Architecture Distribuée

Introduction

Shinken Enterprise peut être configuré pour supporter une architecture distribuée .

L'objectif de cette architecture distribuée est de réduire la charge (CPU...) et d'améliorer les temps de traitement des checks d'une approche "un seul serveur qui traite" vers une approche "plusieurs serveurs qui traitent".
La plupart des environnements simples ne nécessiteront pas la mise en place de cette architecture distribuée, mais quand vous commencez à gérer plusieurs milliers d'hôtes, cela devient crucial.

L’architecture globale

L'architecture de Shinken Enterprise selon les principes Unix : un outil, une tâche. Shinken Enterprise a une architecture où chaque partie est isolée et se connecte aux autres une interface standard HTTP .

Basé sur un back-end HTTP , cela vous permettra de construire une architecture distribuée et hautement disponible très simplement

Le moteur Shinken utilise une programmation distribuée, ce qui veut dire qu'un démon va souvent lancer des invocations de code en asynchrone sur d'autres démons, ce qui implique que la version de code , les chemins et les versions de modules doivent être identiques partout où un démon tourne.

Rôles de chaque démon Shinken Enterprise

Ceci est décrit dans la page démons .

Le load balancing automatique

Distribution des hôtes à travers les schedulers

Shinken Enterprise est capable de couper la configuration en plusieurs parties et les distribuer aux Schedulers .

Le load balancing est fait automatiquement : l'administrateur n'a pas besoin de se souvenir quel hôte est lié à tel autre pour créer les packs

La répartition est basée sur les hôtes : cela veut dire que tous les checks associés à un hôte seront dans le même Scheduler que l'hôte . Cela signifie que l'administrateur n'a pas besoin de connaître toutes les relations entre éléments comme les parents,, dépendances d'hôtes ou dépendances de checks : Shinken Enterprise est capable de lire ses relations et de rassembler tous les éléments liés dans la même partition.

Cette action se fait en 2 parties :

- création de partitions indépendantes pour les éléments
- copie des partitions pour créer N configurations pour N Schedulers

Création de partitions indépendantes

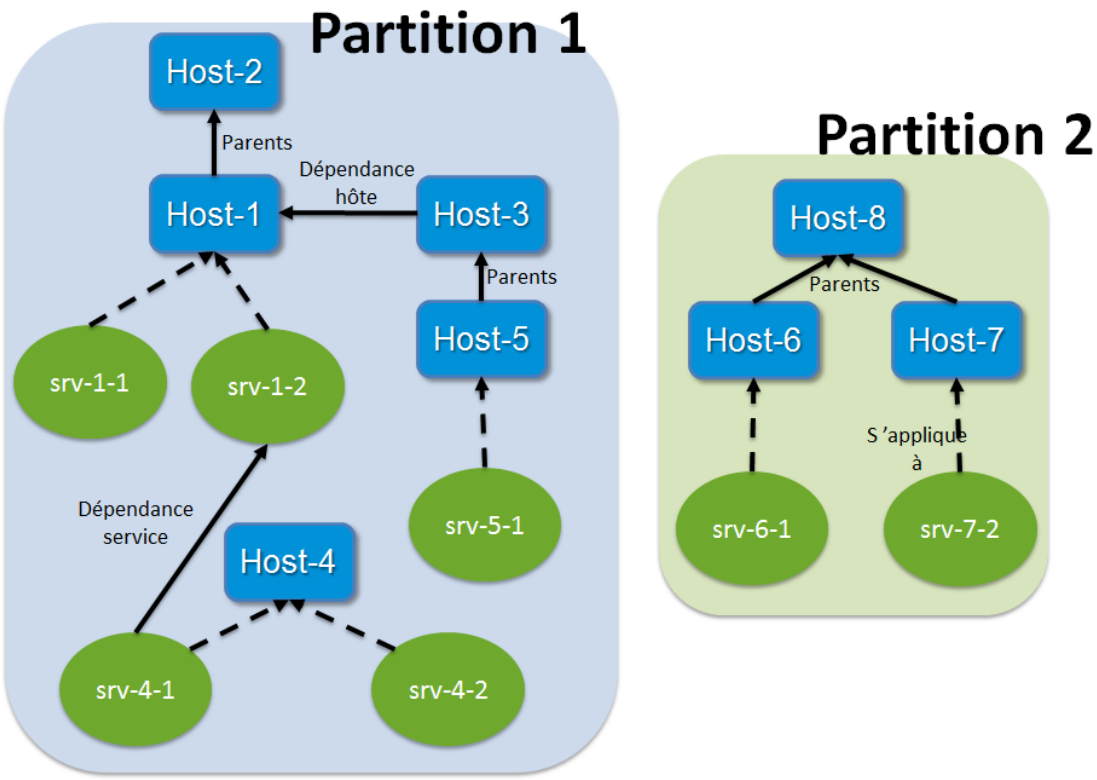
L'action de hachage se fait en se basant sur 2 éléments : les hôtes et les checks. Les checks sont liés à l'hôte donc ils seront dans la même partition.

