ALOHA Load-Balancer - Application Note

Content Switch ou routage de niveau HTTP

Document version: v1.1

Last update: 19 juin 2014

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

Objectif

Le **Content switching** est la capacité à router des requêtes HTTP en se basant sur une information disponible dans le protocole lui-même (URL et en-tête).
Ce document explique comment le faire dans **HAProxy**.

Difficulté

Versions concernées

- Aloha 4.2 and above

Changelog

Version	Description
1.1	2014-06-03 – Mise à jour du thème HAProxy Tech. – Changements mineurs – Prise en compte du firmware **ALOHA** 6.0
1.0	2013-11-08 Première version

Synopsis

Le **content switching** facilite l'évolutivité des applications et rends les architectures webs plus flexibles.
On peut utiliser cette technique pour améliorer la sécurité et la disponibilité des platformes webs ou lorsque certaines applications requièrent différents paramètres, bien qu'étant hébergées sur le même serveur.
Les paramètres peuvent être :

- virtual hosting : routage des requêtes basé on le nom du site web
- application hosting : routage des requêtes basé sur le chemin de contenu dans l'URL
- catégoriser le trafic : statique / dynamique, par exemple
- allocation de ressources en fonction de la catégorie de l'utilisateur (authentifié ou non, etc...)
- contrôle de vitalité
- timeouts
- mode de connexion
- statistiques
- compression
- gestion de file d'attente

Liste non exhaustive.

Limitation

Le **Content Switching** est partiellement compatible avec le mode **tunnel** d'**HAProxy**. Merci de lire le mémo nommé **HTTP Connection mode** pour de plus amples informations sur les différents modes disponibles dans **HAProxy**.
En mode **tunnel**, seule la première requête de la session peut être analysée et routée, toutes les autres seront considérées comme de la donnée brute et transférées sur la même connexion sans analyse.

Si votre application requiert une authentification basée sur **NTLM**, alors vous devez utiliser le mode **tunnel** (ou le mode **http-keep-alive** de l'**ALOHA** 6.0 et supérieur).

ALOHA 4.2 à 5.5

Par défaut, l'**ALOHA** est configuré en mode **server-close**, qui est compatible avec les informations délivrées dans ce document.
Si vous devez activer le mode **tunnel** dans cette version d'**ALOHA**, alors vous ne pourrez router le trafic en cours qu'en vous basant sur la première requête de chaque connexion. C'est suffisant pour la plupart des usages.

ALOHA 6.0 et supérieur

Depuis l'**ALOHA** 6.0, un nouveau mode de connexion a été introduit : **http-keep-alive**. Ce mode est compatible avec le **NTLM** puisqu'il conserve la connexion ouverte avec le serveur, mais il permet aussi l'analyse protocolaire de tout le contenu échangé. Ce qui signifie qu'**HAProxy** est capable de router n'importe quelle requête d'une connexion existante vers un autre serveur.
Le mode **tunnel** est encore disponible, appelé **http-tunnel**, et peut être utilisé dans de rares cas.

Reverse-Proxy

Habituellement, la fonctionnalité de **Content Switching** est associées aux **Reverse-Proxies**.
Le mode interne de fonctionnement d'**HAProxy** est précisément un **Reverse-Proxy**. Dans sa configuration, nous définissons des points d'entrée, appelés **frontend**, et des points de sortie, appelés **backend**.
L'action de choisir vers quel **backend** router la requête HTTP est le **Content Switching**.

Termes en anglais

Cette documentation contient quelques termes en anglais, afin de faciliter au lecteur la recherche d'information dans la documentation **HAProxy** qui est uniquement disponible en anglais.
Ces termes, avec leur traduction, sont :

- **sample** : échantillon
- **fetch** : extraire
- **pattern** : motif
- **match** : correspondre

Content Switching dans l'ALOHA

Fonctionnement du routage à l'intérieur de HAProxy

Dans l'**ALOHA** Load-Balancer, l'outil de traitement au niveau 7 est **HAProxy**. C'est donc dans **HAProxy** que l'on sera capable de configurer les règles de **content switching**.
D'un point de vue HTTP, **HAProxy** est divisé en deux composants principaux :

1. **frontend** : gère tout ce qui est côté client **client**
2. **backend** : gère tout ce qui est côté **serveur**

Quand un client envoi une requête, elle est d'abord traitée par le **frontend**. Puis, en fonction de sa configuration, **HAProxy** route la requête vers un **backend**.

