Open Source Jobscheduler

Bonnes pratiques

Automate d'exploitation

Solutions Open Source Paris
1, impasse des pommerets
92310 Sèvres
http://sos-paris.comhttp://sos-paris.com/http://sos-paris.com/

1 Présentation

Ce document récapitule les bonnes pratiques pour la mise en œuvre et l'utilisation d'Open Source JobScheduler.

Ce document est librement téléchargeable. Nous vous invitons à y participer en indiquant vos suggestions, vos questions. Pour contribuer, il suffit de nous écrire à support@sos-paris.com.

Convention : les bonnes pratiques sont en gras tout au long du document.

La première bonne pratique est de ne jamais hésiter à communiquer avec nous sur vos besoins et vos choix techniques, le support pourra vous préconiser des solutions conformes à l'utilisation du produit, maintenables et donc pérennes dans le temps. Pour toute demande, il est préférable contacter le support par mail à l'adresse support@sos-paris.com.

Vous disposez aussi d'un groupe d'information sur le logiciel, nouvelles versions, nouvelles fonctionnalités, patch.... Ce groupe est animé par nos soins sur LINKEDIN et tient lieu de communauté d'utilisateurs.
Open source JobScheduler - Communauté des utilisateurs francophones
La deuxième bonne pratique est de s'inscrire à ce groupe pour bénéficier de son retour d'expérience.

Si vous souhaitez être acteur, vous pouvez nous rejoindre sur l'espace http://partage.sos-paris.com en vous connectant avec votre compte LinkedIn ou Twitter. Vous pourrez participer aux discussions, échanger des outils et des idées et écrire des articles sur vos réalisations ou sur l'utilisation des outils.

La troisième bonne pratique est de participer, l'échange est l'un des principes fondateur de l'open source.

Ce document ne traite pas des normes que ce soit au niveau de l'organisation des applications ou du nommage des objets qui doivent respecter les règles internes de l'entreprise.

2 Administration

La partie installation est une phase déterminante car elle va conditionner le processus de conception et de pilotage.

2.1 Architecture

Open Source JobScheduler est assez souple pour offrir différents types d'architecture :

- Serveur/Agents pour centraliser l'ensemble des traitements sur une machine et rayonner sur les agents distribués
- En point à point pour une architecture collaborative constituée de serveurs autonomes afin de fiabiliser l'architecture
- En grille de calcul pour du calcul intensif avec un nombre variable de nœuds de traitements.

Dans le cadre d'une production informatique, nous préconisons une architecture centralisée en haute disponibilité avec un serveur dédié pour la qualification et le déploiement.

2.1.1 Dédier un moteur à une application

Il peut être judicieux de se demander si une application indépendante, c'est à dire sans dépendance avec d'autres applications, nécessite d'être centralisée sur un moteur. Cela est d'autant plus vrai si une application est critique car augmenter le volume de traitements génère mécaniquement une consommation des ressources.
Les informations d'exécution sont stockées sur une base de données centralisée permettant ainsi de conserver une vision globale des applications.

Dédier un moteur à une application permet :

- de maximiser les ressources pour l'application
- de minimiser l'impact en cas de maintenance sur le moteur
- d'identifier plus simplement un interlocuteur

Si des dépendances doivent être ajoutées par la suite, plusieurs solutions s'offrent à l'administrateur :

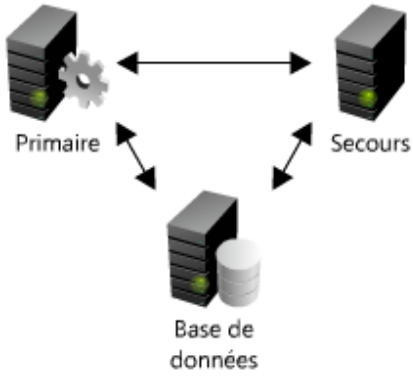
- transférer les traitements sur un moteur mutualisé
- ajouter des dépendances à travers l'orchestration
- mettre en place des dépendances par fichier flag

Bonne pratique : dédier un moteur à une application offre une meilleure qualité de services aux utilisateurs de cette application

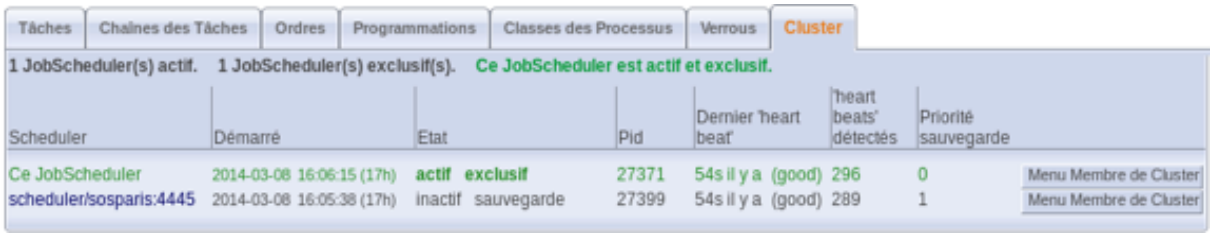
2.1.2 Disposer d'un serveur de secours

L'ordonnanceur est le cœur de la production, en cas d'arrêt de celui-ci c'est l'ensemble des traitements qui se retrouve bloqué. Pour éviter cette situation, Open Source JobScheduler dispose d'un mode cluster actif/passif.

Bonne pratique : installer le mode haute disponibilité pour assurer la continuité de service.



