HAProxy: Zero downtime reloads with HAProxy 1.8 on Ubuntu 16.04 with Systemd

The reload functionality in HAProxy till now has always been "not perfect but good enough", perhaps dropping a few connections under heavy load but within parameters everyone was willing to accept. And because of the potential impact, a reload was typically only done during non-peak traffic times.

But with the popularity of microservices, containerization, continuous deployment, and dynamically scalable architecture, it has become critical for our load balancers to provide zero downtime reloads because reloading can potentially happen every few seconds even during peak production load.

There have been some seminal pieces written on how to achieve this level of availability with HAProxy. Yelp Engineering wrote up how to use qdiscs to delay the SYN packets, then followed up with using a combination of Nginx and HAProxy communicating over unix sockets. An alternative solution used two instances of HAProxy with an iptables flip.

But now with the ability in HAProxy 1.8 to pass listening sockets from the old process, along with Linux kernel 3.9 support of SO_REUSEPORT we finally have a solution that doesn't feel like an ingenious hack of the Linux kernel and networking stack.

This article is a recipe for deploying and then validating the latest HAProxy 1.8 with zero downtime support on Ubuntu 16.04 using Systemd. If you want to implement this on Ubuntu 14.04, see my other article.

Here is a Vagrantfile that can build the custom HAProxy for you, or continue reading for manual steps.

Install HAProxy 1.8.x

Install haproxy 1.8 using a ppa.

```
$ sudo apt-get install software-properties-common -y
$ sudo add-apt-repository ppa:vbernat/haproxy-1.8 -y

$ sudo apt-get update

$ sudo apt-cache policy haproxy
$ sudo apt-get install haproxy -y

$ haproxy -v
HA-Proxy version 1.8.8-1ppa1~xenial 2018/04/19

# which haproxy
/usr/sbin/haproxy
```

Custom Startup Scripts

Place a custom Systemd service file into '/lib/systemd/system'.

```
$ sudo wget
https://raw.githubusercontent.com/fabianlee/blogcode/master/haproxy/ubuntu1604/haproxy.service -O
/lib/systemd/system/haproxy.service

$ sudo systemctl enable haproxy.service

$ sudo systemctl daemon-reload
```

And then modify '/etc/default/haproxy', which is an environment file read by Systemd. Append a line to the file:

```
$ echo 'RELOADOPTS="-x /run/haproxy/admin.sock"' | sudo tee -a /etc/default/haproxy
```

The '-x' option is used in the 'ExecReload' action of the service file and specifies the unix socket used to transfer the listening sockets from the old process. This is the same socket referenced later in the haproxy.cfg 'expose-fd listeners'.

Note that in newer 1.8 releases, the 'haproxy-systemd-wrapper' is obsolete and no longer necessary. There are many example online of using the wrapper, but they are out of date.

SSL cert

As a prerequisite to configuring HAProxy for TLS traffic, we need to create a self-signed cert for the server host.

You can run the script below or follow the instructions I describe in my article here:

```
$ sudo wget https://raw.githubusercontent.com/fabianlee/blogcode/master/haproxy/selfsigned.sh
$ sudo chmod 755 selfsigned.sh
$ ./selfsigned.sh
```

HAProxy configuration

Download the custom haproxy.cfg then modify the filename to your secure certificate. Instead of 'REPLACEME.pem', use the cert name generated in the last section.

```
$ cd /etc/haproxy
$ sudo touch haproxy.cfg
$ sudo cp haproxy.cfg haproxy.cfg.orig
$ sudo wget https://raw.githubusercontent.com/fabianlee/blogcode/master/haproxy/haproxy.cfg -O
haproxy.cfg
$ sudo sed -i "s/REPLACEME/`hostname -f`/" haproxy.cfg
```

The important line in this config is 'expose-fd listeners' which enables the seamless reload functionality.

This would also be a good time to place a quick shell script that allows us to restart HAProxy at different frequencies.

```
$ sudo wget https://raw.githubusercontent.com/fabianlee/blogcode/master/haproxy/haproxytest.sh -O
/usr/sbin/haproxytest.sh
```

```
$ sudo chmod 755 /usr/sbin/haproxytest.sh
```

This script is a simple loop where we pass the type of HAProxy restart we want (restart|reload) as well as the number of seconds between restarts. Usage is described later in this article.

