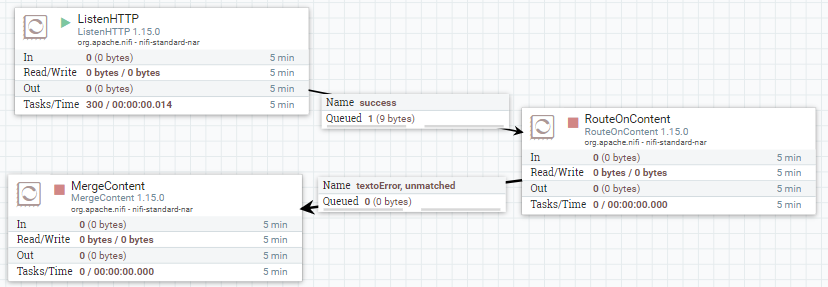
Si comprobamos la cola, podremos ver como se ha creado un FF cuyo contenido es texto de prueba.

### FUSIONANDO CONTENIDO

Si nos fijamos en las propiedades del procesador RouteOnContent, tenemos dos flujos de salida textoError matched (para los mensajes que contienen el texto ERROR) y unmatched (para los que no).

Vamos a añadir el procesador [MergeContent](https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/latest/org.apache.nifi.processors.standard.MergeContent/index.html), el cual recibe un conjunto de FF y los fusiona en uno a partir de la estrategia de fusión que defina el usuario.

Así pues, vamos a conectar las relaciones textoError y unmatched con MergeContent:



Tras conectar los procesadores, vamos a configurar el procesador MergeContent:

* En la pestaña de planificación, vamos a poner que el procesador se ejecuta cada 30 segundos para poder agrupar varios FF.
* En las propiedades:
  + Merge Strategy: Bin-Packing Algorithm (se fusionan en grupos).
  + Merge Format: Binary Concatenation (concatena los contenidos de los FF en un único FF)
  + Correlation Attribute Name: RouteOnContent.Route (crea diferentes grupos para textoError y unmatched)
  + Maximum Number of Entries: 1000 (cantidad máxima de elementos por grupo)
  + Max Bin Age: 300s (fuerza que el fichero fusionado salga como muy tarde a los 300s)
  + Delimiter Strategy: Text (para concatenar los ficheros utilizando una nueva línea como carácter delimitador)
    - Demarcator: al abrir el campo, pulsar Shift/Mayús + Intro para poner el carácter del salto de línea.

Puedes ir probando cada paso ejecutando los siguientes comandos:

*curl --data "texto de prueba" http://localhost:8081/iabd*

*curl --data "este sí que tiene ERROR" http://localhost:8081/iabd*

*curl --data "vaya ERROR más tonto" http://localhost:8081/iabd*

*curl --data "nifi mola mucho" http://localhost:8081/iabd*

### GUARDANDO EL RESULTADO A MONGODB

Para almacenar el resultado de las fusiones anteriores, vamos a guardar los resultados en una colección de MongoDB.

Suponemos que ya tenemos instalado MongoDB en nuestro sistema. Si no, podemos lanzarlo mediante Docker.

Si queremos utilizarlo mediante Docker, necesitamos que MongoDB y Nifi estén dentro del mismo contenedor. Para ello, podemos configurarlo mediante el siguiente archivo [docker-compose.yml](https://aitor-medrano.github.io/iabd/dataflow/resources/nifi-mongodb/docker-compose.yml) (si tuvieras algún problema con la imagen de MongoDB y tu procesador, prueba a cambiar la imagen de mongo por mongo:4.4):

*sudo apt-get install docker-compose*

*nano docker-compose.yml*

***docker-compose.yml***

services:

mongodb:

ports:

- "27017:27017"

image: mongo:latest

Una vez creado el archivo, construimos el contenedor mediante:

*sudo docker-compose -p nifimongodb up -d*

Para poder meter los mensajes en MongoDB, vamos a preparar el contenido para que esté en formato JSON. Además del contenido, vamos a crear un atributo con el nombre del enrutador utilizado para posteriormente poder filtrar los mensajes de error.

Para poder crear el formato JSON, utilizaremos el procesador [AttributesToJSON](https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/latest/org.apache.nifi.processors.standard.AttributesToJSON/index.html). Pero, previamente necesitamos pasar los mensajes desde el contenido de los FF a los atributos (para ello, igual que en el caso anterior, utilizaremos el procesador ExtractText). A su vez, también crearemos un nuevo atributo con el nombre del enrutador mediante el procesador UpdateAttribute.