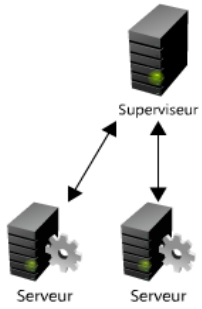
Le principe est de disposer d'un deuxième serveur ayant la même configuration que le premier et les mêmes objets d'ordonnancement. Il vérifie que le premier serveur répond et écrit correctement dans la base de données. S'il ne répond pas au bout de 2 minutes, il reprend la charge de l'automatisation.



2.1.3 Mettre en place un superviseur OJS

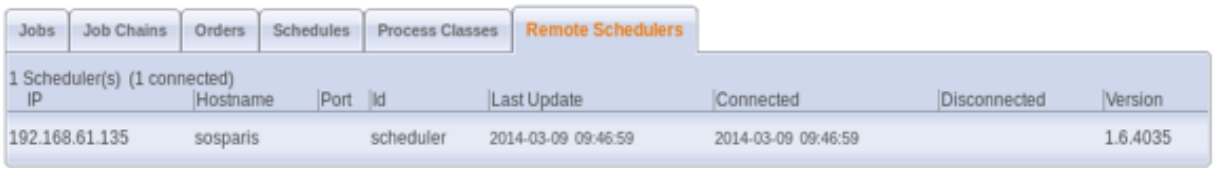
Le superviseur a deux fonctions principales :

- il sert pour le déploiement des objets pour distribuer les définitions vers l'ensemble des machines ou sur une machine spécifique
- il permet de centraliser les règles d'orchestration lorsqu'on souhaite créer des enchaînements complexes sur les ordonnanceurs du réseau



Pour le déploiement, les objets déposés dans le répertoire remote du supervisor sont directement déployés dans le répertoire cache de la machine distante.

Un JobScheduler devient superviseur lorsque d'autres ordonnanceurs s'abonnent à lui. Lorsqu'un nouveau moteur est enregistré, il apparaît sur l'interface du superviseur :



Un autre avantage du superviseur est de disposer d'un outil de surveillance des moteurs car le moindre problème de communication apparaît directement sur l'interface.

L'évaluation d'un grand nombre de règles est consommateur de ressources, dédier un moteur à cet effet permet de préserver les ressources des moteurs des applications.

Bonne pratique : mettre en place le superviseur sur les architectures multi-moteurs ou multi-sites

2.1.4 Connecter une base de données

La base de données sert à stocker les données d'historiques et l'état des objets en cas d'arrêt du système. Bien que facultative d'un point de vue technique, elle apparaît indispensable dans le cadre d'un arrêt de production.

Elle est aussi indispensable pour le mode haute disponibilité.

Bonne pratique : toujours mettre en place la BDD

2.1.5 Interface web

L'interface web (JOC) embarquée doit être réservée aux actions ponctuelles car elle communique directement avec le moteur, le temps de réponse dépend donc du nombre d'événements à traiter. Le moteur devant répondre à chaque utilisateur, les temps de réponses seront croissants avec le nombre de personnes connectées.

Bonne pratiques : Pour le suivi, il est préférable d'utiliser le dashboard ou Arii qui fournissent les informations à travers les données d'historique de la base. Les utilisateurs ne communiquent plus directement avec le moteur.
Avec Arii, l'utilisateur peut avoir des informations de tous les moteurs en simultané.

2.1.6 Mettre en place la supervision Nagios

La remontée des erreurs vers une plateforme de supervision est indispensable pour un serveur de production. SOS Berlin fournit un connecteur pour Nagios en mode passif ou en NCSA.

Une autre solution est l'utilisation d'Arii qui propose une surveillance en fonction d'un filtre sur le nom d'un traitement afin de définir un service Nagios en fonction d'un traitement, d'un groupe de traitements ou de l'ensemble des traitements.

Bonne pratique : le mode passif ou polling est généralement préféré.

2.2 Agents

Un agent n'est ni plus ni moins qu'un serveur dont le rôle est limité à l'exécution des traitements. Nous conseillons de toujours installer un serveur pour profiter pleinement de toutes les fonctionnalités même si ce n'est pas le cas au moment de l'installation.

Le principal intérêt est de pouvoir rendre autonome cette installation en fonction des besoins.

Bonne pratique : installer des serveurs plutôt que des agents.

2.2.1 Agents windows

Il existe de multiples moyens sur Unix d'exécuter une commande avec un compte particulier (su -, sudo, seteuid, etc...) mais il n'en est pas de même pour Windows. JobScheduler peut indiquer un login et un mot de passe dans la définition du traitement mais une meilleure solution est d'installer plusieurs instances du moteur (ou de l'agent). Chacune de ces instances ne comporte qu'un seul utilisateur.

Le principe est de profiter du service Windows, cela apporte plusieurs avantages :

- le mot de passe est stocké dans le système de la machine
- la base de registres de l'utilisateur est accessible car chargée par le service
- l'arrêt d'un jobscheduler n'impacte qu'une application

Bonne pratique : Sur les serveurs Windows, installer installer une instance du moteurtou de l'agent pour chaque compte utilisateur..

2.2.2 Agent SSH

Contrairement à ce qu'on pourrait penser, utiliser un agent SSH au lieu d'un agent JobScheduler est plutôt une bonne pratique car elle apporte plusieurs avantages :

- l'agent SSH ne demande pas d'installation
- on peut indiquer un login particulier
- la clé évite de stocker le mot de passe
- les communications sont sécurisées
- l'ensemble des paramètres peut être redéfini en API

Bonne pratique : utiliser SSH pour les serveurs devant réaliser des jobs simples ou peu nombreux.

