HAPROXY

Powering Your Uptime

HAProxy Technologies

HAProxy best practice

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

HAProxy best practices and common issues

Agenda

- Disclaimer
- Open Source
- Introduction to HAProxy
- How HAProxy works
- A simple configuration
- RTF(W|E|L)M
- Performance: hardware, sysctls
- multi-process
- timeouts
- fetches / acls
- http rules
- weak server/application protection
- stats page / socket

Disclaimer

Don't apply blindly all the tips presented here

- Each application is different and deserves a deep understanding
- Each workload, network stack may interfer
- Don't hesitate to ask the experts on HAProxy's ML: haproxy@formilux.org (no registration required)
- For faster, safer but more expensive support, ask contact@haproxy.com
- Wear 3D glasses to enjoy the graphism of this presentation
- This presentation aims at satisfying people with experience of HAProxy and people who discover HAProxy today
- 26 slides and 8 live demos!

About HAProxy Technologies

- 20 people, in 3 offices: Zagreb (Croatia), Paris, Boston
- 95% of HAProxy open source development is made by us
- HAProxy pure player !
- Willy Tarreau is our CTO
- Deliver services, support, all around HAProxy
- ALOHA appliance: HAProxy based load-blancer
- HAProxy Enterprise: enhanced and supported version of HAProxy, yet open source
- Follow us:
    - Website: http://www.haproxy.com/
    - Blog: http://blog.haproxy.com/
    - Twitter: http://twitter.com/haproxy_tech/
    - Github: http://github.com/haproxytech/
    - Slideshare: http://www.slideshare.net/haproxytech

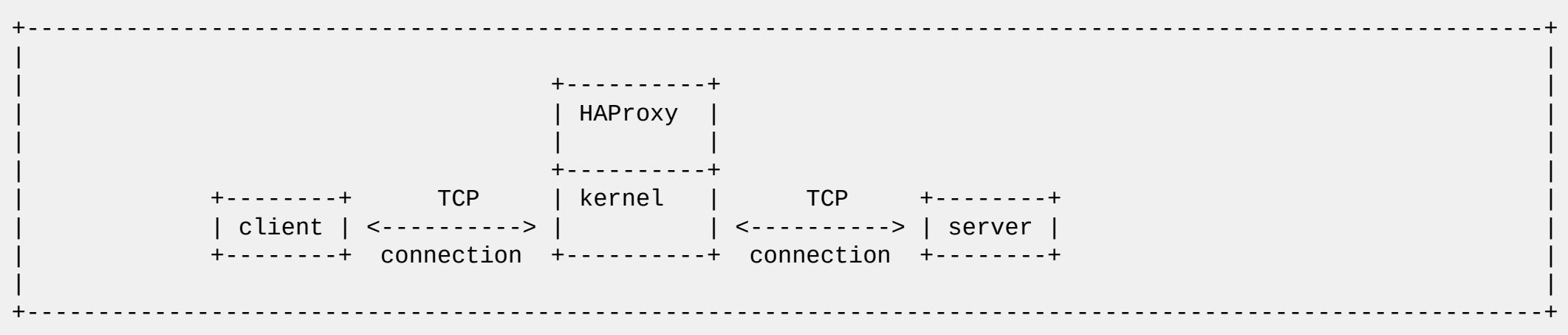
Open Source

- Open source is more than "viewing the code source"
- Even if you don't know any programing language, you can:
  - Answer a question on the mailing list, IRC, stack overflows
  - review/write documentation
  - write tutorials, blog posts
  - screencast a feature
  - file a bug report
  - review issues, bugs
  - test new versions, provide feedback!
  - Advertize, advocate the softwares you love in your company
- The key to open source is **Many people making small improvements**

Source: http://fr.slideshare.net/Docker/docker-opensourceathon-2015

Introduction to HAProxy

- event driven
- simple keyword based configuration
- Userland load-balancer working in a **proxy** mode



