



Módulo Profesional: Big Data Aplicado

Ingestión de datos con NiFi

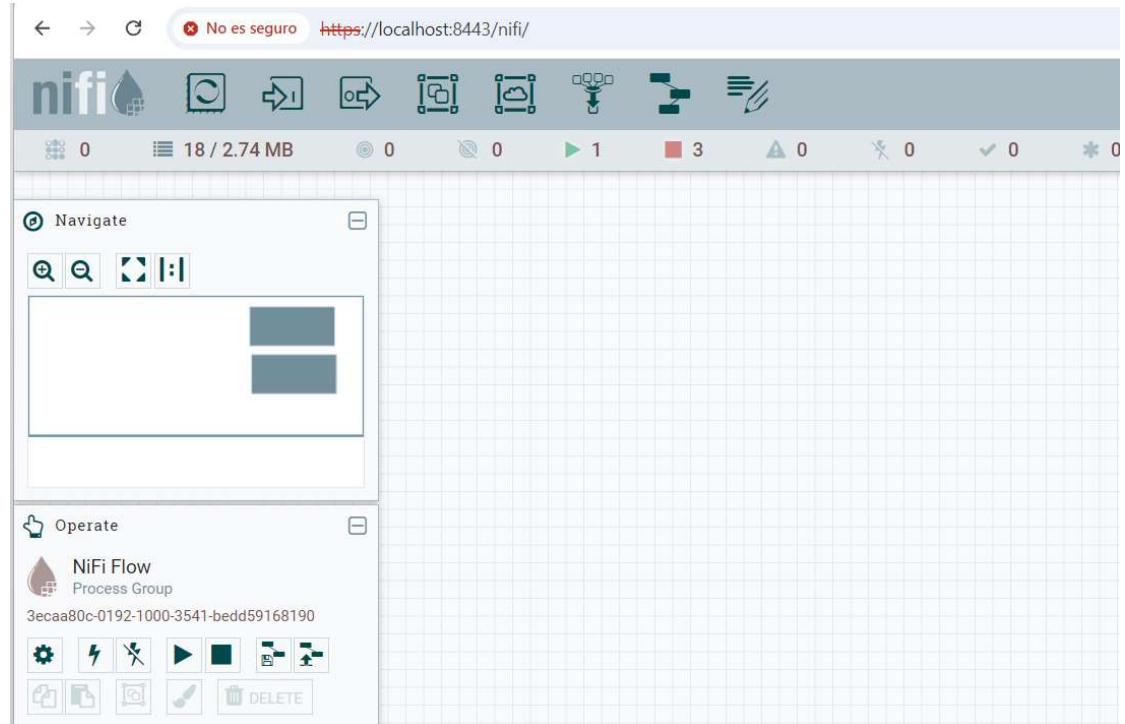
Apache NiFi: Instalación

Pasos:

