

# Bonjour,

BATCH:

Planifié

Tout un lot

+ Souvent

inconvenients:

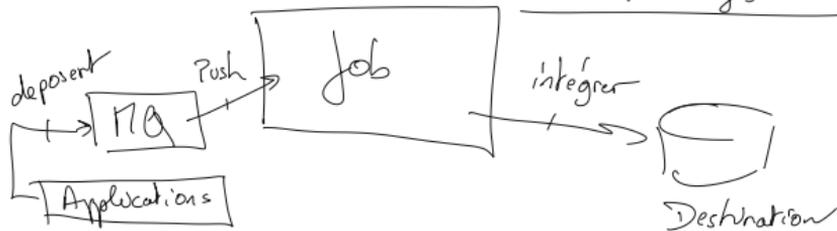
- Montée en charge variable
- Délai de traitement

Flux:

Temps réel  
en fil de l'eau

- Doit tourner  
constamment

Message Application RA = garantie de remise des messages



Talend → Java → Peu performant  
 Java JDBC → Tous les connecteurs de BDD  
 extensible (modules) ⇒ **TRÈS RICHES!**  
 → OLEDB  
 → ODBC  
 → .net provider for ...

SSIS → natif  
 → Limite en connecteurs

MS SQL → (Sybase)

SSAS → natif

SSIS → natif

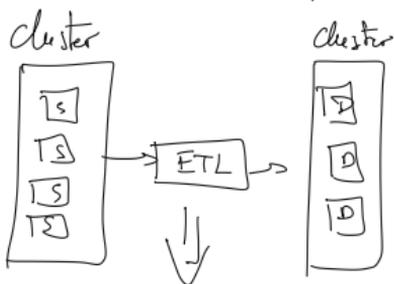
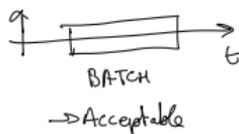
SSRS → App Web .net

---

Data

→ Big Data

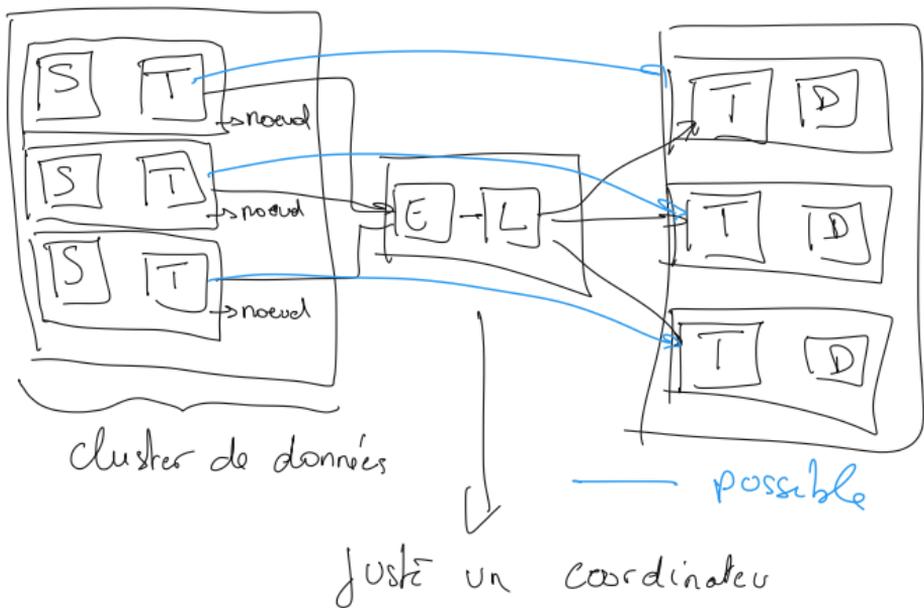
Volume Trop conséquents  
 Durée Trop longues