2.3 Développement

La conception des objets ne doit jamais être réalisée sur le serveur de production car le moteur est nettement plus sollicité pour la conception que pour la production. Lors des phases de conception, les objets sont ajoutés et supprimés jusqu'à obtenir un résultat cohérent, le moteur doit donc gérer l'ensemble des changements, les contrôles de cohérence et le traitement des erreurs.

Bonne pratique : mettre en place un environnement de test OJS

2.3.1 Qualifier sur un serveur dédié

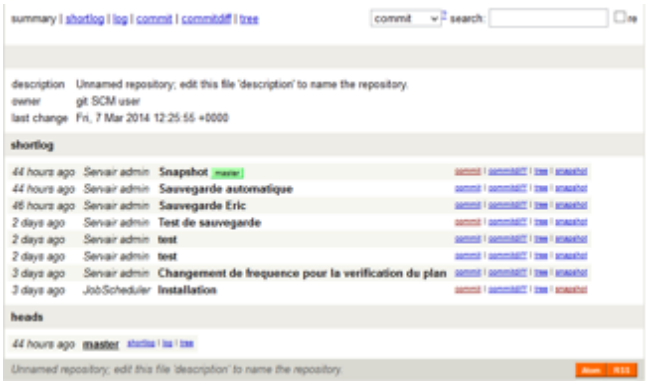
Un moteur de production doit être dédié aux tâches d'ordonnancement, donc à la planification des traitements, à leurs soumissions et au suivi. Utiliser le moteur dans le cadre de la conception consomme les ressources au détriment de l'exploitation. Un autre risque est l'impact d'un nouvel objet sur l'existant et la corruption des enchaînements.

Bonne pratique : Il est très recommandé de concevoir et tester sur un serveur dédié puis de transférer un ensemble cohérent vers le serveur de production.

2.3.2 Gérer les versions avec Git

Les objets d'ordonnancement sont des fichiers XML dans une arborescence, on peut les considérer comme le code source d'un développeur et utiliser les mêmes outils. Il existe de nombreux outils de gestion de versions dont le principe est de conserver des images de l'arborescence des fichiers, il est ainsi possible de connaître les divers ajouts et suppressions des fichiers jusqu'aux changements de contenu de ces fichiers.

Grâce à la sauvegarde de ces images, on peut revenir à une version passée et retrouver un état cohérent.



Solutions Open Source Paris fournit un espace GIT pour sauvegarder l'ensemble de vos objets sur un site distant sans avoir à installer un serveur dédié en interne.

Bonne pratique : installer le client GIT (livré avec la VM), installer un serveur GIT en interne ou utiliser celui de SOS Paris pour sauvegarder l'ensemble des objets et suivre les changements.

Le serveur permet de visualiser les changements de contenu de chaque fichier.

Dans l'exemple ci-contre, le script du traitement a été modifié pour y ajouter la commande *git status*

```
diff --git a/live/sos/git/Add.job.xml b/live/sos/git/Add.job.xml
index 19ab98a..430ef3d 100644 (file)
--- a/live/sos/git/Add.job.xml
+++ b/live/sos/git/Add.job.xml
@@ -6,6 +6,7 @@
         <![CDATA[
 cd config
 git add *.xml
+git status
         ]]>
     </script>
```

2.3.3 Ordonnancement/orchestration

L'orchestration permet de gérer les enchaînements complexes à partir de règles d'événements. Ces événements proviennent des différents ordonnanceurs ou d'applications tierces. Pour une gestion centralisée des règles, un superviseur doit être défini.

Bonne pratique : L'orchestration doit être réservée pour lier des enchaînements entre eux.

3 Conception/Pilotage

La conception et le pilotage sont étroitement liés.

Bonne pratique : Le concepteur doit développer les enchaînements en gardant toujours à l'esprit qu'ils seront exploités par un opérateur, parfois dans des conditions d'urgence et de stress.
Pour cela, il est nécessaire de respecter des règles simples.

Le fonctionnement repose sur des fichiers, correspondant à des objets particuliers, qui sont déposés dans un répertoire pour être pris en compte par le moteur.

3.1 Mise en place

L'organisation des objets est primordial car il facilite la communication avec le pilotage et permet de retrouver rapidement l'information lors du diagnostic.

Bonne pratique : mettre en place une convention de nommage simple et partagée.

3.1.1 Répertoires

Open Source JobScheduler utilise l'arborescence du « hot folder » pour organiser les traitements. Créer des sous-répertoires par application ou domaine applicatif permet d'organiser ses traitements, faciliter leur gestion et accélérer les temps de réponses sur l'interface web car ce dernier interroge le moteur en récupérant les objets d'un même répertoire.

En général on crée un répertoire par application (ex SAP) ou par domaine applicatif (ex RH regroupant tous les objets liés aux différentes applications de RH). Les jobs « techniques » des chaînes (split, merge, début, fin,…) sont laissés avec les autres jobs de la chaîne à laquelle ils appartiennent.

Répertoire « Infra » :
On crée aussi un répertoire « Infra » pour tous les jobs/chaînes destinés au maintien en état opérationnel des systèmes informatiques (sauvegardes, recopies des fichiers, purges des bases,…).

Répertoire « Commun » :
On peut créer un répertoire « Commun » qui regroupe les fichiers partagés par plusieurs domaines, c'est le cas généralement des calendriers jours ouvrés/jours fériés, et de certains jobs génériques avec des paramètres comme des envois de mails, des alertes à la supervision, ou encore certains pré ou post-processing. Il arrive aussi que la racine soit utilisée en lieu et place du répertoire « Commun ».

