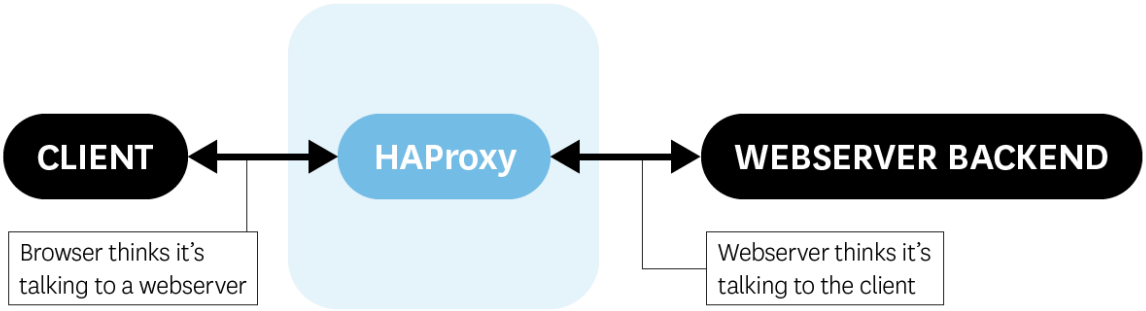
Summary

What is HAProxy?

HAProxy is an open source solution for load balancing and reverse proxying both TCP and HTTP requests—and, in keeping with the abbreviation in its name, it is high availability. HAProxy can continue to operate in the presence of failed backend servers, handling crossover reliably and seamlessly. It also has built-in health checks that will remove a backend if it fails several health checks in a row. With dynamic routing you can transfer incoming traffic to a variety of backend servers, fully configurable with Access Control Lists (ACLs).

HAProxy consistently performs on par or better in benchmarks against other popular reverse proxies like http-proxy or the NGINX webserver. It is a fundamental element in the architecture of many high-profile websites such as GitHub, Instagram, Twitter, Stack Overflow, Reddit, Tumblr, Yelp, and many more.

Like other load balancers or proxies, HAProxy is very flexible and largely protocol-agnostic—it can handle anything sent over TCP.



Glossary

Frontend - A frontend is what a client connects to. As requests enter the load balancer—and as responses are returned to the client—they pass through the frontend. So, it has access to end-to-end timings, message sizes, and health indicators that encompass the whole request/response lifecycle.

Backend - A backend is a pool of load-balanced servers. HAProxy sends requests to a backend and then receives a response from one of the active servers

Session. Session in HAproxy have different approach.

For frontend section a session mean a HTTP session. An HTTP session is initiated by a Web browser each time you visit a website. While each page visit constitutes an individual session, the term is often used to describe the entire time you spend on the website. For example, when you purchase an item on an ecommerce site, the entire process may be described as a session, even though you navigated through several different pages.

For backend session mean connections to backend server, so 1 frontend session can have more backend sessions

Connection - connection between client and server (TCP connection). A connection must be established to start exchange data. For a client to reach backend server (end-to-end connection) must have 2 connections: client to HAProxy and HAProxy to backend server.

HTTP request and response - The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers. HTTP works as a request-response protocol between a client and server. A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

Each request and each response have a time to execute, can generate errors

Queue - applies only to backends and shows how long clients are waiting for a server to become available. We must have queue to avoid server congestion.

ACL - in HAProxy allow you to test various conditions and perform a given action based on those tests. These conditions cover just about any aspect of a request or response such as searching for strings or patterns within them, checking the IPs they are from, recent request rates (via stick tables), TLS status, etc. The action you take can include making routing decisions, redirecting requests, returning static responses and so much more

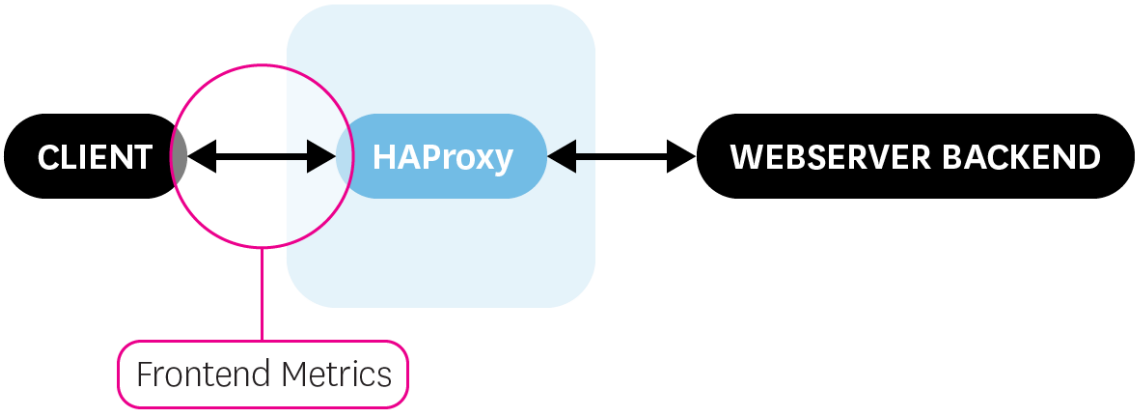
Key HAProxy performance metrics

A properly functioning HAProxy setup can handle a significant amount of traffic. However, because it is the first point of contact, poor load balancer performance will increase latency across your entire stack. The best way to ensure proper HAProxy performance and operation is by monitoring its key metrics in three broad areas:

