Une interface de Visualisation dédiée à un sous-royaume

Introduction

Une utilisation fréquemment rencontrée lors de la mise en place d'une architecture Shinken distribuée contenant plusieurs royaumes est la gestion de plusieurs services/clients.

Supposons qu'on se trouve dans le scénario suivant:

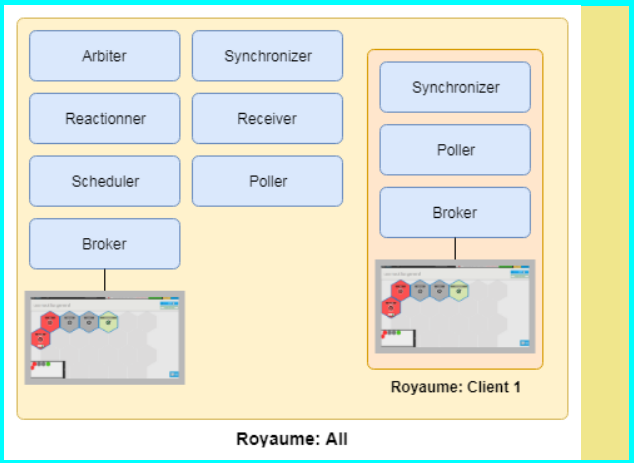
- La plateforme de supervision est mise en place par une équipe dédiée à la supervision
- L'équipe de supervision dispose d'une interface de Visualisation permettant de visualiser l'ensemble des hôtes présents dans la configuration
- Les hôtes supervisés correspondent à des machines/service dédiées à certains clients
- L'équipe de supervision veut pouvoir mettre à disposition une interface de Visualisation à chaque client qui lui permet de visualiser ses hôtes (mais pas ceux des autres clients)

Une solution dans ce cas de figure est de créer un royaume dans lequel est positionné un Broker.

Architecture mise en place

Dans cette page de documentation, on a pour objectif de mettre en place l'architecture suivante:

- L'architecture comporte d'abord un royaume principal (*All*) qui supervise un certain nombre d'éléments
- Un sous royaume *Client 1* s'occupe de superviser les éléments concernant le client 1.
- L'interface de Visualisation sur le royaume permet de visualiser l'ensemble des hôtes, y compris ceux du royaume *Client 1*.
- L'interface de Visualisation du royaume *Client 1* sera mise à disposition au client et ne permet de visualiser que les éléments supervisés par le royaume *Client 1*.



Configuration des démons

La première étape pour la mise en place de cette architecture est la mise en place des démons.

On suppose dans notre exemple que les démons du royaume *All* sont hébergés sur une machine *server_1* et les démons du royaume *Client 1* sont hébergés sur une machine *server_2*.

La configuration se fera alors en majeure partie sur *server_1*, puisque c'est la on est situé le démon Arbiter qui gère la configuration des démons.

Création des royaumes

Sur *server_1*, on commence alors par créer le royaume *Client 1*. Pour cela, on crée le fichier **/etc/shinken/realms/client1.cfg** qui aura le contenu suivant:

```
define realm {
    # Realm name. Must be unique
    realm_name          Client 1

    # default:  0 = this realm is not the default one
    #           1 = this realm is the default, so an element or a daemon is missing a
    #               realm property, then this one will be used
    default             0

    # List of sub-realms for this realm
    #realm_members
}
```

Le royaume *Client 1* est un sous-royaume du royaume *All*. Il faut pour qu'il soit considéré comme un sous royaume modifier la configuration du royaume *All* pour le spécifier, comme dans l'exemple suivant :

/etc/shinken/realms/all.cfg

```
define realm {
    # Realm name. Must be unique
    realm_name          All

    # default:  0 = this realm is not the default one
    #           1 = this realm is the default, so an element or a daemon is missing a
    #               realm property, then this one will be used
    default             1

    # List of sub-realms for this realm
    realm_members Client 1
}
```

Création des démons du royaume *Client 1*

Une fois le royaume *Client 1* créé, il faut ensuite créer les démons qui font partie de ce royaume.

Cette création des démons se fait en 3 temps.

