Le Broker

Sommaire

Rôle

Le démon Broker exporte et gère les données du Scheduler (*les objets **Broks***).

- Sa gestion ne peut se faire qu'à travers des modules.
- Plusieurs modules de gestion peuvent être activés en même temps.

Exemples de modules du Broker :

- Module pour exporter les données de métrologie: Graphite-Perfdata
- Module pour l'API Livedata
- Module pour l'affichage de l'interface de visualisation : WebUI

Données: les Broks

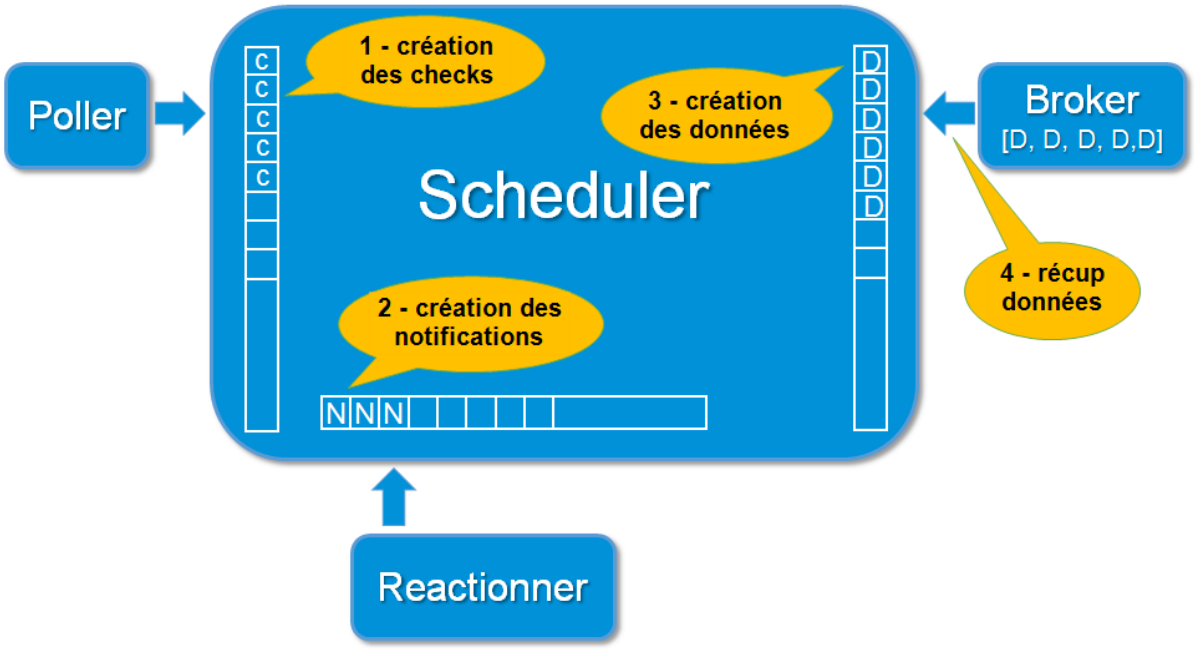
Le Broker reçoit les données des Schedulers sous forme de Broks.

Les Broks

Les Broks sont des conteneur de données échangées entre les Schedulers et les Brokers. Il y a plusieurs type de Broks :

- Des évènements (*comme un Scheduler qui vient de démarrer*).
- La configuration des éléments supervisés (*hôtes, checks, période de temps, utilisateurs*).
- L'état des hôtes, clusters et checks après chaque vérification.

Le rôle du démon Broker est de donner ces données (*Broks*) à tous ses modules.



