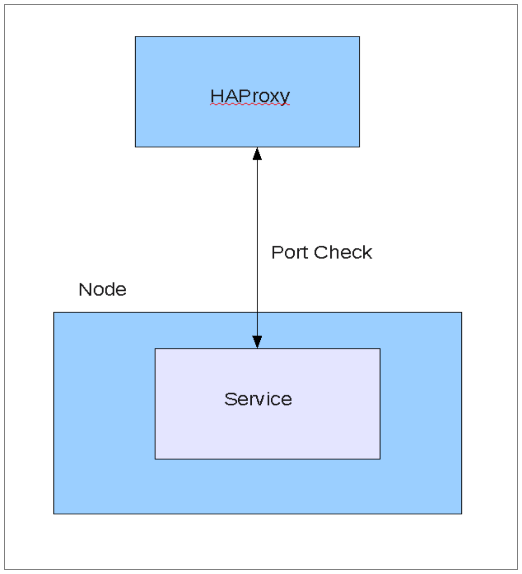
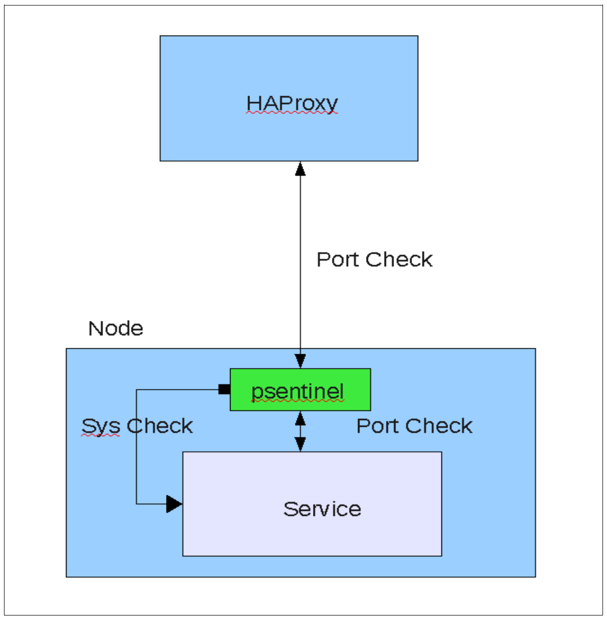
PSentinel

(aka Proxy Sentinel.. or port sentinel or "Put there the word that you like" sentinel)

Introduction

Haproxy is a very interesting product and it's used extensively on the web. At the basic level HAProxy decides the health state of a service checking if the connection TCP/HTTP succeds. If the connection is established the service is OK otherwise is KO.





The idea behind psentinel is pretty simple: insert a daemon between haproxy and the back-end service to decouple the service from the direct check.

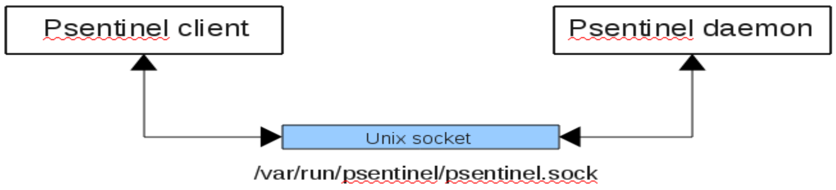
Basic usage

Psentinel executable can act as client or daemon.

In daemon mode, psentinel replies to an haproxy port check and executes a shell script or a binary program to test the real state of the application monitored. That script can be prepared, specifically for monitored applications, by the developers in order to do a better in-deep check about real health state instead of a simple port check.

If the application check succeeds psentinel replies with a 200 OK code to haproxy , if it fails psentinel replies with 500 Internal server error. In that way haproxy can automatically exclude our node from his back-end.

In client mode psentinel can interact, through unix socket, with his daemon counterpart and issue commands to it.



Issuing a command is useful if we want manually control the state of psentinel for example enabling or disabling the port reply to haproxy (and so enabling or disabling our node in the haproxy back-end). This manual intervention compared with automatic port check give us the possibility to temporary disable our application without shutting it down, for example during a jenkins deploy or after an update. When we are confident that our application behaves correctly we can re-enable the port reply and let haproxy re-inserts our node in his back-end.