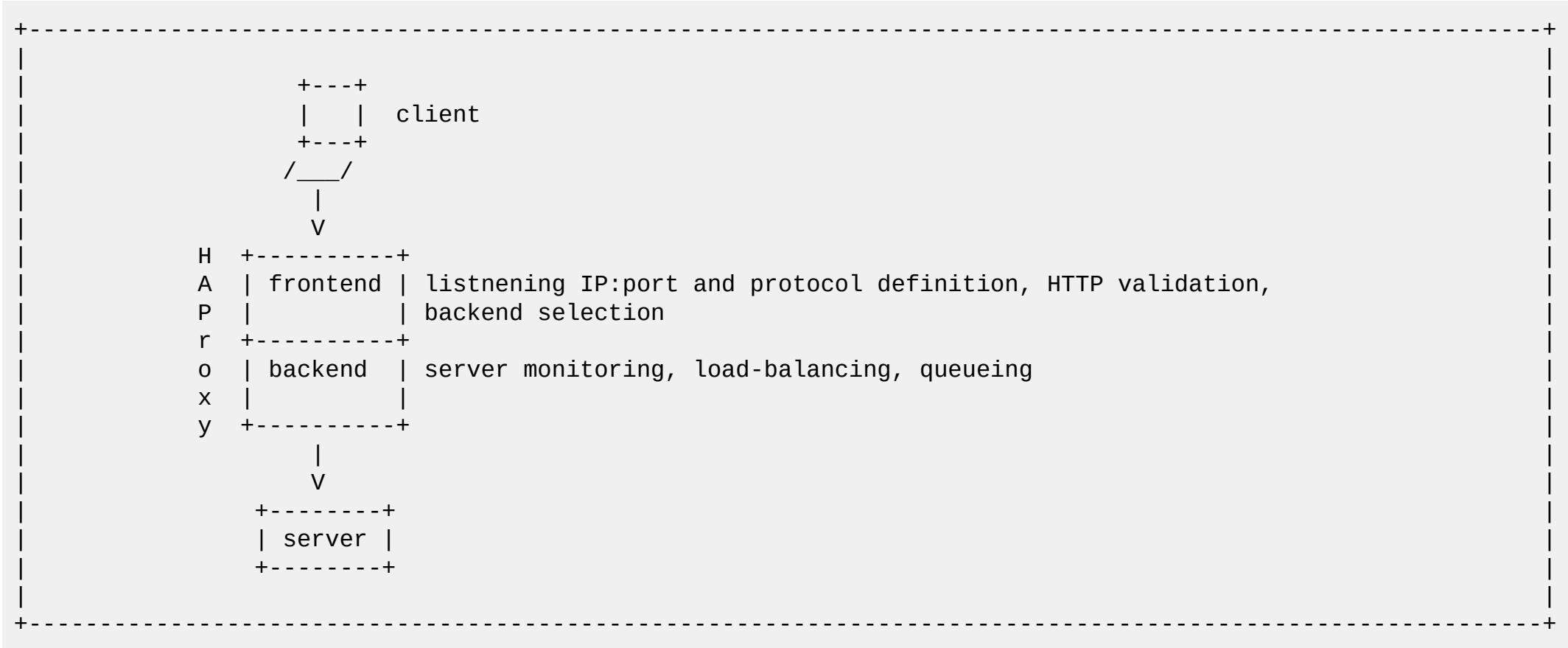
- HTTP reverse proxy (with many advanced HTTP features)
- many persistence method
- TLS/SSL processor (with many advanced TLS/SSL features)
- advanced reporting of both network and application behaviors
- very verbose tool!

Introduction to HAProxy

- HAProxy is amazing at:
  - Load-balancing web sites
  - Ensure high availability and performance of web based API calls
  - protect weak servers and applications
  - managing micro services
  - Splitting and handle traffic differently based on URL or headers
  - Process TLS on behalf of application servers
- HAProxy can also be used to:
  - enforce protection against botnet
  - fight layer 7 floods
  - manage a SSO web service

How HAProxy works

- HAProxy is split into 2 layers:
  - *frontend* : client side
  - *backend* : server side



non-exhaustive list!!

A simple configuration

```
global
 daemon

defaults
 mode http
 timeout client 10s
 timeout connect 4s
 timeout server 30s

frontend fe
 bind 10.0.0.1:80
 bind 10.0.0.1:443 ssl crt ./my.pem
 default_backend be

backend be
 server s1 10.0.0.101:80 check
 server s2 10.0.0.102:80 check
```

We can test a configuration using the `-c` flag: `haproxy -c -f myapp.cfg`

DEMO #1 !!!

RTF(W|E|L)M

Read These F.ck.ng (**Warnings** / **Errors** / **Log**) messages!

- HAProxy reports configuration errors on stderr
- When a configuration is inaccurate a *warning* is emitted on stderr
- these messages are sent to syslog, when logging is enabled
- HAProxy provides a message which explains the mistake
- sometimes a fix is proposed

```
[WARNING] 177/011147 (8652) : Setting tune.ssl.default-dh-param to 1024 by default, if your workload permits it
Configuration file is valid
```

NOTE: this does not prevent you from reading the F.ck.ng manual :)

Hardware recommandation

Due to its way of working, HAProxy requires:

- CPU: prefer the speed and the cache size over the number of cores
- enough memory to handle all the TCP connections + HAProxy + system
- network interfaces: intel ones are usually a good choice
- disk: not required, unless logging locally

In order to get the best performance, a bit of organization is required:

- Network interrupts and kernel on core 0
- HAProxy on next core on the same physical CPU

NOTE: of course, uninstall *irqbalance*

Avoid VM or public cloud with shared resources if the expected workload is important.

