Fred Vellinga BI Services

Apache NiFi Flow Manager

©Fred Vellinga BI Services, (v1.0 December 2019, update jan-2020)

De macro Ctrl+T zet de taal op Engels.
De macro Ctr+L maakt een streepjes lijst.
De macro Ctr+G maakt een grid/tabel.

<u>Content</u>

1. Introduction.. 3
2. Managing data flows 3
3. Architecture.. 4
4. Conclusion.. 6
5. Future Features 6

1. Introduction

According **Apache NiFi** own website, http://nifi.apache.org/docs.html:
*"**NiFi** was built to automate the flow of data between systems. While the term 'dataflow' is used in a variety of contexts, here it means the automated and managed flow of information between systems"*

The marketing wording (http://nifi.apache.org/index.html) is as follows:

"An easy to use, powerful, and reliable system to process and distribute data"

NiFi is an acronym, coming from the '*NiagaraFiles*' software project. It is pronounced as "nye fye" (nī fī).

It is not an **ETL** (**E**xtract **T**ransform **L**oad) tool. It listens for (new) data and process accordingly and saves it at the destination system, as long as **NiFi** has support for it. For instance, you read csv data from an SFTP server, store it locally, do something with it, and saves it at the other system.

In my case, the target system is MarkLogic. **NiFi** has no built-in support for it. You first install a plug-in and then **NiFi** can store the data at MarkLogic. But data manipulation at MarkLogic has to be done in MarkLogic itself. For that they have a "data-hub" application, their ETL tool.

You could argue that you no longer need an ETL tool, which saves license costs. **NiFi** is taken care of pulling data from all of your source systems. The drawback is that you lose the visual coding aspect ETL tools has of the target system. You have to write SQL scripts maintained at the SQL server, and executed by **NiFi**. In the case of MarkLogic you have to write XQuery code. This last statement, executing SQL stored procedures or whatever script language you target system supports, is easier said than done.

2. Managing data flows

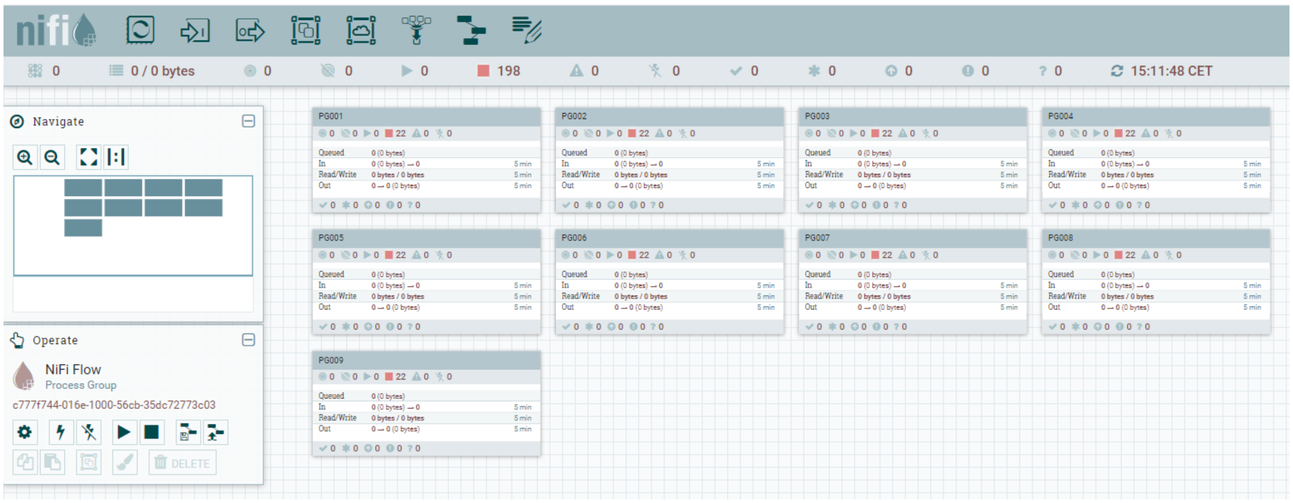
NiFi has one canvas which uses instant auto-saving when you edit the dataflow. Everything is recorded in one compressed file: **flow.xml.gz**. When you have 100 data-flows, it is all in one file. To manage all the **NiFi** flows, an approach is to develop, test and maintain each data-flow as a separate flow, and then later on, merge all the single flows together in one flow, the *merged flow*, or your production flow. This can be done as long as each flow is encapsulated in a so-called *process group*. *Process groups* can be nested, but as long as a flow has only one main *process group*, you can extract the *process group* from the development flow or, terminology, the *other flow*, and merge them all together.

