Module d'exemple pour le Receiver de réception d'actions avec Worker pour une version de Shinken supérieure à la V02.08.02-RC015 incluse (Python 3.X)

Sommaire

Le module d'exemple pour le Receiver de réception d'actions avec Worker a été modifié en V02.08.02-RC015, ainsi que son protocole d'installation (*En raison de la migration en Python3 de Shinken*).

Si vous utilisez une version de Shinken antérieure à la V02.08.02-RC015, veuillez vous référer à la documentation correspondante (*Voir la page* Module d'exemple pour le Receiver de réception d'actions avec Worker pour une version de Shinken inférieure à la V02.08.02-RC015 (Python 2.7)).

Si votre version de Shinken est **supérieure ou égale** à la V02.08.02-RC015, vous pouvez rester sur cette documentation pour créer votre module.

Concept

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 informations de statuts de l'extérieur et de les transmettre aux démons concernés.

Mais comme les sources et les formats de ces informations peuvent être différente, il est nécessaire d'ajouter des modules au Receiver qui traduiront les informations reçues dans un format que Shinken comprendra.

Principe de fonctionnement des modules de Receiver d'écoute

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 :

- Se mettre en écoute pour recevoir ces informations de n'importe où.
- Se connecter à une source pour les recevoir (*exemple : bus de données*).
- Allez chercher directement les informations.

Nous avons fait évoluer l'architecture du Receiver et de ses modules pour qu'ils puissent recevoir les données des hôtes/clusters ainsi que leurs checks associés.
Cela permet, si vous avez besoin d'un identifiant pour vous connecter sur une source de données, 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 (*voir la page* Le 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*).

Mise en place du module d'exemple

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

- module receiver V02.00.00.zip

Le module contenu dans l'archive fonctionnera avec une version de Shinken supérieure ou égale à la V02.08.02-RC015. Si vous utilisez une version de Shinken antérieure à la V02.08.02-RC015, veuillez vous référer à la documentation correspondante (*Voir la page* Module d'exemple pour le Receiver de réception d'actions avec Worker pour une version de Shinken inférieure à la V02.08.02-RC015 (Python 2.7)).

Dans le dossier receiver-module-generic-endpoint de l'archive, vous trouverez :

- Le dossier **receiver_module_generic_endpoint**, qui contient le code Python du module.
- Le fichier **receiver_module_generic_endpoint.cfg** qui correspond à la configuration du module.

Étape 1 : Nommer le module

Pour créer un nouveau module, il faut fournir trois noms :

- Le nom du type de module (*dans le reste de la documentation, on parlera de* **TYPE_MODULE_SHINKEN**).
- Le nom de la classe python du module (*dans le reste de la documentation, on parlera de* **CLASS_PYTHON**).
- Le nom du module dans votre configuration de Shinken (*dans le reste de la documentation, on parlera de* **NOM_MODULE_SHINKEN**).

Pour nommer le type du module (**TYPE_MODULE_SHINKEN**) les règles sont les suivantes :

- Uniquement des caractères alphabétiques,
- Seul le caractère spécial "_" est supporté,
- Pas d'espace,
- Les majuscules sont acceptées,
- Ce nom doit être unique.

Pour nommer la classe python du module (**CLASS_PYTHON**) , les règles sont les suivantes :

- Uniquement des caractères alphabétiques,
- Les majuscules sont acceptées.

Pour nommer son nom dans la configuration Shinken (**NOM_MODULE_SHINKEN**) les règles sont les suivantes :

- Uniquement des caractères alphabétiques,
- Seuls les caractères spéciaux "_" et "-" sont supportés,
- Pas d'espace,
- Les majuscules sont acceptées,
- Ce nom doit être unique.

Nous recommandons les conventions de nommage suivantes pour votre module :

- Pour le type du module (**TYPE_MODULE_SHINKEN**) : **TYPEdeDEMON**_module_**NOM_du_MODULE**
  - Exemple : **receiver_module_generic_endpoint** :
    - **receiver** est le nom du démon,
    - **generic_endpoint** le nom du module.
- Pour la classe python du module (**CLASS_PYTHON**) : **TYPEdeDEMON** concaténé au **NOMduMODULE** en commençant chaque nouveau mot par une majuscule
  - Exemple : **ReceiverGenericEndpoint** :
    - **Receiver** est le nom du démon,
    - **GenericEndpoint** le nom du module.
- Pour son nom dans la configuration Shinken (**NOM_MODULE_SHINKEN**) : **TYPEdeDEMON**-module-**NOM-du-MODULE**
  - Exemple : **receiver-module-generic-endpoint** :
    - **receiver** est le nom du démon,
    - **generic-endpoint** le nom du module.

Étape 2 : Copier le module dans son répertoire

L'installation d'un module ce fait en deux étapes :

- Déployer son code dans le dossier **/opt/shinken/modules** (*sur la machine du Receiver et celle de l'Arbiter*).
- Déployer sa configuration dans **/etc/shinken/modules** (*sur la machine de l'Arbiter*).

