Bonjour

BATCH :

Planifié

Tout un lot

+ Souvent

inconvénients :

- Montré en charge variable
- Délai de traitement

Flux :

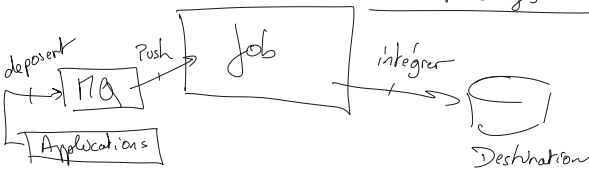
Temps réel

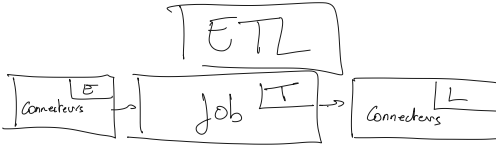
au fil de l'eau

- Doit tourner

Constamment

Messagerie Application MQ = garantie de remise des messages





Talend → Java → Peu performant

Java JDBC → Tous les connecteurs de BDD

extensible (modules) ⇒ TROS RICHE!

SSIS → natif → OLEDB

ODBC

→ Limité en connecteurs

.net provider for ....

NSSQL → (Sybase)

SSAS → natif

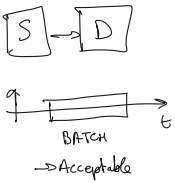
SSIS → natif

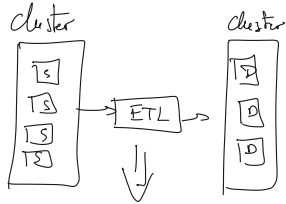
SSRS → App Web .net

Data → Big Data

Volume Trop conséquents

Durée Trop longues



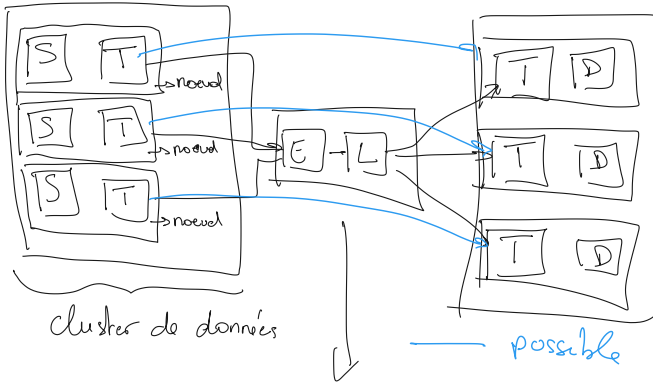


GOULET

BIG COMPUTE

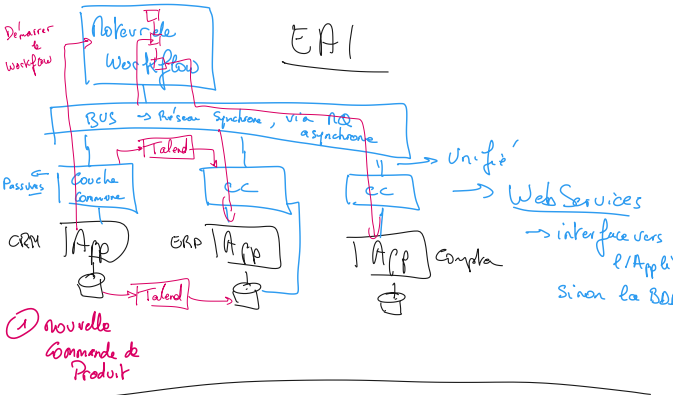
→ On envoie le Traitement à la data,

et non la data au Traitement



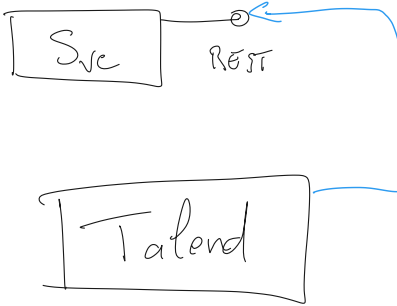
⇒ Talend for BigData

Talend for BigData encapsule le dev. de solutions BigCompute dans Hadoop



2020

→ architectures REST, voir microservices



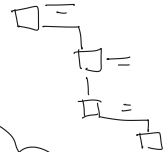
Types de données

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

Listes → n valeurs d'un Type.

