Module Livestatus

Sommaire

Description

Le module Livestatus permet d'ouvrir une API (via TCP) sur le Broker. Les outils externes comme **Thruk** ou **Nagvis** (URIs tierces compatibles Nagios) peuvent alors envoyer des requêtes à cette API afin de récupérer les états des éléments supervisés.

Il est possible de modifier des paramètres (comme l'adresse d'écoute ou le port) via le fichier de configuration ci-dessous.

Activation du module

Le module Livestatus est un module qui peut être activé seulement sur le démon Broker.

- L'activation de la source s'effectue en ajoutant le nom de cette source dans le fichier de configuration du démon Broker.
- Pour ce faire, ouvrer le fichier de configuration à l'emplacement **/etc/shinken/brokers/nom_du_broker.cfg**, et ajouter le nom de votre module "`Livestatus`".

Exemple: par défaut, nous livrons un module dont le nom est "`Livestatus`"

```
define broker {
        ...
    modules                  Module 1, Module 2, Module 3, Livestatus
    ...
}
```

Pour prendre en compte le changement de configuration, redémarrer l'Arbiter:

```
service shinken-arbiter restart
```

Configuration

La configuration du module se trouve par défaut dans le fichier **/etc/shinken/modules/livestatus.cfg**

- Vous trouverez aussi systématiquement un exemple dans **/etc/shinken-user-example/configuration/daemons/brokers/modules /livestatus/livestatus-example.cfg**

Exemple de fichier de configuration

```
#================================================================================
# Livestatus
#================================================================================
# Daemons that can load this module:
# - broker
# This module open a TCP API that can be query by external tools like Nagios
# UIs (Thruk or NagVis)
#================================================================================


define module {

# 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-6e1beab85adc11e5a4ea080027f08538
    _SE_UUID_HASH        bc47ca0c63c8b7cf1a3f3ca1d3c3b79f
# End of Shinken Enterprise part

    #======== Module identity =========
    # Module name. Must be unique
    module_name               Livestatus

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

    #======== Listening address =========
    # host: IP address to listen to.
    #       note: * = all interfaces.
    host                      *

    # port to listen
    port                      50000

    # socket: file path of a unix socket. If set, will listen this socket for query
    #socket                   /var/lib/shinken/live

    #======== Security =========
    # The livestatus protocol allows to set AuthUser: in the query to filter results
    # for a specific user (if the user is a notification contact of hosts/checks)
    # The remote_user_case_sensitive parameter allows to set if the user name should
    # be checked in a case sensitive mode or in case insensitive mode.
    # 0 : Do not look at case when filtering by user name
    # 1 : [default] Look at the case when filtering by user name
    remote_user_case_sensitive        1

    #====== Debug logging =======
    # Enable only for debugging this module
    #debug                    /var/log/shinken/livestatus.debug

    # If set to 1, log queryes. Note: it's very verbose
    #debug_queries            0

}
```

Détails des sections composant le fichier de configuration

Identification du module

Il est possible de définir plusieurs instances de module de type `livestatus` dans votre architecture Shinken.

- Chaque instance devra avoir un nom unique.

Nom	Type	Unités	Défaut	Commentaire

module_name	Texte	---	**Livestatus**	**Nous vous conseillons de choisir un nom en fonction de l'utilisation du module pour que votre configuration soit simple à maintenir.** Doit être unique.
module_type	Texte	---	**livestatus**	Ne peut être modifié.

Options du module

```
define module {
    ...
    host                                *
    port                                50000
    remote_user_case_sensitive          1
    #debug                              /var/log/shinken/livestatus.debug
    #debug_queries                      0
    ...
}
```

Nom	Type	Unités	Défaut	Commentaire
host	IP	---	*	Adresse IP d'écoute.
port	Entier	---	50000	Port d'écoute.
socket	Path	---	/var/log/shinken/livestatus.debug	Chemin vers un ficher d'un socket Unix. Si renseigné, est utilisé pour écouter les requêtes.
remote_user_case_sensitive	Booléen	---	1	Permets d'activer ou désactiver la sensibilité à la casse dans la recherche par nom d'utilisateur (*1 pour activer, 0 pour désactiver*).
debug	Path	---		Chemin vers un fichier qui sera utilisé pour les logues de débogage.
debug_queries	Booléen	---	0	Permets d'activer ou désactiver le logue des requêtes, ceci est très verbeux (*1 pour activer, 0 pour désactiver*).