Execution in daemon mode

```
$>psentinel -d -f /etc/psentinel.cfg
Or
$>psentinel -d       (default is /etc/psentinel/psentinel.cfg)
```

the configuration file options are explained further.

Execution in client mode

```
 $>psentinel -c help  (list all available commands in client mode)
$>psentinel -c status
$>psentinel -c version   -f /home/psentinel.cfg
```

There is also a Debug execution mode (-D) . It put psentinel in “not daemon mode” and issuing all messages in stdout/stderr

Configuration file

Is a INI file syntax with sections and options inside sections.

```
[servicename]   (arbitrary name for section, e.g. jboss,esb1,tomcat3)
 port=8080  (replying port for haproxy check)
pausable=yes (make this port pausable with a single command..see examples)
start_paused=yes   (if pausable, how set the state when psentinel starts)
check_script=/path/scriptname args (the check script name to execute)
check_script_interval=10 (num of seconds interval for executing check script)

        […]  other servicenames with different ports  and check files

[general] (special section where declare control variables for psentinel)
log_level=4 (log level 0..4 0=min level, 4=max level)
pid_file=/path/pidfile (file used to write running pid of daemon)
sock_file=/path/sockfile (unix socket file used for communication with client)
```

Use cases

Case 1

Two back-end nodes with a single application server and haproxy in front of them. The back-end nodes host a complex application and we want to enable or disable it after a deep test called each 2 minutes. The application serves the 8080 port. Psentinel replies to haproxy in port 48080 after executing our test_script each 120 seconds.
In haproxy configuration we have

```
 ...
 acl ourapprule <some rule to select our application backend>
 use backend ourapp if  ourapprule
 ...
backend ourapp
option httpchk OPTIONS * HTTP/1.0
server webapp ouserverip:8080 cookie 1 weight 1 maxconn 1000 check port 48080
```

The *httpchk* tells to haproxy to check the back-end using a http OPTIONS verb with 1.0 protocol (to reply psentinel uses http1.0) and *check port 48080* tells it toward which port. If psentinel replies 200 OK on port 48080 haproxy will call port 8080 for the real service otherwise the node will be declared in error state.
In psentinel configuration we'll have

```
[ourapp]
port=48080
check_script=/usr/local/psentinel/test_ourapp.py
check_script_interval=120
[general]
#Log level 0..4 (0=min level, 4=max level)
log_level=4
pid_file=/var/run/psentinel/psentinel.pid
sock_file=/var/run/psentinel/psentinel.sock
```

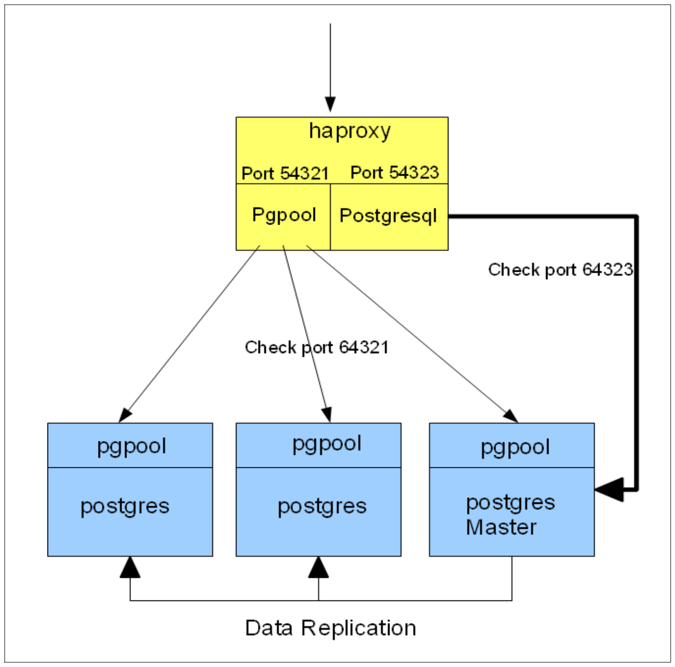
The test_ourapp.py is the smart script that we have to write to check exactly our application. If it fails (returns a non zero result) psentinel replies 500 to haproxy and it jumps our node.