- On commence d'abord par créer les fichiers .ini des démons sur *server_2*

  Sur server_2

  ```
  shinken-daemons-create --ini --type=scheduler --port=7768 --name=scheduler-client-1
  shinken-daemons-create --ini --type=poller --port=7771 --name=poller-client-1
  shinken-daemons-create --ini --type=broker --port=7772--name=broker-client-1
  ```

- On créé ensuite les fichiers de configuration sur l'Arbiter (*server_1*)

  Sur server_2

  ```
  shinken-daemons-create --cfg --type=scheduler --port=7768 --name=scheduler-client-1 --
  address=<adresse_server_2>
  shinken-daemons-create --cfg --type=poller--port=7771 --name=poller-client-1 --
  address=<adresse_server_2>
  shinken-daemons-create --cfg --type=broker--port=7772 --name=broker-client-1 --
  address=<adresse_server_2>
  ```

- On modifie la propriété "*realms*" des fichiers de configuration de ces démons pour les mettre dans le royaume *Client 1* (**/etc/shinken /schedulers/scheduler-client-1.cfg, /etc/shinken/brokers/broker-client-1.cfg, /etc/shinken/pollers/poller-client-1.cfg**).
  Par exemple, pour le Scheduler:

/etc/shinken/schedulers/scheduler-client-1.cfg

```
define scheduler {
    #======== Daemon name and address =========
    # Daemon name. Must be unique
    scheduler_name            scheduler-client-1

    # IP/fqdn of this daemon (note: you MUST change it by the real ip/fqdn of this server)
    address                   <adresse_server_2>


    [... Contenu coupé ...]



    #======== Realm and architecture settings =========
    # Realm to set this daemon into
    realm                     Client 1

    # In NATted environments, you declare each satellite ip[:port] as seen by
    # *this* scheduler (if port not set, the port declared by satellite itself
    # is used)
    #satellitemap             poller-1=1.2.3.4:1772, reactionner-1=1.2.3.5:1773



    #======== Enable or not this daemon =========
    # 1 = is enabled, 0 = is disabled
    enabled                   1

}
```

Gestion des métriques

La configuration des métriques récupérées dans chaque royaume peut se faire de 2 manières différentes:

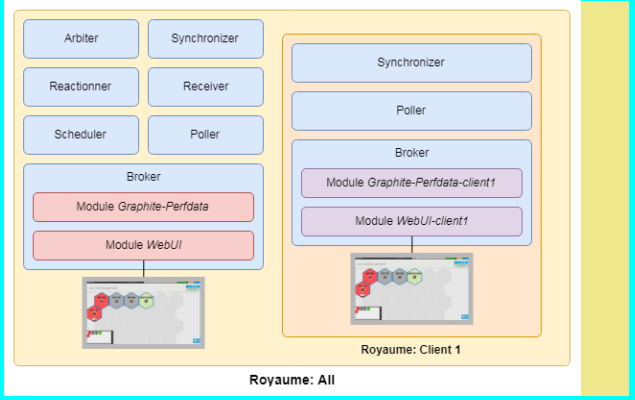
- Toutes les métriques remontées par les hôtes et les checks (tous royaumes confondus) sont sauvegardées et lues sur *server_1*:
  - ➕ **Simplicité de configuration et de maintenance**: Tous les royaumes lisent sur le même serveur, ce qui permet d'avoir un seul module *Graphite-perfdata* utilisé par tous les Brokers
  - ➖ **Performances**: Tous les Brokers font leur requêtes de métriques sur une seule machine centrale. Cette machine peut devenir surchargée lorsqu'on a beaucoup d'hôtes/checks et/ou de sous-royaumes différents
- Les métriques sont sauvegardées et lues de manière indépendante par chaque royaume:
  - ➕ **Montée en charge**: Puisque chaque royaume gère ses propres métriques, l'ajout de nouveaux sous-royaume n'influe pas sur les performances de autres royaumes
  - ➖ **Configuration un peu plus complexe**: Pour profiter de cette architecture, chaque royaume doit être configuré pour gérer ses propres métriques

Dans cette documentation, on détaille la mise en place d'une configuration permettant à chaque royaume de gérer ses propres métriques. Cette procédure de configuration a pour l'avantage d'être utilisable dans un nombre plus élevé de cas de figure.