Colonnées → n valeurs de n Types

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



ex : le client avec ses commandes

Talend peut le faire (+difficile)

Nouvelles Techno = JSON
(Pas Talend)

⇒ ORIENTÉ EN COLONNES

Travailler avec des BDD → Talend

Travailler en Big Data, flux structurés ou semi-structurés

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

→ ELT

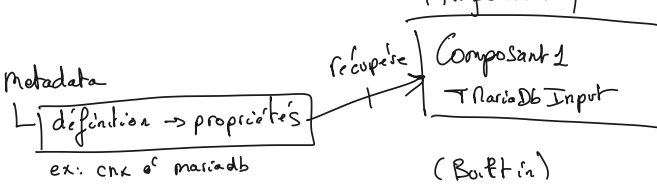
Comparaison des solutions :
https://www.talend.com/fr/products/data-integration-compare-all/

Utilisation de métadonnées communes aux Composants

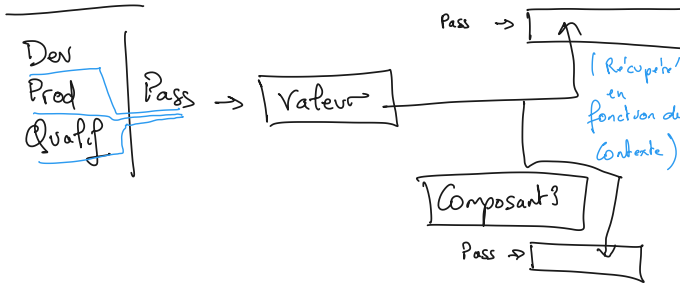
- Built-in = Composant autonome
- Référentiel = Propriétés acquises depuis la Section "metadata"

↓

"Type de Propriété" par Composant



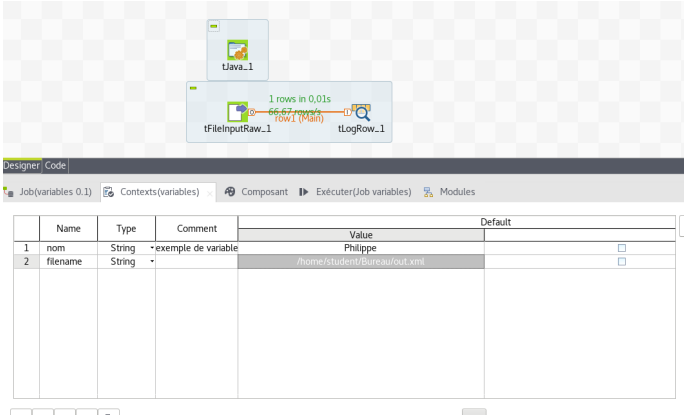
Contextes

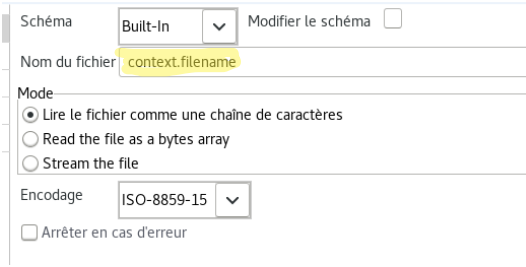


Lieux de définition des variables :
1) Contextes :
liste d'une série de variables
Nom de contexte = les valeurs pour ces valeurs pour ce contexte
Aute contexte = autres pour valeurs pour ces variables
exemple : dev, prod, qualif, preprod, ...