Communiquer avec Livestatus - LQL

Pour cela, utiliser par exemple la commande **nc**

Pour installer le paquet **nc** :

```
yum install nc
```

Vous pouvez alors lancer la commande à destination du port 50000 du serveur hébergeant Livestatus (broker) :

```
echo 'GET hosts' | nc 127.0.0.1 50000
```

Cette commande permettra de récupérer la liste des hôtes avec leurs infos.

LQL

LQL - pronounced "Liquel" as in "liquid" - is a simple language for telling Livestatus what data you want and how it should be formatted. It does much the same as SQL but does it in another, simpler way. Its syntax reflects (but is not compatible to) HTTP.

Each query consists of:

- *A command consisting of the word GET and the name of a table.*
- *An arbitrary number of header lines consisting of a keyword, a colon and arguments.*
- *An empty line or the end of transmission (i.e. the client closes the sending direction of the socket)*

*All keywords including GET are **case sensitive**. Lines are terminated by single linefeeds (no <CR>). The current version of Livestatus implements the following tables:*

- *hosts - your Nagios hosts*
- *services - your Nagios services, joined with all data from hosts*
- *hostgroups - your Nagios hostgroups*
- *servicegroups - your Nagios servicegroups*
- *contactgroups - your Nagios contact groups*
- *servicesbygroup - all services grouped by service groups*
- *servicesbyhostgroup - all services grouped by host groups*
- *hostsbygroup - all hosts group by host groups*
- *contacts - your Nagios contacts*
- *commands - your defined Nagios commands*
- *timeperiods - time period definitions (currently only name and alias)*
- *downtimes - all scheduled host and service downtimes, joined with data from hosts and services.*
- *comments - all host and service comments*
- *log - a transparent access to the nagios logfiles (include archived ones)ones*
- *status - general performance and status information. This table contains exactly one dataset.*
- *columns - a complete list of all tables and columns available via Livestatus, including descriptions!*
- *statehist - 1.2.1i2 sla statistics for hosts and services, joined with data from hosts, services and log.*

Like in an SQL database all tables consist of a number of columns. If you query the table without any parameters, you retrieve all available columns in alphabetical order. The first line of the answer contains the names of the columns. Please note that the available columns will change from version to version. Thus you should not depend on a certain order of the columns!

Des infos sur API Livestatus ici : http://www.naemon.org/documentation/usersguide/livestatus.html

Vous pouvez aussi créer un fichier **query** contant la requête LQL **GET hosts**, puis en lançant :

```
nc 127.0.0.1 50000 < hosts.lql
```

Vous pouvez également envoyer votre sortie dans un fichier CSV afin de l'ouvrir dans un tableur comme MS Excel par exemple :

```
nc 127.0.0.1 50000 < hosts.lql >> hosts.csv
```

ou

```
printf 'GET hosts' | nc 127.0.0.1 50000 >> hosts.csv
```

Vous obtiendrez alors :

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	accept_passive_checks	acknowledged	acknowledgement_type	action_url	action_url_expanded	active_checks_enabled	address	alias	business_impact	check_command	check_flapping_recovery_notification	check_freshness	check_interval	check_options	check_perio
2	1	0	1			1	localhost		5	check-host-alive	1	0	1		24x7
3	1	0	1			1	nautilus Host-01		2	USS-NAUTILUS command	1	0	1		24x7
4	1	0	1			1	nautilus Host-02		2	USS-NAUTILUS command	1	0	1		24x7
5	1	0	1			1	nautilus host cluster		2	bp_rule!1:USS-NAUTILUS host template	1	0	1		24x7
6	1	0	1			1	www.regaz.fr		2	cmd-dummy!0!"test"	1	0	1		24x7
7	1	0	1			1	192.168.1.241		2	check-host-alive	1	0	1		24x7
8	1	0	1			1	192.168.1.241		2	check-host-alive	1	0	1		24x7
9															
10															
11															

Exemples de requêtes LQL

Récupérer tous les checks en état critique :

```
GET services
Columns: host_name description state
Filter: state = 2
Filter: in_notification_period = 1
```

Voici un exemple de retour de la commande

```
test [copy];check-dummy-critical;2
test [copy];check-loop-custom;2
test [copy];check-loop-simple;2
test;check-dummy-critical;2
test;check-random-py;2
```

Récupérer les checks dont le contact est toto :

```
GET services
Columns: host_name description contacts
AuthUser: toto
```