D'autres relations sont prises en compte :

- Liaisons réseau pour un hôte (comme un serveur distant et son routeur).
- Dépendances logiques.

Shinken Enterprise regardent toutes les relations et créée un graphe avec. Un graphe est une partition de relations.

Illustration :



Dans cet exemple, nous avons 2 partitions:

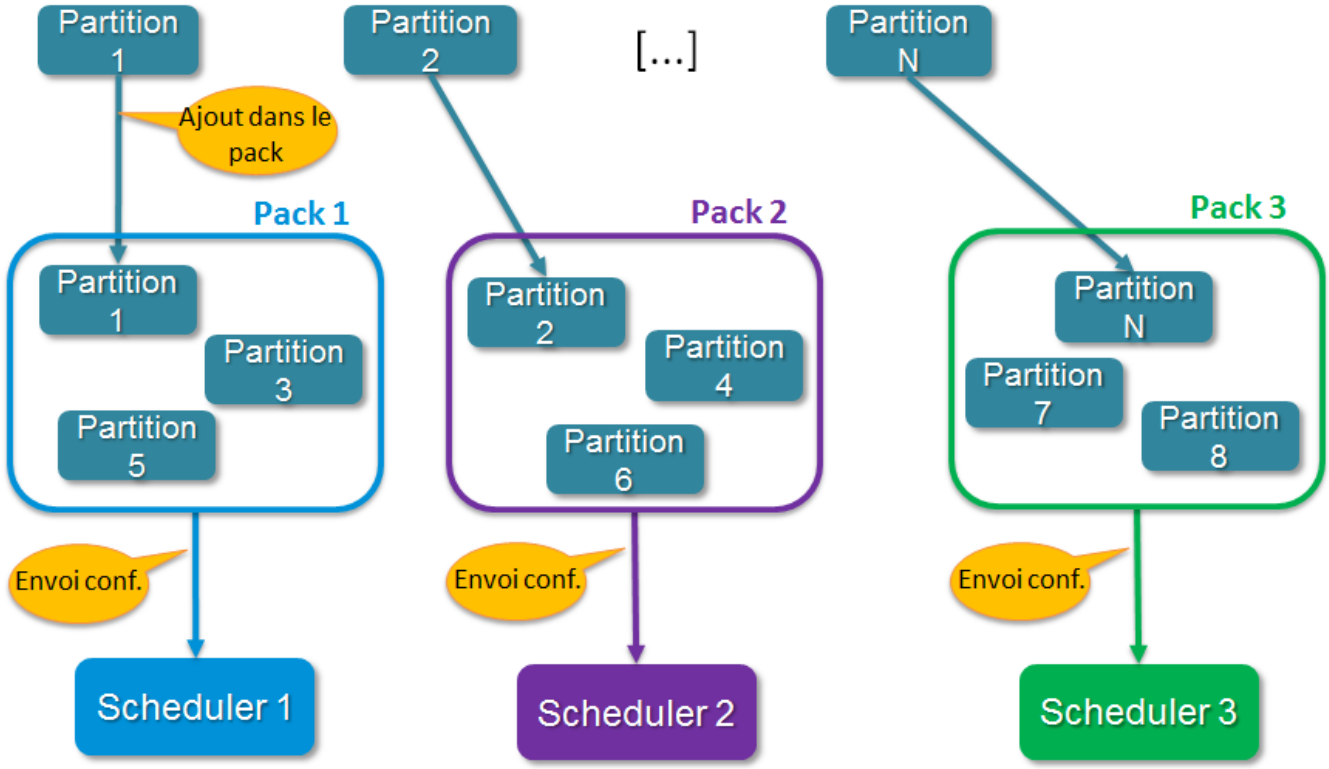
- Shard 1: Host-1 au host-5 et tous leurs checks
- Shard 2: Host-6 au Host-8 et tous leurs checks

L'aggrégation des partitions dans les schedulers

Quand toutes les partitions sont créées, l'Arbiter les agrège dans N configurations si l'administrateur a défini N Schedulers actifs (sans spare).