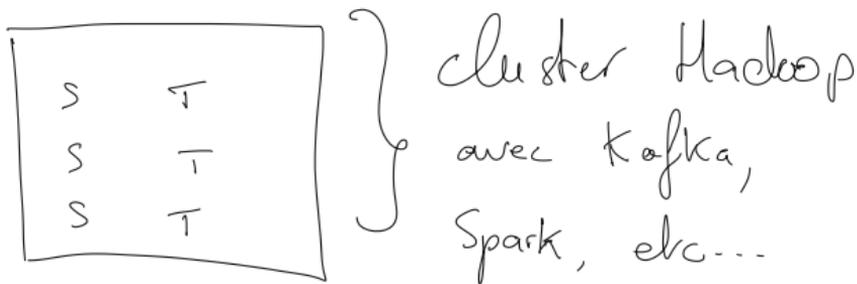
**GOULET**

BIG COMPUTE

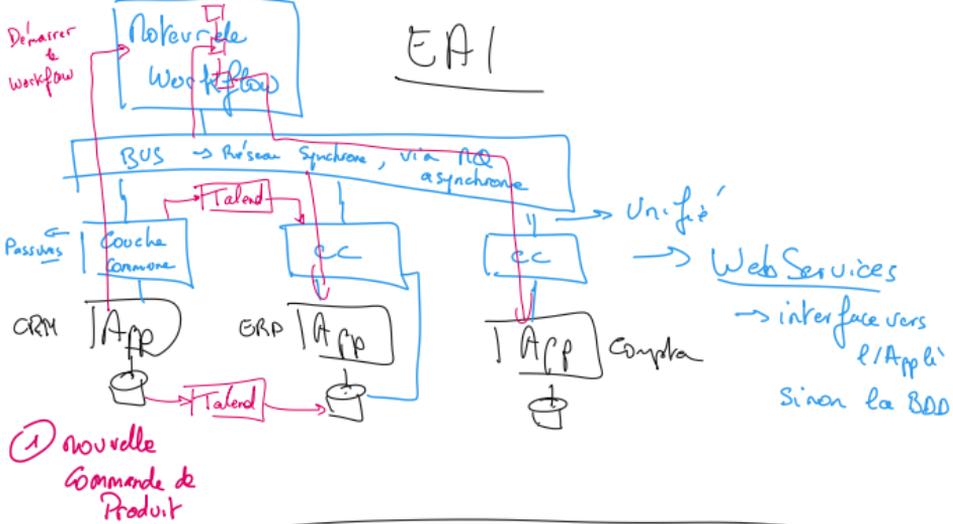
→ On envoie le Traitement à la data,  
 et non la data au Traitement



⇒ Talend for BigData

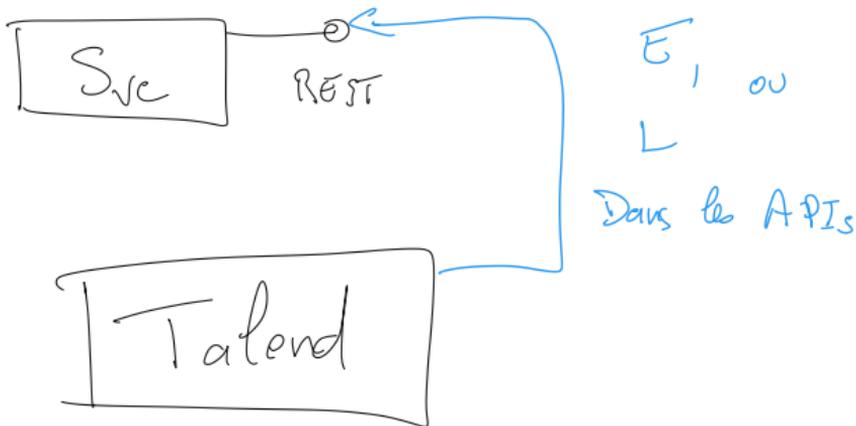


Talend for BigData encapsule  
le dev. de solutions Big Compute  
dans Hadoop



20204

→ architectures REST, voir microservices



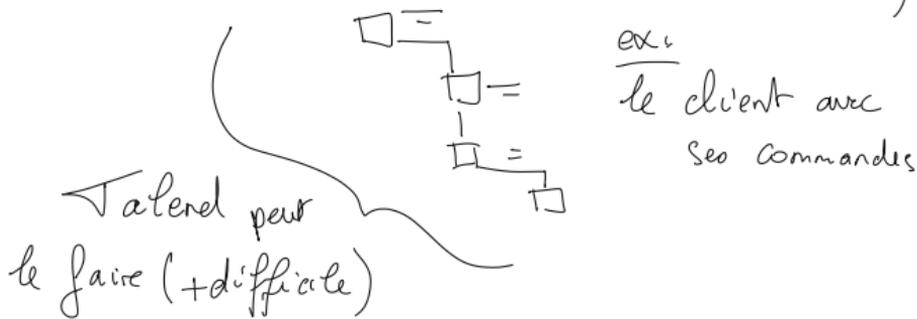
# Types de données

Scalaire → 1 valeur (bool, int, chaîne, ...)

