Module d'exemple pour le Receiver de réception d'actions avec worker

Préambule

Shinken est un ensemble de démons travaillant de manière autonome pour collecter les statuts des équipements supervisés. Mais dans certains cas, Shinken ne peut pas accéder aux équipements à 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 traduiront 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 où
- 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 leurs 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 un 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 parts égales entre les workers.
    - Exemple :
        - 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 répartie entre plusieurs processus.

Mise en place du module d'exemple

Ce lien vous permet de récupérer l'archive contenant le module d'exemple:

- module receiver V01.01.00.zip

Etape 1: Choisir le nom de votre module

Il faut un nom pour votre module. Ce nom doit respecter les règles suivantes :

- Uniquement des caractères ASCII
- Seuls 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 avec 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 copier dans le package d'exemple :

- Le répertoire module en tant que **/var/lib/shinken/modules/NOMduMODULE**
- 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 problèmes 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**

Etape 4: 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 vous 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'inventaire des hôtes vers le Receiver
- elements_sharding_filter_by_template
  - Mettre le nom d'un modèle d'hôte qui va filtrer les hôtes à envoyer au Receiver
- elements_sharding_add_data_of_templates
  - Mettre le nom d'un ou plusieurs modèles d'hôtes/cluster où seront prises les DATA à exporter dans l'inventaire des hôtes
  - Cela permet de limiter le volume de donnée qui ira sur le Receiver (qui peut être conséquent)
  - **Remarque**: les données de checks, elles, sont systématiquement présentes pour les checks.

Etape 5: Tester le module d'exemple

Le module d'exemple a un fonctionnement très simple:

- Il ouvre le port 9000 en HTTP
- Il écoute sur l'uri /my_api en POST
- Il prend comme argument key=VOTREVALEUR
  - Il va afficher cette valeur dans le log
- Il va ensuite envoyer un résultat UP à tous les hôtes qui sont définis dans son inventaire

Exemple de communication avec le module via curl:

```
curl --data 'key=my_value' http://127.0.0.1:9000/my_api
```

Etape 6: A vous de jouer

Modifier le module pour qu'il corresponde à vos attentes

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 divisé 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 existante.
- **want_brok**: vous pouvez le surcharger pour filtrer les broks qu'on ne souhaite pas envoyer au worker.
- Tous les imports doivent être faits depuis **shinkensolutions.api** et pas S**hinken** 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 migrés 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 de 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 existante.
- Tous les imports doivent être faits 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 migrait 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

> **Accès concurrents**
>
> ATTENTION: toutes les autres méthodes (exceptées init_worker_before_main qui est appelée avant le worker_main) sont faites dans des threads différentes. Vous devez donc faire attention **à l'accès concurrent de vos données**.

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

Les méthodes concernant l'inventaire 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

Les objets d'inventaires

Lorsque l'inventaire est mis à jour, les objets d'inventaires vous sont fournis:

- les hôtes
    - avec leur checks accrochés
- les clusters
    - avec leurs checks accrochés

Les méthodes appelables sur ces objets sont:

Pour les hôtes et les clusters:

- **get_uuid**: uuid de l'hôte/cluster
- **get_instance_name**: nom de l'hôte/cluster
- **get_address**: addresse, uniquement pour les hôtes
- **get_realm**: royaume de l'hôte/cluster
- **get_data**: dict avec les DATA de l'élément (sous forme de string)

- **get_checks**: dict des checks accrochés à l'hôte/cluster, avec comme clé l'uuid du check et comme valeur l'objet check
- **get_templates**: liste de string des templates de l'hôte/cluster
- **is_cluster**: booléen si un élément est un cluster ou pas

Concernant les checks, les méthodes appelables sont:

- **get_uuid**: uuid du check
- **get_instance_name** : nom complet du check sous la forme HOST_NAME-CHECK_DESCRIPTION
- **get_name**: champ service description du check
- **get_host**: retourne l'objet host du check
- **get_realm**: royaume où est accroché l'hôte du check
- **get_data:** dict avec les DATA de l'élément (sous forme de string)

> **ATTENTION**
>
> **IMPORTANT:** aucune modification ne doit être faite sur ces objets. Seules ces méthodes sont à appeler, les autres pouvant modifier l'élément et créer de graves incohérences ou bien être supprimer/renommer dans les futures versions.

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 telles 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

Dans le module d'exemple, la méthode **export_http** montre comment définir l'envoi d'un OK pour tous les hôtes de l'inventaire.

