Editer un check appliqué à un modèle d'hôte

Préambule

Shinken est un ensemble de démons travaillant de manière autonome pour collecter les statuts des équipements supervisés. Mais dans certain cas, Shinken ne peut pas accéder aux équipements a superviser.

Le receiver permet alors de recevoir des messages de l'extérieur et de les transmettre aux démons concernés.

Mais comme la source de message peut-être différente, il est nécessaire d'ajouter des modules au receiver qui traduira les messages reçus dans un format que Shinken comprendra.

Principe de fonctionnement des modules de receiver d'écoute avec workers

Le module receiver-module-generic-endpoint est un module qui est utile pour servir de base au développement de vos modules de Receiver.

Il y a plusieurs manières de récolter des informations du monde extérieur:

- Les recevoir de n'importe ou
- Se connecter à une source pour les recevoir (ex: bus de données)
- allez chercher les données
- ...

Nous avons donc fait évoluer l'architecture du Receiver et des modules accrochés au Receiver pour qu'ils puissent recevoir des informations des hôtes/clusters ainsi que leur checks associés à gérer.

- Si vous avez besoin d'un identifiant pour vous connecter sur une source de données, il est intéressant de le définir en tant que donnée sur l'hôte dans le Synchroniser et de le transmettre jusqu'au module de Receiver pour son utilisation.
- Ce point se définit au niveau de la configuration du Receiver.
    - Il est aussi possible de filtrer les hôtes reçus par le Receiver (pour avoir un inventaire plus petit).
    - Enfin on peut choisir quel ensemble de données sera disponible dans cet inventaire (on choisit les modèles dont les données vont être véhiculées).

Nous avons aussi permis que les traitements faits par les modules de Receiver puissent se répartir sur 1 ou plusieurs processus (des workers).

- Sur 1 worker:
    - Dans ce cas, tous les hôtes sont présents dans le module et ce dernier peut traiter n'importe quel message de l'extérieur associé à un hôte en étant sûr qu'il sera traité.

- Avec plusieurs workers:
    - Ceci correspond à une attente particulière car dans ce cas, chaque worker n'aura qu'une partie des hôtes sélectionnés.
    - Les éléments sont divisés à part égale entre les workers.
    - Ex d'utilisation: si vous voulez que votre module se connecte à un bus de données (sens de connexion worker => bus) et que la charge de récupération soit réparti entre plusieurs processus.

Mise en place du module d'exemple

Etape 1: Choisir le nom de votre module

Il faut un nom pour votre module. Le nom doit respecter:

- uniquement des caractères ascii
- en terme de caractères spéciaux, seuls **-** ou _ sont supportés
- pas d'espaces
- tout en lower case

Dans cette documentation on le nommera **NOMduMODULE**

> Nous suivons maintenant pour tout nouveau module la convention suivante:
>
> **NOMduDEMON**_module_**NOMduMODULE**

Etape 2: Copier le module dans son répertoire

Le module est installé via deux actions:

- on met son code en place dans **/var/lib/shinken/modules**
- on met sa configuration en place dans **/etc/shinken/modules**

Il faut donc prendre dans le du package d'exemple et :

- copier le répertoire module en tant que **/var/lib/shinken/modules/NOMduMODULE**

- copier le fichier **Receiver-module-generic-endpoint.cfg** en tant que **/etc/shinken/modules/NOMduMODULE.cfg**

Etape 3: Renommer le contenu du module et sa configuration avec notre nouveau nom

> Il est important de renommer TOUTES les parties du module, si votre module a encore des classes ayant le même nom que le module generic, alors il y aura des problème d'import du code par le daemon et le résultat sera incorrect.

Dans le code livré par défaut, le nom du module est **MODULE_CODE_NAME**.