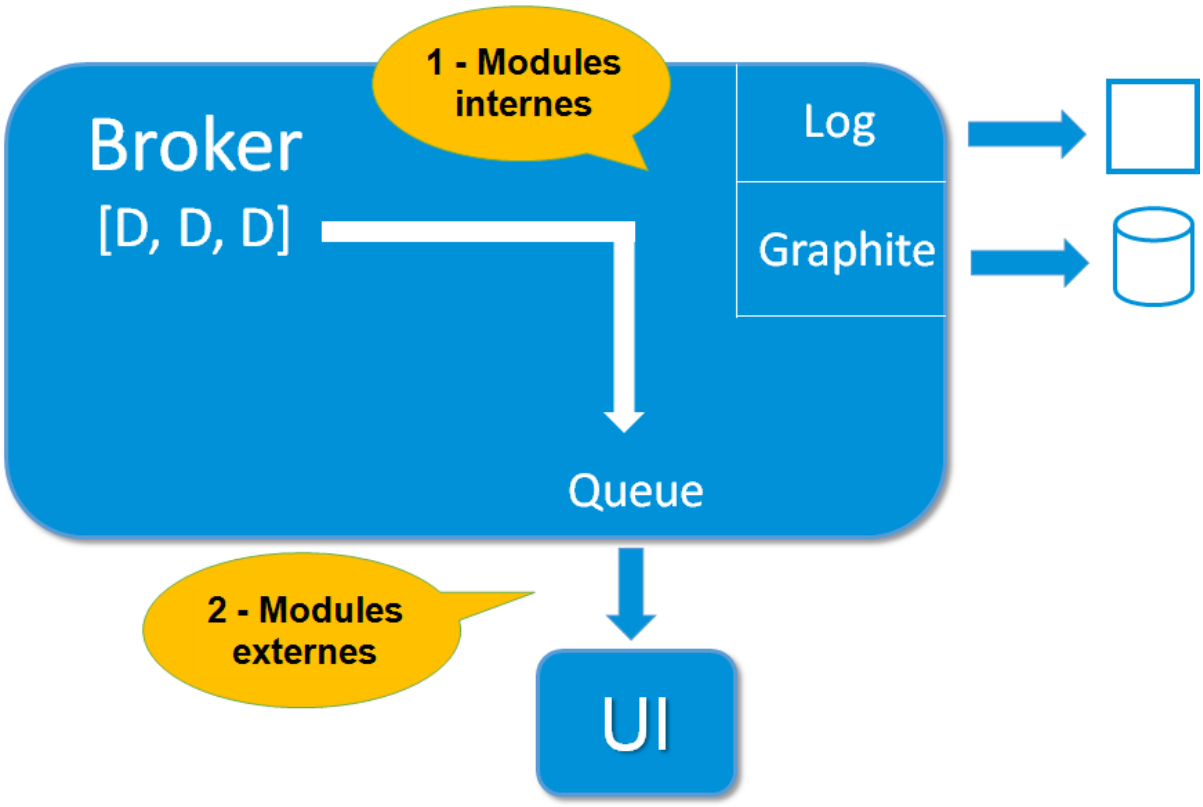
Données de métrologie

Les données de métrologie sont sauvegardées sur le serveur du Broker dans l'application graphite.

Cette application écoute le port 2003, et cette connexion se fait sans authentification.

Cette application doit donc écouter exclusivement sur l'interface réseau locale (*loopback*) du serveur du Broker.

Logique interne du Broker



Résumé des connexions du Broker

Source	Destination	Port	Protocole	Note
Broker	Scheduler	7768	HTTP/HTTPS	

Description des paramètres du Broker (centralisé sur le serveur de l'Arbiter, dans un fichier .cfg)