El resultado final será similar al siguiente flujo:

Diagrama, Esquemático

Descripción generada automáticamente

Utilizamos el procesador ExtractText para pasar el contenido a los atributos:

* Enable DOTALL mode: true, para que incluya los saltos de línea como contenido.
* Añadimos una nueva propiedad contenido, y como expresión regular introducimos (.\*)

Una vez creado, conectamos MergeContent con ExtractText mediante la conexión merged, y el resto de las conexiones las marcamos para que autoterminen.

Añadimos el procesador UpdateAttribute, y dentro de las propiedades, añadiremos una nueva propiedad que vamos a llamar estado cuyo valor será ${RouteOnContent.Route}.

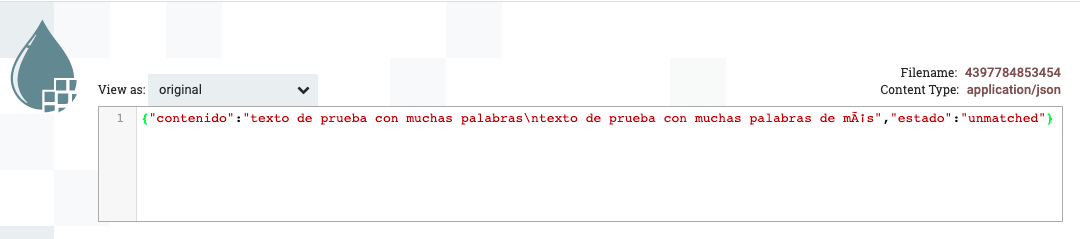
Una vez creado, conectamos ExtractText con UpdateAttribute mediante la conexión matched, y el resto de las conexiones las marcamos para que autoterminen.

A continuación, vamos a utilizar el procesador AttributesToJSON para pasar los atributos contenido y estado como contenido de un FF en formato JSON.

* Attribute List: contenido.0,estado (ponemos contenido contenido.0 en vez de contenido porque si vamos a en la cola a las propiedades de un elemento, en la pestaña Attribute veremos que la propiedad se llama contenido.0 en vez de contenido).
* Destination: flowfile-content

Una vez creado, conectamos UpdateAttribute con AttributesToJSON mediante la conexión success, y el resto de las conexiones las marcamos para que autoterminen.

Si ejecutamos los procesadores anteriores y comprobamos la salida, veremos cómo se están creando FF cuyo contenido es la petición introducida más el estado:



Finalmente, añadimos el procesador [PutMongo](https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-mongodb-nar/latest/org.apache.nifi.processors.mongodb.PutMongo/index.html) para introducir el contenido JSON. Las propiedades que hay que configurar son:

* Cliente Service 🡪 Create new service 🡪 MongoDBControllerService. Después hay que configurar el servicio (felcha hacia la derecha)
  + Mongo URI: mongodb://localhost

Hay que habilitar el servicio (icono del rayo)

* Mongo Database Name: iabd
* Mongo Collection Name: ejemplo4

Unimos AttributesToJSON a PutMongo mediante la conexión sucess y autoterminar en caso de failure.

Por otro lado en PutMongo, autoterminamos la conexión success y reconectamos con el mismo procesador la conexión failure.

Si arrancamos el flujo de datos completo, y tras realizar las peticiones de antes:

*curl --data "texto de prueba" http://localhost:8081/iabd*

*curl --data "este sí que tiene ERROR" http://localhost:8081/iabd*

*curl --data "vaya ERROR más tonto" http://localhost:8081/iabd*

*curl --data "nifi mola mucho"* [*http://localhost:8081/iabd*](http://localhost:8081/iabd)

Sólo nos queda entrar a MongoDB mediante mongosh y comprobar que nos aparecen los datos. Si no tenemos instalado mongosh en nuestro sistema, podemos [instalarlo](https://www.mongodb.com/docs/mongodb-shell/install/) mediante:

*wget -qO - https://www.mongodb.org/static/pgp/server-5.0.asc | gpg --dearmor | sudo tee /usr/share/keyrings/mongodb-archive-keyring.gpg*

*echo "deb [signed-by=/usr/share/keyrings/mongodb-archive-keyring.gpg] http://repo.mongodb.org/apt/ubuntu $(lsb\_release -cs)/mongodb-org/5.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list*

*sudo apt-get update*

*sudo apt-get install -y mongodb-mongosh*

*>mongosh*

*> use iabd*

switched to db iabd

> *db.ejemplo4.find()*

{ "\_id" : ObjectId("6197cca29c63ec4e825b8232"), "contenido.0" : "este sí que tiene ERROR\nvaya ERROR más tonto", "estado" : "textoError" }

{ "\_id" : ObjectId("6197cca29c63ec4e825b8233"), "contenido.0" : "texto de prueba\nnifi mola mucho", "estado" : "unmatched" }

## GRUPOS

En Nifi sólo hay un canvas de nivel superior, pero podemos construir tantos flujos lógicos como deseemos. Normalmente, para organizar los flujos, 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.