- Éditez **/var/lib/shinken/modules/NOMduMODULE/module.py** en changeant toutes les occurrences de **ReceiverGenericEndpoint** en **NOMduMODULE**.
- Éditez **/etc/shinken/modules/NOMduMODULE.cfg** en changeant toutes les occurrences de **Receiver_module_generic_endpoint** en **NOMduMODULE**

Déclarez le module sur votre Receiver

Pour que le module s'active il faut:

- redémarrer le Receiver afin qu'il charge le nouveau code (module.py)
- rajouter votre module dans votre Receiver (typiquement **/etc/shinken/Receivers/Receiver-master.cfg**)

Important: Si vous avez besoin de l'inventaire des hôtes/clusters, il faut que le module soit configuré pour recevoir les données d'inventaires que sous souhaitez gérer via ce Receiver. Pour cela il faut utiliser les paramètres suivants:

- elements_sharding_enabled
  - mettre à 1 pour activer l'envoi de l'inventaires des hôtes vers le Receiver
- elements_sharding_filter_by_template
  - mettre le nom d'un template d'hôte qui va filtrer les hôtes à envoyer au Receiver
- elements_sharding_add_data_of_templates
  - mettre le nom d'un ou plusieurs templates d'hôtes/cluster où seront pris les DATA à exporter dans l'inventaire des hôtes
  - Cela permet de limiter le volume de donnée qui iront sur le receiver (qui peut être conséquent)
  - **Remarque**: les données de checks, elle, sont systématiquement présentes pour les checks.

Directive pour le développement de votre module

Il est important de bien lire les commentaires dans le code d'exemple avant toute modification.

Le code est divisés en deux parties:

- La classe du module en lui même, qui va être chargée par le processus Receiver
- La classe du worker qui va fonctionner dans le processus du worker

Le code du module

Le code dans le module va être très limité, car la seule méthode que vous pouvez modifier/surcharger est get_raw_stats qui est utilisée pour les checks de supervision et le healthcheck.

Il est important de:

- ne pas surcharger la méthode __**init**__ du module
- ne pas surcharger d'autres méthodes de la classe module autre que **get_raw_stats** et **want_brok**
- toujours laisser l'appel au super au sein de **get_raw_stats** et seulement rajouter vos propres données au résultat sans modifier celle existantes.
- **want_brok**: vous pouvez le surcharger pour filtrer les broks qu'on ne souhaite pas envoyer au worker.
- tous les imports doivent être fait depuis **shinkensolutions.api** et pas **shinken** directement car seul **shinkensolutions.api** est considéré comme une API stable entre les versions
  - NOTE: en version v02.07.04 cet espace n'étant pas disponible, les imports dans shinken sont fournis à la place mais devront être migré dès le passage en v02.08.01.

Le code dans le worker

Le code dans le worker est celui où sera votre code métier ainsi que votre boucle principale d'actions.

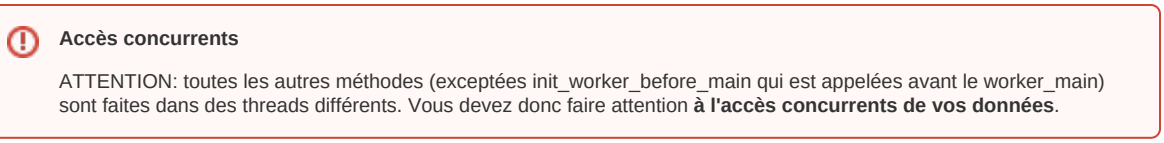
Ici encore certaines règles s'appliquent afin s'assurer une stabilité dans le temps:

- ne pas surcharger la méthode __**init**__ de la classe worker
- toujours laisser l'appel au super au sein de **get_raw_stats** et seulement rajouter vos propres données au résultat sans modifier celle existantes.
- tous les imports doivent être fait depuis **shinkensolutions.api** et pas **shinken** directement car seul **shinkensolutions.api** est considéré comme une API stable entre les versions

    - NOTE: en version v02.07.06 cet espace n'étant pas disponible, les imports dans shinken sont fournis à la place mais devront être migrès dès le passage en v02.08.01.

Les méthodes que vous pouvez surcharger sont multiples.