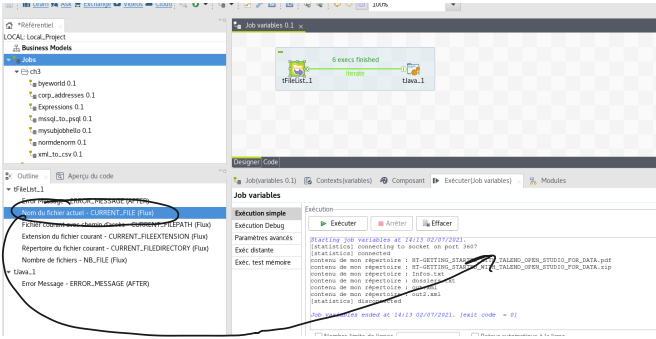
Par défaut, le contexte par défaut s'appelle ***Default***

Lecture :
context.<nom de variable>
System.out.println(context.nom);
System.out.println(context.getOrDefault("nom", "ok"));
System.out.println(context.getProperty("nom"));



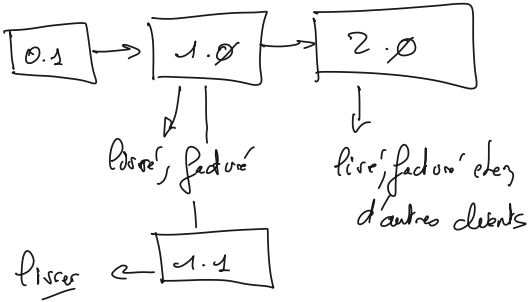


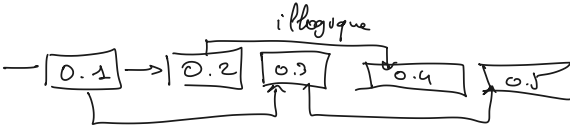
Les variables globales, sont des variables, qui ne sont pas Définies avec le contexte (ne sont pas des paramètres au lancement du job)
Elles sont accessible via
globalMap.get("keyname")
et
globalMap.set("keyname", "keyvalue")

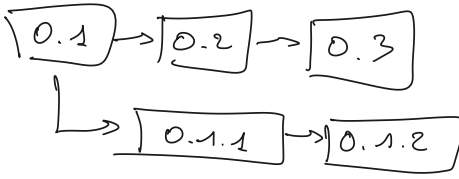


faire un drag'n'drop pour générer le code!

```
System.out.println("contenu de mon répertoire : "+
(String)globalMap.get("tFileList_1_CURRENT_FILE"));
```



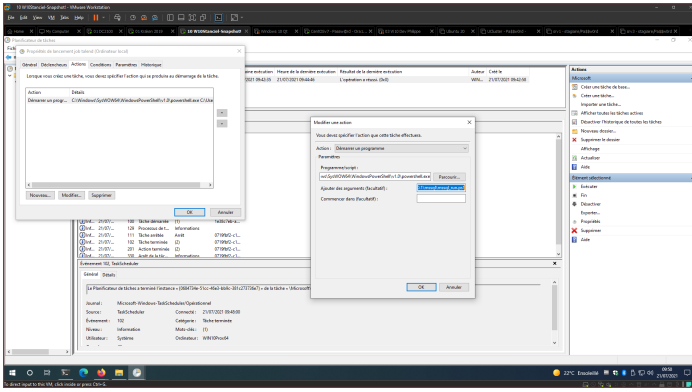




Site d'apprentissage des version avec Git :
https://learngitbranching.js.org/?locale=fr_FR