Frontend metrics such as client connections and requests
Backend metrics such as availability and health of backend servers
Health metrics that reflect the state of your HAProxy setup

Correlating frontend metrics with backend metrics gives you a more comprehensive view of your infrastructure and helps you quickly identify potential hotspots.

Frontend metrics



Frontend metrics provide information about the client's interaction with the load balancer itself.

Name	Description
req_rate	HTTP requests per second
rate	Number of sessions created per second
session utilization (computed)	Percentage of sessions used (*scur / slim * 100*)
ereq	Number of request errors
dreq	Requests denied due to security concerns (ACL-restricted)
hrsp_4xx	Number of HTTP client errors
hrsp_5xx	Number of HTTP server errors
bin	Number of bytes received by the frontend
bout	Number of bytes sent by the frontend

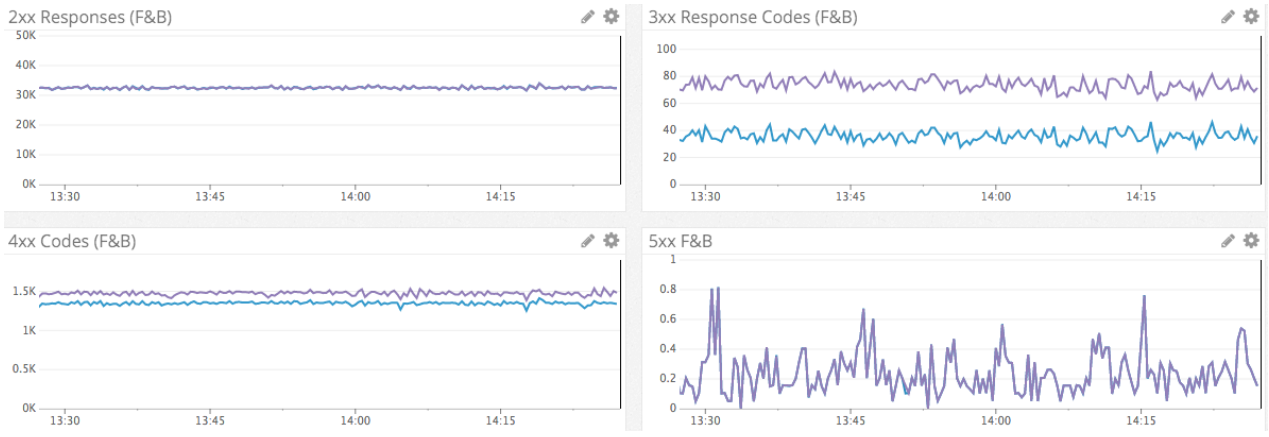
Again a note about terminology: HAProxy documentation often uses the terms sessions, connections, and requests together, and it is easy to get lost. Each client that interacts with HAProxy uses one session. A session is composed of two connections, one from the client to HAProxy, and the other from HAProxy to the appropriate backend server. Once a session has been created (i.e. the client can talk to the backend through HAProxy), the client can begin issuing requests. A client will typically use one session for all of its requests, with sessions terminating after receiving no requests during the timeout window.

Metrics to watch in frontend:

req_rate: The frontend request rate measures the number of requests received over the last second. Keeping an eye on peaks and drops is essential to ensure continuous service availability. In the event of a traffic spike, clients could see increases in latency or even denied connections. Tracking your request rate over time gives you the data you need to make more informed decisions about HAProxy's configuration.

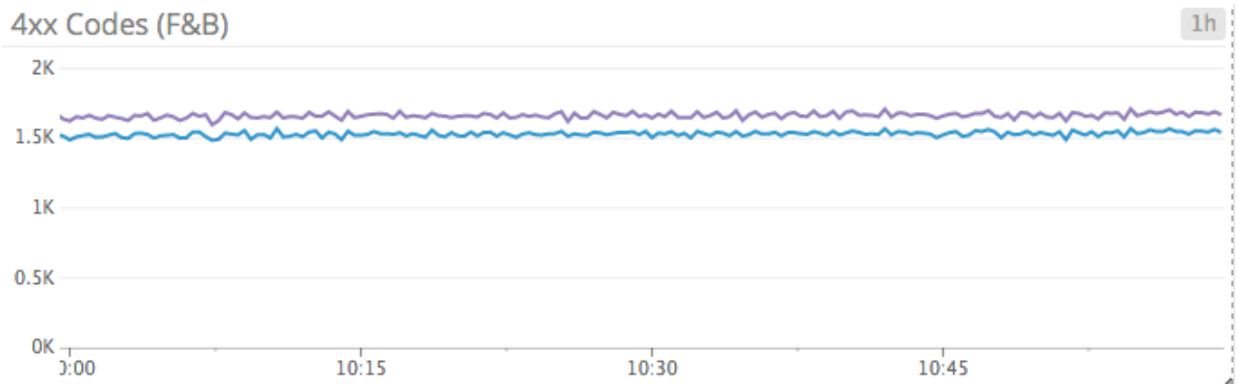
rate: HAProxy allows you to configure the maximum number of sessions created per second. If you are not behind a content delivery network (CDN), a significant spike in the number of sessions over a short period could cripple your operations and bring your servers to their knees.