Le moment où **HAProxy** prend la décision de routage entre le **frontend** et le **backend** est l'étape de "**content switching**".

La décision de routage peut être prise en fonction des critères suivants :

- n'importe quelle chaîne de caractère ou expression régulière dans les en-têtes HTTP
- n'importe quelle chaîne de caractère ou expression régulière dans l'URL
- n'importe quelle valeur d'un paramètre de la query string
- n'importe quelle extension de fichier
- valeurs de cookie
- informations du protocole SSL
- états internes à HAProxy (capacité d'une ferme, file d'attentes, etc...)

Liste non exhaustive.

Profil d'une règle de Content switching

Dans **HAProxy**, les règles de **Content switching** sont divisées en 2 composants :

1. Une **acl** pour extraire un échantillon (**sample fetch**) qui permettra de prendre la décision de routage en fonctions de motifs (**patterns**)
2. Une règle de routage qui pointe vers un **backend** si les **acl**(s) associée(s) sont validées

Ci-dessous, le prototype d'une règle :

acl <nom de l'acl> <fetch> <pattern>
use_backend <nom du backend> **if / unless** <nom de l'acl>

- **acl** : Mot clé **HAProxy** désignant le début d'une nouvelle règle de recherche
- <**nom de l'acl**> : un mot (underscore '_' et moins '-' acceptés) nommant l'**acl** et qui peut être utilisé comme une étiquette (pointeur) plus tard dans la configuration
- <**fetch**> : échantillon à extraire
- <**pattern**> : valeurs comparées avec l'échantillon
- **use_backend** : mot clé indiquant qu'une décision de routage peut s'appliquer si les **acl**s sont validées
- <**nom du backend**> : le nom du backend où envoyer la requête
- **if / unless** : mot clé permettant de valider la décision de routage si le retour de l'**acl** est positif / négatif
- <**nom de l'acl**> : le nom de l'**acl** pour laquelle on va vérifier le résultat

Règles sur le content switching dans HAProxy

Il est important de connaître les quelques règles suivantes lorsque l'on utilise les **acl**s et les **use_backend** dans **HAProxy** :

- Plusieurs **patterns** peuvent être fournies par ligne d'**acl**, un **OR** logique est appliqué.
- Plusieurs **acl**s peuvent partager le même **nom** : un **OU** logique est appliqué entre elles.
- Une **acl** retourne uniquement VRAI ou FAUX lorsque le <**fetch**> correspond à l'une des <**patterns**>.
- Si une configuration nécessite plusieurs **use_backend**, le premier ayant une **acl** positive sera utilisé
- Il est possible de conditionner l'exécution d'un **use_backend** par plusieurs **acl**s. Un **AND** logique est appliqué implicitement. Un **OR** logique peut explicitement être déclaré.

Extraction d'échantillon (fetch)

Il existe plusieurs **fetch** dans **HAProxy**, et chaque nouvelle version d'**ALOHA** Load-Balancer en introduit de nouveaux. Afin de savoir quels **fetch** sont disponibles dans votre **ALOHA**, ouvrez l'onglet **LB Layer7** de l'interface puis cliquez sur le bouton **aide** du coin supérieur droit de la zone de saisie et utilisez le moteur de recherche avec la chaîne de caractère **"Matching at Layer 7"** ou **"Fetching HTTP samples"** (**ALOHA** 6.0 et supérieur).

ALOHA 4.2 à 5.5

Les **critères** les plus communément utilisés sont :

- **src** : extraction de l'adresse IP source (client)
- **nbsrv** : nombre de serveurs disponibles dans une ferme
- **method** : méthode de la requête HTTP
- **hdr** : analyse des en-têtes HTTP
- **path** : extraction du chemin d'URL (query string exclue)
- **url** : extraction de l'URL comme présentée dans la requête, incluant la méthode (GET/POST/-HEAD/etc...) et la version du protocole
- **urlp** : extraction de la première occurence d'un paramètre de la query string

ALOHA 6.0

Cette version de firmware a introduit les nouveaux **fetch** suivant :