Cette disposition permet une gestion facile des jobs lorsqu'on change une application ou lorsqu'on externalise/internalise un domaine.

Le degré de factorisation des objets dans « Commun » dépend de la stabilité du SI : dans un système très stable, une grande partie des jobs est partagée et optimisée. Dans un système appelé à évoluer, on privilégiera la facilité d'entrée et de sortie, donc des domaines indépendants sans aucune partie commune.

Bonne pratique : Créer un répertoire correspondant à un groupe fonctionnel accessible par un groupe d'utilisateurs. Concevoir son arborescence en fonction des droits des utilisateurs facilite la gestion des permissions sur l'interface web.

3.1.2 Choisir un langage API

L'interface de programmation permettre d'étendre les fonctionnalités du produit par l'ajout de fonctions simples écrites dans le langage de votre choix parmi les suivants : Perl, VBScript, Javascript et plus généralement tout langage disposant d'un interpréteur Java (Jytton, Jrubis, etc...).

Bonne pratique : Dans le cadre de l'exploitation, nous privilégions le langage Perl qui généralement utilisé pour les scripts lorsqu'on administre des environnements hétérogènes Unix/Windows.

Nous maintenons une bibliothèque de fonctions étendues, nous vous invitons fortement à consulter le support avant de développer vos propres fonctions afin de vérifier que cette fonctionnalité ne soit déjà présente dans notre bibliothèque. Outre le gain du temps de développement, vous bénéficierez d'un résultat maintenu par nos soins.

3.1.3 Extension des fichiers XML

Le moteur utilise l'extension du fichier pour identifier le contenu, ainsi un traitement test.job.xml est immédiatement comme un objet job. Cette norme n'est pas nécessaire pour les fichiers inclus car le moteur ne les charge qu'à travers un autre objet mais nous conseillons fortement de reprendre la méthode pour identifier ces fichiers xml lorsque vous vous déplacez dans l'arborescence de répertoires.

Bonne pratique : Utiliser le nom de la première balise du fichier comme extension du nom du fichier.

Par exemple, un fichier de paramètres contient le code XML suivant :

```
<params>
      <param name="VAR" value="Value"/>
</params>
```

Le nom du fichier utilisera l'extension **.params.xml**

De même, les fichiers de jours fériés seront suffixés par **.holidays.xml**

3.1.4 Calendrier d'exécution ou calendrier d'exclusions

Tous les ordonnanceurs proposent des calendriers pour planifier les exécutions des tâches.

Bonne pratique : Autoriser tous les jours de l'année puis créer un calendrier d'exclusions pour interdire les exécutions certains jours comme les jours fériés dans un pays donné, les jours de week-end…Ainsi l'année suivante, il suffira d'adapter le calendrier d'exclusions à la nouvelle année.

3.1.5 Documenter les appels à des post ou pré-processing

L'appel à un pré ou post processing est une seule ligne dans le fichier job.xml

En lecture rapide, cette ligne est souvent omise. Il est fortement recommandé de placer un commentaire dans le fichier autour de cet appel afin de rendre plus visible cette ligne dans le job.

3.2 Traitements

Il existe deux types de traitements dont le fonctionnement est assez différent et qui correspondent donc à deux façons de piloter. Il est préférable de toujours utiliser les mêmes types d'objets pour des fonctions particulières afin que le pilotage ait toujours le même d'action.

3.2.1 Codes de sorties

Quel que soit le langage, il est indispensable de terminer un script par un code de sortie, ou exit code, afin de distinguer une bonne fin d'exécution et une erreur. Par défaut, un exit 0 correspond à une fin correcte alors qu'un code supérieur à 0 indique une erreur. Pour être compatible avec la totalité des systèmes, nous vous conseillons d'utiliser un code entre 0 et 255.

Bonne pratique : Toujours terminer ses scripts par un code de sortie.

3.2.2 Journaux

Si un texte génère du texte, il est préférable que ce texte soit renvoyé sur la sortie standard plutôt que dans un fichier car cela permet de visualiser les informations en cours d'exécution.

Bonne pratique : Ne pas stocker les informations du traitement dans un fichier afin de les rendre visible les pendant l'exécution

Il sera toujours possible de centraliser les logs par une fonction de post-exécution.

3.2.3 Traitements indépendants

Les traitements indépendants sont équivalents aux traitements en Crontab Unix ou en Task Scheduler. Ils ne sont pas adaptés au pilotage dans la mesure où l'opérateur dispose d'un minimum d'actions (Lancer ou Stopper).

Bonne pratique : Ces traitements sont à réserver aux traitements techniques.

3.2.4 Traitements ordonnés

Contrairement aux traitements indépendants, les traitements ordonnés sont destinés à être intégrés dans des scénarios. Ces scénarios permettent d'automatiser les actions de l'opérateur en proposant des chemins dégradés, des procédures de reprises ou bien encore l'attente d'une validation.

Bonne pratique : Nous conseillons de systématiser l'utilisation des traitements ordonnés pour tout ce qui est applicatif. L'opérateur n'aura plus qu'à gérer les déclenchements de scénarios.

Une autre différence importante entre les deux types est la gestion des objets par le moteur. Dans le premier cas, le moteur doit surveiller l'ensemble des traitements indépendants alors que dans le deuxième cas, il ne gère que les ordres qui seront liés à une étape à la fois, le moteur n'a donc pas besoin de prendre en compte l'ensemble des traitements mais seulement ceux liés à un ordre en cours.

3.2.5 Mode Script et non Process

Il est possible d'exécuter un traitement en mode script ou en process. Dans le premier cas, le moteur crée un shell temporaire qui sera exécuté par le moteur alors que dans le second, il est directement exécuté à l'intérieur de la machine virtuelle.