Mais ce n'est qu'un exemple. L'important est le bloc pour l'envoi des commandes.

```
                cmd = '[%s] PROCESS_HOST_CHECK_RESULT;%s;0;Is alive' % (int(time.time()), host_name)
                logger.info('[%s] Generating a command UP for %s' % (self.get_name(), host_name))
                ext = ExternalCommand(cmd)
                # give back the check result to the daemon
                self.send_object_to_main_daemon(ext)
```

Envoyer des retours de sondes

Pour pousser un résultat d'hôte/cluster vers les Schedulers

- Il faut créer la commande **PROCESS_HOST_CHECK_RESULT;<nom de l'hôte>;<le code retour>;<le texte de résultat>**
  - Le nom de l'hôte
  - Le code retour (0 ou 2 pour un hôte)
    - 0: OK
    - 2: CRITICAL
  - Le texte de résultat (telle que le retour une sonde)
- **Exemple**: "[1585321359] PROCESS_HOST_CHECK_RESULT**;**Shinken;0;tout va bien|ping_time=100ms"

Pour pousser un résultat de check vers les Schedulers

- Il faut créer une commande **PROCESS_SERVICE_CHECK_RESULT;<nom de l'hôte>;<nom du check>;<le code retour>;<le texte de résultat>**

    - Le nom de l'hôte
    - Le nom du check
    - Le code retour (0 , 1, 2 ou 3 pour un check)
        - 0: OK
        - 1: WARNING
        - 2: CRITICAL
        - 3: UNKNOWN
    - Le texte du résultat (telle que le retour une sonde)

- **Exemple: "[1585321359] PROCESS_SERVICE_CHECK_RESULT;host_name;CPU;1;Attention 90% de CPU utilisé|cpu__all_usage=90.00%"**

Prendre en compte une erreur:

Pour créer une prise en compte sur un hôte/cluster

- Il faut créer une commande **ACKNOWLEDGE_HOST_PROBLEM;<nom de l'hôte>;<sticky>;<notify>;<persistent>;<auteur>;<commentaire>**
    - Le nom de l'hôte
    - L'option sticky:
        - 1: la prise en compte sera automatiquement enlevée sur le prochain changement d'état.
        - 2: la prise en compte restera jusqu'au prochain OK.
    - L'option notify:
        - 0 : pas de notification
        - 1 : envoie de notification aux utilisateurs devant être notifiés.
    - L'option persistent
        - 1 : (Obligatoire, la prise en compte sera persistante, car enregistré dans la rétention du Scheduler)
    - Le nom de l'auteur
    - Un commentaire

- **Exemple: [1585321359] ACKNOWLEDGE_HOST_PROBLEM;host_name;2;1;1;User 1;Incident pris en compte\n"**

Pour créer une prise en compte d'élément check.

- Il faut créer une commande **ACKNOWLEDGE_SVC_PROBLEM;<nom de l'hôte>;<nom du check>;<sticky>;<notify>;<persistent>;<auteur>;<commentaire>**
    - Le nom de l'hôte
    - Le nom du check
    - L'option sticky:
        - 1: la prise en compte sera automatiquement enlevée sur le prochain changement d'état.
        - 2: la prise en compte restera jusqu'au prochain OK.
    - L'option notify:
        - 0 pas de notification
        - 1 envoie de notification aux utilisateurs devant être notifiés.
    - L'option persistent
        - 1 (Obligatoire, la prise en compte sera persistante, car enregistré dans la rétention du Scheduler)
    - Le nom de l'auteur
    - Un commentaire

- **Exemple: "[1585321359] ACKNOWLEDGE_SVC_PROBLEM;host_name;CPU;2;1;1;User 1;Incident pris en compte\n"**

Créer une période de maintenance:

Pour créer une période de maintenance sur un hôte/cluster.

- Il faut créer une commande **SCHEDULE_HOST_DOWNTIME;<nom de l'hôte>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<auteur>;<commentaire>**
    - Le nom de l'hôte
    - Le temps epoch du démarrage
    - Le temps epoch de la fin
    - fixed:
        - *1 (Obligatoire, qui fera que la période de maintenance commencera de la date de début à la date de fin)*
    - trigger_id:
        - *0 (pas pris en compte)*
    - duration:
        - *0 (pas pris en compte)*
    - Le nom de l'auteur
    - Un commentaire

- **Exemple: [1585321359] SCHEDULE_HOST_DOWNTIME;host_name;CPU;1;0;0;User 1;Arrêt de maintenance pris en compte\n"**

Pour créer une période de maintenance sur un check d'hôte/cluster.

- Il faut créer une commande **SCHEDULE_SVC_DOWNTIME;<nom de l'hôte>;<nom du check>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<auteur>;<commentaire>**
  - Le nom de l'hôte
  - le nom du check
  - Le temps epoch du démarrage
  - Le temps epoch de la fin
  - fixed:
    - *1 (Obligatoire, qui fera que la période de maintenance commencera de la date de début à la date de fin)*
  - trigger_id:
    - *0 (pas pris en compte)*
  - duration:
    - *0 (pas pris en compte)*
  - Le nom de l'auteur
  - Un commentaire
- **Exemple: [1585321359] SCHEDULE_SVC_DOWNTIME;host_name;CPU;1;0;0;User 1;Arrêt de maintenance pris en compte\n"**