- **base** : extraction de la concaténation du premier en-tête **Host** et du chemin de l'URL (jusqu'au point d'interrogation)
- **cook** : extraction de la valeur d'un cookie

Liste non exhaustive

Fetch améliorés

Par défaut, un **fetch** s'applique à l'objet visé, en entier. Parfois, il est préférable d'extraire un contenu à un endroit précis ou de différentes manières.
Il est possible de suffixer les **fetch** avec l'un des mots clés ci-dessous pour améliorer l'extraction :

- **_beg** : préfixe
- **_cnt** : nombre d'occurrence
- **_dir** : répertoire (les slashes sont implicites, pas besoin de les déclarer)
- **_dom** : nom de domaine
- **_end** : suffixe
- **_len** : longueur
- **_reg** : Expression régulière PCRE
- **_sub** : sous chaine

Exemples de fetch

- **hdr_sub** : extrait une sous-chaîne dans les en-têtes HTTP
- **hdr_reg** : applique une expression régulière dans les en-têtes HTTP
- **path_beg** : extrait une chaine au début du chemin d'URL
- **path_end** : extrait une chaine qui peut correspondre à l'extension de fichier (plus précisément la fin du chemin de l'URL)
- **path_dir** : recherche un répertoire dans le chemin de l'URL

Types de sample et de patterns

Les données extraites peuvent être de différents types.

Entiers

Certains <**fetch**> retournent des entiers. Il est donc souhaitable de savoir si le **sample** est plus petit, plus grand ou égal à un **pattern**.
Les mots clés ci-dessous permettent ces comparaisons :

- **eq** : vrai si l'échantillon extrait (**fetch**) est **égal** au pattern
- **ge** : vrai si l'échantillon extrait (**fetch**) est **plus grand OU égal** au pattern
- **gt** : vrai si l'échantillon extrait (**fetch**) est **plus grand** au pattern
- **le** : vrai si l'échantillon extrait (**fetch**) est **plus petit OU égal** au pattern
- **lt** : vrai si l'échantillon extrait (**fetch**) est **plus petit** au pattern

Par exemple : tester si le nombre de serveur disponible dans la ferme **bk_web** est inférieur à 2 :

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

L'**acl** ci-dessus retourne **vrai** lorsque le nombre de serveurs disponibles dans la ferme est inférieur à 2 (donc 1 ou 0). Celà peut être utilisé pour router le trafic vers une ferme d'excuse s'il ne reste plus assez de capacité de traitement dans la ferme.

Chaîne de caractère

Quand le <**fetch**> extrait une chaîne de caractères, on veut pouvoir la comparer à d'autres chaines.

- C'est une recherche basique **case-sensitive**.
- Si la chaîne recherchée contient des espaces, alors ils doivent être précédés du caractère antislash ('\').
- Il est possible de rendre la recherche **case-insensitive** en ajoutant le paramètre "**-i**" avant la chaîne recherchée.

Par exemple : vérifier si l'en-tête HTTP "Host" contient la chaîne "www.domain.tld" :

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

Expression régulière

Les expression régulières peuvent être utilisée pour extraire n'importe quel type de contenu.

Ceci dit, pour rechercher une chaîne simple ou un nombre, il est plus efficace d'utiliser l'**ACL** appropriée.

La comparaison est **case-sensitive** par défaut. Il est possible de la rendre **case-insensitive** en ajoutant le paramètre "**-i**" devant l'expression régulière.
Si une regexp contient un espace, alors il doit être précédé du caractère antislash ('\').

Par exemple : vérifier si l'en-tête HTTP **Host** ressemble à "*.domain.tld" :

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

Adresses IPv4 and IPv6

ALOHA 5.0 et précédent peuvent rechercher uniquement des adresses IPv4.
ALOHA 5.5 et supérieur peuvent rechercher des IPv4 et IPv6.

Les adresses IPs peuvent être comparées soit à une adresse de unique, soit dans un sous-réseau en notation CIDR. Le résultat de la recherche est positif si l'adresse IP est trouvée ou si elle appartient à un sous-réseau.
Par exemple : vérifier si l'adresse IP du client appartient à un sous-réseau :

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

L'**acl** retourne **vrai** si l'IP du client appartient au sous réseau passé en argument.