L'exécution en mode script est donc limité au système de la machine alors que le mode process est limité à la machine virtuelle ce qui lui offre moins de ressources.

3.3 Enchaînements

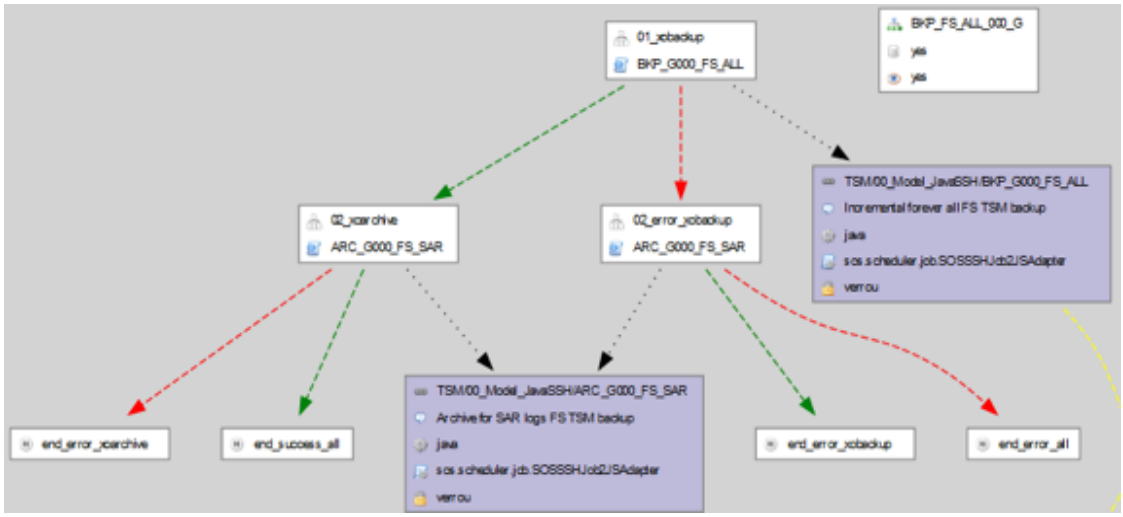
Les enchaînements doivent être considérés comme des scénarios dans lesquels on décrit des étapes. Chaque étape est liée à un traitement et sera déclenchée par un ordre en fonction du statut de ce traitement.

3.3.1 Jobs de début et de fin de chaîne

Bonne pratique : Un enchaînement doit toujours avoir des nœuds de fin, cela permet à l'ordre de sortir du scénario pour obtenir un statut de fin. Dans le cas contraire, votre ordre risque d'être considéré comme « en cours ».

Bonne pratique : Toute étape doit avoir une étape suivante et une étape d'erreur, l'ordre doit toujours avoir un chemin à suivre.

Chaque nœud contient des sorties vers deux successeurs pour obtenir un statut de fin précis, dans l'exemple ci-dessus, on aura quatre statuts différents.



Bonne pratique : Nous conseillons de placer en début ou en fin de chaîne des jobs de démarrage et de fin. Leur présence facilite la maintenance de la chaîne en la délimitant clairement. Le job de démarrage est utilisé pour recueillir ou préparer des paramètres (ceci peut aussi être fait dans l'order), le job de fin sert à détruire tous les orders sauf celui (ou ceux) qu'on souhaite conserver en sortie.

Ce job de fin est impératif à la fin d'une chaîne placée elle-même dans une chaîne de chaine, car si plusieurs orders sortent de la chaîne, la chaîne suivante sera lancée autant de fois.

3.3.2 Job ordonné et stop on error

L'option « stop on error » doit être réservée au traitement indépendant car, n'étant pas dans un processus, il doit bénéficier d'un mécanisme de reprise en cas d'erreur.

Par contre un traitement ordonné ne doit en aucun cas avoir cette option cochée. En cas d'erreur, cela aurait pour incidence du bloquer le processus alors que ce blocage devrait être réalisé en suspendant l'ordre en cours. De plus, cela bloquerait l'ensemble des processus utilisant ce traitement.

Bonne pratique : Ne jamais cocher « stop on error » pour un traitement ordonné.

3.3.3 Etapes en erreur préfixées par !

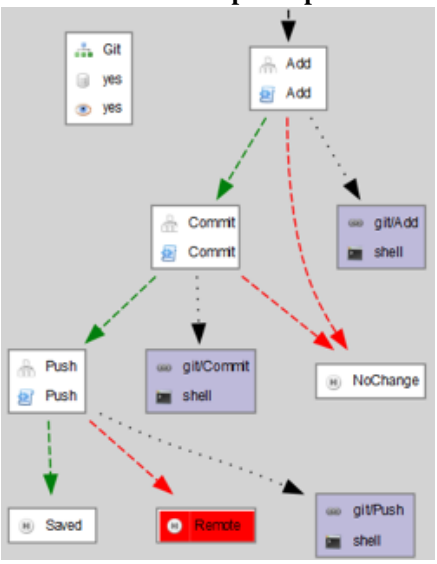
Bien qu'une étape ne puisse suivre qu'un chemin « suivant » ou « erreur », il faut relativiser cette notion d'erreur car il s'agit plutôt de l'évaluation du statut d'un traitement. Si on prend l'exemple de l'évaluation du résultat d'un requête SQL, nous pourrons avoir deux possibilités :

- le résultat contient l'information attendue et on continue le scénario
- le résultat contient une autre information et on suit un autre chemin

Dans ce cas, il ne s'agit pas d'une erreur, la requête SQL est correcte dans tous les cas, le traitement technique a bien tourné. Le scénario se poursuit sans intervention de l'opérateur.