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
```

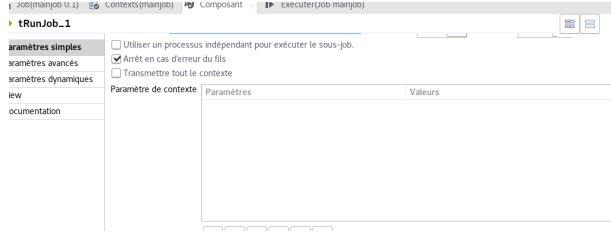
Vérifier l'execution des tâches :
Il faut contrôler le fichier log de cron, dépendant de l'OS que l'on utilise
Sous CentOS :

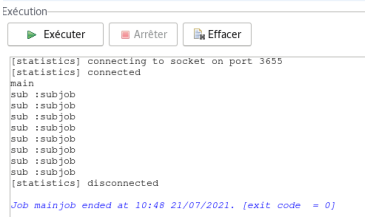
```
[root@linux Bureau]# tail -f /var/log/cron
Jul 21 10:01:01 linux CROND[19339]: (root) CMD (run-parts /etc/cron.hourly)
Jul 21 10:01:01 linux run-parts(/etc/cron.hourly)[19339]: starting 0anacron
Jul 21 10:01:01 linux anacron[19352]: Anacron started on 2021-07-21
Jul 21 10:01:01 linux anacron[19352]: 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

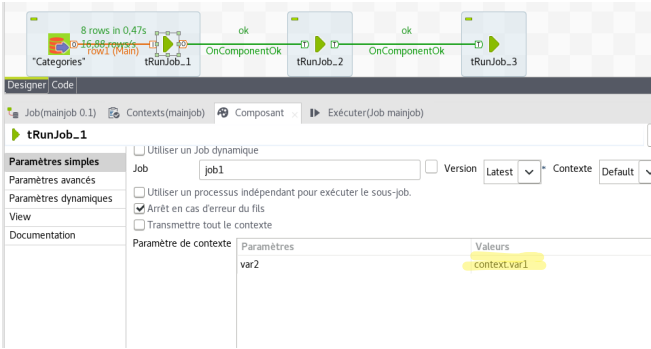
Travailler avec les Sous Jobs

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 "tansmettre tout le contexte" est sélectionné, ou pour chaque correspondance de paramètre dans la liste





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) :

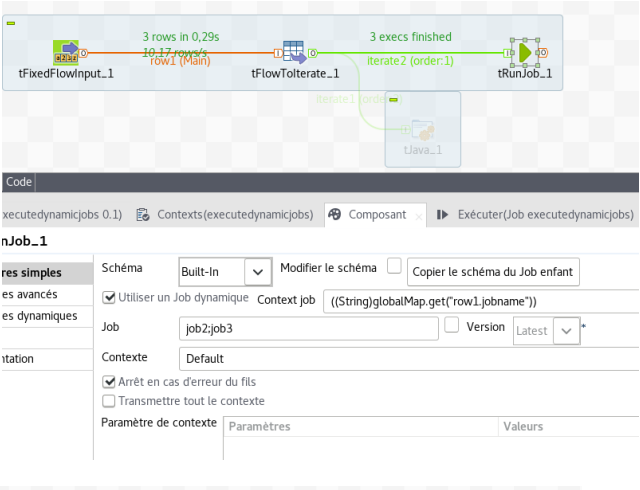


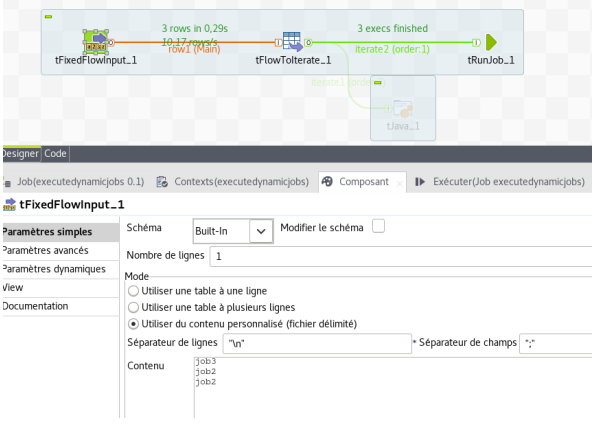
il s'agit finalement de tout type d'expression en Java

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'execution du job principal dans la variable de contexte 'dateexecution')
Pour lire l'heure : java.time.LocalDate.now()
Affichez ces deux variables de context via System.out.println( )

