La scalabilité de la WebUI

Sommaire

Présentation du problème

Sur une installation de Shinken Entreprise volumineuse contenant beaucoup d'hôte, le démon Broker et en particulier l'interface de Visualisation peuvent montrer des ralentissements. C'est principalement dû au fait que le module WebUI du Broker en charge de l'interface de Visualisation ne s'exécute que sur un seul CPU.

Ce module consomme beaucoup de ressources CPU, car il effectue notamment les opérations suivantes :

- Mise à jour des données dans l'interface de Visualisation par communication avec le Scheduler (*Broks*).
- Réponse aux requêtes en parallèle de tout les utilisateurs.

S'il y a un grand nombre d'utilisateurs simultanés, des problèmes de performances peuvent être observés sur l'interface de Visualisation.

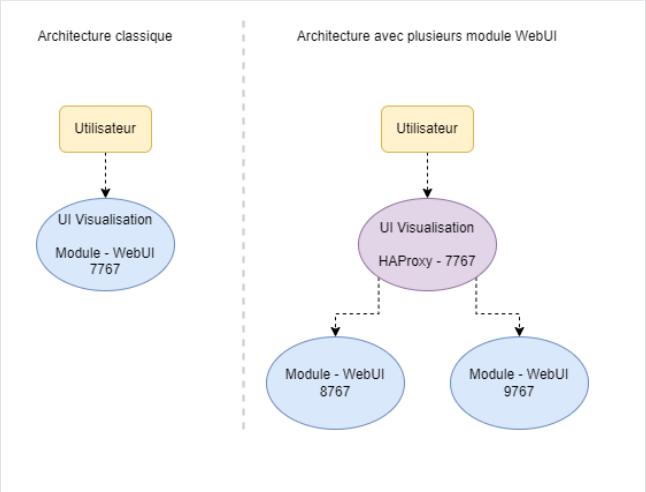
Pour régler ce problème, il est possible d'utiliser plusieurs modules WebUI, qui rendront l'interface de Visualisation accessible via un load balancer (*HAProxy*). Cette modification de l'architecture est donc invisible du point de vue de l'utilisateur.

Mise en place du load balancing

Cette documentation permet de passer de l'Architecture classique à l'Architecture avec plusieurs module WebUI (*voir le schéma à droite*).

À gauche du schéma, on voit une architecture classique, dans laquelle l'utilisateur effectue directement sa requête à l'interface de Visualisation.

À droite, l'utilisateur fait sa requête au HAProxy à la place, qui va répartir les requêtes équitablement et de manière transparente vers une des deux instances du module WebUI.



Définition de plusieurs module WebUI

La première étape pour la mise en place du load balancing est d'avoir plusieurs module WebUI disponibles, sur lesquelles HAProxy va répartir la charge.

On commence donc par dupliquer le module WebUI en copiant **/etc/shinken/modules/webui.cfg** vers **/etc/shinken/modules/webui2.cfg**.

On change ensuite le nom du module en **WebUI2** et le port en **9767** dans notre module fraîchement dupliqué:

/etc/shinken/modules/webui2.cfg

```
define module {
    #======== Module identity =========
    # Module name. Must be unique
    module_name               WebUI2

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



    #======== Listening address =========
    # host: IP address to listen to.
    #       note: 0.0.0.0 = all interfaces.
    host                      0.0.0.0
    # port to listen
    port                      9767


        ...(suite du fichier coupée)
}
```

Pour rester fidèle au schéma ci-dessus, on change également le port du module WebUI (***/etc/shinken/modules/webui.cfg***) en **8767**.

Il faut ensuite ajouter ce module sur le Broker pour qu'il soit activé. Dans la ligne **modules**, on ajoute le module **WebUI2**.