La répartition se fait sur un critère de poids des Schedulers : plus le poids est élevé, plus il y a de packs .

Illustration :



Envoi des configurations vers des satellites

Une fois que toutes les configurations sont créées, l'Arbiter les envoie aux N Schedulers actifs .

Un Scheduler peut commencer à lancer des checks une fois qu'il a reçu et chargé sa configuration sans avoir à attendre que TOUS les Schedulers soient prêts.

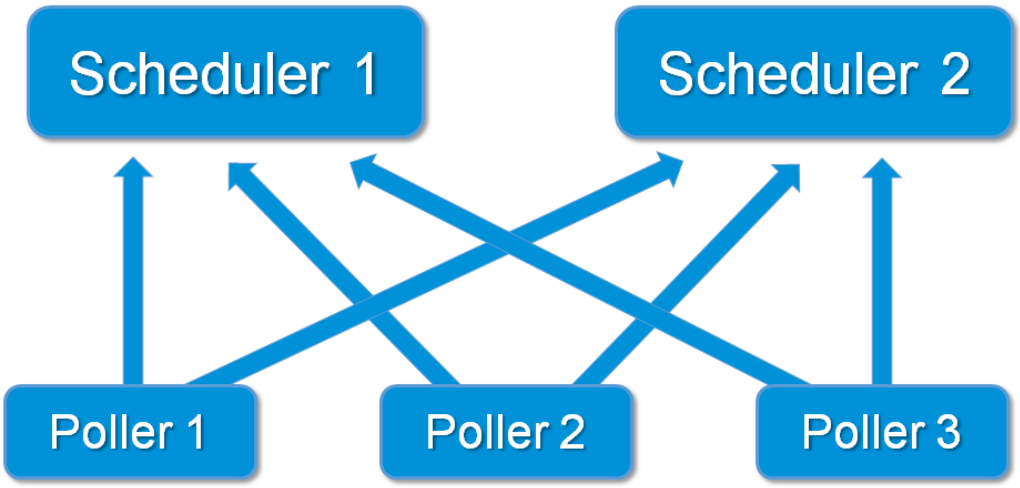
Pour des configurations plus importantes, avoir plusieurs Schedulers (même sur un seul serveur) est fortement recommandé car ils chargeront leur configuration beaucoup plus vite (nouvelle ou modification)

L'Arbiter crée également les configurations pour ses satellites (pollers, reactionners et brokers) avec les liens permettant de savoir où réaliser les tâches .

Après avoir envoyé les configurations, l'Arbiter commence à traiter les ordres (appelées commandes externes) des utilisateurs et est responsable de vérifier la disponibilité des satellites.

.

Connexions aux Pollers avec plus d'un Scheduler



La haute disponibilité

L'architecture de Shinken Enterprise est hautement disponible. Avant de rentrer dans le détail, regardons comment fonctionne le load balancing.

Quand un nœud meurt

Un serveur peut crasher, une application également. C'est pour cela que les administrateurs ont des back up : ils peuvent recharger la configuration des éléments tombés .