You need one flow, called the *common flow*, where you put all the extracted flow data in. The *common flow* must **not** contain a *process group*. The *common flow* contains all the shared parameters (which can be encrypted, passwords etc.) that can be used be the different flows. Thus, it is an empty flow, which is then populated with extracts from the *other flows*, which become the *merged flow*. The *other flow* should be developed as a fully self-supporting stand-alone flow. When you use parameters, you have to maintain them also in the *common flow*.

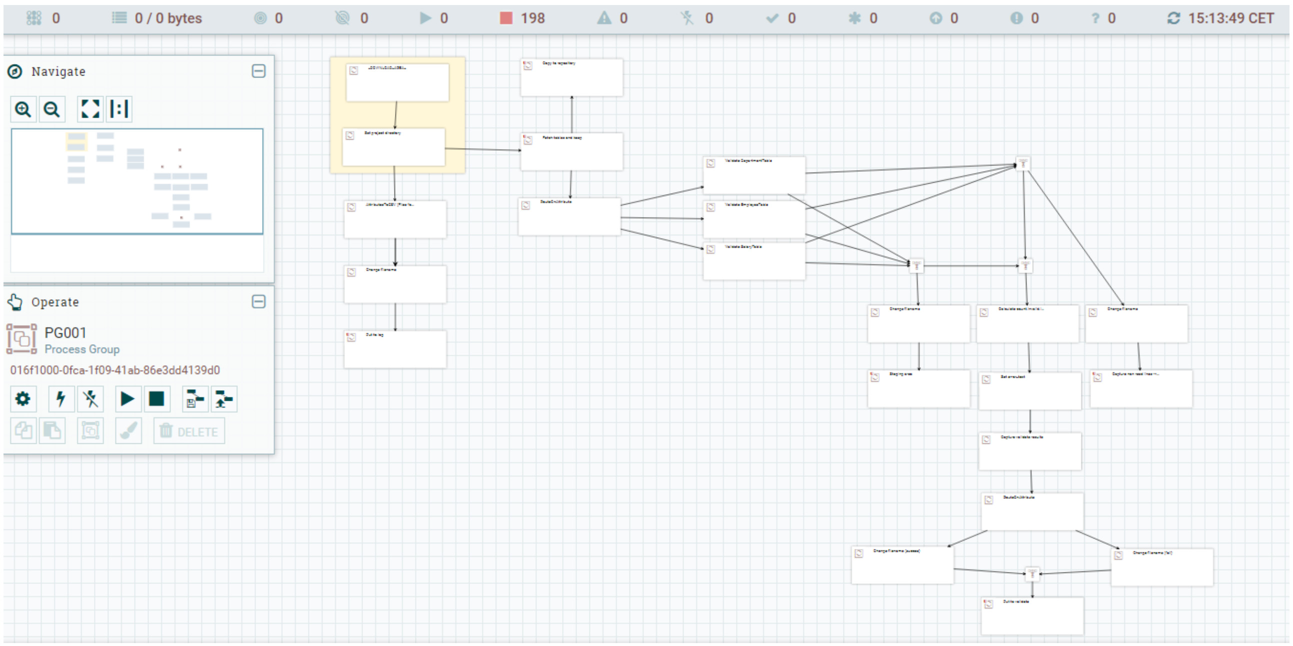
The ***NiFi Flow Manager*** application automates this approach. It is a fully file-driven approach. It operates on the file system. When you have 100 data-flows, you have 100 **flow.xml.gz** files, all stored at a different subdirectory. When you want to edit a flow, you copy it config directory where **NiFi** read the flows. When you are done you copy back the flow to the subdirectory.