4 -Arrancar localhost:**puerto**

6.- Buscar en Docker “**Generated**” para buscar usuario y contraseña

7.- Accedemos a la aplicación NiFi

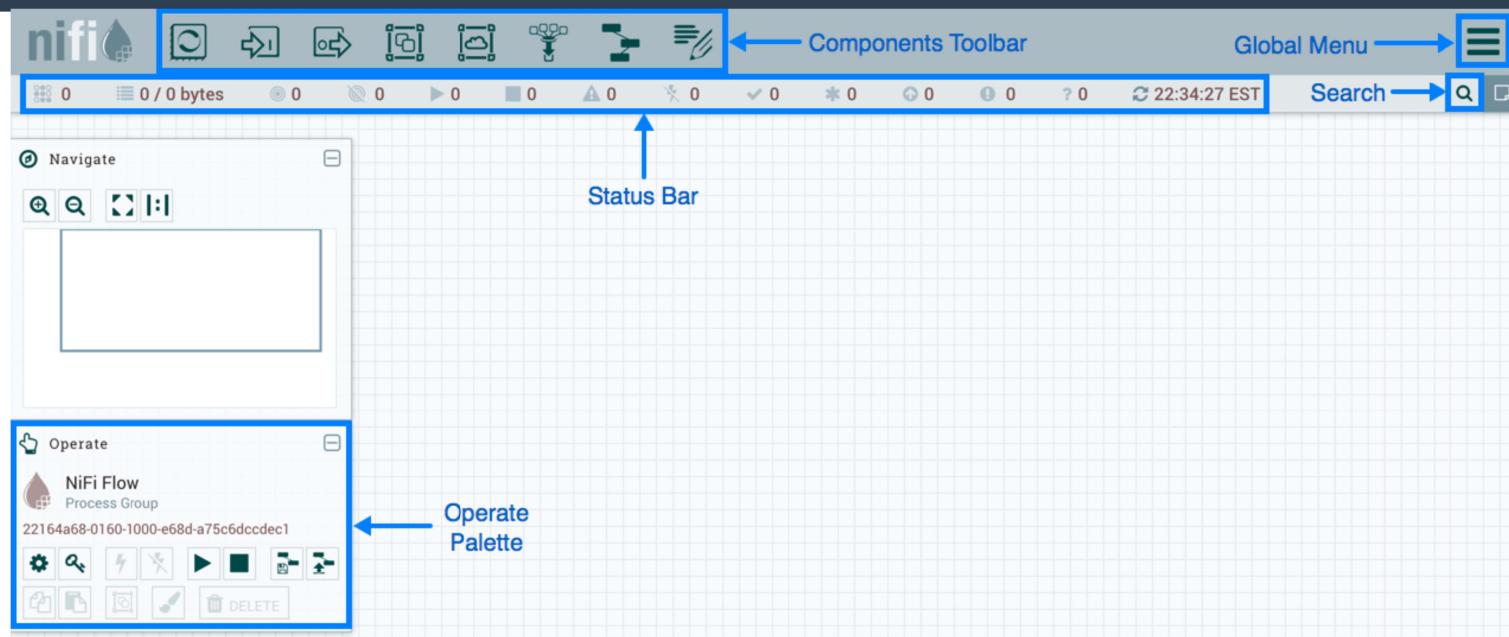


Configurando Apache NiFi

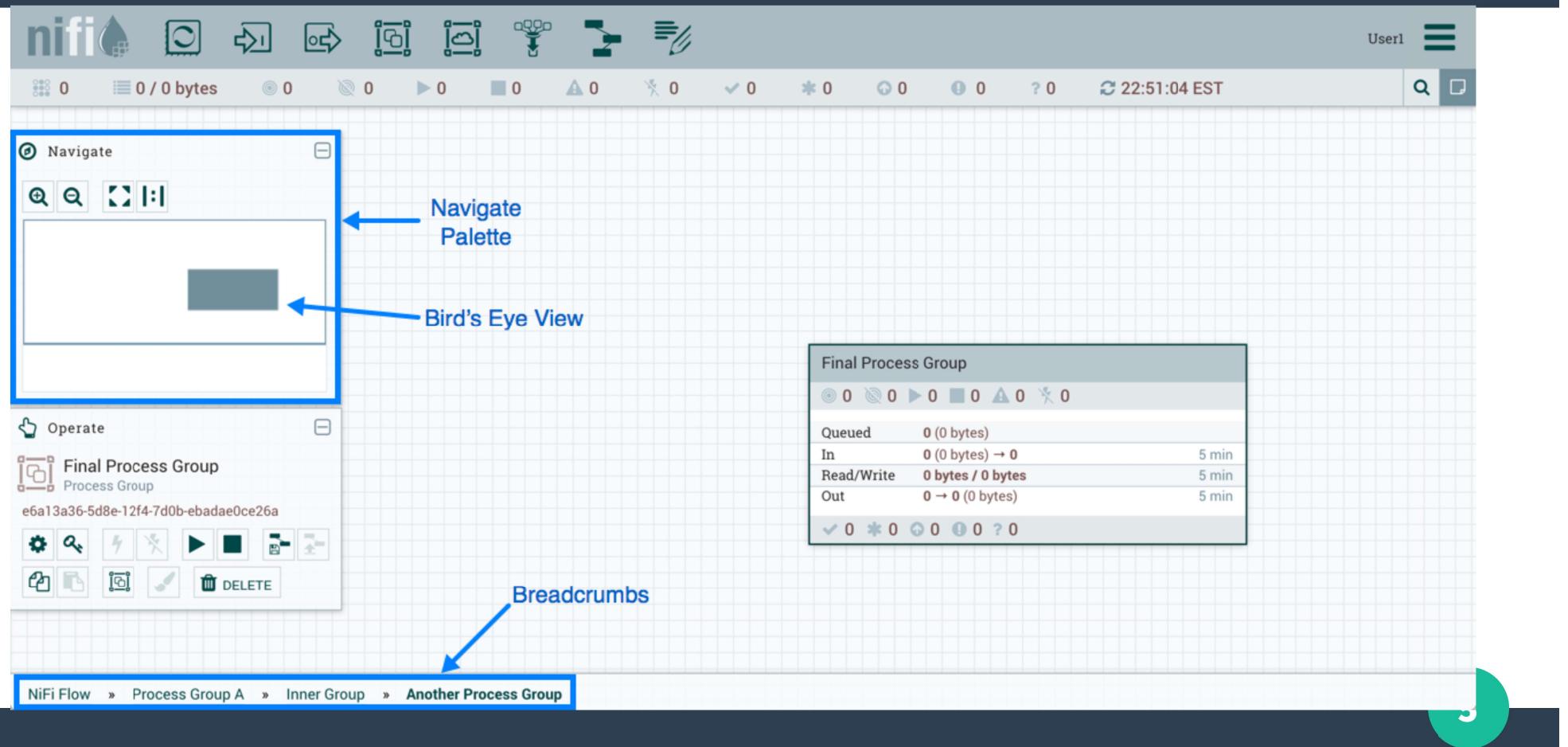
¡¡¡Comenzamos con NiFi!!!