Tracking your session creation rate over time can help you discern whether a traffic spike was a unique event or part of a larger trend. You can then set a limit based on historic trends and resource availability so that a sudden and dramatic spike does not result in a denial of service.



Frontends are purple, backends blue

hrsp_4xx and hrsp_5xx: HAProxy exposes the number of responses by HTTP status code. Ideally, all responses forwarded by HAProxy would be class **2xx** codes, so an unexpected surge in the number of other code classes could be a sign of trouble. Correlating the denial metrics with the response code data can shed light on the cause of an increase in error codes. No change in denials coupled with an increase in the number of **404** responses could point to a misconfigured application or unruly client. Over the course of our internal testing, we graphed the frontend and backend metrics together, one graph per response code, with interesting results.



The difference in the **4xx** responses is explained by the tendency of some browsers to preconnect, sometimes resulting in a **408** (request timeout) response. If you are seeing excessive **4xx** responses and suspect them to be **408** codes, you can try this temporary workaround and see if that reduces the number of **4xx** responses.

bin/bout: When monitoring high traffic servers, network throughput is a great place to start. Observing traffic volume over time is essential to determine if changes to your network infrastructure are needed. A 10Mb/s pipe might work for a startup getting its feet wet, but is insufficient for larger traffic volumes. Tracking HAProxy's network usage will empower you to scale your infrastructure with your ever-changing needs.

Frontend metrics to alert on:

session usage (computed): For every HAProxy session, two connections are consumed—one for the client to HAProxy, and the other for HAProxy to your backend. Ultimately, the maximum number of connections HAProxy can handle is limited by your configuration and platform (there are only so many file descriptors available).

Alerting on this metric is essential to ensure your server has sufficient capacity to handle all concurrent sessions. Unlike requests, upon reaching the session limit HAProxy will deny additional clients until resource consumption drops. Furthermore, if you find your session usage percentage to be hovering above 80%, it could be time to either modify HAProxy's configuration to allow more sessions, or migrate your HAProxy server to a bigger box.

Remember that sessions and connections are related—consistently high traffic volumes might necessitate an increase in the number of maximum connections HAProxy allows or even require you to add an additional HAProxy instance. Upon reaching its connection limit, HAProxy

will continue to accept and queue connections unless or until the backend server handling the requests fails.

This metric is not emitted by HAProxy explicitly and must be calculated by dividing the current number of sessions (scur) by the session limit (slim) and multiplying the result by 100. Keep in mind that if HAProxy's keep-alive functionality is enabled (as it is by default), your number of sessions includes sessions not yet closed due to inactivity.

dreq: HAProxy provides sophisticated information containment controls out of the box with ACLs. Properly configured ACLs can prevent HAProxy from serving requests that contain sensitive material. Similarly to a web application firewall, you can use ACLs to block database requests from non-local connections, for example, but ACLs are highly configurable—you can even deny requests or responses that match a regex.

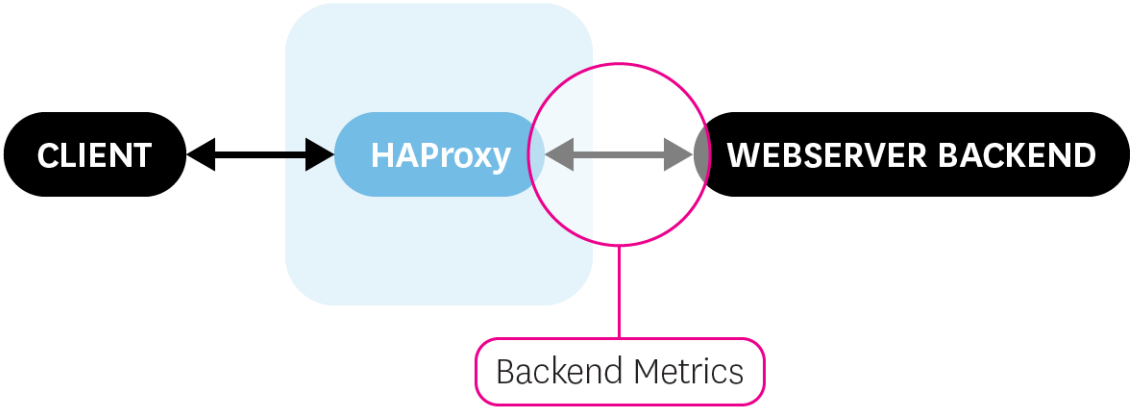
This metric tracks the number of requests denied due to security restrictions. You should be alerted in the event of a significant increase in denials—a malicious attacker or misconfigured application could be to blame. More information on designing ACLs for HAProxy can be found in the documentation. An increase in denied requests will subsequently cause an increase in **403 Forbidden** codes.