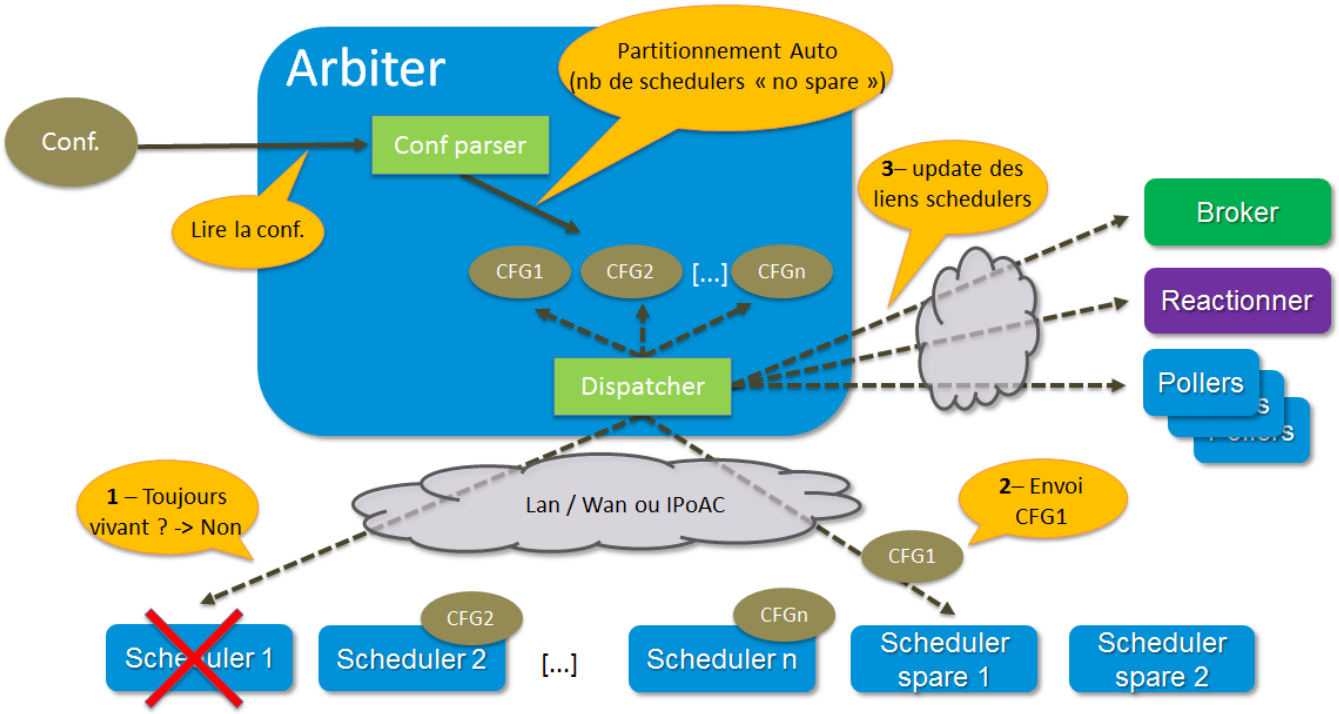
L'Arbiter vérifie régulièrement que tous les démons sont disponibles. Si un Scheduler ou un autre satellite est tombé, l'Arbiter envoit sa configuration au nœud spare défini par l'administrateur .

- Tous les satellites sont informés de ce changement, de façon à recevoir leur tâches du nouvel élément sans essayer de joindre le démon tombé .
- Si un nœud est perdu du à une coupure réseau, puis revient , l'Arbiter en prend note et lui renvoie sa configuration. Le nœud spare l'ayant remplacé redeviendra spare.

Les critères de disponibilité peuvent être modifiés dans les paramètres par défaut lorsqu'il s'agit d'une grosse installation car les Schedulers et Brokers peuvent être surchargés et du coup avoir des temps de réponse de disponibilité plus longs.

Les délais sont volontairement très courts pour de petites installations (Voir paramètres de configuration des Démons pour plus d'information).

Illustration :



Distribution par Commande Externe

L'administrateur doit envoyer des ordres aux Schedulers (comme par exemple un nouveau statut pour un check passif).

Dans Shinken Enterprise, l'administrateur envoie uniquement l'ordre à l'Arbiter, c'est tout. Les commandes externes sont de 2 types :

- commandes qui concernent tous les Schedulers.
- commandes qui sont spécifiques à un seul élément (hôte/check).

Pour chaque commande, Shinken Enterprise détecte si c'est global ou particulier:

- si global, il envoie les ordres à tous les Schedulers.
- Si particulier, il détecte quel Scheduler gère l'élément concerné par la commande (hôte/check) et envoie l'ordre au bon Scheduler.

Dès réception de l'ordre par les Schedulers, il est appliqué.

Différents types de Pollers: poller_tag

L'architecture de Shinken Enterprise est très pratique quand on utilise le même type de Poller pour tous les checks. Mais il peut également être nécessaire d'avoir plusieurs types de Pollers, comme par exemple un GNU/Linux et un Windows.Nous avons déjà vu que tous les Pollers communiquent avec tous les Schedulers. En fait, les Pollers peuvent être caractérisés ("tag") afin qu'ils n'exécutent que certains checks définis.