Listes → n valeurs d'un type.

Colonnes → n valeurs de n types

Structurée → - soit colonnes -  
- soit hiérarchiques (complexes)



---

Nouvelles Technos = JSON  
(Pas Valent)

⇒ ORIENTÉ EN COLONNES

↳ Travailler avec des BDD → Talend

↳ Travailler en Big Data, flux  
structurés ou Semi-structurés

⇒ Spark, Jupyter Notebook  
(Python, Java, Scala)

→ ELT

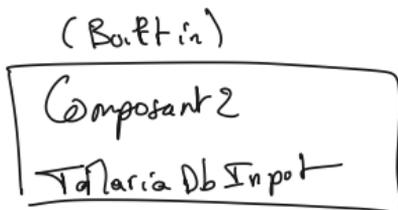
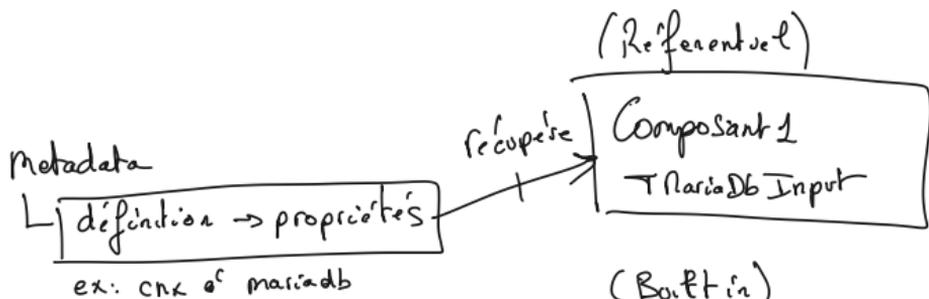
---

Comparaison des solutions :

<https://www.talend.com/fr/products/data-integration-compare-all/>

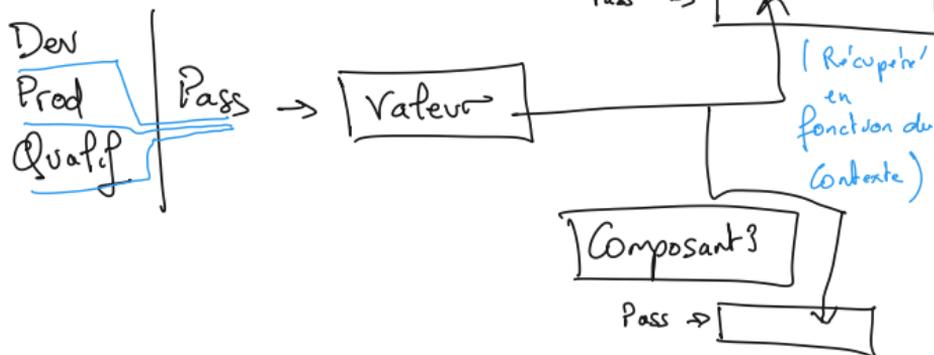
# Utilisation de métadonnées communes aux Composants

- Built-in = Composant autonome
  - Referentiel = Propriétés acquises depuis la Section "metadata"
- ↓
- "Type de Propriété" par Composant



host → spécifique  
user →  
pass →

## Contextes



Lieux de définition des variables :

1) Contextes :

liste d'une série de variables

Nom de contexte = les valeurs pour ces valeurs pour ce contexte

Aute contexte = autres pour valeurs pour ces variables

exemple : dev, prod, qualif, preprod, ...