Correlating the two can help you discern the root cause of an increase in **4xx** responses. **ereq**: Similar to dreq, this metric exposes the number of request errors. Client-side request errors could have a number of causes:

- client terminates before sending request
- read error from client
- client timeout
- client terminated connection
- request was tarpitted/subject to ACL

Under normal conditions, it is acceptable to (infrequently) receive invalid requests from clients. However, a significant increase in the number of invalid requests received could be a sign of larger, looming issues. For example, an abnormal number of terminations or timeouts by numerous clients could mean that your application is experiencing excessive latency, causing clients to manually close their connections.

Backend metrics



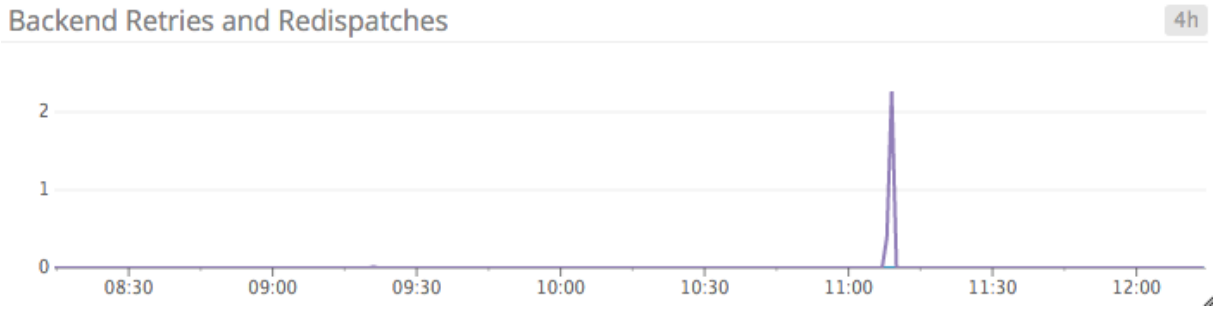
Backend metrics measure the communication between HAProxy and your backend servers that handle client requests. Monitoring your backend is critical to ensure smooth and responsive performance of your web applications.

Name	Description
rtime	Average backend response time (in ms) for the last 1,024 requests
econ	Number of requests that encountered an error attempting to connect to a backend server
dresp	Responses denied due to security concerns (ACL-restricted)
eresp	Number of requests whose responses yielded an error
qcur	Current number of requests unassigned in queue
qtime	Average time spent in queue (in ms) for the last 1,024 requests
wredis	Number of times a request was redispatched to a different backend
wretr	Number of times a connection was retried

Backend metrics to watch:

econ: Backend connection failures should be acted upon immediately. Unfortunately, the econ metric not only includes failed backend requests but additionally includes general backend errors, like a backend without an active frontend. Thankfully, correlating this metric with eresp and response codes from both your frontend and backend servers will give you a better idea of the causes of an increase in backend connection errors.

dresp: In most cases your denials will originate in the frontend (e.g., a user is attempting to access an unauthorized URL). However, sometimes a request may be benign, yet the corresponding response contains sensitive information. In that case, you would want to set up an ACL to deny the offending response. Backend responses that are denied due to ACL restrictions will emit a **502** error code. With properly configured access controls on your frontend, this metric should stay at or near zero. Denied responses and an increase in **5xx** responses go hand-in-hand. If you are seeing a large number of **5xx** responses, you should check your denied responses to shed some light on the increase in error codes.



wretr: Some dropped or timed-out connections are to be expected when connecting to a backend server. The retry rate represents the number of times a connection to a backend server was retried. This metric is usually non-zero under normal operating conditions. Should you begin to see more retries than usual, it is likely that other metrics will also change, including econ and eresp. Tracking the retry rate in addition to the above two error metrics can shine some light on the true cause of an increase in errors; if you are seeing more errors coupled with low retry rates, the problem most likely resides elsewhere. **wredis**: The redispatch rate metric tracks the number of times a client connection was unable to reach its original target, and was subsequently sent to a different server. If a client holds a cookie referencing a backend server that is down, the default action is to respond to the client with a **502** status code. However, if you have enabled option redispatch in your haproxy.cfg, the request will be sent to any available backend server and the cookie will be ignored. It's a matter of preference whether you want to trade user session persistence for smoother transitions in the face of failed backends.