C'est très utile quand un utilisateur a des hôtes dans le même Scheduler (comme les dépendances) mais nécessite de lancer des checks depuis un Poller dédié.

Ces checks peuvent être caractérisés à 3 niveaux :

- Hôte
- Check
- Commande

Le paramètre pour qualifier une commande, un hôte ou un check est le "poller_tag".

L'existence du paramètre est déterminé dans l'ordre de cascade suivant:

- On regarde d'abord sur la commande
- Puis le check
- Puis sur l'hote.

Les pollers peuvent être identifiés par plusieurs poller_tags. Si ils ont des tags, ils ne prendront que les checks qui correspondent à ce tag.
Pour avoir les checks caractérisés spécifiquement par une liste de poller_tag et ceux non défini, il faut juste rajouter explicitement le tag "None". Ceci permet d'exclure certains tags sans avoir à définir une liste exhaustive.

Cas d'usage

Cette fonctionnalité est très utile pour une DMZ.

Dans un 1er cas, il peut être utile d'avoir une boîte Windows dans un domaine avec un Poller tournant dans un compte domaine. Si ce Poller lance les requêtes WMI, il est ainsi aisé de mettre en place une supervision windows.

Le 2ème cas de figure est assez classique : quand vous avez une DMZ, vous devez avoir un Poller dédié qui est DANS la DMZ et qui renvoie les résultats des checks à un Scheduler via le LAN. Ainsi, vous pouvez avoir des dépendances entre des hôtes en DMZ et des hôtes en LAN, et être certains que les checks seront réalisés à l'intérieur de l'espace sécurisé grâce au Poller dédié..

Différents types de Reactionners: reactionner_tag

D'une manière totalement symétrique aux pollers, les Reactionners peuvent être caractérisés avec un ou plusieurs "reactionner_tags". Leur logique est complètement identique au paramètre poller_tags, mais elle permet de distribuer les exécutions dans les reactionners plutôt que les pollers.

Ce paramètre peux être caractérisés à 3 niveaux :

- Hôte
- Check
- Commande

Là encore, l'existence du paramètre est déterminé dans l'ordre de cascade suivant:

- On regarde d'abord sur la commande
- Puis le check
- Puis sur l'hote.

Les reactionners peuvent être identifiés par plusieurs reactionner_tags. Si ils ont des tags, ils ne prendront que les commandes qui correspondent à ce tag.
Pour avoir les commandes caractérisées spécifiquement par une liste de reactionner_tag et ceux non défini, il faut juste rajouter explicitement le tag "None". Ceci permet d'exclure certains tags sans avoir à définir une liste exaustive.

Architecture avancée: les Royaumes

L'architecture de Shinken Enterprise permet à l'administrateur d'avoir un point unique de gestion pour tous les Schedulers, Pollers, Reactionners et Brokers. Les hôtes sont répartis avec leur propres checks vers les Schedulers et ses satellites récupèrent directement leurs tâches. Jusque là tout se passe bien.

Ou presque bien... Imaginons un administrateur qui a une infrastructure répartie dans le monde entier. En version simple avec Shinken Enterprise, l'administrateur peut installer plusieurs couples Scheduler/Poller dans différentes zones géographiques (par exemple Europe et Asie), mais il ne peut pas forcer les checks "Asie" à être réalisés par le Scheduler "Asie". Essayer de lancer un check en Asie depuis le serveur Europe peut également se révéler très lent, les hôtes étant répartis sur tous les Schedulers et leurs satellites, donc il n'y a aucune garantie que les hôtes locaux seront checkés par leurs serveurs locaux.

Shinken Enterprise permet alors de gérer la notion de zones géographiques ou organisations séparées.

Le terme utilisé pour gérer cette notion de séparation dans l'outil est le **Royaume**.

Les Royaumes en quelques mots

Un royaume est un ensemble de ressources (scheduler, poller, reactionner et broker) auxquelless sont rattachés des hôtes ou des groupes d'hôtes. Un hôte ou un groupe d'hôte ne peut être rattaché qu'à un seul royaume. Toutes les dépendances ou parents des hôtes doivent également être dans le même royaume. Un royaume peut être défini par défaut, les hôtes non affectés lui seront automatiquement rattachés. Dans un royaume, les pollers, reactionners et brokers ne recevront les tâches que des schedulers du même royaume.

Les Royaumes ne sont pas des poller_tags!