Interfaz Apache NiFi



Interfaz Apache NiFi



Procesadores - Fuentes



{ REST }

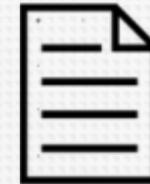


http://



and many more ...

Procesadores - Destinos



and many more ...



Ejercicio 1: Construcción del primer Data Flow

Crear un flujo de datos simple donde se obtiene una carpeta y se copiar a otra carpeta.

- Conectarse a <https://localhost:8443>
- Crear dos carpetas temporales en el contenedor NiFi Docker Docker

“/tmp/in” donde se encuentran los ficheros

“/tmp/out” donde se almacenarán los ficheros

SUGERENCIA: use el "docker exec" para obtener acceso al contenedor o use el panel de Docker Desktop.

```
C:\Users\SONIA>docker exec -it nifi /bin/bash  
nifi@8a4e9d01d347:/opt/nifi/nifi-current$
```

DOCKER EXEC -IT

Ejercicio 1: Construcción del primer Data Flow

2.- Una vez dentro del contenedor, escribimos en el terminal **ls**

3.- Creamos dentro de la carpeta tmp dos carpetas in y out, escribiendo **mkdir /tmp/in** y **mkdir /tmp/out**

```
nifi@8a4e9d01d347:/opt/nifi/nifi-current$ mkdir /tmp/in  
nifi@8a4e9d01d347:/opt/nifi/nifi-current$ mkdir /tmp/out
```

4.- Comprobamos que dentro de la carpeta /tmp existen in y out. Para ello, escribimos en el terminal **ls /tmp**

```
nifi@8a4e9d01d347:/opt/nifi/nifi-current$ ls /tmp  
hsperfdata_nifi  in  out  
nifi@8a4e9d01d347:/opt/nifi/nifi-current$
```

5.- Comprobamos con “**ls**” los ficheros que hay y copiamos el fichero, por ejemplo, README dentro de la carpeta **cp README /tmp/in**

```
nifi@8a4e9d01d347:/opt/nifi/nifi-current$ ls  
bin  content_repository  docs          flowfile_repository  LICENSE  NOTICE          README  state  
conf database_repository  extensions  lib           logs      provenance_repository  run      work  
nifi@8a4e9d01d347:/opt/nifi/nifi-current$ cp README /tmp/in
```

Ejercicio 1: Construcción del primer Data Flow

6.- Comprobamos que en la carpeta ls /tmp/in existe README.

```
nifi@8a4e9d01d347:/opt/nifi/nifi-current$ ls /tmp/in  
README
```

7.- Comprobamos que en la carpeta ls /tmp/out no hay nada.

```
nifi@8a4e9d01d347:/opt/nifi/nifi-current$ ls /tmp/out  
nifi@8a4e9d01d347:/opt/nifi/nifi-current$
```