Propriété	Défaut	Description
broker_name	N/A	Cette variable est utilisée pour identifier le **nom réduit** du Broker auquel les données sont associées.
address	N/A	Cette directive est utilisée pour définir l'adresse permettant de joindre ce Broker. Par défaut "localhost", changez-le par un nom DNS ou une adresse IP.
port	7772	Cette directive est utilisée pour définir le port TCP utilisé par le démon.
use_ssl	0	Cette variable est utilisée pour définir si le Broker doit être contacté en HTTPS (***1***) ou HTTP (***0***). La valeur par défaut est 0 (***HTTP***).
spare	0	Cette variable est utilisée pour définir si le Broker peut être géré comme un spare (prendra uniquement la configuration si le maître échoue). La valeur par défaut est *0* (maître).
spare_daemon	N/A	Nom du démon spare qui sera utilisé pour reprendre le travail de ce démon s'il vient à ne plus être disponible
broker__manage_spare__spare_must_have_the_same_list_of_module_type	1	Si cette option est désactivée (0), alors la configuration des modules du spare défini par **spare_daemon** ne sera pas vérifiée pour voir si elle corresponds bien à celle du master
timeout	3	Cette variable est utilisée pour définir le temps en secondes avant que l'Arbiter ne considère ce démon comme à l'arrêt. Si ce démon est joignable en HTTPS (*use_ssl à 1*) avec une latence élevée, nous vous conseillons alors d'augmenter cette valeur de timeout (*l'Arbiter aura besoin de plus d'allers/retours pour le contacter*).
data_timeout	120	Cette variable est utilisée pour définir le temps en secondes avant de considérer un transfert de configuration ou de données comme échoué.
max_check_attempts	3	Si le ping permettant de détecter la disponibilité réseau du nœud est en échec N fois ou plus, alors le nœud est considéré comme mort (*par défaut, 3 tentatives*)
check_interval	60	Intervalle de Ping toutes les N secondes.
modules	N/A	Cette variable est utilisée pour définir les modules chargés par le Broker.
realm	N/A	Cette variable est utilisée pour définir le royaume où le Broker doit être. Si aucun n'est sélectionné, celui par défaut lui sera assigné.
manage_sub_realms	1	Cette variable est utilisée pour définir si le Broker prendra des tâches des Schedulers des sous-royaumes .
manage_arbiters	1	Prends les données de l'Arbiter. Il ne devrait y avoir qu'un seul Broker pour l'Arbiter.
satellitemap	N/A	Cette variable est utilisée pour définir, pour des environnements NATés, les différents satellites comme vus depuis ce Broker.
broks_packet_size	204800	Si présent, les demandes vers les Schedulers vont avoir comme limite haute de taille de paquet cette valeur (*en Ko*). Par défaut les envois sont illimités.
broker__manage_brok__enable_sub_processes_memory_usage_protection	1	Si activé, le Broker va vérifier qu'il y a assez de RAM disponible sur le système avant de lancer ses processus workers qui poussent les broks vers les modules externes (*comme WebUI*)
broker__manage_brok__sub_process_memory_usage_system_reserved_memory	0	Dans le cas de la protection de mémoire, on peut réserver un pourcentage de RAM pour le système qui ne sera pas considérée comme disponible par le démon
broker__manage_brok__sub_processes_memory_usage_protection_max_retry_time	5	Dans le cas de la protection mémoire, pendant combien de temps le Broker va attendre (*en secondes*) avant de considérer qu'il n'a pas assez de mémoire, ce qui aura comme conséquence de tuer le module externe concerné.
broker__manage_brok__sub_process_broks_pusher_min_execution_timeout	5	Temps (*en secondes*) que le Broker va laisser aux workers qui poussent les broks vers les modules externes pour s'exécuter.

broker__manage_brok__sub_process_broks_pusher_security_ratio	5	Le Broker va estimer le temps d'exécution des workers qui poussent les broks en se basant sur leur moyenne passée, et va appliquer ce ratio multiplicateur comme timeout d'exécution.
broker__manage_brok__sub_process_broks_pusher_max_execution_timeout	240	Temps (*en secondes*) que le Broker va laisser aux workers qui poussent les broks vers les modules externes pour s'exécuter.
broker__manage_brok__sub_process_broks_pusher_max_retry	3	Nombre de tentatives où le Broker va relancer les workers qui poussent les broks avant d'arrêter et tuer le module lié.
broker__manage_brok__sub_process_broks_pusher_queue_batch_size	100000	Taille maximum en nombres de Broks que peuvent faire les workers qui poussent les broks aux modules externes (*comme WebUI*). Attention, trop augmenter cette limite peux poser des problèmes d'envoi trop importants pour la socket de communication.
enabled	N/A	Cette variable est utilisée pour définir si le Broker est activé ou non.

Définition - exemple

Dans le répertoire **/etc/shinken/brokers/**, voici un exemple de définition qui permet la définition du Broker (*à placer dans un fichier CFG*) :