/etc/shinken/brokers/broker-master.cfg

```
define broker {

# 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-broker-060340145ade11e5b703080027f08538
    _SE_UUID_HASH       8e00136f9e61061e07ca0f4a63509b68
# End of Shinken Enterprise part

    #======== 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                   172.16.0.5

    ...(contenu
coupé)




    #======== 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, WebUI2, Graphite-Perfdata, sla,
Livestatus


    ...(suite du fichier coupée)
}
```

Enfin on redémarre l'Arbiter et le Broker pour prendre en compte le changement de configuration.

```
service-shinken-arbiter restart
```

```
service-shinken-broker restart
```

Mise en place du HAProxy

La première étape est d'installer HAProxy et de l'activer au démarrage:

- **Installer HAProxy**

  ```
  yum install haproxy
  ```

- **Activer HAProxy au démarrage**

```
systemctl enable haproxy
```

On copie ensuite la configuration de HAProxy présente dans le fichier suivant dans **/etc/haproxy/haproxy.cfg** sur la machine du Broker:

Pour explication, cette configuration de HAProxy se découpe comme suivant:

- **Déclaration d'un frontend**: Tout ce qui arrive sur la machine en TCP sur le port 7767 utilise le backend "*webuis*"

```
frontend  main *:7767
    mode  tcp
    default_backend             webuis
```

- **Déclaration d'un backend**: On configure ensuite comme traiter les requêtes envoyées par le frontend. Dans notre cas, on repartit les requêtes entre 2 serveurs "*webui1*" et "*webui2*" qui ont chacun leur adresse et leur port.

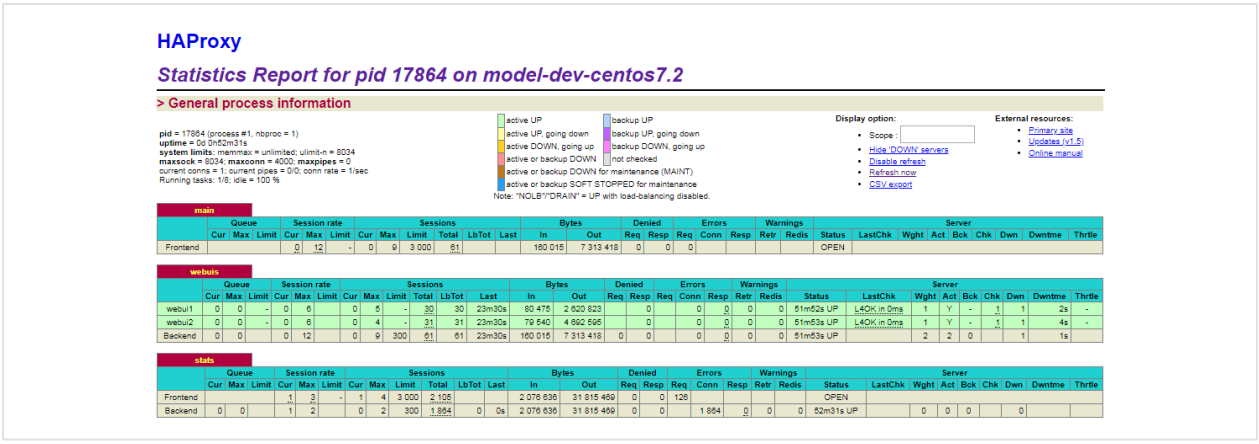
```
backend webuis
    mode tcp
    balance     roundrobin

    server  webui1 127.0.0.1:8767  check  fall 1  rise 1  inter 1s
    server  webui2 127.0.0.1:9767  check  fall 1  rise 1  inter 1s
```

- **Mise à disposition d'une page de statistiques HAProxy**: HAProxy permet de mettre à disposition une page qui présente en temps réel l'état des backends (*dans notre cas les 2 WebUI*).
Cette page de statistiques se configure de la manière suivante:

```
listen stats
    bind :8080
    mode http
    stats enable
    stats hide-version
    stats refresh 1s
    stats show-node
    stats auth admin:shinken
    stats uri  /loadbalancer-stats
```

Dans cet exemple, elle est accessible via http://ip-haproxy:8080/loadbalancer-stats, protégée par une authentification (***user**: "admin", **password**: "shinken"*) et se présente comme suivant:



HAProxy

Statistics Report for pid 17864 on model-dev-centos7.2

> General process information

pid = 17864 (process #1, nbproc = 1)
uptime = 0d 0h52m31s
system limits: memmax = unlimited; ulimit-n = 8034
maxsock = 8034; **maxconn** = 4000; **maxpipes** = 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/8; idle = 100 %

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
backup UP
backup UP, going down
backup DOWN, going up
not checked
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Disable refresh
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.5)
- Online manual

main

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	12	-	0	9	3 000	61			160 015	7 313 418	0	0	0					OPEN								

webuis

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
webui1	0	0	-	0	6		0	5	-	30	30	23m30s	80 475	2 620 823		0		0	0	0	0	51m52s UP	L4OK in 0ms	1	Y	-	1	1	2s	-
webui2	0	0	-	0	6		0	4	-	31	31	23m30s	79 540	4 692 595		0		0	0	0	0	51m53s UP	L4OK in 0ms	1	Y	-	1	1	4s	-
Backend	0	0		0	12		0	9	300	61	61	23m30s	160 015	7 313 418	0	0		0	0	0	0	51m53s UP		2	2	0		1	1s	

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	3	-	1	4	3 000	2 105			2 076 636	31 815 469	0	0	126					OPEN								
Backend	0	0		1	2		0	2	300	1 864	0	0s	2 076 636	31 815 469	0	0		1 864	0	0	0	52m31s UP		0	0	0		0		

La dernière étape est de démarrer le service HAProxy (*ou le redémarrer pour prendre en compte la configuration*):

```
systemctl restart haproxy
```

Les URL suivantes sont alors disponibles:

- http://ip-haproxy:7767 ou https://ip-haproxy:7767 si les modules WebUI sont en HTTPS
- http://ip-haproxy:8080 pour la page de statistiques

Superviser le HAProxy

Le HAProxy peut aussi être supervisé depuis Shinken. Pour ça, il faut mettre la sonde **check_haproxy** en place sur le **serveur du broker** (*dans **/var /lib/shinken-user/libexec/***).

Cette sonde nécessite la dépendance perl-Switch pour fonctionner. Pour installer celle-ci :

sous Centos7 :

```
yum install perl-Switch
```

sous Almalinux 8 :

```
dnf --enablerepo=powertools install perl-Switch
```

Ou via RPM directement au lien suivant : https://almalinux.pkgs.org/8/almalinux-powertools-aarch64/perl-Switch-2.17-10.el8.noarch.rpm.html

Le fichier de la sonde à mettre en place est le suivant:

Il faut penser à lui donner les droits d'exécutions:

```
chmod a+x /var/lib/shinken-user/libexec/check_haproxy
```

Pour la lancer:

```
/var/lib/shinken-user/libexec/check_haproxy -S /var/lib/haproxy/stats -D 'C<u,.501,0.01,25,50>'
```

- **u**: on ne compte que les serveurs up
- **0.501** : on passe en WARNING si on passe en dessous de 50.1% des serveurs disponibles
- **0.01**: on passe en CRITICAL si on passe en dessous de 0.1% des serveurs disponibles
- **25**: on passe en WARNING si on dépasse les 25% de sessions autorisées
- **50**: on passe en CRITICAL si on dépasse les 50% de sessions autorisées

Voici un exemple de rendu de la sonde:

```
[root@centos-7 ~]# ./check_haproxy -S /var/lib/haproxy/stats -D 'C<u,.501,0.01,25,50>'
HAPROXY WARNING: BACKEND webuis has fallen below 0 available server(s) (1 up, 1 down, 0 disabled) (check_haproxy 1.0.3)
```

Résolutions de problèmes courants

HAProxy affiche dans les l'erreur "Starting frontend main: cannot bind socket [0.0.0.0:7767]"