AHORA YA PASAMOS A REALIZAR EL FLOWFILE

Ejercicio 1: Añadiendo el procesador

The screenshot shows the Apache NiFi user interface. At the top, there is a toolbar with various icons. Below the toolbar, a modal window titled "Add Processor" is open. The left side of the modal has a sidebar with a list of processor types: "amazon", "attributes", "avro", "aws", "consume", "csv", "database", "fetch", "files", "get", "hadoop", "ingest", "input", "insert", "json", "listen", "logs", "message", "put", "remote", "restricted", "source", "sql", and "text". The "files" icon is highlighted with a pink border. The main area of the modal displays a table with 11 rows, each representing a different processor type. The columns are "Source", "Type", "Version", and "Tags". The first row, "FetchFTP", is highlighted with a yellow background. The table shows the following data:

Source	Type	Version	Tags
	FetchFTP	1.2.0	input, ftp, get, fetch, retrieve, files, source, remote, ingest
	FetchFile	1.2.0	ingress, input, restricted, get, files, source, local, filesystem, ingest
	FetchSFTP	1.2.0	input, get, fetch, retrieve, files, sftp, source, remote, ingest
	GetFTP	1.2.0	input, FTP, get, fetch, retrieve, files, source, remote, ingest
	GetFile	1.2.0	ingress, input, restricted, get, files, source, local, filesystem, ingest
	GetHDFS	1.2.0	restricted, get, fetch, HDFS, hadoop, source, filesystem, ingest
	GetSFTP	1.2.0	input, get, fetch, retrieve, files, sftp, source, remote, ingest
	ListFTP	1.2.0	input, ftp, files, source, list, remote, ingest
	ListFile	1.2.0	file, get, source, list, filesystem, ingest
	ListHDFS	1.2.0	get, HDFS, hadoop, source, list, filesystem, ingest
	ListSFTP	1.2.0	input, files, sftp, source, list, remote, ingest

Below the table, a detailed description of the "FetchFTP 1.2.0" processor is shown:

FetchFTP 1.2.0 org.apache.nifi - nifi-standard-nar
Fetches the content of a file from a remote SFTP server and overwrites the contents of an incoming FlowFile with the content of the remote file.

GETFILE

PUTFILE

Ejercicio 1: Configurando el procesador GETFILE

Processor Details | GetFile 1.27.0

▶ Running STOP & CONFIGURE

SETTINGS SCHEDULING PROPERTIES RELATIONSHIPS COMMENTS

Name Leer en /tmp/in → Nombre de nuestro procesador

Id 4391c6fc-0192-1000-9d68-4b45cd2d6c43 → Identificador del procesador en NiFi

Type GetFile 1.27.0 → Identifica el tipo de procesador

Bundle org.apache.nifi - nifi-standard-nar → Es el paquete que contiene el procesador y sus dependencias
Penalty Duration: Tiempo que un archivo problemático debe esperar antes tras un error.

Penalty Duration ? 30 sec Yield Duration ? 1 sec → **Yield Duration:** Tiempo que el procesador espera antes de reintentar cuando no tiene trabajo

Bulletin Level ? → **WARN** Determina el nivel de severidad de los mensajes que se mostrarán en los boletines del procesador en el panel de NiFi. Los boletines son mensajes de log que te informan sobre eventos importantes relacionados con el comportamiento del procesador. Los niveles disponibles son:
DEBUG: Muestra todos los mensajes, incluidos los detalles más técnicos y específicos para depuración.
INFO: Muestra información general sobre la operación del procesador.
WARN: Muestra advertencias sobre posibles problemas que no necesariamente son críticos.
ERROR: Muestra solo mensajes de error que indican que algo falló de manera crítica.

Configuración

Ejercicio 1: Configurando el procesador GETFILE

Configure Processor | GetFile 1.27.0

Stopped

SETTINGS

SCHEDULING

PROPERTIES

RELATIONSHIPS

COMMENTS

Scheduling Strategy ?

Timer driven



Se define cómo se va a activar el procesador. Diferentes estrategias:

Timer driven: el procesador se ejecuta periódicamente basado en un intervalo de tiempo (**Run Schedule**)

Cron driven: se utiliza una **expresión de cron** para definir cuándo debe ejecutarse el procesador

Concurrent Tasks ?

1



Run Schedule ?

0 sec



Run Schedule: define el intervalo de tiempo entre cada ejecución del procesador cuando la estrategia de programación es Timer Driven. Es la frecuencia con la que el procesador se activará para realizar su tarea. Es importante definirlo para no sobrecargar el sistema.

Execution ?