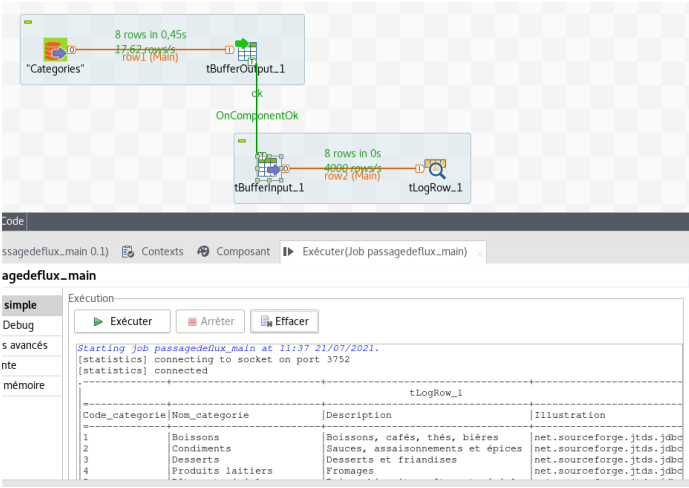
Un job dynamique est une opportunité d'executer des jobs dont les noms sont définis dans une variable
ex : job1, job2, job3
jobs = ["job1", "job2"]
-> job3 ne sera pas éxecuté.

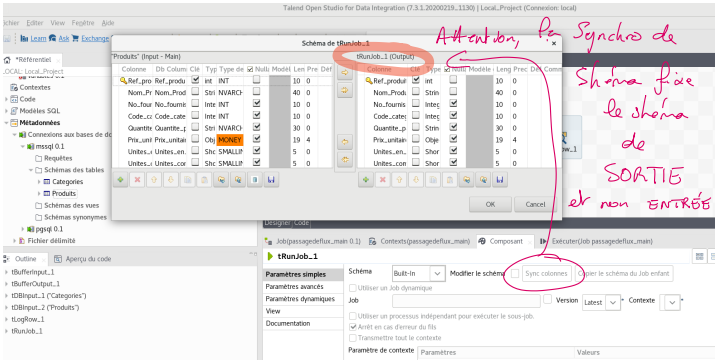
Il faut que la valeur passée dans la variable de "contexte de job" (rien à voir avec la notion de contexte) fasse partie de la liste des jobs possibles :





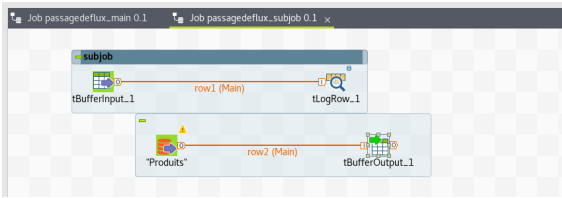
Utilisation des Buffers :
BufferInput permet de lire le contenu d'un buffer
BufferOutput permet d'injection un flux dans le buffer
Il n'y a qu'un buffer
C'est une zone mémoire partagée entre les sous jobs (d'un même job, ou inter jobs)
Cette zone mémoire est invalidée si la lecture (bufferinput se fait depuis un autre process Java (mais pas un autre thread))

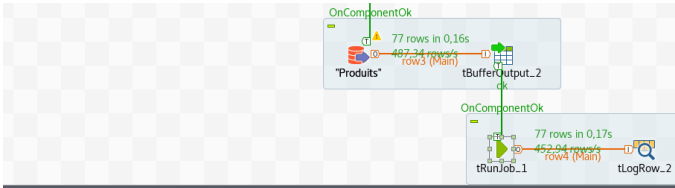




Il n'est pas possible de passer un flux en entrée dans un sous Job
On a vu que la connexion d'un flux engendre l'appel de n fois le sous job pour n lignes
Passage des lignes dans le sous job via bufferinput / output :

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





ux_main 0.1) Contexts(passagedeflux_main) Composant Exécuter(Job passagedeflux_main)

ux_main

Exécution

Exécuter Arrêter Effacer

70	Outback Lager	7	1	24 bouteilles (355 ml)
71	Fløtemysost	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]

Nombre limite de lignes Retour automatique à la ligne