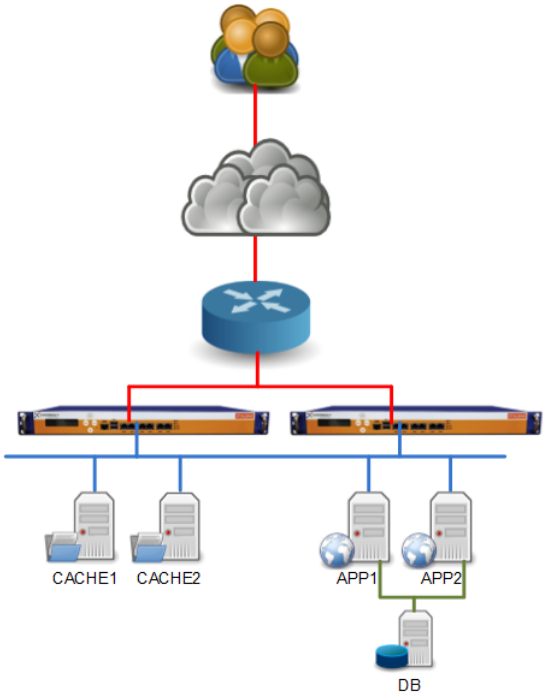
Exemple d'utilisation du Content switching

Séparation statique / dynamique

Quand une application est hébergée sur un nom de host unique, la seule manière de séparer les flux statiques et dynamiques est d'utiliser les information contenus dans le chemin d'URL de la requête ou l'extension du fichier.

Schéma

Le schéma ci-dessous illustre ce cas. L'**ALOHA** Load-Balancer reçoit tout le trafic puis le renvois vers chacune des fermes.



Veuillez noter qu'un serveur unique peut être utilisé pour délivrer à la fois le contenu statique et dynamique.
La segmentation du trafic permet de gérer chaque ferme individuellement avec différents paramètres chacune : health checks, régulation de trafic, persistance...

Configuration

Frontend

Tout le trafic pour ce site-web arrice sur ce **frontend**. **HAProxy** prend la décision de routage en fonction des informations fournies par le protocole HTTP (niveau 7) :

```
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

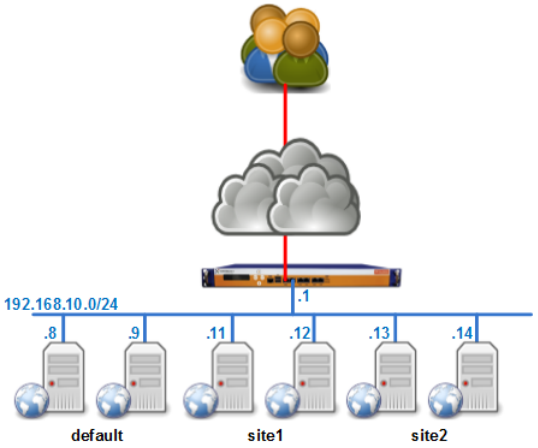
```
# static farm configuration
backend bk_static
  mode http
  balance roundrobin
  option forwardfor
  # dedicated health check for static content
  option httpchk HEAD /images/pixel.png
  default-server inter 3s rise 2 fall 3 slowstart 0
  server srv1 192.168.10.11:80 weight 10 maxconn 1000 check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

L'hébergement virtuel basé sur le nom de domaine du site web est une technique utilisée lorsque l'on possède peu d'adresse IP publique et que l'on doit héberger un grand nombre d'application.
Dans ce genre de configuration, l'**ALOHA load-balancer** est utilisé en mode reverse-proxy.

Schéma

Dans l'exemple ci-dessous, il y a 2 noms de domaines pointant sur l'adresse IP publique de l'**ALOHA**. En fonction du nom de domaine, l'**ALOHA Load-Balancer** décide de la ferme à utiliser.



Configuration

Frontend

Le trafic pour tous les sites web arrivent sur ce **frontend**. **HAProxy** prend la décision de routage en fonction des informations fournie par le protocole HTTP (niveau 7)

```
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

Chaque virtual host possède son propre **backend** avec sa propre configuration.

```
# 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
```

Toutes les requêtes qui ne correspondent à aucun noms de domaines connus sont routées vers ce **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
```