HAProxy logging

First, create the log file manually to avoid permissions issues.

```
$ sudo touch /var/log/haproxy.log
$ sudo chown haproxy:haproxy /var/log/haproxy.log
$ sudo chmod ug+r+w /var/log/haproxy.log
```

Then modify '/etc/rsyslog.conf' so that it is listening on UDP port 514.

```
$ sudo sed -i '/load=\"imudp\"/s/^#//' /etc/rsyslog.conf
$ sudo sed -i '/type=\"imudp\"/s/^#//' /etc/rsyslog.conf
```

And restart rsyslog

```
$ sudo systemctl restart rsyslog
```

HAProxy logs will go to "/var/log/haproxy.log". If you want to see the Systemd logs (latest entries at the top):

```
$ sudo journalctl -u haproxy.service -r
```

Node.js backend content

What we have currently is an HAProxy instance that has no content to serve. The haproxy.cfg has a backend that grabs content from a pool of localhost ports at 9000, 9001, and 9002.

We will use a simple Node.js server to deliver content on these ports. Open a new console to the HAProxy host so that we can run the server in the foreground and see its console output as calls are made.

```
$ cd ~
$ sudo apt-get install nodejs nodejs-legacy -y

$ wget https://raw.githubusercontent.com/fabianlee/blogcode/master/haproxy/server.js

$ sudo ufw allow 9000:9002/tcp

$ node server.js 9000
Server started at 9000

started server on 0.0.0.0:9000
```

```
started server on 0.0.0.0:9001
started server on 0.0.0.0:9002
```

Smoke Test

We have taken a lot of steps to get here, it would be prudent to make sure that Node.js and HAProxy are correctly serving pages.

```
$ sudo systemctl restart haproxy.service


$ sudo apt-get install curl -y
$ curl https://localhost --insecure

{"user-agent":"curl/7.35.0","host":"localhost","accept":"*/*","x-forwarded-proto":"https","x-forwarded-
for":"127.0.0.1","connection":"close"}
endpoint: 0.0.0.0:9000
requestcount: 1
echo: undefined

$ curl https://localhost --insecure

{"user-agent":"curl/7.35.0","host":"localhost","accept":"*/*","x-forwarded-proto":"https","x-forwarded-
for":"127.0.0.1","connection":"close"}
endpoint: 0.0.0.0:9001
requestcount: 2
echo: undefined
```

And in the other console where the Node.js server is running you should see output that looks like:

```
served request: 1 on port 9000
served request: 2 on port 9001
```

This shows that our HAProxy instance is serving as the SSL termination point, going back to a pool of HTTP instances.

Load Tester Installation

With the setup and smoke test complete, now we want to put a high-concurrency load on the system while we reload HAProxy at a high frequency. Run the load generator from a different VM than where HAProxy is installed to minimize resource interference and simulate true network access.

For this article I will use Apache Workbench which has proven to be good at reporting back errors; whether those be at connection, handshake, or mid-stream.

The one issue with Apache Workbench is that it has hardcoded fatal failure after 10 exceptions, which doesn't work for our purposes since we are purposely testing failures. So I have patched the program so that we can pass an '-R' argument that allows execution to continue despite the number of individual errors.

Apache Workbench is part of the Apache HTTP download, so below are instructions for creating a custom binary. Or you can use my Vagrantfile (vagrant up) or Dockerfile (docker build -t my-ab && docker run -it my-ab) .

```
$ wget http://apache.mirrors.pair.com/httpd/httpd-2.4.33.tar.gz
$ tar xvfz httpd-2.4.33.tar.gz
$ cd httpd-2.4.33

$ cp support/ab.c support/ab.c.old
$ wget https://raw.githubusercontent.com/fabianlee/blogcode/master/haproxy/ab.c -O support/ab.c

$ sudo apt-get install libapr1-dev libaprutil1-dev libpcre3 libpcre3-dev -y
$ ./configure
$ make

$ support/ab -V
This is ApacheBench, Version 2.3mymod <$Revision: 1807734 $>

$ sudo cp support/ab /usr/sbin/ab
```

