



Introduction à la Programmation en Flux

« Lazy » flow-based programming :
Utilité, concepts de base et application avec PyF.

Présenté par :

Jonathan Schemoul, consultant JMSI



Qu'est-ce que la programmation en flux ?

- Dans les années 70 : John Paul Morrison au Canada
 - Applications Bancaires
- On traite les données unitairement
 - Elles passent d'un bloc à un autre, formant un flux
- On transforme les données dans le flux, comme dans une chaîne de production.



Qu'est-ce que la programmation en flux ?

Dans la théorie :

- L'ensemble de blocs (composants) constitue un réseau
- Chaque composant a des ports d'entrée et de sortie
- Le réseau est géré par un « scheduler » (ordonnanceur)
Il gère les communications entre les processus, et vérifie l'état du réseau
- Les données échangées sont des paquets standards, appelés « Information Packets ».

En Python : une chaine de générateurs

Pour utiliser les générateurs en programmation en flux...

On définit des boites, étant chacune un générateur

- On passe l'itérateur à un autre générateur, qui renverra lui aussi un flux
- On dit que l'on « adapte » le flux au fur et à mesure

```
>>> def mon_adaptateur(flux) :  
...     for v in flux:  
...         print « J'ai reçu », v  
...         yield dict(valeur=v,  
...                     double=v*2)  
...  
>>> it1 = xrange(5) # 0 à 5  
>>> it2 = mon_adaptateur(it1)  
  
>>> def mon_consommateur(flux_adapte) :  
...     for nv in flux_adapte:  
...         print nv['double']  
...         yield True # c'est ok.  
...  
>>> it3 = mon_consommateur(it2)  
  
>>> for value in it3:  
...     if value is True:  
...         print 'ok so far'  
...     else:  
...         print 'not ok'  
...  
J'ai reçu 0  
0  
ok so far  
J'ai reçu 1  
2  
ok so far  
[...]
```



Avantages de la prog. en flux avec des générateurs

- Pas de montée en RAM, traitement de gros volumes
- Méthode de programmation facile à apprendre (on prend un item en entrée, on en sort un... ou pas :-)
- Tous les objets python peuvent être utilisés (pas uniquement des packets)
- Pas besoin de threads
- Pas besoin de stackless, utilisation de la librairie standard