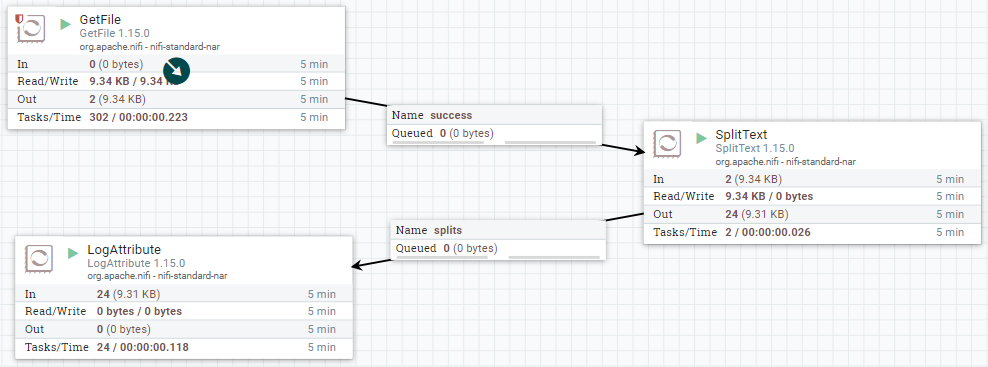
Dentro de los grupos, para indicar como entran los datos se utiliza un Input port, el cual va a definir un puerto de entrada al grupo. Del mismo modo, para indicar cómo salen los datos, se utiliza un Output port como puerto de salida para transferir la información fuera del grupo.

Así pues, los grupos de procesos nos van a permitir refactorizar nuestros flujos para agruparlos y poder reutilizarlos en otros flujos, o bien mejorar la legibilidad de nuestro flujo de datos.

En todo momento podemos ver el nivel en el que nos encontramos en la parte inferior izquierda, con una notación similar a Nifi Flow >> Subnivel 1 >> Subnivel 2…

### CREAR UN GRUPO

Vamos a partir de un ejemplo sencillo de leer un fichero de texto, partirlo en fragmentos y nos saque por el log alguno de sus atributos.



GetFile:

* Input Directory: /opt/nifi/README
* File Filter: README
* Keep Source File: true

SplitText

* Line Split Count: 1 (cada línea del archivo se convertirá en un nuevo flowfile).

LogAttribute

* Log Payload: true (para imprimir también el contenido del flowfile en el log).

Podemos probarlo

Una vez lo tenemos funcionando, vamos a colocar el procesador SplitText dentro de un grupo. Así pues, un grupo encapsula la lógica de un procesador haciéndolo funcionar como una caja negra.

Para ello, desde la barra superior arrastramos el icono de Process Group, le damos un nombre por ejemplo Grupo.

### PUERTOS

Una vez creado, vamos a copiar nuestro procesador SplitText.

* Botón derecho copiar sobre SplitText
* Doble clic sobre el grupo para entrar en ese subnivel
* Botón derecho copiar
* Añadir Input port (con el nombre entrada)
* Añadir Output port (con el nombre salida)
* Conectamos los puertos al procesador con las mismas conexiones que antes.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Ahora salimos del grupo, y conectamos los procesadores del nivel superior con el grupo creado y comprobamos como sigue funcionando.

Diagrama

Descripción generada automáticamente

### IMPORTAR UN FLUJO

Otra manera de crear un grupo es a partir de una definición de flujo.

En cualquier momento, sobre el área de trabajo de un flujo, con el botón derecho, podemos seleccionar la opción Download flow definition, el cual nos descarga un archivo JSON con la definición del flujo.

Para importarlo arrastramos de nuevo el icono de Process Group y en este caso hay que hacer clic en el icono Browse para seleccionar el json descargado.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## FUNNELS

Los funnels (el icono está en la barra superior) son un tipo de componente que permite trabajar en paralelo y después unir los diferentes flujos en un único flujo de ejecución, además de poder definir su propia prioridad de forma centralizada.

Para ello vamos a poner varios GenerateFlowFile (4 en este caso) para mostrar sus datos mediante LogAttribute.

Diagrama

Descripción generada automáticamente

Si quisiéramos cambiar el procesador de LogAttribute por otro tipo de procesador, deberíamos borrar todas las conexiones y volver a conectarlo todo. Para evitar esto añadimos un Funnel que va a centralizar todas las conexiones en un único punto.

Diagrama

Descripción generada automáticamente