⚠️ Il est conseillé d'éditer les fichiers .cfg avec l'encodage utf-8

```
#===============================================================================
# BROKER
#===============================================================================
# Description: The Broker is responsible for:
# - Exporting centralized logs of all Shinken daemon processes
# - Exporting status data
# - Exporting performance data
# - Exposing Shinken APIs:
#   - Status data
#   - Performance data
#   - Command interface
#===============================================================================

define broker {

    #======== Daemon name and address =========
    # Daemon name. Must be unique
    broker_name               broker-master

    # IP/fqdn of this daemon (note: you MUST change it by the real ip/fqdn of this server)
    address                   localhost

    # Port (HTTP/HTTPS) exposed by this daemon
    port                      7772

    # 0 = use HTTP, 1 = use HTTPS
    use_ssl                           0


    #======== Master or spare selection =========
    # 1 = is a spare, 0 = is not a spare
    spare                     0

    # spare_daemon: name of the daemon that will take this daemon job if it dies
    # IMPORTANT:
    #   * a spare_daemon can only be the spare of 1 (and only one) master daemon
    #   * a spare_daemon cannot have a spare_daemon
    #   * the spare must have modules with the same module_type as the master
    #     - depending of the value of the broker__manage_spare__spare_must_have_the_same_list_of_module_type
parameter
    # Example: spare_daemon             broker-spare
    spare_daemon

    # 1 = (default) the spare defined with spare_daemon must have the same module_type as this master
    # 0 = the spare module_type are not checked
    # broker__manage_spare__spare_must_have_the_same_list_of_module_type     1
```

```
    #======== Daemon connection timeout and down state limit =========
    # timeout: how many seconds to consider a node don't answer
    timeout                   3

    # data_timeout: how many second to consider a configuration transfert to be failed
    # because the network brandwith is too small.
    data_timeout              120

    # max_check_attempts: how many fail check to consider this daemon as DEAD
    max_check_attempts        3

    # Check this daemon every X seconds
    check_interval            60



    #======== Modules to enable for this daemon =========
    # Available:
    # - Simple-log            : save all logs into a common file
    # - WebUI                 : visualisation interface
    # - Graphite-Perfdata     : save all metrics into a graphite database
    # - sla                   : save sla into a database
    # - Livestatus            : TCP API to query element state, used by nagios external tools like NagVis or
Thruk
    modules                   Simple-log, WebUI, Graphite-Perfdata, sla, event-manager-writer


    #======== Realm and architecture settings =========
    # Realm to set this daemon into
    realm                    All

    # 1 = take data from the daemon realm and its sub realms
    # 0 = take data only from the daemon realm
    manage_sub_realms        1


    # In NATted environments, you declare each satellite ip[:port] as seen by
    # *this* Broker (if port not set, the port declared by satellite itself
    # is used)
    #satellitemap    scheduler-1=1.2.3.4:7768, poller-1=1.2.3.5:7771

    # Exchange between brokers <- schedulers can be limited by packet size (in kB)
    # Note: as compression is automatic, this is a higher limit, and in real case the
    #       packets will be lower than this value
    # broks_packet_size 1024


    #======== Memory protection =========
    # Are the daemon module process and worker process are waiting for enough
    # memory to be available before being launch. Default: 1 (enabled)
    broker__manage_brok__enable_sub_processes_memory_usage_protection   1

    # The sub process memory usage protection can have a system reserved memory
    # that won't be used by theses sub process when launched
    # By default: 0 (no reserved memory)
    # Example: 10  (means 10% of the total memory is reserved for the system)
    broker__manage_brok__sub_process_memory_usage_system_reserved_memory    0

    # If a sub process cannot be started because of the protection, how many seconds
    # it will be retry and wait that the system memory is freed until it fail to start
    # By default: 5 (seconds)
    broker__manage_brok__sub_processes_memory_usage_protection_max_retry_time   5


    #======== Brok pusher worker =========
    # The Broker spawn broks pusher sub process to push to external modules (like WebUI)
    # the Broker will look at this worker execution time, and will kill if it timeout
    # The Broker will compute the average execution time of previous workers to
    # decide about how many time this worker will take based on:
    # number of broks to send / past average send speed (broks/s)
```

```
    # If this time is reach, it means that the pusher process is killed

    # For small amount of broks to send, it should lead to ridicusly small allowed execution time
    # and the fac to spawn the sub process can be higher than this value, so we are using a minimal
    # execution timeout
    # Default: 5 (second)
    broker__manage_brok__sub_process_broks_pusher_min_execution_timeout           5

    # In order to manage the fact that the server can slow down during this send, you can setup a
    # ratio that will be used to increase the allowed timeout by multiply it
    # Default: 5
    broker__manage_brok__sub_process_broks_pusher_security_ratio                  5

    # At the broker start without stats, this valud will be used for the timeout
    # Default: 240 (seconds)
    broker__manage_brok__sub_process_broks_pusher_max_execution_timeout           240

    # If a sub process reach a timeout, it will be killed and relaunched. After max retry,
    # the attached module will be restarted
    # Default: 3
    broker__manage_brok__sub_process_broks_pusher_max_retry                       3

    # broker__manage_brok__sub_process_broks_pusher_queue_batch_size:
    #   * defines the maximum number of broks the "queue brok pusher"
    #     process will handle per send to external module ( like WebUI ) .
    #   * Remaining broks will be handled in next send.
    #   * IMPORTANT: increase this value can lead to error on the socket
    # Default: 100000 (broks/batch)
    # broker__manage_brok__sub_process_broks_pusher_queue_batch_size      100000


    #======== Enable or not this daemon =========
    # 1 = is enabled, 0 = is disabled
    enabled                  1

}
```