Par contre, une requête SQL qui ne peut se connecter sur une base de données est un traitement en erreur qui demande une intervention humaine.

Bonne pratique : Pour les erreurs nécessitant une intervention de l'opérateur, préfixer le nom de l'étape par un point d'exclamation (!). Ce point sera pris en compte sur les interfaces Dashboard et Arii comme une erreur à traiter par le pilote.



L'exemple ci-contre est un envoi de données vers un serveur GIT, il est possible qu'aucun changement n'ait eu lieu, dans ce cas on peut sortir du scénario avec un statut **NoChange**.

Par contre, un problème sur la connexion du serveur distant est une erreur qui doit être remontée au pilotage.

3.3.4 Aiguillage interne

Le point d'interrogation préfixant une étape est dans la continuité de la logique précédente et indique les étapes qui ne servent qu'au routage dans le processus.
Si on reprend l'exemple de le requête SQL, cette étape ne sert qu'à choisir le chemin suivant, le chemin dégradé est alors considéré comme un simple chemin alternatif, le caractère d'erreur disparait totalement.

Bonne pratique : Préfixer une étape par un code permet de distinguer rapidement l'état du processus.

3.3.5 Etapes obligatoires et facultatives

Il est pertinent d'identifier la criticité d'une étape en cas d'erreur mais cela peut aussi être très utile dans le cadre de l'exécution car il sera alors possible de définir les étapes facultatives. Nous conseillons de mettre en place la norme suivante :

- Toute étape préfixée par + est obligatoire et ne peut être en aucun cas contournée.
- Toute étape préfixée par – est facultative et peut être sautée pour accélérer le processus.

Bonne pratique : Identifier les étapes critiques d'une chaine aide à la prise de décision en cas d'incident ou de nuit batch trop longue

3.3.6 Reprises et suspension

Certains traitements, comme le JobSynchronise qui permet de synchroniser différents enchaînements entre eux, peuvent se mettre en attente. Cette attente peut être réalisée de deux manières :

- en mode reprise, le traitement suit la configuration du setback et se replanifie au bout d'un temps défini
- en mode suspendu, le traitement apparaît bloqué et sera débloqué automatiquement

Bonne pratique : Dans le cadre d'un pilotage, il est nécessaire de réserver la suspension des ordres aux actions du pilotage et utiliser les reprises pour les actions internes du moteur. L'intérêt est d'indiquer à l'opérateur qu'il n'a pas besoin d'intervenir lors des reprises mais qu'il doit réagir sur un ordre suspendu.

Dans l'exemple ci-dessous, l'ordre est en mode reprise sur l'étape de synchronisation.

L'indication setback indique au pilote qu'il s'agit d'une action interne ne nécessitant aucune action pour l'instant.

Si l'ordre ne peut rester perpétuellement dans cet état, le concepteur a indiqué le nombre maximum de reprise pour aboutir à une nouvelle étape.

Dans cette même chaîne, l'ordre est suspendu à l'étape 002, ce mode indique au pilote qu'une action est requise pour débloquer cette situation.

