By default haproxy has central and only configuration file, stored in /etc/haproxy/haproxy.cfg. But due to our needs, which are dialed with configs for each user to be stored in separate files, I had to make changes in daemon starter (/etc/init.d/haproxy). That is why we have 3 files of global configuration and a directory for customer's config files. They are:

- /etc/haproxy/haproxy.cfg.def – this one stores global directives that are common for all customers;
- /etc/haproxy/haproxy.cfg – this file is being created each time haproxy starts, and comes as default file that haproxy uses for it's run. It is a combination of all client's configurations and haproxy.cfg.def.
- /etc/haproxy/haproxy.cfg.old – this file is being created and rewritten every time haproxy starts, it stores previous working config just in case if something goes wrong it can be used to start haproxy. I'm working on check config mechanism that will automaticaly bring back previous configuration if there are any errors in compiled new config.
- /etc/haproxy/Clients – directory that has configurations for every server cluster that are served with current loadbalance system.

*names of files can be changed. They don't play big role.
Config file with default definitions looks like:

```
global
        log 127.0.0.1:514   local0
        log 127.0.0.1:514   local1
        #log loghost    local0 info
        maxconn 10000
        #debug
        #quiet
        user haproxy
        group haproxy
defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        retries 3
        option redispatch
        no option checkcache
        maxconn 10000
        contimeout      5000
        clitimeout      50000
        srvtimeout      50000
       cookie SERVERID insert nocache indirect
```

I has two sections global
There are 2 ways of configuring haproxy ballansing:

- Simple way, is used when cluster requires only one group of servers on backend to be used (one listen section).

```
listen <instance_name>
bind <ip_address>:<port>
mode <layer mode>
balance <ballance_mode>
option <option1>
option <option2>
…….....
option <optionN>
server <server_name1> <node_ip_address>:<port> <option1> <option2> … <option N>
server <server_name2> <bind <option1> <option2> … <option N>
```

- More complicated way, used when cluster requires more than one group of servers on backend to be used, in this case each backend need to be configured in separate from frontend section. (Frontend that holds description for few backends)

```
frontend <instance_name>
bind <ip_address:port>
mode <layer mode>
option <option1>
option <option2>
…….....
```

```
option <optionN>
acl <acl_name1> <acl_type> <acl_definition>

use_backend <backend_name> if <acl_name1>
default_backend <backend_name>

backend <backend_name>
balance <ballance mode>
option <option1>
option <option2>
…..................
option <optionN>
server <server_name1> <nod_ip_address>:<port> <option1> <option2> … <option N>
server <server_name2> <nod_ip_address>:<port> <option1> <option2> … <option N>
```

Options and keywords can be found at: http://code.google.com/p/haproxy-docs/wiki/Keywords
Modes: http://code.google.com/p/haproxy-docs/wiki/mode
Ballances: http://code.google.com/p/haproxy-docs/wiki/balance

1. HTTP Load Balancing based on Layer 3 (sticky sessions)
Config sample:

```
listen <listen_name> <ip_address_to_listen>:801
      mode tcp
      balance source
      option httpchk OPTIONS * HTTP/1.1\r\nHost:\ www
      server sv1 <srv1_ip>:80 check port 80
      server sv2 <srv2_ip>:80 check port 80
      ....................................................................
      server svN <srvN_ip>:80 check port 80
      server backup <srv_with_maintance_page>:80
```

2. HTTP Load Balancing based on Layer 3 (random)
Config sample:

```
listen <listen_name> <ip_address_to_listen>:801
      mode tcp
      balance roundrobin
      option httpchk OPTIONS * HTTP/1.1\r\nHost:\ www
      server sv1 <srv1_ip>:80 check port 80
      server sv2 <srv2_ip>:80 check port 80
      ....................................................................
      server svN <srvN_ip>:80 check port 80
      server backup <srv_with_maintance_page>:80
```

The difference is only in balance modes:

- first uses **source balance** mode where The source IP address is hashed and divided by the total weight of the running servers to designate which server will receive the request. This ensures that the same client IP address will always reach the same server as long as no server goes down or up. If the hash result changes due to the number of running servers changing, many clients will be directed to a different server. This algorithm is generally used in TCP mode where no cookie may be inserted.
- Second uses **roundrobin balance** mode, where each server is used in turns, according to their weights. This is the smoothest and fairest algorithm when the server's processing time remains equally distributed.