Sysctls tuning

The most important sysctls are:

- net.ipv4.ip_local_port_range = "1025 65534"
- net.ipv4.tcp_max_syn_backlog = 100000
- net.core.netdev_max_backlog = 100000
- net.core.somaxconn = 65534
- ipv4.tcp_rmem = "4096 16060 64060"
- ipv4.tcp_wmem = "4096 16384 262144"

Depending on the workload:

- tcp_slow_start_after_idle = 0

iptables tuning:

- net.netfilter.nf_conntrack_max = 131072

=> when improperly configured, conntrack will prevent HAProxy from reaching high performance.

NOTE: just enabling iptables with connection tracking takes 20% of CPU, even with no rules.

HAProxy multi-process

- Advantages:
  - **ability to dedicate a process to a task (or application, or protocol)**
    In example: 1 process for HTTP and 1 process for **MySQL**
  - scale up: same hardware, more processing capacity by binding processes to different CPU cores
  - **useful when massive SSL offloading processing is required**
    key generation scales almost linearly with number of processes, but TLS session resumption gets little gain over 3 processes

HAProxy multi-process

- Limitations:

  Each process has its own memory area, which means:

  - debug mode cancels multi-process (a single process is started)
  - frontend(s) and associated backend(s) must run on the same process
  - not compatible with `peers` section (*stick table synchronization*)
  - information is stored locally in each process memory area and can't be shared:
    - stick table + tracked counters
    - statistics
    - server's maxconn (queue management)
    - connection rate
  - Each **HAProxy** process performs its health check:
    - a service is probed by each process
    - a service can temporarly have different status in each process
  - managing a configuration which starts up multiple processes can be more complicated

HAProxy multi-process

```
 1 # **DON'T RUN IN PRODUCTION, THERE ARE NO TIMEOUTS**
 2 global
 3   nbproc 2
 4   cpu-map 1 1
 5   cpu-map 2 2
 6   stats socket /var/run/haproxy/socket_web process 1
 7   stats socket /var/run/haproxy/socket_mysql process 2
 8
 9 defaults HTTP
10   bind-process 1
11   mode http
12 frontend f_web
13   bind 192.168.10.1:9000
14   default_backend b_web
15 backend b_web
16   server w1 192.168.10.21:8000 check
17
18 defaults MYSQL
19   bind-process 2
20   mode tcp
21 frontend f_mysql
22   bind 192.168.10.1:3306
23   default_backend b_mysql
24 backend b_mysql
25   server m1 192.168.10.11:3306 check
```

DEMO #2 !!!

Logging

- HAProxy logs are **very** verbose
- When the traffic workload allows it, they should be enabled all the time! Otherwise, we must have the ability to enable them on demand
- HAProxy can be configured to selectively log part of the traffic
- the log line can be customized to your needs (beware to not break the comptibility with **halog**)
- Logging can be enabled either in the `global` or in the `defaults`/`frontend` section
- Log format is configured per `frontend`, but log level must be reported in the `backend` too
- To setup your own log format, use the .... `log-format` directive

Logging

- Example in `global` section

```
global
  log 127.0.0.1:514 local1

defaults
  log global # 'pointer' to the global section
  option httplog
```

- Example in `frontend`/`backend` section

```
frontend fe
  log 127.0.0.1:514 local1
  option httplog
  default_backend be

backend be
  option httplog
```

- Split traffic and events logs:

```
global
  log 127.0.0.1:514 local1        # traffic logs
  log 127.0.0.1:514 local2 notice # event logs
```

Logging

- Log only errors:

```
defaults
  option dontlog-normal
```

- Don't log empty connections or browser's pre-connect

```
defaults
  option dontlognull
  option http-ignore-probes
```

- Log only dynamic traffic:

```
frontend fe
  http-request set-log-level silent unless { path_end .php }
```

DEMO #3 !!!

Timeouts

Bear this in mind: **timeouts** are not the problem!!!!

Without any timeouts, a public facing HAProxy won't last too long and run out of connections quickly.

Must set up at least the following timeouts:

- `timeout client` : client side inactivity
- `timeout connect` : time to establish the TCP connection on the server
- `timeout server` :
  - in TCP mode: server side inactivity
  - in HTTP mode: time for the server to process the response (504 returned)

Other important, but facultative, timeouts

- `timeout client-fin` : maximum time to wait in FIN_WAIT state on the client side
- `timeout server-fin` : maximum time to wait in FIN_WAIT state on the server side

Timeouts

In http mode, the following timeouts are important too:

- `timeout http-request` : timeout for the client to send a whole request (protection against slowlowris-like attacks)
- `timeout http-keep-alive` : maximum time to wait for the next request when doing HTTP keep-alive
- `timeout tunnel` : inactivity timeout for tunnel mode and websockets