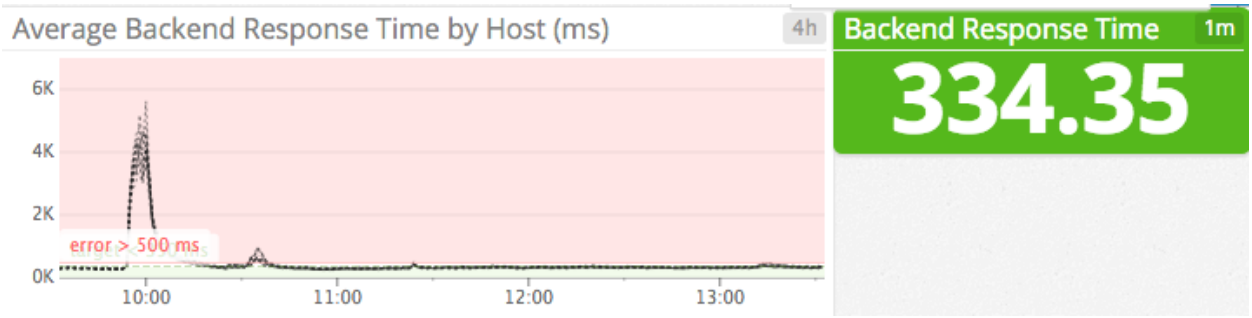
Metrics to alert on:

qcur: If your backend is bombarded with connections to the point you have reached your global maxconn limit, HAProxy will seamlessly queue new connections in your system kernel's socket queue until a backend server becomes available. The qcur metric tracks the current number of connections awaiting assignment to a backend server. If you have enabled cookies and the listed server is unavailable, connections will be queued until the queue timeout is reached. However, if you have set the option redispatch directive in your global HAProxy configuration, HAProxy can break the session's persistence and forward the request to any available backend.

Keeping connections out of the queue is ideal, resulting in less latency and a better user experience. You should alert if the size of your queue exceeds a threshold you are comfortable with. If you find that connections are consistently enqueueing, configuration changes may be in order, such as increasing your global maxconn limit or changing the connection limits on your individual backend servers.

An empty queue is a happy queue

qtime: In addition to the queue size, HAProxy exposes the average time spent in queue via the qtime metric. This metric represents an average of the last 1,024 requests, so an abnormally large queue time for one connection could skew results. It goes without saying that minimizing time spent in the queue results in lower latency and an overall better client experience. Each use case can tolerate a certain amount of queue time but in general, you should aim to keep this value as low as possible.



rtime: Tracking average response times is an effective way to measure the latency of your load-balancing setup. Generally speaking, response times in excess of 500 ms will lead to degradation of application performance and customer experience. Monitoring the average response time can give you the upper hand to respond to latency issues before your customers are substantially impacted. *Keep in mind* that this metric will be zero if you are not using HTTP (see #60).

eresp: The backend error response rate represents the number of response errors generated by your backends. This includes errors caused by data transfers aborted by the servers as well as write errors on the client socket and failures due to ACLs. Combined with other error metrics, the backend error response rate helps diagnose the root cause of response errors. For example, an increase in both the backend error response rate and denied responses could indicate that clients are repeatedly attempting to access ACL-ed resources.

Health metrics

HAProxy can also expose information about the health of each front and backend server, in addition to the metrics listed above. Health checks are not enabled by default, and require you to set the check directive in your HAProxy configuration. Once set up, HAProxy will regularly perform health checks on all enabled servers. If a health check fails three times in a row (configurable with the rise directive), it is marked in a **DOWN** state.

Monitoring the health of your HAProxy servers gives you the information you need to quickly respond to outages as they occur.

Health checks are highly configurable, with specific checks available for MySQL, SMTP, Redis and others. Refer to the documentation to take full advantage of HAProxy's powerful health-checking features.

Collecting the HAProxy metrics you need

Now that you know the key HAProxy metrics to monitor, it's time to collect them! You can either use HAProxy's built-in tools or a third-party tool. HAProxy gives you two means by which you can monitor its performance: via a status page, or via sockets. Both of the methods below give you an immediate and detailed view into the performance of your load balancer. The main difference between the two is that the status page is static and read-only, whereas the socket interface allows you to modify HAProxy's configuration on the fly.

Stats page

The most common method to access HAProxy metrics is to enable the stats page, which you can then view with any web browser. This page is not enabled out of the box, and requires modification of HAProxy's configuration to get it up and running.