Par défaut, le contexte par défaut s'appelle Default

Lecture :

context.<nom de variable>

```
System.out.println( context.nom );
```

```
System.out.println( context.getOrDefault("nom", "ok") );
```

```
System.out.println( context.getProperty("nom") );
```

The screenshot shows a Java IDE interface. At the top, there are two graphical components: a 'tJava\_1' component and a 'tFileInputRaw\_1' component connected to a 'tLogRow\_1' component. The 'tLogRow\_1' component displays the text '1 rows in 0,01s' and '66,67 rows/s row1 (Main)'. Below the graphical interface, the 'Designer' tab is active, showing a table of variables. The table has columns for Name, Type, Comment, Value, and Default. The first row has Name 'nom', Type 'String', Comment 'exemple de variable', Value 'Philippe', and Default is empty. The second row has Name 'filename', Type 'String', Comment is empty, Value '/home/student/Bureau/out.xml', and Default is empty. The 'Code' tab is also visible, showing the code snippets from the text above.

Name	Type	Comment	Value	Default
1 nom	String	exemple de variable	Philippe	
2 filename	String		/home/student/Bureau/out.xml	

Schéma	Built-In	▼	Modifier le schéma	<input type="checkbox"/>
Nom du fichier	context.filename			
Mode				
<input checked="" type="radio"/> Lire le fichier comme une chaîne de caractères				
<input type="radio"/> Read the file as a bytes array				
<input type="radio"/> Stream the file				
Encodage	ISO-8859-15	▼		
<input type="checkbox"/> Arrêter en cas d'erreur				

Les variables globales, sont des variables, qui ne sont pas Définies avec le contexte ( ne sont pas des paramètres au lancement du job )

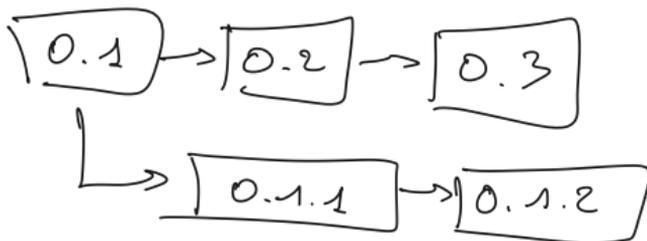
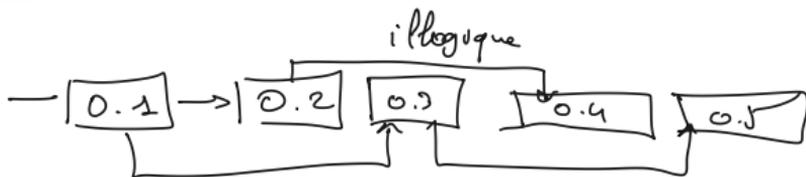
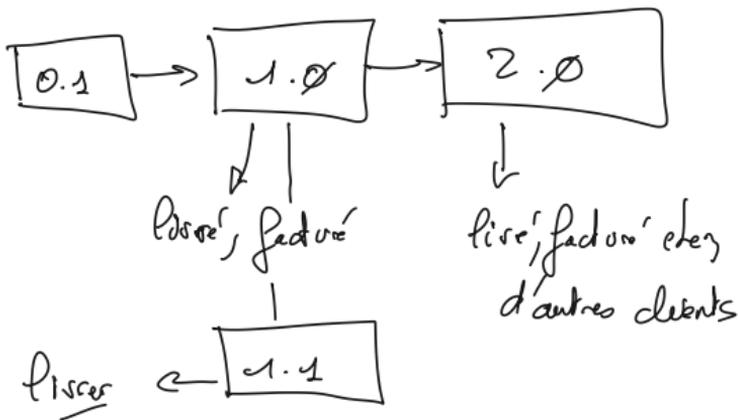
Elles sont accessible via

```
globalMap.get("keyname")
```

et

```
globalMap.set("keyname", "keyvalue")
```





Site d'apprentissage des version avec Git :

[https://learngitbranching.js.org/?locale=fr\\_FR](https://learngitbranching.js.org/?locale=fr_FR)

Planification : Utiliser CRON ( par exemple ) :