Il se peut que HAProxy refuse de démarrer puisqu'il n'arrive pas à utiliser le port 7767. Dans ce cas, on a l'erreur suivante dans le journal du système sous CentOS 7 (*obtenu avec la commande "journalctl -xe"*):

```
Mar 11 14:50:46 model-dev-centos7.2 systemd[1]: Starting HAProxy Load Balancer...
-- Subject: Unit haproxy.service has begun start-up
-- Defined-By: systemd
-- Support: http://lists.freedesktop.org/mailman/listinfo/systemd-devel
--
-- Unit haproxy.service has begun starting up.
Mar 11 14:50:46 model-dev-centos7.2 polkitd[669]: Unregistered Authentication Agent for unix-process:4715:
1812926 (system bus name :1.306, object path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_US.
UTF-8) (disconnected from bus)
Mar 11 14:50:46 model-dev-centos7.2 haproxy-systemd-wrapper[4721]: haproxy-systemd-wrapper: executing /usr
/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -Ds
Mar 11 14:50:46 model-dev-centos7.2 haproxy-systemd-wrapper[4721]: [ALERT] 069/145046 (4722) : Starting
frontend main: cannot bind socket [0.0.0.0:7767]
Mar 11 14:50:46 model-dev-centos7.2 haproxy-systemd-wrapper[4721]: haproxy-systemd-wrapper: exit, haproxy
RC=1
Mar 11 14:50:46 model-dev-centos7.2 systemd[1]: haproxy.service: main process exited, code=exited, status=1
/FAILURE
Mar 11 14:50:46 model-dev-centos7.2 systemd[1]: Unit haproxy.service entered failed state.
Mar 11 14:50:46 model-dev-centos7.2 systemd[1]: haproxy.service failed.
```

- On peut vérifier dans ce cas que le port n'est pas déjà réservé par un processus. On fait cette vérification avec la commande:

  ```
  netstat -lapute | grep 7767
  ```

  Si cette commande a une sortie vide, on voit alors qu'aucun processus n'a effectué de *bind* sur le port 7767. Sinon, on a une ligne qui indique le protocole, l'utilisateur et le PID/processus responsable de l'écoute:

  ```
  $ netstat -lapute | grep 7767
  tcp        0      0 0.0.0.0:7767            0.0.0.0:*               LISTEN      shinken    201712
  4600/shinken-broker
  ```

- Si le port est déjà réservé par un processus, on peut éteindre ce processus pour libérer le port
- Si personne n'utilise le port, la cause peut se situer du côté de SELinux qui empêche l'utilisation du port par HAProxy.
- On peut confirmer cette hypothèse en commençant par vérifier si SELinux est activé sur la machine avec la commande suivante:

```
$ sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      31
```

- On peut alors désactiver SELinux ou bien autoriser HAProxy à utiliser des ports (*conseillé*):

```
setsebool -P haproxy_connect_any=1
```

Le check Haproxy affiche l'erreur "HAPROXY UNKNOWN: Cannot connect to socket /var/lib/haproxy /stats (check_haproxy 1.0.3)"

Si shinken affiche le le message suivant :

```
/var/lib/shinken-user/libexec/check_haproxy -S /var/lib/haproxy/stats -D 'C<u,.501,0.01,25,50>'
HAPROXY UNKNOWN: Cannot connect to socket /var/lib/haproxy/stats (check_haproxy 1.0.3)
```

La **raison** est que la sonde a été exécutée manuellement via l'utilisateur root, le fichier temporaire **/var/lib/haproxy/stats** créé lors de son exécution, sera en root si c'est le fichier n'existait pas (*creation du fichier*) .

- De ce fait, la prochaine execution de la sonde par le poller sous l'utilisateur Shinken, le fichier **stats** ne pourra pas etre utilisé.

Pour corriger le problème, on peut :

- Supprimer le fichier de stats :

  ```
  rm -f /var/lib/haproxy/stats
  ```

- Modifier les droits du fichier pour les attribuer à Shinken :

  ```
  chown shinken:shinken /var/lib/haproxy/stats
  ```