ALOHA Load-Balancer - Application Note

Routing HTTP requests

Document version: v1.1

Last update: 19th June 2014

EMEA Headquarters
3, rue du petit robinson
ZAC des Metz
78350 Jouy-en-Josas
France
http://www.haproxy.com/

Purpose

Content switching is the ability to route HTTP requests based on any information available in the HTTP protocol (URL and headers).
This document explains how to do it in **HAProxy**.

Complexity

Versions concerned

- Aloha 4.2 and above

Changelog

Version	Description
1.1	2014-06-19 – HAProxy Tech. theme update – minor changes – Updates for ALOHA 6.0
1.0	2013-11-08 Initial release

Synopsis

Content Switching helps improving application scalability and makes web architectures more flexible.
It can be used as well to improve security and reliability of web platforms or when some applications require different settings despite being hosted on the same server.
These settings could be:

- virtual hosting: routing requests based on website host name
- application hosting: routing requests based on url path
- categorizing HTTP traffic, like static and dynamic
- resource allocation based on user category (authenticated or not)
- health check
- timeouts
- connection mode
- statistics
- compression
- queue management

Non exhaustive list.

Limitation

Content switching rules are partialy compatible with **HAProxy's tunnel** mode. Please read the memo named **"HTTP connection mode"** to know more about **HAProxy**'s modes.
In **tunnel** mode, only the first request of the session can be routed, upcoming data of the connection is forwarded as payload in the tunnel established by **HAProxy** between the **client** and the **server**.

If your application requires authentication based on **NTLM**, then you must use the tunnel mode (or **http-keep-alive** mode in **ALOHA** 6.0 and above)

ALOHA 4.2 to 5.5

By default, the **ALOHA** is configured in the **server-close** mode, which is compatible with the information provided in this document.
If you need to enable **tunnel** mode on this **ALOHA** version, then you'll be able to route the whole traffic based on the first request of each connection. This can be sufficient in most cases.

ALOHA 6.0 and above

Since the **ALOHA** 6.0, a new connection mode has been introduced: **http-keep-alive**. This mode is compatible with **NTLM** since it keeps the connection opened on the server side, but it is also able to analyse the whole content exchanged on this connection. It means it is able to route any request from an established connection to an other server when required.
The **tunnel** mode is still available, called **http-tunnel**, and could be used in very rare cases.

Reverse Proxy

Usually, the **Content Switching** ability is associated to **Reverse-Proxies**.
HAProxy's way of working is typically a **Reverse proxy**. In your configuration, you define entry points, called **frontend**, and outgoing points, called **backend**. The way to choose which backend to route an HTTP request is **content switching**.

Content Switching in the ALOHA Load-Balancer

HAProxy's internal routing process

In the **ALOHA**, the software responsible to processing HTTP is **HAProxy**. So this is in **HAProxy** that we'll be able to configure **content switching** rules.
From an HTTP point of view, **HAProxy** is split into two main components:

1. **frontend**: manages everything related to the **client** side
2. **backend**: manages everything related to the **server** side

Basically, when a client sends a request, it is first processed by the **frontend**. Then, based on its rules, it routes the request to a **backend**.

The step when **HAProxy** takes routing decision between the **frontend** and the **backend** is the "**content switching**" step.

Routing decision can be taken on the following items:

- any string or regexp matching in HTTP headers
- any string or regexp matching in the URL (including scheme and protocol version)
- any value of a query string parameter
- any file extension
- cookie values
- ssl protocol
- HAProxy's internal status (farm capacity, etc...)

Non-exhaustive list.