This means every flow can be handled by a version issue system, for instance SourceTree. Al the flows are stored in a central repository, and you have local copies of it.

Next screen-shot shows what is meant by merging. Here you have nine flows. The objects are lined automatically, once merged. Here it is 4 column grid, but that it adjustable.



In this case all flows are the same, and look like this:



It is not so relevant what this is doing, but here you see a draw-back of **`NiFi`**; You need wide monitor screens to show a significant readable part the complete flow. The above does not tell you much. If you develop at a 14inch laptop, you are constantly moving the flow to get the processor (that is name of a block or box) in focus. The interface does not support scroll-bars. It also very easy to accidently moving boxes and lines etc. There is no Ctrl+Z function to undo that. Also, automatic lining/formatting does not exist. These draw-back are minor things.

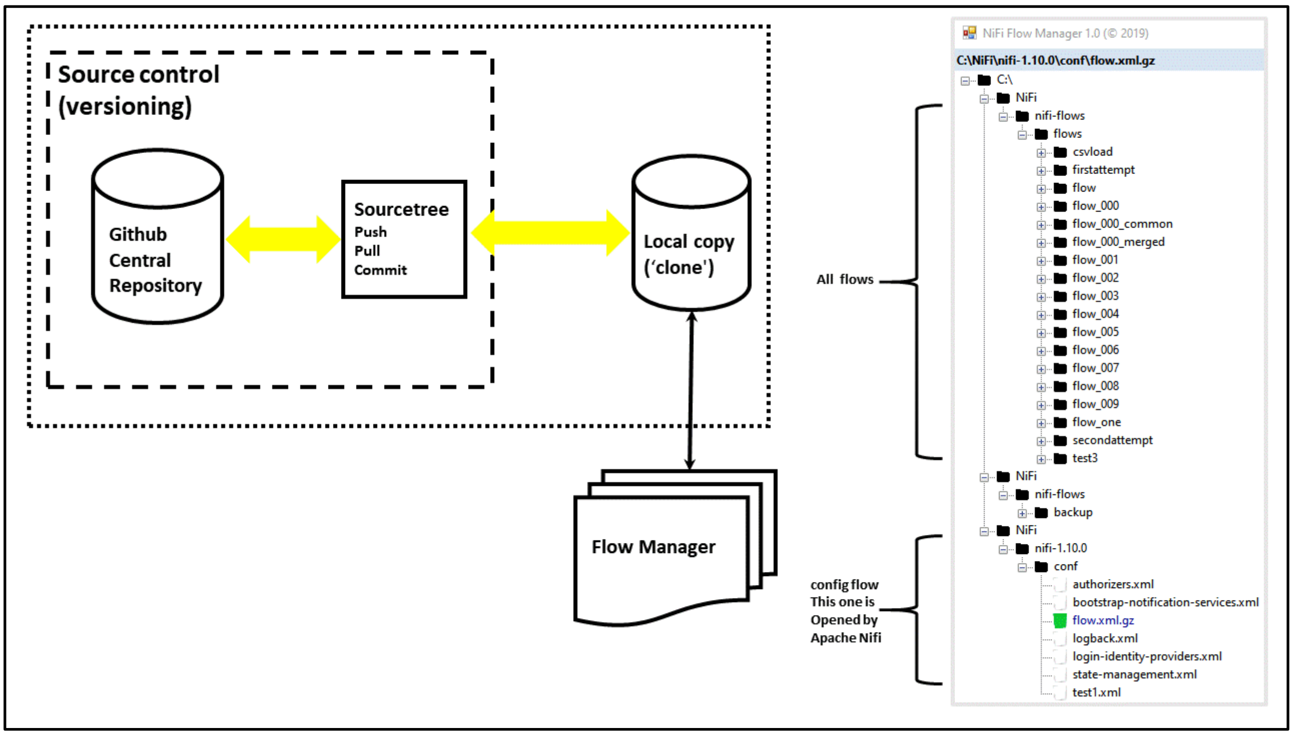
Note: When you copy a flow and want to work with the copied flow, you have to first copy the process group to a new one and delete the original process group. This is needed because every processor/parameter must have a unique processor id. The copying is doing that for you. When a flow has objects (processors) with the same id, Apache **`NiFi`** won't run. In your single flow this is no issue, but when you are going to merge, then you get problems. Thus, when you have copied a flow, running it in **`NiFi`**, then first copy the process group, delete the original process-group.