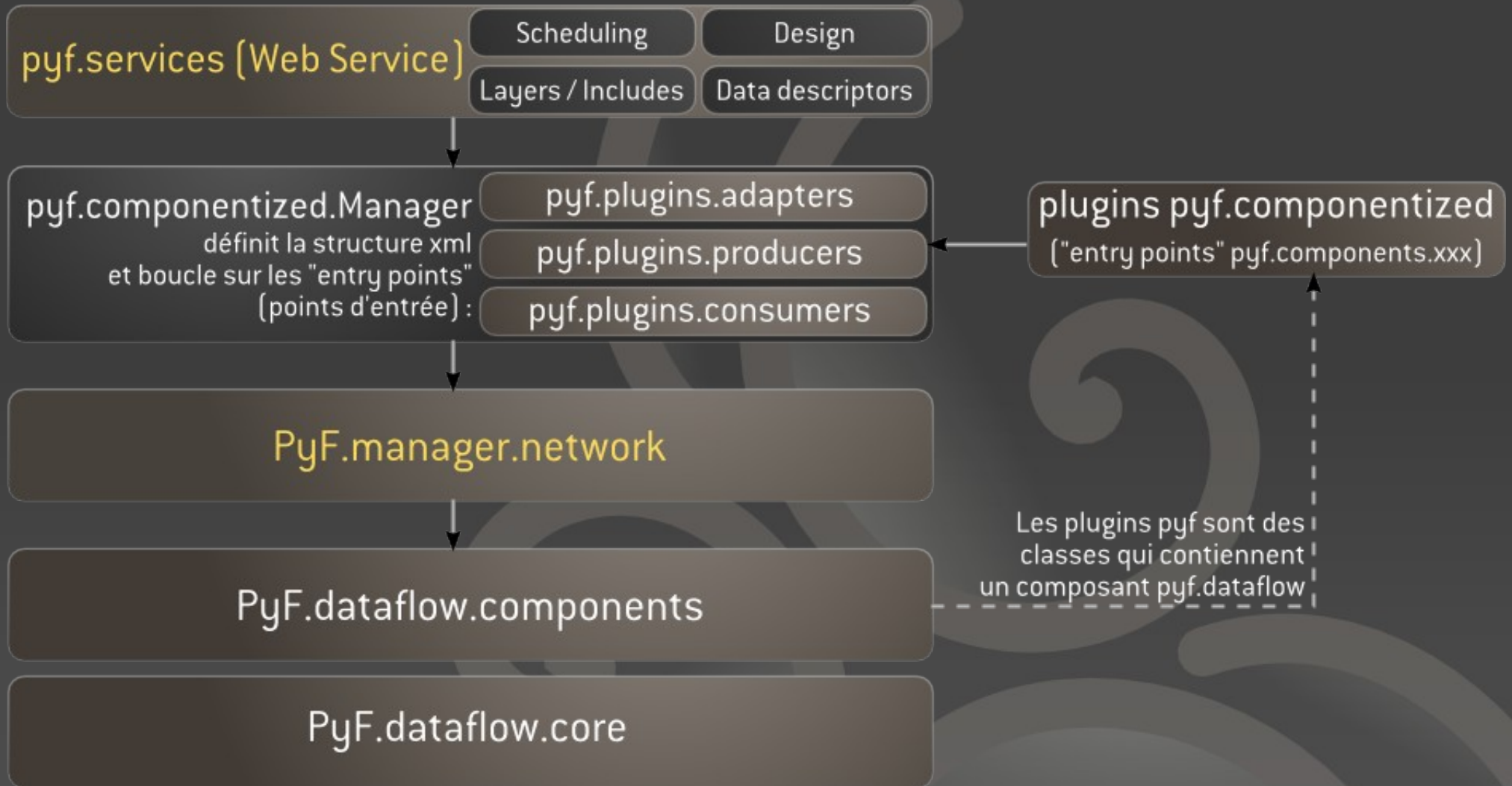
En Python : PyF

- PyF est un framework basé sur un fork de Zflow pour les couches basses (pyf.dataflow.core)

Fork « ami »

- PyF contient beaucoup plus que la base, et est « batteries included ».

Architecture de PyF





En Python : PyF

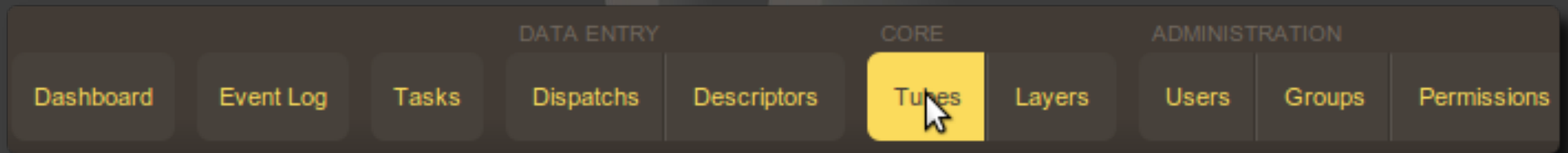
- Système de Plugins
 - De nombreux existants
 - pour l'extraction (site web, rss, sql alchemy...),
 - pour l'adaptation,
 - pour l'écriture (csv, xml, fixed-length, xlsx, pdf avec xhtml, odt ou rml, etc.)
- Du code python peut être mis dans le code source d'un tube
- Beaucoup d'outils annexes pour la gestion des flux (designer graphique, scheduler, ...)



Les avantages

- Votre business c'est vos données
- PyF les traite, rapidement et de façon « scalable »
 - Lecture,
 - Transformation,
 - Adaptation,
 - Reporting...