All nodes



Concurrent Tasks: se refiere a la cantidad de hilos (threads) que un procesador puede utilizar de forma simultánea para realizar su trabajo. Si lo configuras, por ejemplo, a 5, el procesador podrá ejecutar hasta 5 hilos en paralelo, procesando múltiples flujos de datos simultáneamente, lo que incrementa la eficiencia, especialmente en entornos de alta carga.

Programación, cuándo y cómo se ejecuta el procesador

Ejercicio 1: Configurando el procesador GETFILE

The screenshot shows the NiFi interface with the 'SETTINGS' tab selected. Under the 'SCHEDULING' section, the 'Scheduling Strategy' is set to 'Timer driven'. Below it, 'Concurrent Tasks' is set to '1'. In the 'Execution' section, the dropdown menu is open, showing 'All nodes' as the current selection. A red arrow points from the text 'Primary Node (Nodo primario):' to this dropdown menu.

Tipos de Configuración de Execution:

All Nodes (Todos los nodos):

En esta configuración, el procesador se ejecuta en todos los nodos del clúster. Esto significa que cada nodo del clúster procesará sus propios datos de forma independiente y paralela.

Esta configuración es útil cuando quieras que todos los nodos trabajen simultáneamente y balanceen la carga de trabajo, por ejemplo, al leer archivos distribuidos en varias ubicaciones o al procesar datos que llegan a cada nodo por separado.

Ejemplo de uso: Si tienes un procesador como GetFile en un clúster con tres nodos, y está configurado para ejecutarse en All Nodes, cada nodo buscará archivos en sus respectivos directorios y los procesará de forma independiente.

Primary Node (Nodo primario):

En esta configuración, el procesador se ejecuta solo en el nodo primario del clúster. El nodo primario es un único nodo que NiFi designa como el encargado de ejecutar tareas críticas en un clúster. En esta configuración, el procesador no se ejecutará en los demás nodos.

Es útil cuando tienes tareas que solo deben ejecutarse una vez en todo el clúster, como la consulta a una base de datos o la obtención de datos de una fuente externa.

Ejemplo de uso: Si tienes un procesador QueryDatabaseTable para leer datos de una base de datos, podrías configurarlo en Primary Node para evitar que múltiples nodos realicen la misma consulta y dupliquen el trabajo.

Ejercicio 1: Configurando el procesador

Ejemplo de Uso en Clústeres:

Si tienes un clúster NiFi de 5 nodos y quieres leer archivos de una ubicación central compartida, puedes configurar el procesador GetFile con Execution = Primary Node, de manera que solo un nodo (el primario) lea los archivos, evitando duplicados. Sin embargo, si cada nodo tiene su propio directorio con archivos separados, entonces puedes configurarlo con Execution = All Nodes, permitiendo que cada nodo procese su propia carga.

Consideraciones

Escalabilidad: Configurar la ejecución en All Nodes puede mejorar la escalabilidad de tu flujo de datos, ya que permite distribuir la carga de trabajo entre todos los nodos del clúster.

Procesos únicos: Si el procesamiento de un flujo de datos debe ocurrir solo una vez, como en el caso de trabajos de consulta o escritura a una base de datos central, la opción Primary Node asegura que solo un nodo se encargue de esa tarea.

Fallo del nodo primario: En caso de que el nodo primario falle, otro nodo será promovido a nodo primario y continuará ejecutando los procesadores configurados en Primary Node.

Ejercicio 1: Configurando el procesador GETFILE

Running STOP & CONFIGURE

SETTINGS SCHEDULING PROPERTIES RELATIONSHIPS COMMENTS

Required field

Property	Value
Input Directory	/tmp/in
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

Todas las propiedades en color negrita son obligatorias.

En este caso, indicamos
“Input Directory”

Ejercicio 1: Configurando el procesador GETFILE

Configure Processor | GetFile 1.27.0

⚠ Invalid

SETTINGS SCHEDULING PROPERTIES RELATIONSHIPS COMMENTS

Automatically Terminate / Retry Relationships ⓘ

success
 terminate retry

All files are routed to success

Cada uno de los procesadores tienen diferentes relaciones.

En este caso, lo podemos dejar así y cuando hagamos la conexión la única relación que tiene es success.