Note: **`NiFi`** support also flow variables, sort of constants. Don't use them. The official documentation status you have to use parameters instead.

3. Architecture

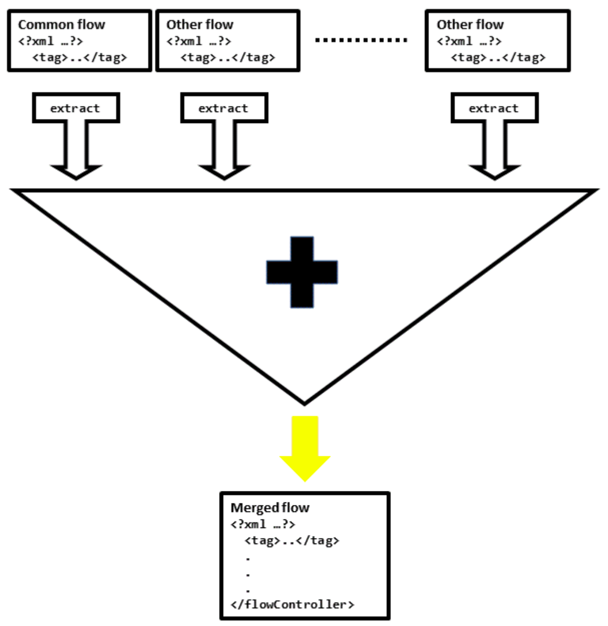
A common development approach is to develop and test, then do acceptance testing, then put into production. In the case of **`NiFi`** you can argue what do you want to test at acceptance, the whole *merged* *flow*? Or a single flow? Do you want to maintain three versions of a single flow, or only one, and three *merged flows*; development, acceptance testing, production.

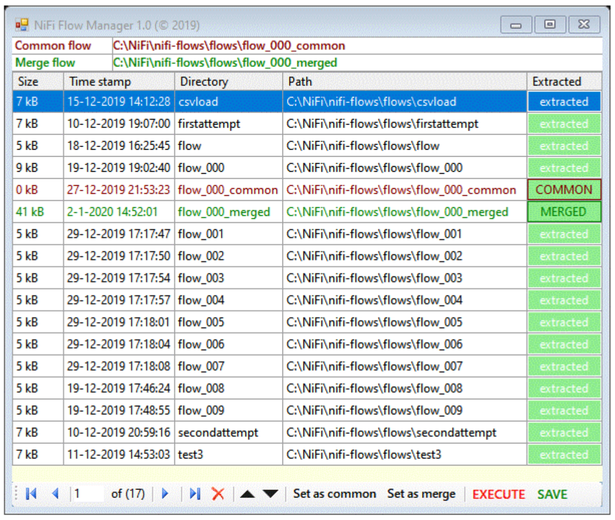
The picture below shows the development architecture. Any acceptance/production environment looks the same, at a different file system. At the right you see the tree view that list the file system. The bottom list the path to the conf(ig) directory, that is where **`Apache NiFi`** installs itself, which is installation dependent.