Configuration

To enable the HAProxy stats page, add the following to the bottom of the file /etc/haproxy/haproxy.cfg (adding your own username and password to the final line):

```
listen stats :9000 #Listen on localhost port 9000
mode http
stats enable #Enable statistics
stats hide-version #Hide HAPRoxy version, a necessity for any public-facing site
stats realm Haproxy\ Statistics #Show this text in authentication popup (escape space characters with backslash)
stats uri /haproxy_stats #The URI of the stats page, in this case localhost:9000/haproxy_stats
stats auth Username:Password #Set a username and password
```

For HAProxy 1.6+ :

```
listen stats
bind :9000
mode http
stats enable #Enable statistics
stats hide-version #Hide HAPRoxy version, a necessity for any public-facing site
stats realm Haproxy\ Statistics #Show this text in authentication popup (escape space characters with backslash)
stats uri /haproxy_stats #The URI of the stats page, in this case localhost:9000/haproxy_stats
stats auth Username:Password #Set a username and password
```

This sets up a listener on port 9000 in HTTP mode with statistics enabled.

Next you'll need to restart HAProxy, which can interrupt client sessions and cause downtime. If you want to be very careful about how you restart HAProxy, check out Yelp's research on the least disruptive means by which you can reload HAProxy's configuration.

If you're comfortable with session interruption, you can restart HAProxy with sudo service haproxy restart. After restarting HAProxy with your modified configuration, you can access a stats page like the one below after authenticating via the URL: http://<YourHAProxyServer>:9000/haproxy_stats

You can even mouse over some stats for more information, as seen in this screenshot:

If you prefer machine-readable output, you can choose to view the page as CSV output instead by appending ;csv to the end of your stats URL. The stats page is great for a quick, human-readable view of HAProxy.

However, there are a couple of downsides:

- static: the page must be refreshed to be updated
- ephemeral: old statistics are lost on each refresh

If you plan on scraping HAproxy's metrics for a script or otherwise, communicating over the socket interface is a much more practical method.

Unix Socket Interface

The second way to access HAProxy metrics is via a Unix socket. There are a number of reasons you may prefer sockets to a web interface: security, easier automation, or the ability to modify HAProxy's configuration on the fly. If you are not familiar with Unix interprocess communication, however, you may find the statistics page served over HTTP to be a more viable option. Enabling HAProxy's socket interface is similar to the HTTP interface—to start, open your HAProxy configuration file (typically located in /etc/haproxy/haproxy.cfg). Navigate to the global section and add the following lines:

```
global #Make sure you add it to the global section

    stats socket /var/run/haproxy.sock mode 600 level admin
    stats timeout 2m #Wait up to 2 minutes for input
```

Although only the stats socket line is necessary to open the socket, setting a timeout is useful if you plan on using the socket interactively. For the curious, the mode 600 level admin parameters tell HAProxy to set the permissions of the socket to allow only the owner to read and write to it (mode 600) with administrative privileges. Admin rights let you alter HAProxy's configuration through the socket interface; without them the socket is essentially read-only.

Socket communication

Accessing HAProxy's interface requires a means to communicate with the socket, and the popular netcat tool is perfect for the job. Most *NIX platforms have a version of nc either installed by default or available from your package manager of choice, but if not you can always compile from source.

Although the HAProxy's official socket documentation recommends socat as the preferred socket client, during our tests we found that many popular versions of socat lack readline support (necessary for using the socket interactively). Using OpenBSD's nc is just as easy and works out

of the box. HAProxy's socket interface offers two modes of operation: *interactive* or *non-interactive*.

In **non-interactive mode**, you can send one string of commands to the socket before the connection closes. This is the most common method for automated monitoring tools and scripts to access HAProxy statistics. Sending multiple commands in this mode is supported by separating each command with a semicolon. In the examples to follow we will assume your HAProxy socket is located at /var/run/haproxy.sock.

The following command will return:
- information about the running HAProxy process (pid, uptime, etc)
- statistics on your frontend and backend (same data returned from your stats URI) echo "show info;show stat" | nc -U /var/run/haproxy.sock

Interactive mode is a similar one-liner. Running: nc -U /var/run/haproxy.sock connects to the socket and allows you to enter one command. Entering the prompt command drops you into an interactive interface for the HAProxy socket.

```
$ nc -U /var/run/haproxy.sock
$ prompt
> show info
Name: HAProxy
Version: 1.6.1
Release_date: 2015/10/20
Nbproc: 1
Process_num: 1
Pid: 6515
Uptime: 0d 0h08m22s
Uptime_sec: 502
Memmax_MB: 0
Ulimit-n: 4030
Maxsock: 4030
Maxconn: 2000
Hard_maxconn: 2000
CurrConns: 0
CumConns: 2
CumReq: 2
[...]
```

Send an empty line or quit to exit the prompt. For more information on the available socket commands, refer to the HAProxy documentation.

Show me the metrics!