Soyez sûrs de bien comprendre quand utiliser les Royaumes et quand utiliser les poller_tags .

- Les royaumes sont utilisés pour séparer les schedulers
- les poller_tags sont utilisés pour utiliser les pollers séparés

Dans certains cas, la fonctionnalité poller_tag peut aussi être réalisée en utilisant les royaumes. La vraie question à se poser est : est que le poller_tag est suffisant, ou est-ce nécessaire de totalement séparer les environnements au niveau du scheduler à travers la notion de royaume. Dans un royaume, les schedulers ne communiquent qu'avec les autres schedulers du même royaume, et jamais avec d'autres.

- si vous avez juste besoin d'un poller dans une DMZ, utiliser le poller_tag.
- Si vous avez besoin d'un scheduler/poller dans un LAN client, utiliser les royaumes.

Sous royaumes

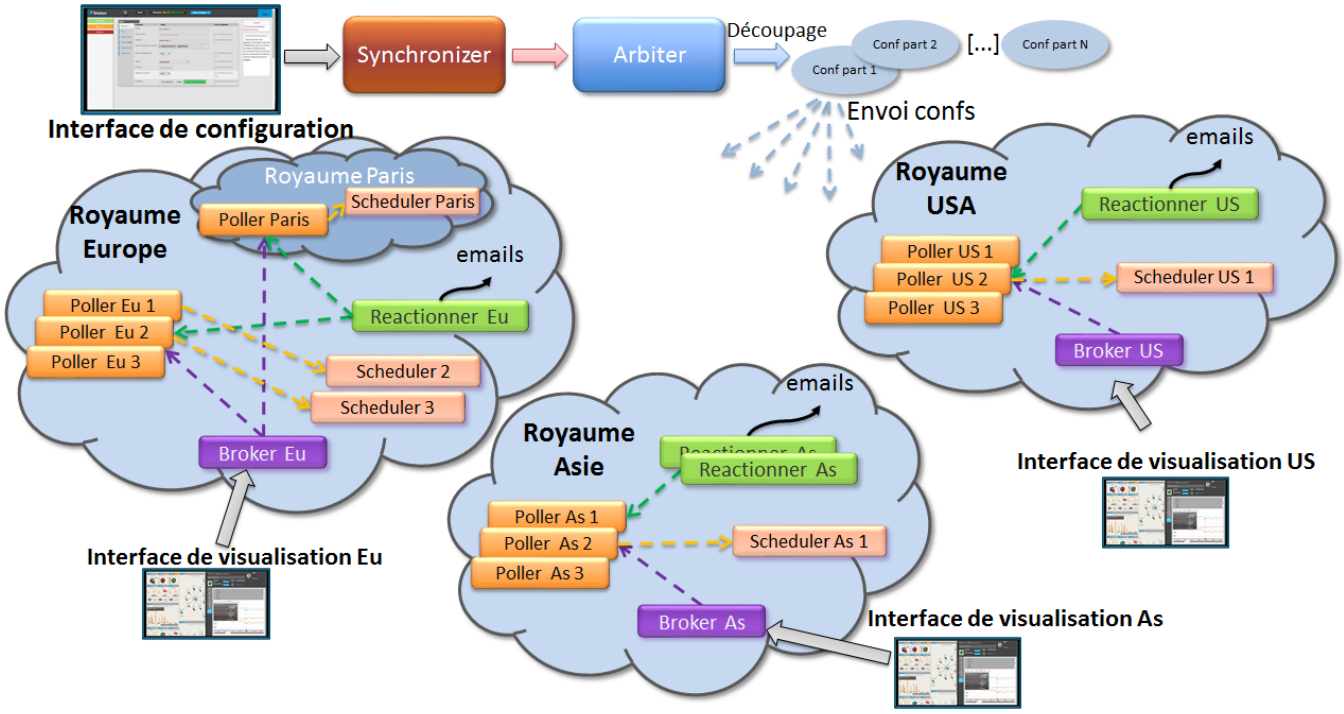
Un royaume peut contenir un autre royaume (sous royaume). Cela ne change rien pour les schedulers : ils ne sont responsables que des hôtes de leur royaume/sous royaume. L'arborescence royaumes est utile pour les satellites comme les reactionners ou brokers: ils peuvent recevoir les tâches des schedulers de leur royaume mais aussi des schedulers des sous royaumes. Les pollers peuvent également recevoir les tâches des sous royaumes, mais c'est moins intéressant donc désactivé par défaut.