La configuration mise en place dans la suite de cette procédure est celle représentée sur le schéma ci-contre.

La configuration d'un Broker au niveau des métriques s'organise de la façon suivante:

- **Lecture des métriques**: Le Broker a un module *WebUI*, qui en plus des paramètres de l'interface de Visualisation, définit quel serveur Graphite est utilisé pour la lecture des métriques.
- **Ecriture des métriques**: Le module *Graphite-Perfdata* du Broker permet de définir ou seront écrites les métriques des éléments reçus par ce Broker.



Dans le schéma ci-contre, chaque Broker a un module WebUI et un module Graphite différent pour permettre de dissocier la configuration des 2 Brokers au niveau des métriques.

Création des modules *WebUI-client1* et *Graphite-Perfdata-client1*

Pour mettre en place la configuration décrite dans le schéma précédent, on commence par créer les modules *WebUI-client1* et *Graphite-Perfdata-client1*. La manière la plus simple de créer ces modules est de dupliquer les modules existants *WebUI* et *Graphite-Perfdata*.

Cette duplication s'effectue en copiant les fichiers de configuration et en renommant les modules:

- Module *WebUI-client1*
  - Copie du module

```
cp /etc/shinken/modules/webui.cfg /etc/shinken/modules/webui-client1.cfg
```

  - Renommage du module

/etc/shinken/modules/webui-client1.cfg

```
define module {
    #======== Module identity =========
    # Module name. Must be unique
    module_name              WebUI-client1

    # Module type (to load module code). Do not edit.
    module_type              webui


[... Suite du fichier ...]


}
```

- Module *Graphite-Perfdata-client1*
  - Copie du module

```
cp /etc/shinken/modules/graphite.cfg /etc/shinken/modules/graphite-client1.cfg
```

  - Renommage du module

/etc/shinken/modules/graphite-client1.cfg

```
define module {

    #======== Module identity =========
    # Module name. Must be unique
    module_name                Graphite-Perfdata-client1

    # Module type (to load module code). Do not edit.
    module_type                graphite_perfdata



[... Suite du fichier ...]


}
```

Copie de fichiers et SEUUID

Les fichiers de configuration des démons et modules livrés par défaut contiennent un bloc SEUUID permettant à Shinken d'identifier les éléments livrés par défaut. Lorsqu'on copie ces fichiers dans le but de créer des nouveaux modules, il faut supprimer ce bloc pour éviter les avertissements dans l'interface de Configuration.

Les blocs à supprimer dans les fichiers de configuration des modules créés ont le format suivant:

```
# Shinken Enterprise. Lines added by import core. Do not remove it, it's used by Shinken Enterprise
to update your objects if you re-import them.
    _SE_UUID            core-module-2e0f08c45adc11e59a90080027f08538
    _SE_UUID_HASH       99215795c7e21b658e7c34989e872fcc
# End of Shinken Enterprise part
```

Une fois les modules créés, on les attache sur le Broker du royaume "*Client 1*":

/etc/shinken/brokers/broker-client-1.cfg

```
define broker {

    #======== Daemon name and address =========
    # Daemon name. Must be unique
    broker_name               broker-client1

    # IP/fqdn of this daemon (note: you MUST change it by the real ip/fqdn of this server)
    address                   client1

    [... Contenu caché ...]


    #======== Modules to enable for this daemon =========
    # Available:
    # - Simple-log            : save all logs into a common file
    # - WebUI                 : visualisation interface
    # - Graphite-Perfdata     : save all metrics into a graphite database
    # - sla                   : save sla into a database
    # - Livestatus            : TCP API to query element state, used by nagios external tools like NagVis or
Thruk
    modules                   Simple-log, WebUI-client1, Graphite-Perfdata-client1, sla, Livestatus


        [... Suite du fichier ...]
}
```