Other timeouts:

- `timeout queue` : how long a request can remain in the queue
- `timeout tarpit` : how long the tarpitted connection is maintained

Timeouts

- Configuration example for an HTTP service

```
defaults HTTP
  mode http
  timeout http-request 10s
  timeout client 20s
  timeout connect 4s
  timeout server 30s
  timeout http-keep-alive 4s
  # for websockets:
  timeout tunnel 2m
  timeout client-fin 1s
  timeout server-fin 1s
```

- Configuration example for a TCP service with long time connections (POP, IMAP, etc)

```
defaults HTTP
  mode http
  timeout client  1m
  timeout connect 4s
  timeout server  1m
  timeout client-fin 1s
  timeout server-fin 1s
```

DEMO #4 !!!

Fetches

- Fetches can be used to form condition or to get samples from the request or response
- There many type of fetches, preventing the use of regexes
- Some internal states fetches:
  - `nbsrv(<backend>)` : number of server *UP* in *backend*
  - `fe_sess_rate` : session rate on the frontend
  - `queue(<backend>)` : number of queued connections on *backend*
- Some Layer 3 / 4 fetches examples:
  - `dst` : destination IP
  - `dst_port` : destination port
  - `src` : client IP address
- Some layer 7 fetches examples:
  - `url` : whole URL (from the method to the protocol version)
  - `path` : URL path
  - `url_param` : focused on the query string
  - `hdr` : HTTP header fields
  - `cook` : dedicated to cookie

Fetches

- Layer 7 fetches can get a sample of data at:
  - `beg` : beginning of a string
  - `end` : end of a string
  - `dir` : directory
  - `dom` : domain name
  - `len` : string length
  - `cnt` : number of occurence of the fetch
  - `sub` : sub-string
  - `reg` : regex (last chance :))
- Forming Layer 7 fetches:
  - match at the begining of the path: `path_beg -i /api/`
  - match at the end of the *Host* header: `hdr_end(Host) -i domain.com`
  - does a cookie exist: `cook_cnt(PHPSESSID) gt 0`
- When multiple strings are given to the fetch, a logical implicit `OR` is applied:

```
hdr_end(Host) -i .domain.com .domain.fr .domain.net .domain.org
```

ACLs

- ACLs can be either anonymous or named
- when named, they are configured using the keyword `acl`

```
acl api_path path_beg -i /api/
use_backend bk_api if api_path
```

- when anonymous, acls declared between curve brackets `{` and `}`

```
use_backend bk_api if { path_beg -i /api/ }
```

- Mutliple ACLs can have the same name, a logical `OR` is then applied:

```
acl myapi path_beg -i /api/
acl myapi hdr_beg(Host) -i api.
use_backend bk_api if myapi
```

- This is the equivalent to:

```
acl api_path path_beg -i /api/
acl api_vhost hdr_beg(Host) -i api.
use_backend bk_api if api_path || api_vhost
```

DEMO #5 !!!

HTTP rules

- HAProxy supports rules at HTTP layer. HAProxy can:
  - allow or deny request or response
  - redirect traffic
  - manipulate headers and url
  - capture content
  - update ACLs or maps content
  - ...
- Each rule can be conditionned by an ACL
- Each rule can use content of fetches

```
http-request deny unless { req.hdr(Host) -i www.mydomain.com }

http-request redirect location /%[req.hdr(Host)]%[path] if { path_beg -i /api/ }
```

DEMO #6 !!!

Server and application protection

- HAProxy can queue request to protect weak applications and servers
- Avoiding processing too many requests in parallel allows the following benefits:
  - the server never crashes
  - the application remains fast
  - response time remains good
- Simply setup `maxconn` parameters on the `server` line statement in HAProxy's `backend`
- there are no magic values. Benchmarking the application is the only way
- From our experience, `maxconn` value is from 50 to 300

DEMO #7 !!!

- When different workloads are expected, it is possible to route requests to different backends with different maxconn values

```
frontend f_myapp
 use_backend b_light if { path_beg /api/ /foo/ /bar/ }
 use_backend b_heavy if { path_beg /search /massivefoo /heavybar }

backend b_light
 server s1 server1:80 maxconn 300

backend b_heavy
 server s1 server1:80 maxconn 10
```

Stats page

- HAProxy maintain many different type of counters
- They are available through a fancy web page or a UNIX socket in CSV format

```
listen stats
 bind-process 1
 bind :9010
 stats enable
 stats uri /
 stats auth demo:demo
 stats realm Demo
 stats admin if TRUE
```

- Stats page is per process! In case of `nbproc` > 1, it is recommended to create one stats page or one UNIX socket per process
- in case of `nbproc` > 1, one unix path and one TCP port should be provided per process

DEMO #8 !!!