Source control (versioning)
Github Central Repository
Sourcetree
Push
Pull
Commit
Local copy ('clone')
Flow Manager
All flows
config flow This one is Opened by Apache Nifi
NiFi Flow Manager 1.0 (© 2019)

```
C:\NiFi\nifi-1.10.0\conf\flow.xml.gz
C:\
  NiFi
    nifi-flows
      flows
        csvload
        firstattempt
        flow
        flow_000
        flow_000_common
        flow_000_merged
        flow_001
        flow_002
        flow_003
        flow_004
        flow_005
        flow_006
        flow_007
        flow_008
        flow_009
        flow_one
        secondattempt
        test3
  NiFi
    nifi-flows
      backup
  NiFi
    nifi-1.10.0
      conf
        authorizers.xml
        bootstrap-notification-services.xml
        flow.xml.gz
        logback.xml
        login-identity-providers.xml
        state-management.xml
        test1.xml
```

Below you see how the merging works. You select the flows that are part of the merging. The *common flow* and *merged flow* are mandatory. You can select the merge order. The button tells you the status of the merge. Clicking on it result in viewing the extract and original content of the flow. When the buttons are green, all fine. Otherwise red. It is possible that the *merged flow* will not run due to duplicate processer id, or local flow parameters were not passed through to the common flow.



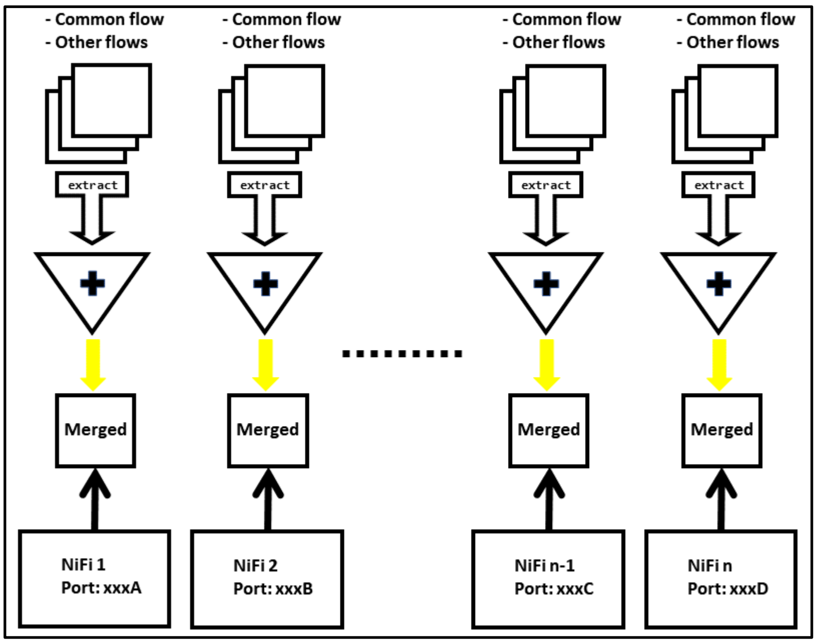


Common flow	C:\NiFi\nifi-flows\flows\flow_000_common
Merge flow	C:\NiFi\nifi-flows\flows\flow_000_merged