Once you've enabled the socket interface, getting the metrics is a simple one-liner: echo "show stat" | nc -U /var/run/haproxy.sock This command pipes the text "show stat" into the socket, producing the output below:

http-in,FRONTEND,,,0,85,2000,4655,380562,991165,0,0,14,,,,,OPEN,,,,,,,,,1,2,0,,,,0,0,0,3305,,,,0,0,0,14,4641,0,,0,3305,4655,,,0,0,0,0,,,,,,,,

appname,lamp1,0,0,0,0,,0,0,0,,0,,0,0,0,0,DOWN,1,1,0,1,1,134,134,,1,3,1,,0,,2,0,,0,L4TOUT,,20
02,0,0,0,0,0,0,0,,,,0,0,,,,,-1,,,0,0,0,0,
appname,lamp2,0,0,0,0,,0,0,0,,0,,0,0,0,0,DOWN,1,1,0,1,1,133,133,,1,3,2,,0,,2,0,,0,L4TOUT,,20
02,0,0,0,0,0,0,0,,,,0,0,,,,,-1,,,0,0,0,0,
appname,BACKEND,0,0,0,77,200,4641,380562,988533,0,0,,4641,0,0,0,DOWN,0,0,0,,1,133,13
3,,1,3,0,,0,,1,0,,3292,,,,0,0,0,0,4641,0,,,,,0,0,0,0,0,0,-1,,,0,0,0,0,

As you can see, if you want to access your HAProxy metrics in a human-readable format, the stats page is the way to go.

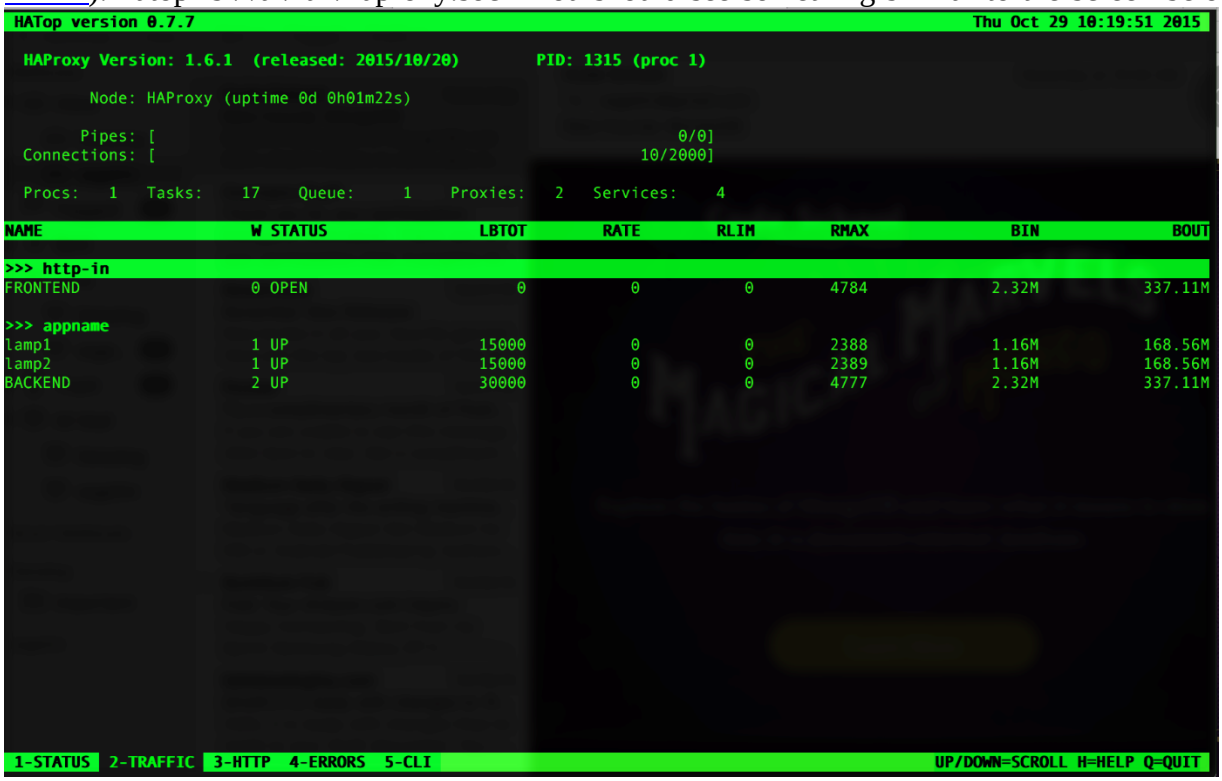
The output metrics are grouped by server. In the snippet above you can see four devices: *http-in*, a frontend; the backend *appname*; and the backend servers associated with *appname*, *lamp1* and *lamp2*. The metric names are abbreviated and some should look familiar: qcur is the current queue size, bin is the number of bytes in, etc. You can find a full list of HAProxy metrics in the documentation.

Third party tools

There is no shortage of third party tools available in the HAProxy community, though many are unmaintained. Luckily, there is HATop.

Though unpatched for over five years, HATop to this day remains the go-to tool for taking a closer look at a running HAProxy service. HATop was designed to mimic the appearance of htop. It is an excellent management and diagnostics tool capable of overseeing the entirety of your HAProxy service stack.