ATTENTION : avoir plusieurs brokers dans un royaume n'est pas une bonne idée. Même chose pour l'arbiter, il n'y a qu'un seul arbiter et une seule configuration quelque soit le nombre de royaumes. .

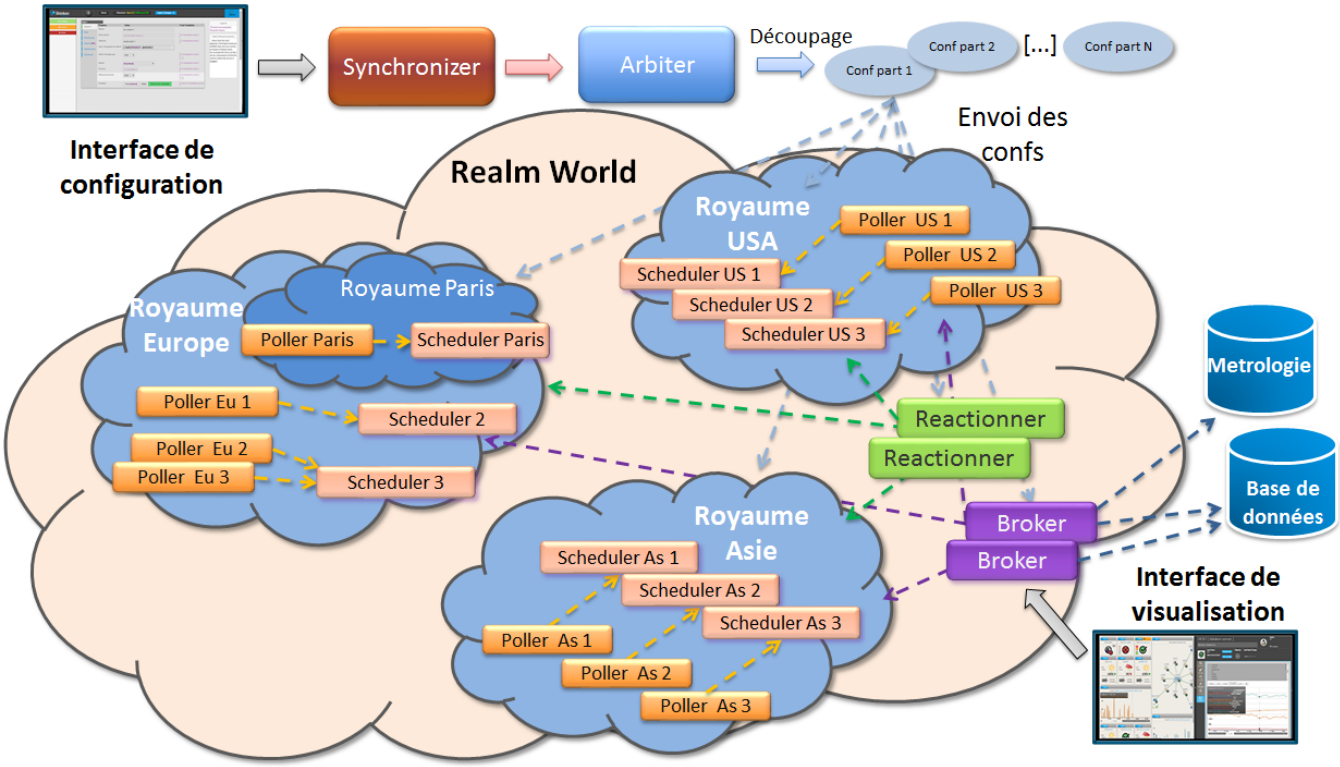
Exemple d'utilisation des royaumes

Regardons 2 environnements distribués. Dans le 1er cas, l'administrateur veut des démons totalement distincts. Dans le second cas, il veut simplement avoir des schedulers/pollers distincts, tout en gardant un seul endroit pour envoyer les notifications (reactionners) et un seul endroit pour l'export de base (broker).

Royaumes distincts :



Usage plus classique: le royaume global (avec reactionner/broker) et les sous-royaumes (avec schedulers/pollers):



Les satellites peuvent être rattachés également aux royaumes et sous-royaumes.Ce n'est qu'un paramètre dans la configuration de l'élément.