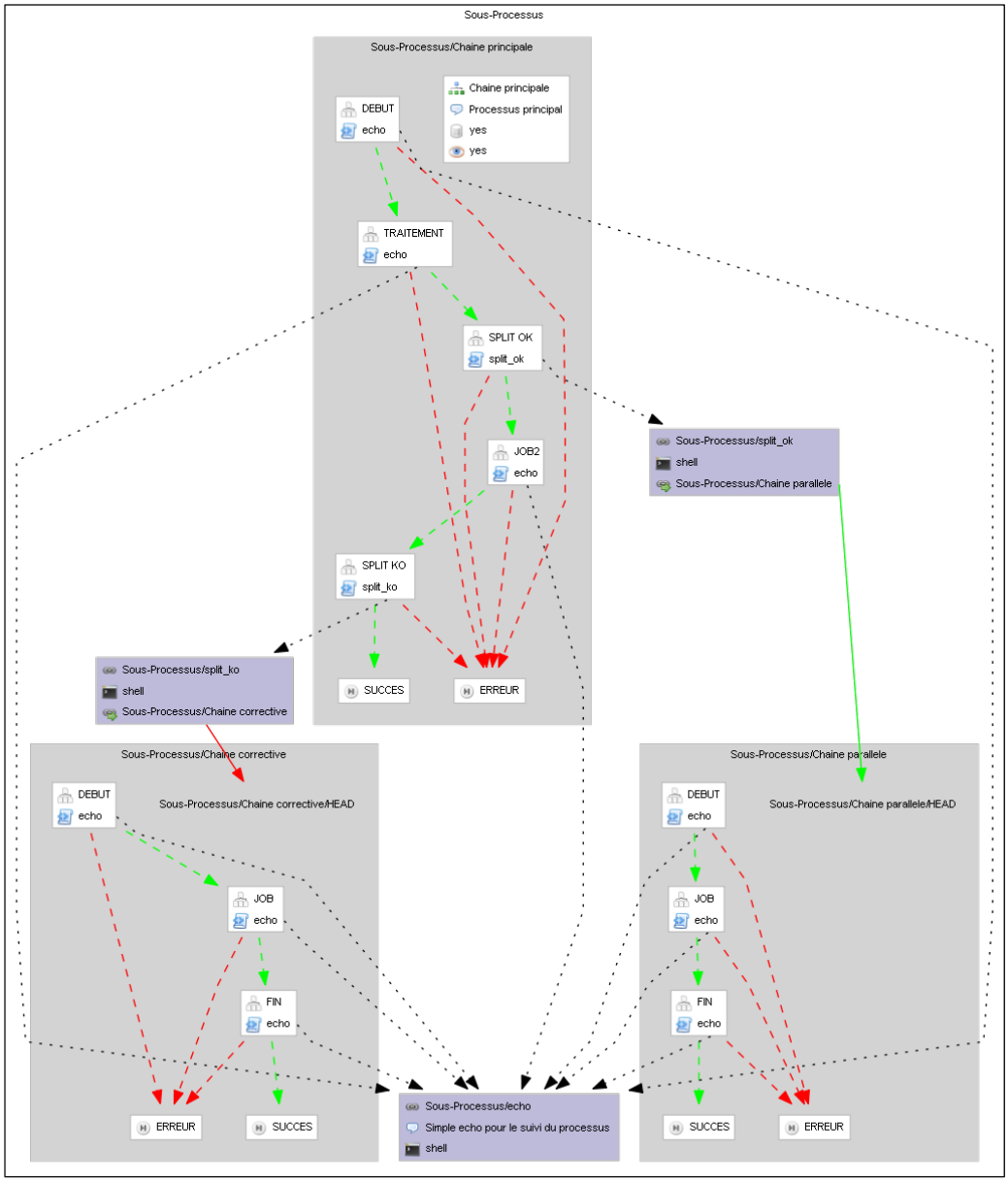
3.3.7 Processus parallèles

La parallélisation des traitements peut être réalisée de différentes manières :

- Par la fonction commande du traitement
- En utilisant le job JobSplitter fourni dans la bibliothèque JITL
- En utilisant les fonctions internes (APIs)
- A travers les règles d'orchestration

Nous conseillons la première méthode qui présente plusieurs avantages :

- Elle est simple à mettre en œuvre à travers l'éditeur de traitements
- L'exécution est gérée par le moteur interne et non à travers une classe externe
- Elle ne demande pas de connaissance en développement
- Elle ne requiert pas d'installation supplémentaire



3.4 Synchronisations

La synchronisation est le pendant de la parallélisation et permet de réunir les chaines de traitement qui n'ont pas de liens directes entre elles. Elle n'est pas directement réalisée par le moteur mais s'appuie sur des traitements techniques. Il est donc utile de nommer ces jobs d'une certaine manière afin de les distinguer des jobs applicatifs.

Bonne pratique : Distinguer les traitements applicatifs des traitements techniques

3.4.1 Traitement de synchronisation

Le nom des traitements de synchronisation doit être préfixé par un souligné (_), ce caractère est arbitraire mais le souligné est pris en compte par nos outils de visualisation graphique.

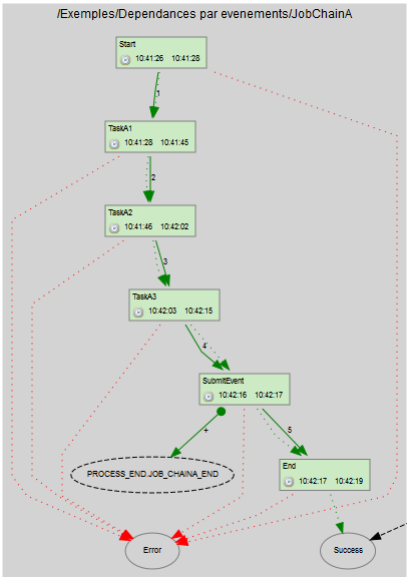
Bonne pratique : Identifier les traitements de synchronisation par un caractère souligné améliore la visibilité

3.4.2 Dépendances par évènements

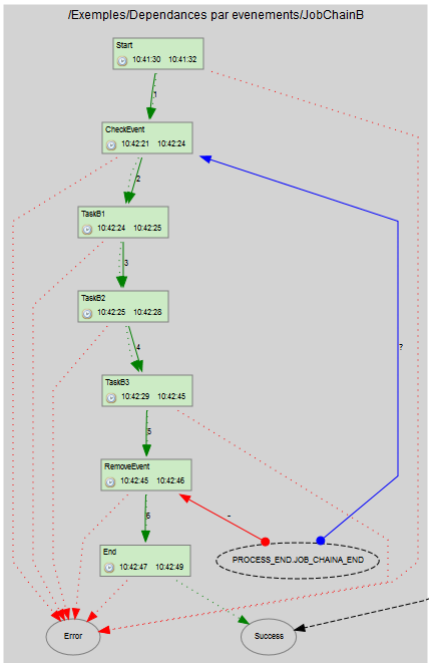
La synchronisation par évènements permet de résoudre les problèmes de dépendances complexes, par exemple en évaluant les fins de différents traitements à travers des règles ou en conditionnant les traitements de différents moteurs.
On distingue 3 phases dans l'utilisation des évènements :

- L'envoi de l'évènement dans la base de données
- La vérification de l'évènement
- La suppression de l'évènement

Pour chacune des phases, on indiquera respectivement un caractère particulier : +,_,-
Si on ajoute le caractère _ indiquant la synchronisation, nous aurons alors les noms de jobs suivants :
-+CLASS.ID envoi l'évènement référencé par la classe CLASS et l'ID
__CLASS.ID vérification de l'évènement précédent
_-CLASS.ID suppression de l'évènement



La chaine A, à gauche, contient un job qui envoie un évènement. L'envoi est symbolisé par une flèche verte et le signe +



La chaine B attend l'évènement (en bleu) et le supprime en fin de processus (en rouge).