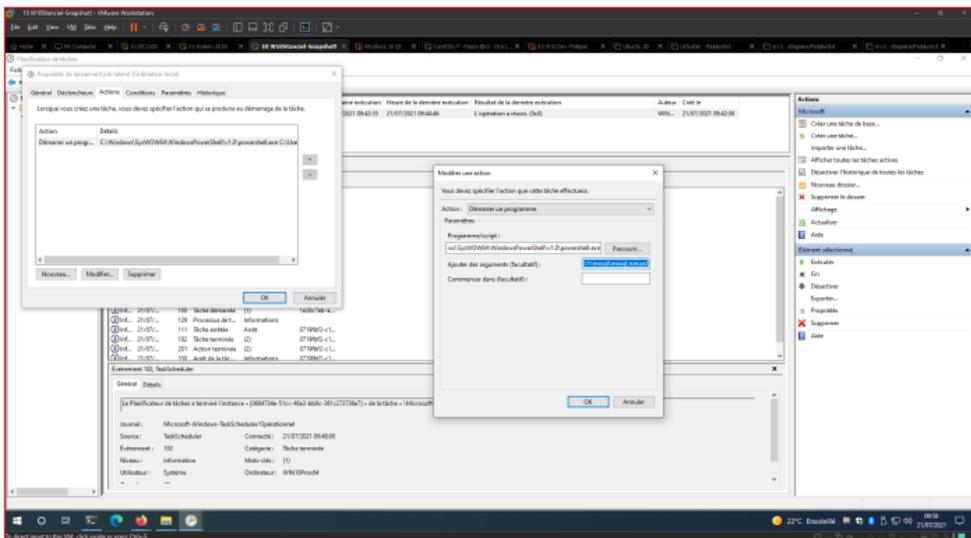
Site de tutos : <https://www.server-world.info/en/>

Planification du script via Windows planificateur de tâches :

Soit lancer cmd.exe et paramètres : /K <chemin du script>

Ou powershell :

powershell.exe paramètre : chemin du .ps1



Sous Linux :

Afficher cat /etc/crontab pour avoir la liste des champs :

```
[student@linux mssql]$ cat /etc/crontab
```

```
SHELL=/bin/bash
```

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin
```

```
MAILTO=root
```

```
# For details see man 4 crontabs
```

```
# Example of job definition:
```

```
# .----- minute (0 - 59)
```

```
# | .----- hour (0 - 23)
```

```
# | | .----- day of month (1 - 31)
```

```
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
```

```
# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR
```

```
sun,mon,tue,wed,thu,fri,sat
```

```
# | | | | |
```

```
# * * * * * user-name command to be executed
```

```
[student@linux mssql]$ cat /etc/crontab
```

Ensuite, editer notre crontab :

```
crontab -e
```

( le fichier est vide, on peut copier/coller dedans le modèle de /etc/crontab

Attention, il ne faut pas spécifier le nom d'utilisateur s'il on est pas root

Exemple :

```

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user-name command to be executed

```

```
05 10 * * * /work/app/TOS_DI-20200219_1130-V7.3.1/jobs/mssql/mssql_run.sh
```

Vérifier l'exécution des tâches :

Il faut contrôler le fichier log de cron, dépendant de l'OS que l'on utilise

Sous CentOS :