Ejercicio 1: Configurando el procesador PUTFILE

Configure Processor | PutFile 1.27.0

Stopped

SETTINGS SCHEDULING PROPERTIES RELATIONSHIPS COMMENTS

Name: Grabar en /tmp/out Enabled

Stopped

SETTINGS SCHEDULING PROPERTIES RELATIONSHIPS COMMENTS

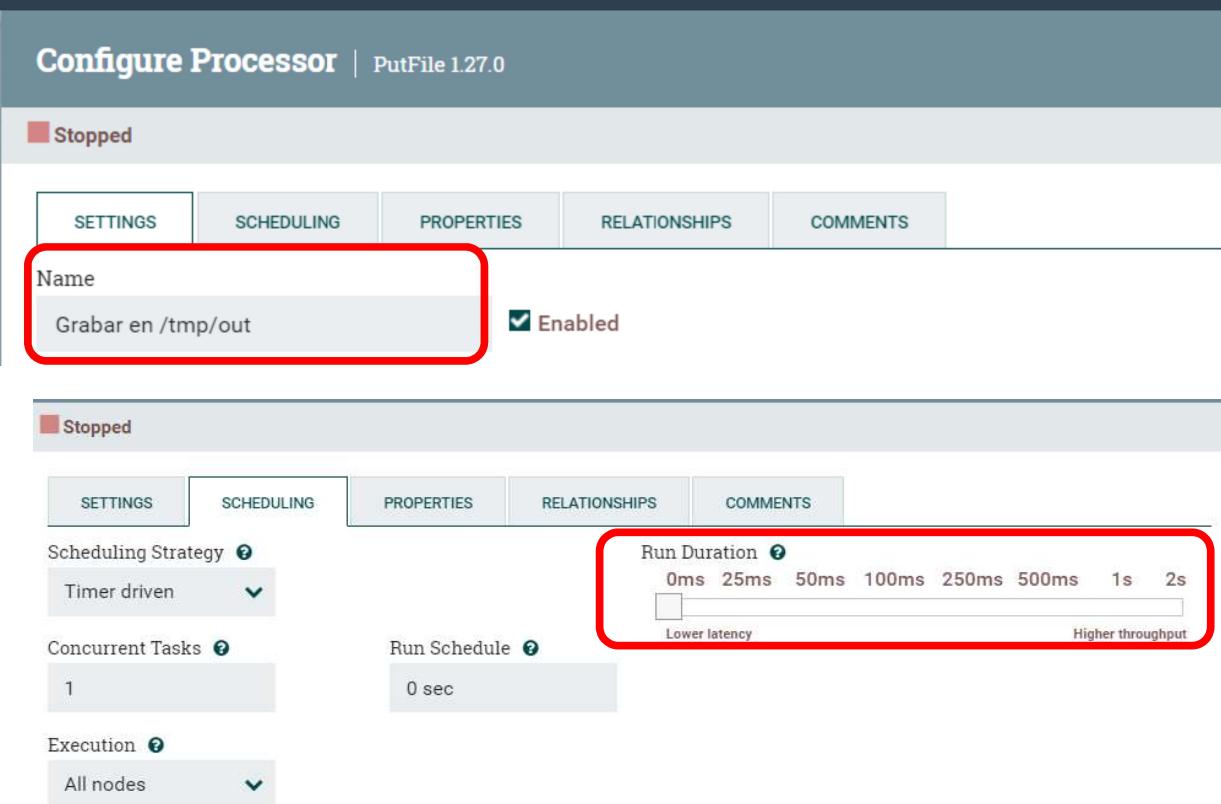
Scheduling Strategy: Timer driven

Concurrent Tasks: 1

Execution: All nodes

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

Run Schedule: 0 sec



Run Duration: Este parámetro define la cantidad de tiempo que un procesador debería ejecutarse de manera continua en cada invocación, antes de liberar el control y permitir que otros procesadores se ejecuten.

Ejercicio 1: Configurando el procesador PUTFILE

Configure Processor | PutFile 1.27.0

Stopped

SETTINGS SCHEDULING PROPERTIES RELATIONSHIPS COMMENTS

Required field

Property	Value
Directory	/tmp/out
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