3. HTTP Load Balancing based on Layer 7 (sticky sessions)
This layer allow more options to use. Here is statistics for hosts, http life checking, cookie assignment and others. This is config sample:

```
listen <listen_name> <ip_address_to_listen>:801
      mode http
      cookie SERVERID insert nocache indirect
      stats enable
      stats auth admin:123456
      stats uri /stats
      balance roundrobin
```

```
    cookie JSESSIONID prefix
    option httpclose
    option forwardfor
    option httpchk OPTIONS * HTTP/1.1\r\nHost:\ www
    server sv1 <srv1_ip>:80 cookie S1 check port 80
    server sv1 <srv2_ip>:80 cookie S2 check port 80
    .....................................................................
    server svN <srvN_ip>:80 check port 80
    server backup <srv_with_maintance_page>:80
```

In this case each session will get cookie added (S1, S2, .. , Sn) that will send visitor to the same server in the cluster every time.

Description on options that are used:

Stats enable enable http statistics, that will be availiable using web-url/stats and login+pass combination specified in stats auth field.

option forwardfor HAProxy works in reverse-proxy mode, the servers see its IP address as their client address. This header contains a value representing the client's IP address.

option httpchk a complete HTTP request is sent once the TCP connection is established, and responses 2xx and 3xx are considered valid.

option httpclose By default, when a client communicates with a server, HAProxy will only analyze, log, and process the first request of each connection. If *option httpclose* is set, it will check if a "Connection: close" header is already set in each direction, and will add one if missing. Each end should react to this by actively closing the TCP connection after each transfer, thus resulting in a switch to the HTTP close mode. Any "Connection" header different from "close" will also be removed.

In this case frontend+backend configuration, for more complicated cases, will have a view:

```
frontend <frontend_name>
        bind <ip_address_to_listen>:80
        mode http
        capture request header Cache-Control len 100
        acl backoffice path_beg /backend
        acl backoffice2 path_beg /BackEnd
        use_backend  <back_end_name1> if backoffice
        use_backend  <back_end_name2> if backoffice2
        default_backend <dafault_back_end_name>

backend  <back_end_name1>
       server <server_name> <server_listen_ip>:80 cookie SC1 check inter 2000 fall 3

backend  <back_end_name2>
       server <server_name2> <server_listen_ip2>:80 cookie SC2 check inter 2000 fall 3

backend  <dafault_back_end_name>
        balance roundrobin
        stats enable
        stats realm LoadBalancer_statistics
        stats auth admin:adminpassword
        stats scope <frontend_name>
        stats scope <back_end_name1>
        stats scope <back_end_name2>
        stats scope <dafault_back_end_name>
        stats uri /stats
        server <server_name3> <server_listen_ip3>:80 cookie SC3 check inter 2000 fall 3
        server <server_name4> <server_listen_ip4>:80 cookie SC4 check inter 2000 fall 3
        server backup <backup_server_listen_ip>:80
```

In this example each server response gets sticky cookie (SC1, SC2, SC3, SC4), that will bring customer to the same server in cluster on next visit.

4. HTTP Load Balancing based on Layer 7 (random)

Config for random ballancing will look the same as configs in previous example, but without “cookie SCn” assignment. In this case extra “private” cookie will be added to header, to make it possible to get content if customers will be behind some proxy server, but that customer wouldn't be assigned to any of nodes in cluster, and next time visit the decision will be taken by roundrobin algorithm.

5. HTTP and HTTPS Load Balancing

Http load ballansing is used in regular haproxy way – the way it was described in previous examples. SSL content

ballansing can be done on layer 3 in tcp mode. But in case when cluster hosts are unreachable – their certificates are unreachable too. So we need to store those in 2 places – on web server and at loadballancer itself. To show 503th error pages we'll use nginx. And everything is gonna work.

6. Load Balancing traffic to one server, with high load the extra server will be added to the Load Balance Pool.

In case if too many connections will come to servers and we'll need to add one more server in cluster it can be easily done by adding it into needed config file and after gentle restart (/etc/init.d/haproxy reload) haproxy will start proxying connections to it.

The only stuff needed to do is checking daemon starter parameters at reload section. Pay attention to -sf and -st parameters:

> -sf <pidlist> Send FINISH signal to the pids in pidlist after startup. The processes which receive this signal will wait for all sessions to finish before exiting. This option must be specified last, followed by any number of PIDs. Technically speaking, SIGTTOU and SIGUSR1 are sent.
>
> -st <pidlist> Send TERMINATE signal to the pids in pidlist after startup. The processes which receive this signal will wait immediately terminate, closing all active sessions. This option must be specified last, followed by any number of PIDs. Technically speaking, SIGTTOU and SIGTERM are sent.