```

[root@linux Bureau]# tail -f /var/log/cron
Jul 21 10:01:01 linux CROND[19339]: (root) CMD (run-parts /etc/cron.hourly)
Jul 21 10:01:01 linux run-parts(/etc/cron.hourly)[19339]: starting 0anacron
Jul 21 10:01:01 linux anacron[19352]: Anacron started on 2021-07-21
Jul 21 10:01:01 linux anacron[19352]: Will run job `cron.daily' in 28 min.
Jul 21 10:01:01 linux anacron[19352]: Will run job `cron.weekly' in 48 min.
Jul 21 10:01:01 linux anacron[19352]: Jobs will be executed sequentially
Jul 21 10:01:01 linux run-parts(/etc/cron.hourly)[19354]: finished 0anacron
Jul 21 10:02:00 linux crontab[19053]: (student) REPLACE (student)
Jul 21 10:02:00 linux crontab[19053]: (student) END EDIT (student)
Jul 21 10:02:05 linux crontab[19437]: (student) LIST (student)
Jul 21 10:05:01 linux CROND[19834]: (student) CMD
(/work/app/TOS_DI-20200219_1130-V7.3.1/jobs/mssql/mssql_run.sh)

```

Semble ok

## Travailler avec les Sous Jobs

### 1) Les variables de contexte :

-On a la possibilité de les passer dans les sous jobs pour exploiter les paramètres transmis à l'appel du job principal au lancement.

-Ces variables doivent être créés avec exactement le même nom dans chaque sous job

-Si ces variables existent, elles seront donc écrasées par les valeurs du job principal si le paramètre "transmettre tout le contexte" est sélectionné, ou pour chaque correspondance de paramètre dans la liste

The screenshot shows a configuration window for a job named 'tRunJob\_1'. The window has a title bar with tabs for 'job(mainjob v.1)', 'Contextes(mainjob)', and 'Composant', and a play button labeled 'executer(job mainjob)'. On the left is a sidebar with a tree view containing 'Paramètres simples', 'Paramètres avancés', 'Paramètres dynamiques', 'Iew', and 'Documentation'. The main area contains several checkboxes: 'Utiliser un processus indépendant pour exécuter le sous-job.' (unchecked), 'Arrêt en cas d'erreur du fil' (checked), and 'Transmettre tout le contexte' (unchecked). Below these is a table titled 'Paramètre de contexte' with two columns: 'Paramètres' and 'Valeurs'. The table is currently empty.

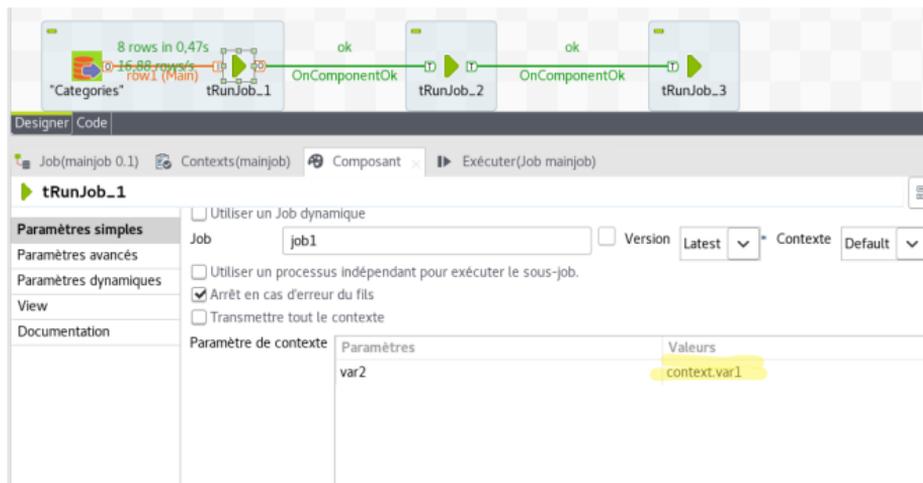
### Exécution

Three buttons are visible: 'Exécuter' with a play icon, 'Arrêter' with a red square icon, and 'Effacer' with a trash icon.

```
[statistics] connecting to socket on port 3655
[statistics] connected
main
sub : subjob
[statistics] disconnected