Content switching rule profile

In **HAProxy**, Content switching rules are split in 2 components:

1. an **acl** to **fetch** samples from current traffic and to **match** it against **patterns**
2. a routing decision which points to a **backend** if the associated **acl**(s) are true

Below is the prototype of a rule:

acl <acl name> <fetch> <patterns>
use_backend <backend name> **if / unless** <acl name>

- **acl**: **HAProxy** keyword pointing the beginning of a new matching rule
- <**acl name**>: a word (underscore '_' and hyphen '-' allowed) naming the **acl** which will be used as a label (or pointer) later in the configuration
- <**fetch**>: sample to be extracted
- <**patterns**>: values to be compared to the sample fetched
- **use_backend**: **HAProxy** keyword indicating that a routing decision may occur if the **acl** matches
- <**backend name**>: the name of the backend to route to
- **if / unless**: keyword to tell whether to match (or not) an **acl**
- <**acl name**>: the name of the **acl** to get the matching result from

Rules about HAProxy's content switching

When working with **acl** and **use_backed** in **HAProxy**, it is important to keep in mind the following rules:

- Many **patterns** can be provided on a single **acl** line, a logical **OR** is applied between them
- Many **acl**s can have the same **name**: a logical **OR** is applied between them
- An **acl** returns only TRUE or FALSE when the <**sample fetched**> matches any <**pattern**>
- When a frontend configuration owns many **use_backend**, the first one matching an **acl** will be used
- A **use_backend** can be triggered by many **acl**s. In such case, just append **acl** names. A logical **AND** is implicitly applied. An explicit **OR** is also available.

Fetch samples

There are many **fetch** methods available in **HAProxy**, and each new **ALOHA** release comes with new ones.
To know which **fetches** are available in your **ALOHA**, open the WUI, go in the **LB Layer7** tab, click on the **help** button on the top right corner of the textarea and use the search engine and look for the string **"Matching at Layer 7"** or **"Fetching HTTP samples"** (ALOHA 6.0 and above).

ALOHA 4.2 to 5.5

The most common **fetches** are:

- **src**: fetch source IP address
- **nbsrv**: number of available servers in a farm
- **method**: request HTTP method
- **hdr**: fetch a header value
- **path**: fetch a url path (query string excluded)
- **url**: fetch the request's URL as presented in the request, including the method (GET/POST/-HEAD/etc...) and the protocol version
- **urlp**: fetch the first occurrence of a URL parameter in a query string

ALOHA 6.0

This firmware release introduced the new following **sample fetch**:

- **base**: fetch the concatenation of the first **Host** header and the path of the url (until the question mark)
- **cook**: fetch a cookie value

(Non exhaustive list)

Enhanced fetches

By default, any **fetch** method applies to the whole targetted object. Sometimes, we just want to **fetch** samples at different locations or through different ways in the object.
You can suffix the **fetch** method by the one of the keyword below to change the location:

- **_beg**: prefix
- **_cnt**: number of occurrence
- **_dir**: directory (slashes are implicit, no need to declare them)
- **_dom**: domain name
- **_end**: suffix
- **_len**: length
- **_reg**: PCRE regex
- **_sub**: substring

fetch examples

- **hdr_sub**: fetch a substring in a header
- **hdr_reg**: fetch a regular expression in a header
- **path_beg**: fetch the beginning of the url path
- **path_end**: fetch file extension (basically, the end of the url)
- **path_dir**: fetch a directory in the path

Sample / pattern types

There are many ways to match **pattern** against a **fetched** sample, depending on the type of the sample.

Integers

When a <**fetch**> returns an integer, we may want to know if the number returned is lower, greater or equal than a **pattern**.
The keyword below allows these comparisons:

- **eq**: true if the **sample fetched** is **equal** to the **pattern**
- **ge**: true if the **sample fetched** is **greater OR equal** than the **pattern**
- **gt**: true if the **sample fetched** is **greater** than the **pattern**

- **le**: true if the **sample fetched** is **lower OR equal** than the **pattern**
- **lt**: true if the **sample fetched** is **lower** than the **pattern**

In example: test if the number of available servers in the backend **bk_web** is lower than 2:

```
acl low_capacity nbsrv(bk_web) lt 2
```

The **acl** above returns **true** when the number of available servers in a farm is LOWER than 2 (thus 1 or 0).
It can be used to route traffic to a sorry backend if you know your application requires at least 2 servers to handle the load properly.