Now suppose that we want to upgrade our application in production.We can manually enable and disable the haproxy backend.
The steps could be:

1) execute *psentinel -c disable ourapp* → returns 503 to haproxy and haproxy exclude the node
2) upgrade application
3) check the script status with *psentinel -c status ourapp.* Meanwhile psentinel still call, in a different thread, our check script and we can see the result in “Last check return status” line (and the time too)
4) if everything is ok issue *psentinel -c enable ourapp* → return 200 to haproxy and haproxy re-include our node in back-end.

Case 2

We have three postgresql back-end servers with pgpool for query caching and master/server replica. We want to create a rule in haproxy to balancing all read-only requests between pgpool nodes but we want to send read/write operations (as insert and update)only toward postgres master node . Obviously if we change master (e.g we elect a new server as master) we want that haproxy automatically selects the new one.



The figure shows a possible configuration.
We define two front-end in haproxy:
One for pgpool on port 54321 and the other for postgresql read/write in master only on port 54323. Clients that do select-only query will open the port 54321 the others will open th 54323 port.
The pgpool front-end balances select queries to three pgpool back-end nodes.
Postgresql front-end send the read/write operations to one node only , the node declared as master in replication (the others are simply server and receive updates from master)
The requirements here are:
1) if the DBA declares as "master" one of others, haproxy have to switch back-end automatically
2) if an node upgrade is needed the DBA can pause pgpool and postgres haproxy check but he wants to be able to use both for testing until he is confident with upgrade. Then the DBA can re-activate node.

3) If one of them fails obviously should be possible disable them automatically from psentinel and check_script.

In haproxy configuration:

```
 frontend db_read
     bind :54321
     mode tcp
     default_backend db_read

frontend db_write
     bind :54323
     mode tcp
     default_backend db_write
  ...
  backend db_read
    balance leastconn
    mode tcp
    option httpchk OPTIONS * HTTP/1.0
    server db1 db1:6432 weight 1 check port 64321 #pgpool is waiting on 6432
    server db2 db2:6432 weight 1 check port 64321
    server db3 db2:6432 weight 1 check port 64321

  backend db_write
    balance leastconn
    mode tcp
    option httpchk OPTIONS * HTTP/1.0
    server db1 db1:5432 weight 1 check port 64323
    server db2 db2:5432 backup check port 64323
    server db3 db3:5432 backup check port 64323
```

For psentinel configuration we have:

```
[pgpool]

port=64321
pausable=yes
start_paused=no
check_script=/usr/local/psentinel/test_pgpool.sh
check_script_interval=15

[pgwrite]

port=64323
pausable=yes
start_paused=no
check_script=/usr/local/psentinel/test_pgwrite.sh
check_script_interval=10

[general]
#Log level 0..4 (0=min level, 4=max level)
log_level=4
pid_file=/var/run/psentinel/psentinel.pid
sock_file=/var/run/psentinel/psentinel.sock
```

We declared all services pausable. It means that we can disable them with a single command *psentinel -c pause* (*psentinel -c run* to re-enable).
Now is interesting to see the *test_pgwrite.sh* script. This script check if the postgresql is in master mode. A possibility could be

```
#!/bin/sh
ps -fupostgres | grep "wal sender" 2>&1 > /dev/null
exit $?
```

This simply check the running postgresql executable. If there is the word “sender” between his arguments then the postgresql is in master mode and haproxy have to select it. When the DBA wants to change master this script will fail here and will succeed in the other server and haproxy automatically select it.
Th pg_pool.sh can be as simple as we like. It could be a plain ps or a port check.

Compile and Install

Short story:

make && make install   (from root user)

Long story:
psentinel needs root permission to create dirs /var/run/psentinel , /etc/psentinel and /usr/local/psentinel. I prefer to create a specific user and run psentinel as that specific user.
So probably the best thing to do is :

```
$> adduser psentinel
$> sudo make install
$> chown psentinel.psentinel /usr/local/psentinel -R
$> chown psentinel.psentinel /var/run/psentinel -R
```

After this we can execute the psentinel using

```
$> sudo su -c “/usr/local/psentinel/psentinel -d -f /etc/psentinel/psentinel.cfg” psentinel
```