Job mainjob ended at 10:48 21/07/2021. [exit code = 0]
```

Transmettre la valeur d'une variable de contexte depuis le principal vers l'enfant ( et non pas tout le contexte avec les valeurs en correspondance uniquement ) :



il s'agit finalement de tout type d'expression en Java

Exercice :

Créez un job principal, qui appelle un sous job, et y transmet en correspondance automatique, une variable de contexte ( n'importe laquelle ), et transmet également l'heure d'exécution du job principal dans la variable de contexte 'dateexecution' )

Pour lire l'heure : `java.time.LocalDate.now()`

Affichez ces deux variables de contexte via `System.out.println()`

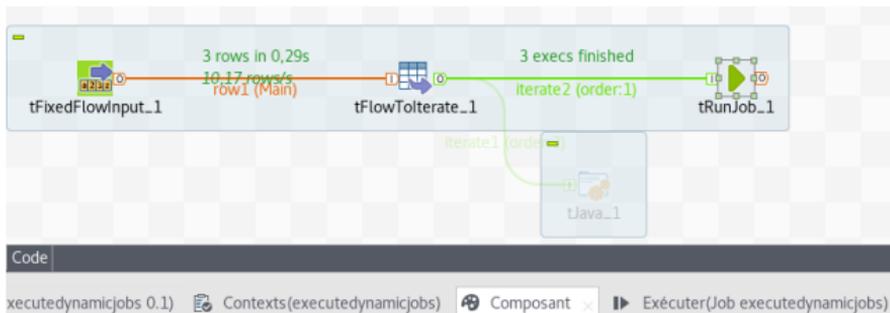
Un job dynamique est une opportunité d'exécuter des jobs dont les noms sont définis dans une variable

ex : job1, job2, job3

jobs = ["job1", "job2"]

-> job3 ne sera pas exécuté.

Il faut que la valeur passée dans la variable de "contexte de job" ( rien à voir avec la notion de contexte ) fasse partie de la liste des jobs possibles :



### nJob\_1

**res simples** Schéma Built-In Modifier le schéma Copier le schéma du Job enfant

**es avancés**  Utiliser un Job dynamique Context job ((String)globalMap.get("row1.jobname"))

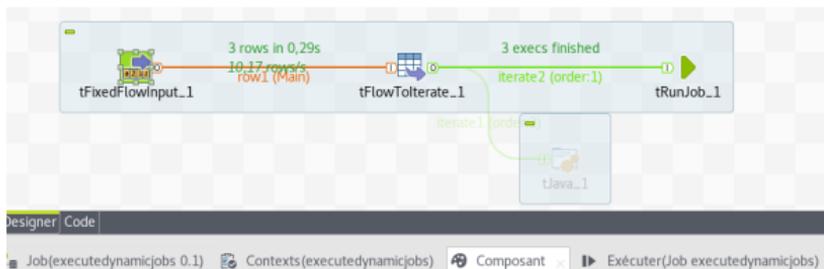
**es dynamiques** Job job2;job3 Version Latest

**itation** Contexte Default

Arrêt en cas d'erreur du fils

Transmettre tout le contexte

Paramètre de contexte Paramètres Valeurs



### tFixedFlowInput\_1

**Paramètres simples** Schéma Built-In Modifier le schéma

**Paramètres avancés** Nombre de lignes 1

**Paramètres dynamiques** Mode

**View**  Utiliser une table à une ligne

Utiliser une table à plusieurs lignes

Utiliser du contenu personnalisé (fichier délimité)

**Documentation** Séparateur de lignes "\n" Séparateur de champs ";"

Contenu job3  
job2  
job2

Utilisation des Buffers :

BufferInput permet de lire le contenu d'un buffer

BufferOutput permet d'injection un flux dans le buffer

Il n'y a qu'un buffer

C'est une zone mémoire partagée entre les sous jobs ( d'un même job, ou inter jobs )

**Cette zone mémoire est invalidée si la lecture ( bufferinput se fait depuis un autre process Java ( mais pas un autre thread ))**

Code

ssagedeflux...main 0.1) Contexts Composant ▶ Exécuter(Job passagedeflux...main) x

**agedeflux...main**

Exécution

simple  
Debug  
s avancés  
nte  
mémoire

▶ Exécuter   ■ Arrêter   🗑 Effacer

Starting job passagedeflux\_main at 11:37 21/07/2021.  