Strings

When a <**fetch**> returns a string, we may want to compare with other strings.

- A raw **case-sensitive** comparison is performed
- It is possible to make the search **case-insensitive** by adding the flag "**-i**" before the **patterns**
- If a **pattern** contains a space character, then it must be escaped by a backslash ('\')

In example: check if the Host header is 'www.domain.tld':

```
acl host_www.domain.tld hdr(Host) www.domain.tld
```

Regular expressions

Regular expressions can be used to match any type of content.

if you want to match a simple string or an integer, it is better to use the appropriate **enhanced fetch**.

The comparison is **case-sensitive** by default. It can be turned **case-insensitive** by adding the flag "**-i**" before the regexp.
If a regexp contains a space character, then it must be escaped by a backslash ('\').

In example: check if the Host header looks like '*.domain.tld':

```
acl host_domain.tld hdr_reg(Host) .*domain\.tld$
```

IPv4 and IPv6 addresses

ALOHA 5.0 and below can only do IPv4 matching.
ALOHA 5.5 and above can match both IPv4 and IPv6.

IPs addresses can be provided either as a single host address or as a subnet in CIDR notation. The result is positive if the **sample** matches or if it belongs to the subnet (**patterns**).
In example: check if the client IP belong to the users subnet:

```
acl users_subnet src 10.0.0.0/24
```

The acl returns **true** if the client IP belongs to the supplied subnet.

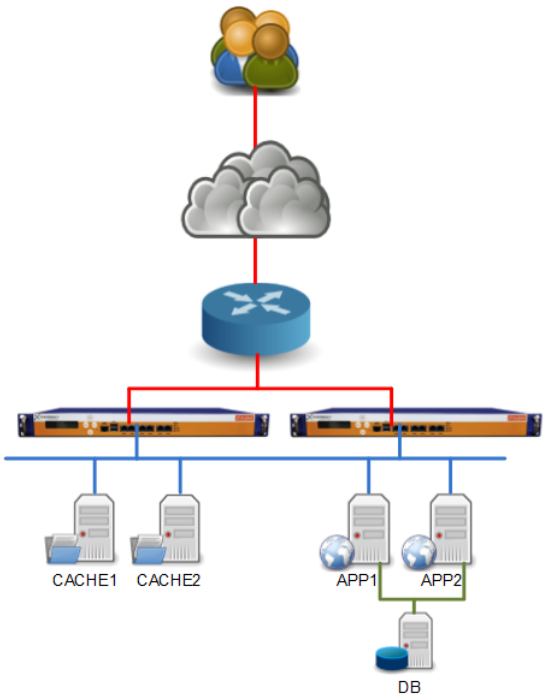
Content switching examples

Static and dynamic traffic split

When an application is hosted over a single host name, one way to split requests for static and dynamic content is to use either URL path or file extension.

Diagram

The diagram below illustrates this case. The **ALOHA** gets all traffic and splits it against 2 farms.



Note that a single server could be used to deliver both static and dynamic content.
Splitting the traffic that way allows to manage each farm individually with different type of queueing and health checks.

Configuration

Frontend

Whole website traffic reaches this **frontend**. **HAProxy** takes rooting decision based on layer 7 information.

```
frontend ft_websites
   mode http
   bind 0.0.0.0:80
   log global
   option httplog

# detect static content by file extension or URL path
   acl static path_end .gif .jpg .jpeg .png
   acl static path_end css js
   acl static path_beg /images/ /users/
   acl static path_dir static
   use_backend bk_static if static

# default route
   default_backend bk_dynamic
```