À partir de l'archive d'exemple :

- Copier le répertoire du code du module en tant que **/opt/shinken/modules/TYPE_MODULE_SHINKEN.**
- Copier le fichier **receiver_module_generic_endpoint.cfg** en tant que **/etc/shinken/modules/TYPE_MODULE_SHINKEN.cfg .**

Étape 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 **ReceiverGenericEndpoint.**

- Éditez **/opt/shinken/modules/TYPE_MODULE_SHINKEN/module.py** en changeant toutes les occurrences de **ReceiverGenericEndpoint** en **CLASS_PYTHON.**
- Éditez **/opt/shinken/modules/NOM_MODULE_SHINKEN/module_info.json** en changeant toutes les occurrences de **receiver_module_generic_endpoint** en **TYPE_MODULE_SHINKEN**
- Éditez **/etc/shinken/modules/TYPE_MODULE_SHINKEN.cfg :**
  - Changer la valeur du paramètre module_name avec **NOM_MODULE_SHINKEN** .
  - Changer la valeur du paramètre module_type avec **TYPE_MODULE_SHINKEN** .

Étape 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*),
- Ajouter votre module à la configuration de votre Receiver (*par défaut dans **/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,
    - Ce modèle peut être à différent niveau d'héritage.
      - Si la modèle **Mylinux** hérite de Linux et qu'il est utilisé comme modèle dans ce paramètre, ne seront transférés que les données de **linux**, même si sur l'hôte, il n'y a que **Mylinux**.
      - 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.

Étape 5 : Tester le module d'exemple

Voici le fonctionnement du module d'exemple :

- 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
```

Étape 6 : À 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

La seule méthode que vous pouvez modifier/surcharger dans le code du module 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**,
- Toujours laisser l'appel au "super" au sein de **get_raw_stats** et seulement rajouter vos propres données au résultat sans modifier celle existante.

Le code dans le Worker

Le code dans le Worker correspond à votre code métier ainsi qu'à votre boucle principale d'action.

Ici encore, certaines règles s'appliquent afin d'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 ajouter vos propres données au résultat sans modifier celles existantes.

Les méthodes que vous pouvez surcharger sont les suivantes :

- Méthodes sur chargeables concernant le fonctionnement global du Worker :
  - **init_worker_before_main**: cette méthode permet de récupérer les paramètres de configuration de votre module, définis dans le fichier .cfg, sous forme de chaine de caractère comme propriétés de l'objet **module_configuration**.
  - **worker_main**: (***obligatoire***) cette méthode correspond à la boucle principale de votre Worker. Si elle s'arrête, le Worker s'arrête également et le module est mis en erreur.

> **Accès concurrents**
>
> ATTENTION : toutes les autres méthodes (*à l'exception de init_worker_before_main, qui est appelée avant le worker_main*) sont exécutées dans des threads différents. 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_

- Les méthodes concernant l'inventaire des hôtes :
  - **callback__a_new_host_added**: cette méthode est appelée avec l'uuid d'un hôte qui vient d'être ajouté,
  - **callback__a_host_updated**: cette méthode est appelée avec l'uuid d'un hôte qui vent d'être modifié,
  - **callback__a_new_realm_added**: cette méthode est appelée à la fin du chargement d'un royaume,
    - et donc tous les appels callback__a_new_host_added/callback__a_host_updated ont déjà été effectués,
  - **callback__a_realm_updated**: cette méthode est appelée à la fin de la mise à jour d'un royaume,
    - et donc tous les appels des callback__a_new_host_added/callback__a_host_updated ont déjà été effectués,

Les objets d'inventaires

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

- les hôtes :
  - avec leurs 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**: dictionnaire Python contenant les DATA de l'élément (*sous forme de string*),
  - **get_checks**: dictionnaire 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:** dictionnaire Python avec les DATA de l'élément (*sous forme de string*).

ATTENTION

IMPORTANT : aucune modification ne doit être faite sur ces objets. Seules les méthodes listées doivent être appelées, les autres pouvant modifier l'élément et créer de graves incohérences ou bien être supprimées/renommées dans de 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 la forme suivante :

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

- **EPOCH** : epoch (*int*),
- **NOM_COMMANDE** : nom de la commande,
- **ARG1**, **ARG2**, **ARG3** : Arguments de la commande (*Tous les arguments de la commande doivent être présent*).

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

La partie importante de l'exemple est le bloc qui correspond à l'envoi des commandes.

Encodage des commandes

ATTENTION : Shinken attend que les commandes soient soit de type **str** de Python, soit encodées en **UTF-8** dans le cas de **bytes**. Dans le second cas, Shinken décodera les bytes en utilisant UTF-8 et ignorera les caractères qu'il ne parvient pas à décoder, pouvant entraîner des comportements inattendus si la commande n'était pas encodée en UTF-8.

Notre recommandation est d'utiliser le type **str** de Python pour vos commandes.

```
            cmd = '[%s] PROCESS_HOST_CHECK_RESULT;%s;0;Is alive' % (int(time.time()), host_name)
            self.logger.info('[%s] Generating a command UP for %s' % (self.get_name(), host_name))
            ext = ExternalCommand(cmd)  # create an object the receiver will accept
            self.send_object_to_main_daemon(ext)
```

Envoyer des retours de sondes

Pour les hôtes / cluster

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 les checks

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 les hôtes / cluster

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 les checks

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 les hôtes / cluster

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 les checks

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"** .