We are interested in gentle connections completing, so that line should look like this:

```
$HAPROXY -f "$CONFIG" -p $PIDFILE -D $EXTRAOPTS -sf $(cat $PIDFILE)
```

None of connected customers will see that anything happened to the environment.

7. Maintenance page with error code 503 && 12. Possibility to see the Maintenance Page per customer. So it is possible for the customers to see which maintenance page it is using but it can not see the pages of our other customers.

For this we will need to use some light web server (nginx). All maintenance pages will be stored in one place on loadballancer (/var/www/503_pages), and web server will refer to that location. Common maintenance page is a combination of html message file and few jpg images. Viewing it as compiled web page will return "200 OK" http response in header. So all content should be uploaded with "503 Temporary unavailable" http response in header. To do that rewrite rule is required for nginx host.

Config sample for that:

```
server {
    listen <ip_address_to_listen>:80;
        server_name server-name.nl;
    location / {
        return 503;
        }
    error_page 503 @maint;
    location @maint {
        root /var/www/503_pages/failover-customer/;
        if (!-f $request_filename) {
        rewrite  (some\.jpg|some2\.gif)$ /$1 break;
        rewrite ^(.*)$ /index.html break;
              }
        }
}
```

For https content it is possible to store all certificates in one place (/etc/nginx/SSL/<customer_name>), and use the same nginx hosts to view 503 maintenance pages for https. For that it is required to change port in listen section (80 → 443), and add ssl configuration to server section in virtual host configuration:

```
ssl on;
ssl_protocols       SSLv3 TLSv1;
ssl_certificate /etc/nginx/SSL/<customer_name>/server.pem;
ssl_certificate_key /etc/nginx/SSL/<customer_name>/server.key;
```

All client's configarution could be stored in two files (for http and https content) or separate host's configs can be creaed fot each customer.

8. **One General Maintenance Page but each customer can use their own Maintenance Page**

It can be described in global configuration file (/etc/haproxy/haproxy.cfg.def).

For example:

```
errorfile 503 /etc/haproxy/errors/503.http
```

This line will point to error page, that will be used globally for all customers, who donn't have custom error pages.

9. New visitors will see a heavy load page when it reach a to high amount of visitors
10. All our websites will be used behind the load balancer even if it is a singekl server website. The load balancer can be used for Maintenance or Heavy load pages.
11. Management tools. Overview of active sessions (real time and historical).

13. **Config of different customers does not influence each other :**

Make haproxy read separate configs:
add next to /etc/init.d/haproxy after first variables section:

```
#This is path to directory where client's configs are stored
CONFIG_DIR=/etc/haproxy/Clients
#This part makes it possible to use separate configs for every customer
#Next command reads all cfg files from customer's dirrectory
#and combines them into one config file
#
#Renaming current haproxy configuration file, just in case if something goes wrong
        mv /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.old
#Creating default config
        cp /etc/haproxy/haproxy.cfg.def /etc/haproxy/haproxy.cfg
#Reading config files from Client's directory
        client=$(ls $CONFIG_DIR)
        cd $CONFIG_DIR
        for cfg in $client;
        do
#Make config file more comfortable:
        echo " " >> $CONFIG
        echo "#Using $cfg config" >> $CONFIG
        cat $cfg >>  $CONFIG 2>&1
        done;
```

It is important to realize that we don't have any working static configuration file for our loadballance, because it is being compiled during haproxy start, so we can't check it in regular way after making any kind of changes.

After new file compilation it is highly important to check it on flow, and start haproxy only if it is fine, if it is not – restore working config, – we don't need our loadballancer to fail. For that one more sickle is used:

```
#Checking config file - we donn't want our live haproxy to fail. Steps to take:
# - check compiled config file for errors if any;
# - if everything is OK - goto last step;
# - show number of errors and display errors if any;
# - use previous config if errors found;
# - goto starting haproxy;
test_condition=$(haproxy -c -f $CONFIG 2>&1 |grep parsing |wc -l)
if [ $test_condition != "0" ]
        then
        echo " * There are $test_condition error(s) in configuration" && echo " "
        haproxy -c -f $CONFIG 2>&1 |grep parsing && echo " "
        echo " "
        echo " * Using previously saved config"
        mv /etc/haproxy/haproxy.cfg.old /etc/haproxy/haproxy.cfg
        echo " "
        haproxy -c -f $CONFIG 2>&1 && echo " "
        else echo " " && echo " * Config file is OK" && echo " "
fi
```