Ejercicio 1: Configurando el procesador GETFILE

Configure Processor | PutFile 1.27.0

Stopped

SETTINGS

SCHEDULING

PROPERTIES

RELATIONSHIPS

COMMENTS

Automatically Terminate / Retry Relationships [?](#)

failure

terminate retry

Files that could not be written to the output directory for some reason are transferred to this relationship

success

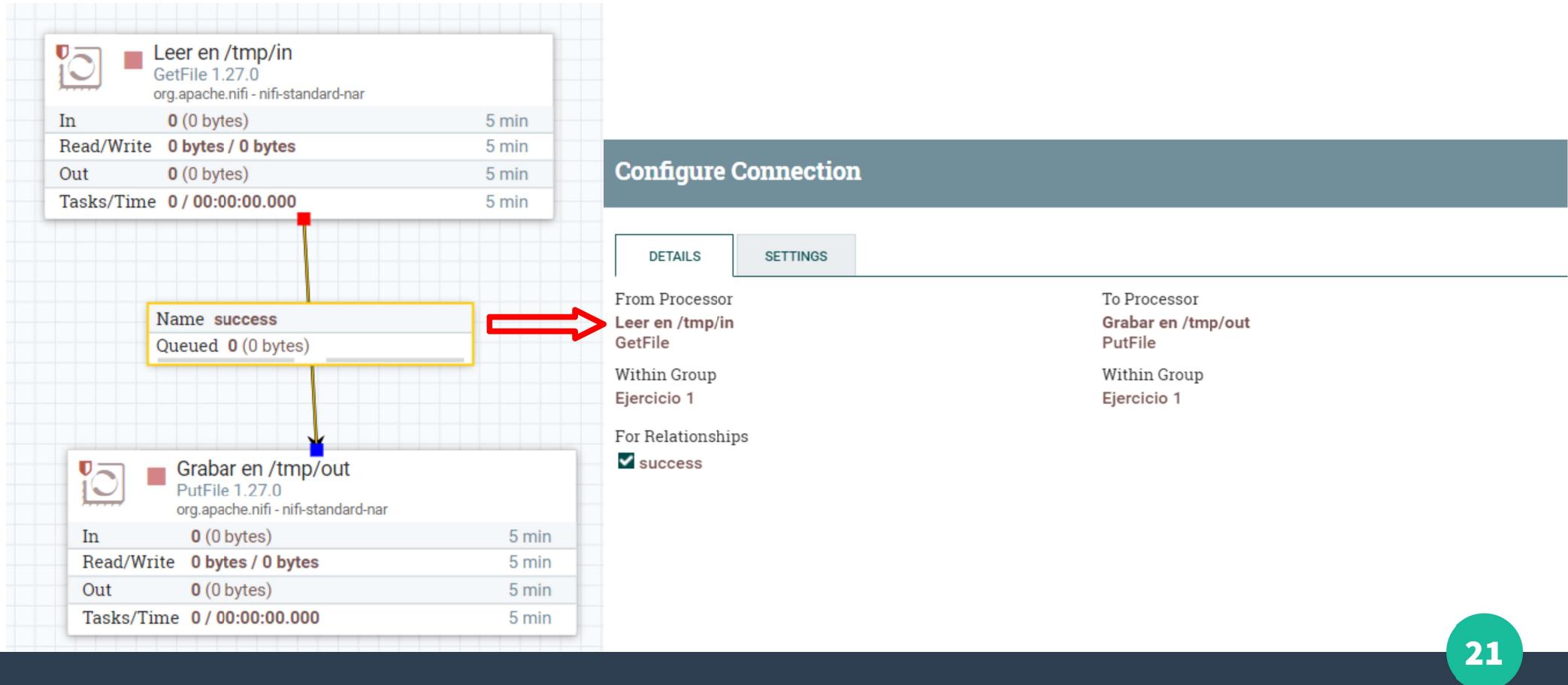
terminate retry

Files that have been successfully written to the output directory are transferred to this relationship

Indicamos que si falla que termine.

Indicamos si ha ido bien que termine también.

Ejercicio 1: Configurando Connection



Ejercicio 1: Configurando Connection

Available Prioritizers 

FirstInFirstOutPrioritizer

NewestFlowFileFirstPrioritizer

OldestFlowFileFirstPrioritizer

PriorityAttributePrioritizer

Selected Prioritizers 

- **FirstInFirstOutPrioritizer:** se procesen en el orden en que llegaron, dando prioridad al primero en entrar.
- **NewestFlowFileFirstPrioritizer:** prioriza el procesamiento de los más recientes antes que los más antiguos.
- **OldestFlowFileFirstPrioritizer:** prioriza el procesamiento de los más antiguos antes que los más recientes.
- **PriorityAttributePrioritizer:** prioriza el procesamiento según el valor de un atributo específico, que define la prioridad.