Once downloaded, start HATop with the following command (assuming HATop is in your PATH): hatop -s /var/run/haproxy.sock You should see something similar to the screen below:



With a tool like HATop, you get human-readable metrics (updated in real time), along with the ability to perform common tasks, such as changing backend weights or placing servers into maintenance mode. The HATop documentation has details on what you can do with this useful tool.

Conclusion

HAProxy's built-in tools provide a wealth of data on its performance and health. For spot checking your HAProxy setup, HAProxy's tools are more than enough. For monitoring systems in production, however, you will likely need a dedicated monitoring system that can collect and store your HAProxy metrics at high resolution, and that provides a simple interface for graphing and alerting on your metrics.

Integrating Zabbix and HAProxy

Integration logic

This solution will provide you with a comprehensive template, a lot of controls without loading the HAProxy server with requests. Monitoring process will be done in next way:

a. Script haproxy_discovery.sh will be executed by zabbix and will discovery all the Frontends, Backends and servers under those backends. All data will be returned in JSON format to Zabbix. For reading data zabbix will use data from socket file /var/lib/haproxy/stats
b. The script haproxy_stats.sh will extract ALL data from the socket /var/lib/haproxy/stats and will write them in file /var/tmp/haproxy_stats.cache. Script will take care that data from this file to not be older than 59 sec, if more – the file will be rewrited with the new info from socket.
c. The script haproxy_stats will be executed with parameters $1, $2, $3, $4.
    a. $1 – will provide path to the socket
    b. $2 – name for the backend
    c. $3 – name for backend server according to haproxy.cfg
    d. $4 – the statistic key that we need (specified in item key)

An example:
The string: haproxy_stats.sh $1 $2 $3 $4, where $4 is key status will be executed as: haproxy_stats.sh /var/lib/haproxy/stats www-backend www01 status – will get status info for the backend www-backend unsing socket file /var/lib/haproxy/stats

Verify HAProxy's status

Before you begin, you must verify that HAProxy is set to output metrics over HTTP. To read more about enabling the HAProxy status page, refer to Chapter 2 of this doc. Simply open a browser to the stats URL listed in haproxy.cfg.

Install the Zabbix Agent

The Zabbix Agent is open-source software that collects and reports metrics from all of your hosts so you can view, monitor, and correlate them on the Zabbix platform. Installing the Agent usually requires just a single command. Installation instructions are platform-dependent and can be found here. As soon as the Zabbix Agent is up and running, you should add your host in your Zabbix server and see that is running.

Configure the Agent

* Place `userparameter_haproxy.conf` into `/etc/zabbix/zabbix_agentd.d/` directory, assuming you have Include set in `zabbix_agend.conf`, like so:

```
### Option: Include
# You may include individual files or all files in a directory in the configuration file.
# Installing Zabbix will create include directory in /usr/local/etc, unless modified during the compile time.
#
# Mandatory: no
```

 # Default:
Include=/etc/zabbix/zabbix_agentd.d/
```

Place `haproxy_discovery.sh` into `/etc/zabbix/scripts/` directory and make sure it's executable (`sudo chmod +x /etc/zabbix/scripts/haproxy_discovery.sh`)

Place `haproxy_stats.sh` into `/etc/zabbix/scripts/` directory and make sure it's executable (`sudo chmod +x /etc/zabbix/scripts/haproxy_stats.sh`)

Import `haproxy_v4_template.xml` template via Zabbix Web UI interface (provided by `zabbix-frontend-php` package)

Restart zabbix agent to load new config: systemctl restart zabbix-agent

## Configure HAProxy control socket

Configure HAProxy to listen on `/var/lib/haproxy/stats` by adding/configuring in HAProxy configuration file (/etc/haproxy/haproxy.cfg):

# haproxy.conf snippet
# haproxy read-only non-admin socket
## (user level permissions are required, admin level will work as well, though not necessary)
global
# default usage, through socket
stats socket /var/lib/haproxy/stats mode 666 level admin
## alternative usage, using tcp connection (useful e.g. when haproxy runs inside a docker and zabbix-agent in another)
## replace socket path by ip:port combination on both scripts when using this approach, e.g. 172.17.0.1:9000
#stats socket *:9000

Reload HAProxy conf restarting the HAProxy or reload: service haproxy restart

Test if HAProxy user can read from socket

1. Install socat and nc utilities: yum install nc socat –yum
2. Test: sudo -uhaproxy echo "show info;show stat" | socat stdio unix-connect:/var/lib/haproxy/stats
   We must get data

**Show me the metrics!**

If all is configured rightly, in some time (1h) we must have data. We can set the discovery time from template to 1 minute, but after we will get data, don't forget to put again to 1h/1d, to not make a lot of request to the server

**Conclusion**

In this post we've walked you through integrating HAProxy with Zabbix to visualize your key metrics and notify the right team whenever your infrastructure shows signs of trouble. If you've followed along using your own Zabbix account, you should now have improved visibility into what's happening in your environment, as well as the ability to create automated alerts tailored to your infrastructure, your usage patterns, and the metrics that are most valuable to your organization.
```