Notice that the version reported back is "2.3mymod" which proves it is our custom binary. Here is the syntax for running the load test:

```
$ ab -r -R -c <numberUsers> -n <numberOfRequests> -f TLS1.2 -l https://<HAProxyHost>:443/
```

As an example, running as 10 parallel users with 100 requests total, the results should look something like below, and make sure you have the trailing slash on the URL as shown below:

```
$ ab -r -R -c 10 -n 100 -f TLS1.2 -l https://src1:443/
.
.
.
Server Software:
Server Hostname:        src1
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,2048,256
TLS Server Name:        src1

Document Path:          /
Document Length:        Variable

Concurrency Level:      10
Time taken for tests:   0.340 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      36100 bytes
HTML transferred:       18600 bytes
Requests per second:    293.96 [#/sec] (mean)
Time per request:       34.018 [ms] (mean)
Time per request:       3.402 [ms] (mean, across all concurrent requests)
Transfer rate:          103.63 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        6   16   7.9     14      39
Processing:     6   18   5.5     17      31
Waiting:        6   17   5.5     16      31
Total:         21   34   9.5     30      65
```

```
Percentage of the requests served within a certain time (ms)
  50%     30
  66%     33
  75%     36
  80%     38
  90%     44
  95%     62
  98%     65
  99%     65
 100%     65 (longest request)
```

Changing reload mechanism

In the next section we are going to be running multiple tests. In order to change the reload mechanism from the legacy reload to the new socket transfer, you have to modify the following files:

- **/etc/haproxy/haproxy.cfg** – contains line with 'expose-fd' that enables socket transfer
- **/etc/default/haproxy** – contains RELOADOPTS variable that has '-x' option that passes socket value to Systemd reload command

I have a script that can quickly make these changes for you:

```
$ sudo wget
https://raw.githubusercontent.com/fabianlee/blogcode/master/haproxy/ubuntu1604/switchhaproxy.sh -O
/usr/sbin/switchhaproxy.sh
$ sudo chmod 755 /usr/sbin/switchhaproxy.sh
```

In order to run using the legacy reload mechanism:

- sudo systemctl stop haproxy.service
- ps -ef | grep haproxy (to verify no haproxy processes are running)
- sudo switchhaproxy.sh reload
- sudo systemctl start haproxy.service
- sudo haproxtest.sh reload 0.2

In order to run using the new socket transfer mechanism:

- sudo systemctl stop haproxy.service
- ps -ef | grep haproxy (to verify no haproxy processes are running)
- sudo switchhaproxy.sh reload-socket
- sudo systemctl start haproxy.service
- sudo haproxtest.sh reload 0.2

To run the hard restart test:

- sudo systemctl stop haproxy.service
- ps -ef | grep haproxy (to verify no haproxy processes are running)
- sudo systemctl start haproxy.service
- sudo haproxytest.sh restart 0.2

Run Load Tests

Here are the four variations on restart/reload we will test:

1. **No restarts or reloads** – running 10k requests with absolutely no restarts or reloads. *Expect:* no errors and fast response times
2. **Hard restart** – the HAProxy service will be hard stopped and restarted. *Expect:* lots of errors as connections are killed abruptly
3. **Legacy reload** – attempts a port rebind to the new HAProxy process. *Expect:* some level of errors during high traffic or high frequency reload
4. **Reload using socket** – retrieves and transfers listening sockets from the old process to the new process. *Expect:* no errors

We will run with 200 parallel users, running 10k requests total. Below are the results in table form:

Test	#req/users	#fail	resp avg	95% resp
baseline	10k/200	0	513ms	560ms
restart 0.2	10k/200	4332	147ms	253ms
reload 0.2 (legacy)	10k/200	10	455ms	815ms
reload 0.2 (socket)	10k/200	0	567ms	852ms

These results match well to our expectations. There was a high failure count for hard restarts, and a low percentage error for the legacy reload mechanism. The new reload based on socket transfer was a success and yielded 100% success rates.