Backend

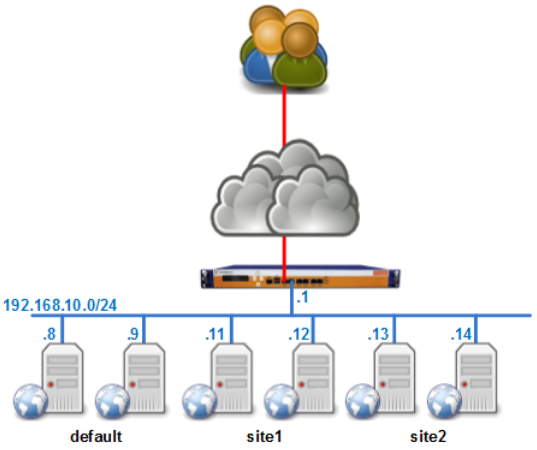
```
# static farm configuration
backend bk_static
  mode http
  balance roundrobin
  option forwardfor
  # dedicated health check for static content
  option httpchk HEAD /images/pixel.png
  default-server inter 3s rise 2 fall 3 slowstart 0
  server srv1 192.168.10.11:80 weight 10 maxconn 1000 check
  server srv2 192.168.10.12:80 weight 10 maxconn 1000 check

# dynamic farm configuration
backend bk_dynamic
  mode http
  balance roundrobin
  cookie SERVERID2 insert indirect nocache
  option forwardfor
  # dedicated health check for dynamic content
  option httpchk GET /check.php
  http-check expect string OK
  default-server inter 3s rise 2 fall 3 slowstart 0
  server srv1 192.168.10.11:8080 cookie s1 weight 10 maxconn 100 check
  server srv2 192.168.10.12:8080 cookie s2 weight 10 maxconn 100 check
```

Virtual Hosting

It is very common to use Virtual Hosting based on website host name when we own a very few public IP addresses and need to give a access to a very large number of applications.
Basically, the **ALOHA load-balancer** is used as a reverse proxy in such case.

Diagram

In the example below, there are 2 domains pointing to the **ALOHA** public IP address. Depending on the domain name, the **ALOHA Load-Balancer** decides which farm to use.



Configuration

Frontend
Whole website traffic reaches this **frontend**. **HAProxy** takes rooting decision based on layer 7 information.

```
frontend ft_websites
   mode http
   bind 0.0.0.0:80
   log global
   option httplog

# Capturing Host header in logs is important for
# troubleshooting to know whether rules matches or not
   capture request header host len 64

# site1 routing rules based on Host header
   acl site1 hdr_end(host) site1.com site1.eu
   use_backend bk_site1 if site1

# site2 routing rules based on Host header
   acl site2 hdr_reg(host) site2\.(com|ie)$
   use_backend bk_site2 if site2

# default route
   default_backend bk_default
```

Backend

Each virtual host has its own backend with its specific configuration (mind the health check).

```
# site1 backend configuration
backend bk_site1
  mode http
  balance roundrobin
  cookie SERVERID1 insert indirect nocache
  option forwardfor
  # dedicated health check for site1
  option httpchk HEAD /check.php HTTP/1.1\r\nHost:\ www.site1.com
  default-server inter 3s rise 2 fall 3 slowstart 0
  server srv1 192.168.10.11:80 cookie s1 weight 10 maxconn 1000 check
  server srv2 192.168.10.12:80 cookie s2 weight 10 maxconn 1000 check

# site2 backend configuration
backend bk_site2
  mode http
  balance roundrobin
  cookie SERVERID2 insert indirect nocache
  option forwardfor
  # dedicated health check for site2
  option httpchk HEAD /check.jsp HTTP/1.1\r\nHost:\ www.site2.com
  default-server inter 3s rise 2 fall 3 slowstart 0
  server srv1 192.168.10.13:80 cookie s1 weight 10 maxconn 1000 check
  server srv2 192.168.10.14:80 cookie s2 weight 10 maxconn 1000 check
```

All requests which did not match the content switching rules are routed to this **backend**:

```
backend bk_default
   mode http
   balance roundrobin
   option forwardfor
   option httpchk HEAD /
   default-server inter 3s rise 2 fall 3 slowstart 0
   server srv1 192.168.10.8:80 weight 10 maxconn 1000 check
   server srv2 192.168.10.9:80 weight 10 maxconn 1000 check
```