Méthodes surchargeables concernant le fonctionnement global du worker:

- **init_worker_before_main**: (init_worker en version v02.07.06) cette fonction vous permet de récupérer les paramètres de votre module dans le fichier .cfg sous forme de string comme propriétés de l'objet module_configuration.
- **worker_main**: (**obligatoire**) c'est la boucle principale de votre worker. Si elle s'arrêt, le worker s'arrête également et le module est mis en erreur.
- 

- **get_raw_stats**: vous pouvez la surcharger pour retourner les stats de votre module, qui seront récupérées par le get_raw_stats que vous avez surchargés dans la classe module.

Les méthodes concernant l'inventaires des hôtes:

- **callback__a_new_host_added**: appelé avec l'uuid d'un hôte qui vient d'être rajouté
- **callback__a_host_updated**: appelé avec l'uuid d'un hôte qui vent d'être modifié
- **callback__a_new_realm_added**: appelé lorsqu'un royaume vient d'être fini d'être chargé
    - et donc tous les appels des callback__a_new_host_added/callback__a_host_updated sont déjà effectués
- **callback__a_realm_updated**: appelé lorsqu'un royaume vient d'être fini d'être mis à jour
    - et donc tous les appels des callback__a_new_host_added/callback__a_host_updated sont déjà effectués

Envoi des ordres/commandes vers shinken depuis le worker

Dans le worker il est possible actuellement d'effectuer les commandes suivantes:

- pousser un résultat vers les schedulers
- créer une prise en compte d'un élément
- créer une période de maintenance

Toutes les commandes doivent avoir une forme telle que:

[**EPOCH**] **NOM_COMMANDE**;**ARG1**;**ARG2**;**ARG3**

- **EPOCH**: epoch en int
- **NOM_COMMANDE**: nom de la commande qui sera donné ensuite
- **ARG1**, **ARG2**, **ARG3**: les arguments de la commande. Attention, tous les arguments sont obligatoires

Envoyer des retours de sondes

Pour pousser un résultat d'hôte/cluster vers les schedulers il faut créer la commande **PROCESS_HOST_CHECK_RESULT** avec comme arguments:

- le nom de l'hôte
- le code retour (0 ou 2 pour un hôte)
- le texte de output

Pour pousser un résultat de check vers les schedulers il faut créer une commande **PROCESS_SERVICE_CHECK_RESULT** avec comme arguments:

- le nom de l'hôte
- nom du check
- le code retour (0 , 1, 2 ou 3 pour un check)
- le texte de output

Prendre en compte une erreur:

Pour créer une prise en compte d'élément hôte/cluster il faut créer une commande **ACKNOWLEDGE_HOST_PROBLEM**:

- le nom de l'hôte
- le nombre 1
- 1 ou 0 suivant si vous souhaitez qu'une notification soit envoyée
- le nombre 1
- le nom de l'auteur
- un commentaire

Pour créer une prise en compte d'élément check il faut créer une commande **ACKNOWLEDGE_SVC_PROBLEM**:

- le nom de l'hôte
- le nom du check
- le nombre 1

- 1 ou 0 suivant si vous souhaitez qu'une notification soit envoyée
- le nombre 1
- le nom de l'auteur
- un commentaire

Créer une période de maintenance:

Pour créer une période de maintenance d'élément hôte/cluster il faut créer une commande **SCHEDULE_HOST_DOWNTIME**:

- le nom de l'hôte
- le temps epoch du démarrage
- le temps epoch de la fin
- le chiffre 1
- le chiffre 0
- le chiffre 0
- le nom de l'auteur
- un commentaire

Pour créer une période de maintenance d'élément hôte/cluster il faut créer une commande **SCHEDULE_SVC_DOWNTIME**:

- le nom de l'hôte
- le nom du check
- le temps epoch du démarrage
- le temps epoch de la fin
- le chiffre 1
- le chiffre 0
- le chiffre 0
- le nom de l'auteur
- un commentaire