Size	Time stamp	Directory	Path	Extracted
7 kB	15-12-2019 14:12:28	csvload	C:\NiFi\nifi-flows\flows\csvload	extracted
7 kB	10-12-2019 19:07:00	firstattempt	C:\NiFi\nifi-flows\flows\firstattempt	extracted
5 kB	18-12-2019 16:25:45	flow	C:\NiFi\nifi-flows\flows\flow	extracted
9 kB	19-12-2019 19:02:40	flow_000	C:\NiFi\nifi-flows\flows\flow_000	extracted
0 kB	27-12-2019 21:53:23	flow_000_common	C:\NiFi\nifi-flows\flows\flow_000_common	COMMON
41 kB	2-1-2020 14:52:01	flow_000_merged	C:\NiFi\nifi-flows\flows\flow_000_merged	MERGED
5 kB	29-12-2019 17:17:47	flow_001	C:\NiFi\nifi-flows\flows\flow_001	extracted
5 kB	29-12-2019 17:17:50	flow_002	C:\NiFi\nifi-flows\flows\flow_002	extracted
5 kB	29-12-2019 17:17:54	flow_003	C:\NiFi\nifi-flows\flows\flow_003	extracted
5 kB	29-12-2019 17:17:57	flow_004	C:\NiFi\nifi-flows\flows\flow_004	extracted
5 kB	29-12-2019 17:18:01	flow_005	C:\NiFi\nifi-flows\flows\flow_005	extracted
5 kB	29-12-2019 17:18:04	flow_006	C:\NiFi\nifi-flows\flows\flow_006	extracted
5 kB	29-12-2019 17:18:08	flow_007	C:\NiFi\nifi-flows\flows\flow_007	extracted
5 kB	19-12-2019 17:46:24	flow_008	C:\NiFi\nifi-flows\flows\flow_008	extracted
5 kB	19-12-2019 17:48:55	flow_009	C:\NiFi\nifi-flows\flows\flow_009	extracted
7 kB	10-12-2019 20:59:16	secondattempt	C:\NiFi\nifi-flows\flows\secondattempt	extracted
7 kB	11-12-2019 14:53:03	test3	C:\NiFi\nifi-flows\flows\test3	extracted

```
EXECUTE: Extract the processgroup from the flow.
SAVE   : Save the result to the merged flow.
```

`Apache NiFi` is a webserver application default running at port 8080. You can cluster **`NiFi`**, but you can also run multiple instances of **`NiFi`** at different ports. You can install at different physical virtual machines, or at different locations on the same physical machine, all having its own port. When you install at different machines, having different IP addresses, you can use the same port number. This means you spread workload but also can make the **`merged flow`** smaller. Instead of 100 flows, you can split it in 4 times 25 flows. Or group per flow complexity. You get something as depicted below.



As far as I can see, you are quite flexible in installing a **NiFi** instance. You can also make it as complex as possible. Anyway, you can get lost in configuration optimisation, or simple write robust flows who do not break and take the throughput times for granted.

4. Practical use

NiFi supports something called templates. The idea is that flows share the same repetitive steps. You put those steps in a dedicated template, and then import it in your flow. (Analog to importing a module in Python or Class in C#). The ***NiFi Flow Manager*** is an extractor application. It is very easy to store any re-usable flow steps in a dedicated flow, call it the *template flow*, and extract it from there in your own flow. You are merging your flow you are working on with the *template flow,* thus a merge of 2 flows, and you have a *merged flow* which is your new flow you work on.

5. Conclusion

I am new to **Apache NiFi**, and when trying it, I was a bit confused by the concept of one flow, and not able to save and opening a flow you want. That is why I came up with this approach. It is Windows based, a C# Windows Forms application, and drive letters are needed to address the file system. All doable on a local machine, I think.

The assumption is you have a local version of **Apache NiFi** running, and develop locally. When **NiFi** is installed centrally, thus everybody is developing on the same server, this approach won't work. You then have to look at the **Apache NiFi Registry** application.

The ***NiFi Flow Manager*** application is a simple and straightforward approach of maintaining your flows. It is completely file-based and the whole authorization of who is allowed to edit what, is handled by how you set up he git repository or any other version/issue system you are using. The application itself is not aware of it. It simple sees the local file system. You do not need complex auxiliary software to develop/maintain flows. And yes, the extracting and merging of flows can all be done manually.

6. Future Features

- A check on unique processor ids in the *merged flow*. (`<id>d21d34ea-016e-1000-4266-b27a573e08c3</id>`)
- The tree view also shows the acceptance and production environment.
- Support for multiple instances of **NiFi**. (Also, a matter of enhanced tree view behaviour)

You could argue that parameter extracting could be handy, but I think this makes things unnecessary complex. It is much easier to add them to *common flow* when needed.