[statistics] connecting to socket on port 3752  
[statistics] connected

tLogRow\_1

Code_categorie	Nom_categorie	Description	Illustration
1	Boissons	Boissons, cafés, thés, bières	net.sourceforge.jtds.jdbc
2	Condiments	Sauces, assaisonnements et épices	net.sourceforge.jtds.jdbc
3	Desserts	Desserts et friandises	net.sourceforge.jtds.jdbc
4	Produits laitiers	Fromages	net.sourceforge.jtds.jdbc

Talend Open Studio for Data Integration (7.3.1.20200219\_1130) | Local\_Project (Connexion: local)

Attention, Pas Synchronisation de Schéma fixe le schéma de SORTIE et non ENTRÉE

Colonne	Db	Column	Clé	Type	Type de	Null	Modèle	Len	Pre	Def
Ref_pro	Ref_produ	INT	INT					10	0	
Nom_Prod	Nom_Prod	Stri	NVARCHAR					40	0	
No_fourn	No_fourns	Inte	INT					10	0	
Code_cat	Code_cate	Inte	INT					10	0	
Quantite	Quantite_p	Stri	NVARCHAR					30	0	
Prix_unit	Prix_unitair	Obj	DECIMAL					19	4	
Unites_1	Unites_em	Shc	SMALLINT					5	0	
Unites_1	Unites_cor	Shc	SMALLINT					5	0	

Paramètres simples: Schéma: Built-in, Modifier le schéma:  Sync colonnes, Copier le schéma du Job enfant

Paramètres avancés:  Utiliser un Job dynamique

Paramètres dynamiques: Job: [ ], Version: Latest, Contexte: [ ]

View:  Utiliser un processus indépendant pour exécuter le sous-job.

Documentation:  Arrêt en cas d'erreur du fils,  Transmettre tout le contexte

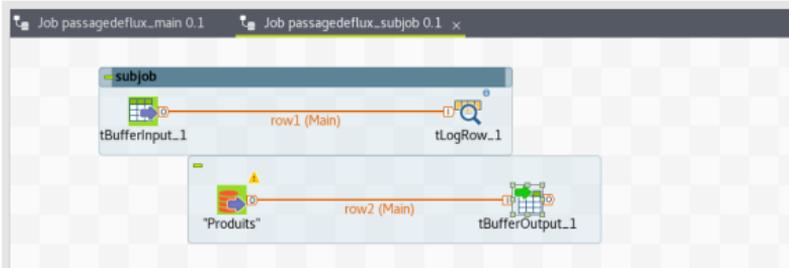
Paramètre de contexte: Paramètres, Valeurs

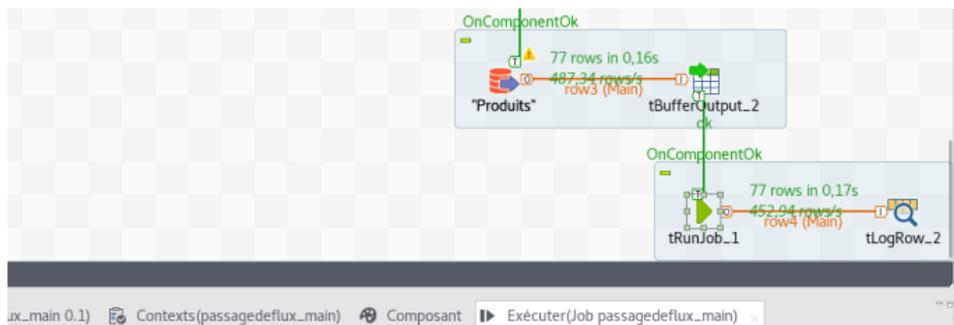
Il n'est pas possible de passer un flux en entrée dans un sous Job  
On a vu que la connexion d'un flux engendre l'appel de n fois le sous job pour n lignes

Passage des lignes dans le sous job via bufferinput / output :

Récupération des lignes issues d'un sous job :

Il faut envoyer le résultat dans un bufferinput dans le sousjob,  
pour ensuite simplement consulter le flux de sortie du trunjob  
Attention, les schémas doivent correspondre





## ix\_main

Exécution

▶ Exécuter    ⏹ Arrêter    🗑 Effacer

70	Outback Lager	7	1	24 bouteilles (355 ml)
71	Flätmysost	15	4	10 cartons (500 g)
72	Mozzarella di Giovanni	14	4	24 cartons (200 g)
73	Röd Kaviar	17	8	24 pots (150 g)
74	Longlife Tofu	4	7	1 carton (5 kg)
75	Rhönbräu Klosterbier	12	1	24 bouteilles (0,5 litre)
76	Lakkalikööri	23	1	1 bouteille (500 ml)
77	Original Frankfurter grüne Soße	12	2	12 boîtes

[statistics] disconnected

Job passagedeflux\_main ended at 11:57 21/07/2021. [exit code = 0]

Masquer limite de lignes     Retour automatique à la ligne