Summary

The older reload mechanism used by HAProxy was sufficient for most cases, but under high load or triggering reloads of high frequency it would drop connections.

If you are using HAProxy 1.8 and Linux kernel >=3.9 then you can take advantage of a new reload mechanism which can transfer sockets from the old process to the new process without dropping any connections.

This enables dynamic scalability and continuous integration all the way into production datacenters.

REFERENCES

https://www.haproxy.com/blog/truly-seamless-reloads-with-haproxy-no-more-hacks/

https://engineeringblog.yelp.com/2015/04/true-zero-downtime-haproxy-reloads.html

https://engineeringblog.yelp.com/amp/2017/05/taking-zero-downtime-load-balancing-even-further.html

https://githubengineering.com/glb-part-2-haproxy-zero-downtime-zero-delay-reloads-with-multibinder/

https://serverfault.com/questions/580595/haproxy-graceful-reload-with-zero-packet-loss

http://inside.unbounce.com/product-dev/haproxy-reloads/

https://www.mail-archive.com/haproxy@formilux.org/msg25632.html (original patch mailing list discussion)

http://louwrentius.com/how-to-compile-haproxy-from-source-and-setup-a-basic-configuration.html

http://www.haproxy.org/download/1.4/src/snapshot/?C=M;O=A (explains that haproxy-ss has latest source code with applied patches)

http://zhgwenming.blogspot.com/2012/09/tcp-sysctl.html (tcp related sysctl)

https://cbonte.github.io/haproxy-dconv/1.8/snapshot/configuration.html (haproxy 1.8 docs)

https://httpd.apache.org/docs/2.4/programs/ab.html (apache workbench docs)

https://launchpad.net/~ondrej/+archive/ubuntu/apache2 (ppa for apache)

http://apache.mirrors.pair.com/httpd/ (download apache src)

http://ibm-blue-box-help.github.io/help-documentation/troubleshooting/How_many_connections_can_HAProxy_handle/ (using gobench)

https://serverfault.com/questions/580595/haproxy-graceful-reload-with-zero-packet-loss (draining using iptables)

https://stackoverflow.com/questions/9436860/apache-httpd-setup-and-installation (apache install)

https://www.mail-archive.com/haproxy@formilux.org/msg26250.html (note about the master work mode making the haproxy-systemd-wrapper obsolete)

NOTES

Build latest HAProxy 1.8.x snapshot

```
$ sudo apt-get install build-essential libssl-dev -y

$ cd /usr/src

$ sudo wget http://www.haproxy.org/download/1.8/src/snapshot/haproxy-ss-LATEST.tar.gz

$ sudo tar xfz haproxy-ss-LATEST.tar.gz
$ sudo mv haproxy-ss-<YYYYMMDD> haproxy

$ cd haproxy

$ cat VERSION
1.8-<VER>
$ sudo chmod ugo+w VERSION
$ sudo echo "1.8-<VER>mymod" > VERSION
```

Then make and compile the HAProxy 1.8 binary and copy it into the location expected by the Systemd scripts.

```
$ sudo make TARGET=linux2628 CPU=native USE_LIBCRYPT=1 USE_LINUX_SPLICE=1 USE_LINUX_TPROXY=1
USE_OPENSSL=1 USE_LIBPCRE=1
$ sudo make install
$ /usr/local/sbin/haproxy -v
HA-Proxy version 1.8-dev2mymod-64cc49c 2017/10/13
```

```
$ ./haproxy -v
HA-Proxy version 1.8-dev2mymod-64cc49c 2017/10/13

$ sudo systemctl stop haproxy.service
$ sudo cp ./haproxy /usr/sbin/haproxy
```

Docker build of custom Apache Workbench

wget https://raw.githubusercontent.com/fabianlee/blogcode/master/haproxy/ab/Dockerfile

docker build -t my-ab .
docker image rm my-ab
docker run -it my-ab -c 2 -n 10 https://www.google.com/

Fabian / October 16, 2017 / Containers, Linux / downtime, haproxy, reload, restart, seamless, sla, systemd, zero