Configuration du royaume *All*

Pour les métriques, on veut faire en sorte que toutes les métriques soient lues sur *server_1*. Pour cela, on modifie la configuration du module WebUI attaché sur le Broker du royaume *All* comme suivant:

```
define module
{

    #======== Module identity
==========

    # Module name. Must be unique
    module_name               WebUI

    # Module type (to load module code). Do not edit.
    module_type               webui




    [... Contenu caché ...]



    #======== Metrology access
==========
    # Multi-realm graphite parameter
    graphite_backends›        *:<adresse_server_1>


    #======== Extended configuration ========
    [OVERLOAD_FROM]            /etc/shinken/_default/daemons/brokers/modules/webui.cfg
    [OVERLOAD_FROM]            /etc/shinken-user/configuration/daemons/brokers/modules/webui
/webui_cfg_overload.cfg
}
```

La paramètre modifié est "*graphite_backends*". La valeur spécifiée dans l'exemple ci-dessus permet de dire que les métriques de tous les royaumes (*) seront lues sur la machine *server_1*.

On modifie ensuite le module Graphite-Perfdata pour dire au Broker du royaume All d'écrire l'ensemble de ses métriques sur la machine *server_1*.

Pour cela, on modifie le fichier **/etc/shinken/modules/graphite.cfg**:

/etc/shinken/modules/graphite.cfg

```
define module {
    #======== Module identity ==========
    # Module name. Must be unique
    module_name                Graphite-Perfdata

    # Module type (to load module code). Do not edit.
    module_type                graphite_perfdata



    #======== Graphite address ==========
    # host:  graphite server address (ip or fqdn)
    host                       <adresse_server_1>

    # port:  tcp port of the graphite server
    port                       2003



    #======== realm filtering ==========
    # By default, this module will save metrics from all realm and subrealms of the broker realm.
    # You can use realm_store_only to save olly the realm you want into the graphite server
    #realm_store_only           Realm1, Realm2, Realm3
}
```

Configuration du royaume *Client 1*

Pour le royaume *Client 1*, on veut écrire et lire les métriques sur la machine *server_2*. On modifie alors les réglages des modules *WebUI-client1* et *Graphite-Perfdata-client1* sur le même modèle que pour le royaume *All*:

- On dit au module *WebUI-client1* de lire l'ensemble des métriques du royaume *Client 1* sur la machine *server_2*

/etc/shinken/modules/webui-client1.cfg

```
define module
{


    #======== Module identity
=========

    # Module name. Must be unique
    module_name              WebUI-client1

    # Module type (to load module code). Do not edit.
    module_type              webui




    [... Contenu caché ...]



    #======== Metrology access
=========
    # Multi-realm graphite parameter
    graphite_backends›       Client 1:<adresse_server_2>


    #======== Extended configuration ========
    [OVERLOAD_FROM]           /etc/shinken/_default/daemons/brokers/modules/webui.cfg
    [OVERLOAD_FROM]           /etc/shinken-user/configuration/daemons/brokers/modules/webui
/webui_cfg_overload.cfg
}
```

- **/etc/shinken/modules/graphite-client1.cfg**

```
define module {
    #======== Module identity =========
    # Module name. Must be unique
    module_name                Graphite-Perfdata-client1

    # Module type (to load module code). Do not edit.
    module_type                graphite_perfdata



    #======== Graphite address =========
    # host:  graphite server address (ip or fqdn)
    host                       <adresse_server_2>

    # port:  tcp port of the graphite server
    port                       2003



    #======== realm filtering =========
    # By default, this module will save metrics from all realm and subrealms of the broker realm.
    # You can use realm_store_only to save olly the realm you want into the graphite server
    #realm_store_only           Realm1, Realm2, Realm3
}
```

Gestion des SLA

Pour les SLA, la configuration par défaut du module SLA permet de sauvegarder les SLA de chaque royaume sur la même machine que le Broker. On peut donc conserver la configuration par défaut des Brokers et du module SLA pour obtenir le même comportement que celui configuré précédemment pour les métriques.

De plus, le volume des données de SLA est bien moins important que celui des données de métrologie, ce qui permet de s'affranchir de la configuration séparée par royaumes pour la SLA pour la grande majorité des installations Shinken Entreprise.