Le service web



Créer un tube

Info

Name:

Display Name:

Active

Needs Source

Components

- producer plugin
- consumer plugin
- producer code
- consumer code
- adapter code
- DescriptorSource
- WebExtractor
- SetAttributes
- ...

Controls

producer code

unique name

Name	Config
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	

Du code...

producer code

► Joiner Info

Name

Config

```
1 class User(object):
2     def __init__(self, name, email, level):
3         self.name = name
4         self.email = email
5         self.level = level
6
7     def get_source():
8         for index in range(0, 10):
9             yield User(
10                "John%02d" % index,
11                "john%02d" % index,
12                ['high', 'low'][index % 2])
13
14
15
16
17
18
19
20
21
22
23
```

adapter code

► Joiner Info

► Advanced

Name

Config [Add](#)

```
1 from itertools import groupby
2 from operator import attrgetter
3 from pyf.transport import Packet
4
5 def aggregate_lines(lines):
6     for account_code, acc_lines in groupby(lines, key=attrgetter('account_code')):
7         yield Packet(dict(account_code=account_code,
8                            lines=list(acc_lines)))
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

Ou des plugins...

pyf.components.adapters.simple_filter

▶ Joiner Info

▶ Advanced

Name

Filter expression

pyf.components.adapters.summarizer

▶ Joiner Info

▶ Advanced

Name

Attribute	<input type="text" value="total_amount"/>
Summary Type	<input type="text" value="sum"/>
Data Getter	<input type="text" value="attribute"/>
Getter	<input type="text" value="amount"/>
	remove

Attribute	<input type="text" value="average_amount"/>
Summary Type	<input type="text" value="average"/>
Data Getter	<input type="text" value="attribute"/>
Getter	<input type="text" value="amount"/>
	remove

[Add](#)

Yield Items (or just the summary at the end)

Attribute of summary

21
22
23



pyf.components.adapters.simple_filter X

▶ **Joiner Info**

Name

Filter expression

21
22
23



pyf.components.adapters.simple_filter X

▶ **Joiner Info**

Name

Filter expression

21
22
23



pyf.components.adapters.simple_filter X

▶ **Joiner Info**

Name

Filter expression

▼ **Advanced**

Separate Process

```

producer code
> Joiner Info
Name: source1
Config: Add
1 class User(object):
2     def __init__(self, name, email, level):
3         self.name = name
4         self.email = email
5         self.level = level
6     def get_source():
7         for index in range(0, 10):
8             yield User(
9                 "John%04d" % index,
10                "john%04d@some-where.com" % index,
11                ["high", "low", "high", "high", "low"][index%5])
12
13
14
15
16
17
18
19
20
21
22
23

```

```

pyf.components.adapters.simple_filter X
> Joiner Info
Name: filter1
Filter expression: item.level == "high"

```

```

adapter code
> Joiner Info
Name: filter2
Config: Add
1 def low_or_med(items):
2     for item in items:
3         if item.level == "low" or item.level == "med":
4             yield item
5         else:
6             yield Ellipsis
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

```

```

pyf.components.adapters.compute_attrbu X
> Joiner Info
Name: adapter1
Attribute: numeric_level
Type: eval
Value: 10
remove Add

```

```

adapter code
> Joiner Info
Name: adapter2
Config: Add
1 def set_numeric_level(items):
2     for item in items:
3         item.numeric_level = 0
4         yield item
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

```

```

pyf.components.consumers.csvwriter X
> Joiner Info
Name: csvoutput
Encoding: UTF-8
Target filename: user_levels.csv
Delimiter: ;
Write header line: [checked]
Title: name
Attribute: Source object attribute
Renderer: Renderer eval (optional) remove
Title: email
Attribute: Source object attribute
Renderer: Renderer eval (optional) remove
Title: level
Attribute: numeric_level
Renderer: Renderer eval (optional) remove
Add

```


A bientôt !

- Pour plus d'informations :

<http://www.pyfproject.org/>

<http://groups.google.com/group/pyf-users>