Bonne pratique : L'utilisation des caractères _+,_- et __ permet d'identifier les traitements liés à des évènements

3.5 Ordres

Les ordres sont les déclencheurs des étapes des enchaînements, ils vont démarrer les traitements en fonction du chemin décrit dans le scénario.

3.5.1 Rendre les paramètres visibles

Bonne pratique : Les paramètres peuvent être précisés dans le traitement ou dans la création du scénario mais si vous souhaitez les rendre visible pour l'opérateur afin qu'il puisse éventuellement les modifier avant le démarrage, vous devrez indiquer ces paramètres au niveau de l'ordre.

Inversement, si vous ne souhaitez pas que l'opérateur ait accès à ces paramètres, il sera préférable de les supprimer de l'ordre.

3.5.2 Variables dans l'ordre

Si ces paramètres concernent une étape en particulier, il faut préfixer le paramètre par le nom de l'étape suivi d'une barre oblique /.

Du point de vue du pilotage, ce système permet de donner une visibilité sur l'ensemble des paramètres de l'enchaînement même s'ils sont spécifiques à une étape en particulier.

Au niveau de la conception, le système offre le moyen de concevoir des enchaînements génériques utilisant le paramétrage défini au niveau des ordres.

Bonne pratique : Utiliser les variables de l'ordre permet de voir et de modifier rapidement des paramètres internes au processus.

3.5.3 Choix des paramètres

JobScheduler offre de multiples possibilités dans l'utilisation des paramètres que l'on peut résumer comme ceci :

- Les paramètres du job peuvent être considérés comme des paramètres par défaut
- Les paramètres des processus sont propres à un processus
- Les paramètres d'étapes permettent de modifier l'état entre deux traitements successifs
- Les paramètres d'ordres permettent de définir les paramètres globaux du processus

Bonne pratique : Limiter l'utilisation des paramètres à quelques objets comme les jobs et les ordres ou mettre en place une norme permettant de retrouver la source du paramètre

3.5.4 Nom pour les chaînes de chaînes

Une « nested chain » est un scénario pour les ordres qui ne gère pas des traitements unitaires mais des ensembles de traitements. L'ordre va passer de chaîne en chaîne en indiquant systématiquement la chaîne en cours, seul le nom de la sous-chaîne est stocké en base de données.

Bonne pratique : Lorsqu'un ordre concerne une chaîne de chaines, indiquer systématiquement le nom de la chaîne globale en préfixe de l'ordre en les séparant par un point (.) comme suit : CHAINE_GLOBAL.ORDRE

3.6 Planification horaire

Bonne pratique : Un déclenchement horaire peut être directement défini dans le traitement ou dans l'ordre mais il est préférable de créer un objet à part pour en améliorer la visibilité.

3.6.1 Liste des planifications

L'onglet « Schedules » affiche les différentes définitions et leur utilisation.

Dans cet exemple, on voit que la planification lundi_18h est actuellement utilisée par un ordre.

3.6.2 Substitutions de planification

Une option intéressante pour le pilotage est la possibilité de substituer une planification par une autre pendant un laps de temps défini.

L'exemple ci-contre montre que la planification du lundi_18h remplacera celle du jeudi_19h, tous les traitements habituellement programmés le jeudi seront décalés au jeudi.

4 Verrous

4.1 Lissage des lancements

Lorsqu'une série de traitements peut démarrer à partir d'une certaine heure mais que ces traitements ne peuvent pas être exécutés en même temps, il est tentant de mettre une planification différente pour chacun de ces jobs.

Prenons le cas de sauvegardes, elles peuvent être démarrées à partir de 22h mais pour des raisons de ressources systèmes ou matérielles, il peut être contre-productif de les exécuter simultanément. Le mauvais réflexe est de considérer que chaque sauvegarde dure moins de 30minutes et qu'on peut donc planifier la première à 22h, puis la seconde à 22h30 et ainsi de suite.

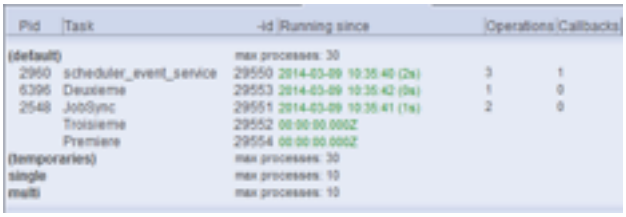
Cette vision à court terme atteindra ses limites si l'une des sauvegardes dure plus de 30 minutes ou si on a besoin d'en ajouter ou d'en enlever car il faudra alors générer manuellement ces planifications. La bonne pratique est de planifier toutes les sauvegardes à partir de 22h et de limiter les exécutions parallèles par l'utilisation de verrous.

Bonne pratique : Le lissage des exécutions doit être réalisé par une gestion des ressources et non par des planifications gérées manuellement.

4.2 File d'attente

Une file d'attente permet de soumettre un certain nombre de traitements en local ou à distance. Une file d'attente est un fichier XML, il peut être modifié à tout moment pour modifier la taille de la file ou envoyer les traitements vers une autre machine.

L'interface http embarqué propose un filtre par file d'attente, l'utilisation de cet objet peut aussi servir à regrouper des traitements d'une même machine.



Bonne pratique : Il est conseillé de toujours soumettre les traitements sur une file d'attente, cette file d'attente fait office d'alias de machine de soumission. En cas d'arrêt de la machine, ou plus globalement en cas de bascule sur un site de secours, il suffit de modifier les fichiers XML pour redéfinir l'environnement.

L'autre avantage de la file d'attente est de pouvoir la limiter le nombre de traitements concurrents.