# shinken-poller

La configuration du module se trouve par défaut dans le fichier suivant: **/etc/shinken/modules/broker-module-livedata.cfg**

```
#============================================================================
# broker module livedata
#============================================================================
# Daemons that can load this module:
# - broker
# This module is an api getting information from the broker
#============================================================================

define module {

    #======== Module identity =========
    # Module name. Must be unique
    module_name             broker-module-livedata

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

    #======== Listening address =========
    # host: IP address to listen to.
    #       note: 0.0.0.0 = all interfaces.
    host                    0.0.0.0
    # port to listen
    port                    50100

    # HTTPs part, enable if you want to set the visualisation interface listen in HTTPs mode
    # disabled by default. Set your own certificates.
    use_ssl                 0
    ssl_cert                /etc/shinken/certs/server.cert
    ssl_key                 /etc/shinken/certs/server.key
    token                   change_me

    lang                    en
}
```

## Configuration de l'interface et du port d'écoute

Par défaut, le port de l'API rendue disponible par le module "*broker-module-livedata*" est **50100**. Ce port peut être changé via le paramètre "*port*" dans le fichier de configuration du module: **/etc/shinken/modules/broker-module-livedata.cfg.**

En plus du port, il est également possible de configurer l'interface réseau sur laquelle est mise à disposition l'API. Si par exemple l'API ne doit être accessible seulement via un réseau local, il est possible de n'écouter les requêtes que sur cette interface réseau.

Cette modification de configuration se fait via le paramètre "*host*" du fichier de configuration du module: **/etc/shinken/modules/broker-module-livedata.cfg**. Dans l'exemple suivant, l'API ne sera disponible que sur l'interface avec l'adresse **192.168.1.27**:

**/etc/shinken/modules/broker-module-livedata.cfg**

```
define module {
    [...]
    host                    192.168.1.27
    [...]
}
```

L'API du module est par défaut mise à disposition sur toutes les interfaces: le paramètre "*host*" est à *0.0.0.0*

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

```
service shinken-arbiter restart
```

## Activation du SSL

L'API du module est accessible via HTTP. Si pour des raisons de sécurité, cette API doit être accessible via HTTPS, il faut passer le paramètre "*use_ssl*" à 1 dans le fichier de configuration du module:

**/etc/shinken/modules/broker-module-livedata.cfg**

```
define module {
    [...]
    # HTTPs part, enable if you want to set the visualisation interface listen in HTTPs mode
    # disabled by default. Set your own certificates. Set your own token, it is usefull to get access to the
API
    use_ssl                 1
    ssl_cert                /etc/shinken/certs/server.cert
    ssl_key                 /etc/shinken/certs/server.key


    [...]
}
```

Les paramètres "*ssl_cert*" et "*ssl_key*" permettent de spécifier la clé et le certificat SSL à utiliser pour la connexion.

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

```
service shinken-arbiter restart
```

## Absorption des broks (éléments de supervision)

Le fonctionnement du thread de récupération des **broks** de ce module peut être configuré via certains paramètres, afin de modifier son "agressivité".

Pendant la mise à jour des données de supervision, le module ne peut pas répondre aux requêtes via son API.

> Une mauvaise configuration de ces paramètres peut compromettre le bon fonctionnement du module, se rapprocher du support si vous avez le moindre doute.

Principe de l'algorithme d'absorption des broks :

1. Attente de broks à traiter
2. Récupération de broks en retard (fonctionnalité de rattrapage)
3. Désérialisation des broks
4. Entrée en session critique (les requêtes à l'API sont bloquées)
5. Traitement des broks
6. Libérer la session critique et attendre de nouveaux broks, **ou** continuer l'absorption de broks (en cas de retard important, on repart à l'étape 1. en restant sur la session critique)

| Clé | Type | Valeur par défaut | Description |
|---|---|---|---|
| | Booléen | 0 | Entrer en session critique après la récupération d'un premier **brok set**.<br><br>La récupération des **broks** en retard, et la désérialisation se font alors dans la session critique (Locké) pour disposer d'un maximum de temps CPU<br><br>• **1** : Activer<br>• **0** : Désactiver |

| | | | |
|---|---|---|---|
| broker_module_livedata__broks_getter__activate_late_set_catchup | Booléen | 1 | Utilisation de la fonctionnalité de rattrapage pour absorber des **broks** en retard :<br><br>• **1** : Activer<br>• **0** : Désactiver |
| broker_module_livedata__broks_getter__nb_late_set_allowed_before_catchup | Entier | 10 | Nombre de **brok set** en attente tolérés.<br>Au dessus de ce nombre, les **brok set** sont immédiatement récupérés par l'algorithme de rattrapage pour être traités maintenant |
| broker_module_livedata__broks_getter__catchup_broks_managed_by_module_in_a_catchup_loop | Entier | 200000 | Nombre maximal de **broks** que l'algorithme de rattrapage récupère avant de lancer le traitement<br><br>Ce paramètre permet de borner la consommation mémoire et le temps d'exécution d'un tour de boucle de traitement |

| | | | |
|---|---|---|---|
| broker_module_livedata__broks_getter__catchup_run_endless_until_nb_late_set_allowed_reached | Booléen | 1 | Après traitement des **broks**, si le nombre de **brok set** en retard est trop élevé,<br><br>• **1** : continuer le rattrapage et absorber des **broks** en retard (en restant sur la session critique ("avec le lock"))<br>• **0** : arrêter l'absorption de **brok** et libérer la session critique (rendre le lock) |
| broker_module_livedata__broks_getter__include_deserialization_and_catchup_in_lock | Booléen | 0 | Dans le cas ou vous voulez disposer d'un maximum de temps CPU pour traiter les broks en retard, vous pouvez inclure la phase 2 ( Récupération de broks en retard ) et Phase 3 ( Désérialisation des broks ) dans la phase Critique ( Phase 4 )<br><br>La récupération des **broks** en retard, et la désérialisation se font alors dans la session critique (Locké) pour<br><br>• **1** : Activer<br>• **0** : Désactiver |