****
Must watch this video >>
https://www.youtube.com/watch?v=mIOw4a34LCk

****

HAProxy

****

Introduction

HAProxy, which stands for High Availability Proxy, is a popular open source software TCP/HTTP Load Balancer and proxying solution which can be run on Linux, Solaris, and FreeBSD. Its most common use is to improve the performance and reliability of a server environment by distributing the workload across multiple servers (e.g. web, application, database). It is used in many high-profile environments, including: GitHub, Imgur, Instagram, and Twitter.

In this guide, we will provide a general overview of what HAProxy is, basic load-balancing terminology, and examples of how it might be used to improve the performance and reliability of your own server environment.

****

HAProxy Terminology

****

There are many terms and concepts that are important when discussing load balancing and proxying. We will go over commonly used terms in the following sub-sections.

Before we get into the basic types of load balancing, we will talk about ACLs, backends, and frontends.

Access Control List (ACL)

In relation to load balancing, ACLs are used to test some condition and perform an action (e.g. select a server, or block a request) based on the test result. Use of ACLs allows flexible network traffic forwarding based on a variety of factors like pattern-matching and the number of connections to a backend, for example.

Example of an ACL:

acl url_blog path_beg /blog

This ACL is matched if the path of a user's request begins with */blog*. This would match a request of *http://yourdomain.com/blog/blog-entry-1*, for example.

For a detailed guide on ACL usage, check out the HAProxy Configuration Manual.

Backend

A backend is a set of servers that receives forwarded requests. Backends are defined in the *backend* section of the HAProxy configuration. In its most basic form, a backend can be defined by:

- which load balance algorithm to use
- a list of servers and ports

A backend can contain one or many servers in it--generally speaking, adding more servers to your backend will increase your potential load capacity by spreading the load over multiple servers. Increase reliability is also achieved through this manner, in case some of your backend servers become unavailable.

Here is an example of a two backend configuration, *web-backend* and *blog-backend* with two web servers in each, listening on port 80:

```
backend web-backend
  balance roundrobin
  server web1 web1.yourdomain.com:80 check
  server web2 web2.yourdomain.com:80 check

backend blog-backend
  balance roundrobin
  mode http
  server blog1 blog1.yourdomain.com:80 check
  server blog1 blog1.yourdomain.com:80 check
```

balance roundrobin line specifies the load balancing algorithm, which is detailed in the Load Balancing Algorithms section.

mode http specifies that layer 7 proxying will be used, which is explained in Types of Load Balancing section.

The check option at the end of the server directives specifies that health checks should be performed on those backend servers.

Frontend

A frontend defines how requests should be forwarded to backends. Frontends are defined in the *frontend* section of the HAProxy configuration. Their definitions are composed of the following components:

- a set of IP addresses and a port (e.g. 10.1.1.7:80, *:443, etc.)
- ACLs
- *use_backend* rules, which define which backends to use depending on which ACL conditions are matched, and/or a *default_backend* rule that handles every other case

A frontend can be configured to various types of network traffic, as explained in the next section.

****

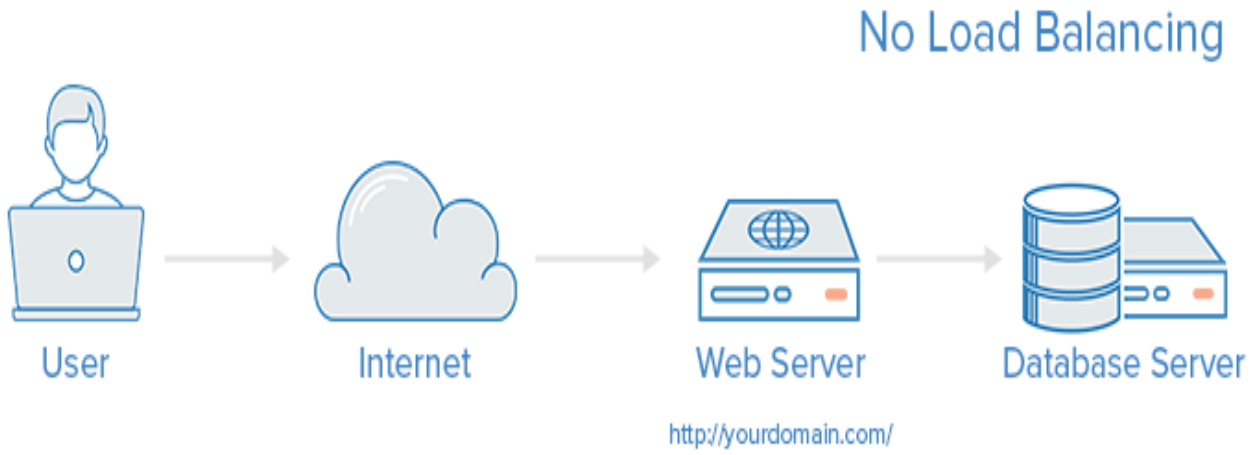
Types of Load Balancing

****

Now that we have an understanding of the basic components that are used in load balancing, let's get into the basic types of load balancing.

No Load Balancing

A simple web application environment with no load balancing might look like the following:

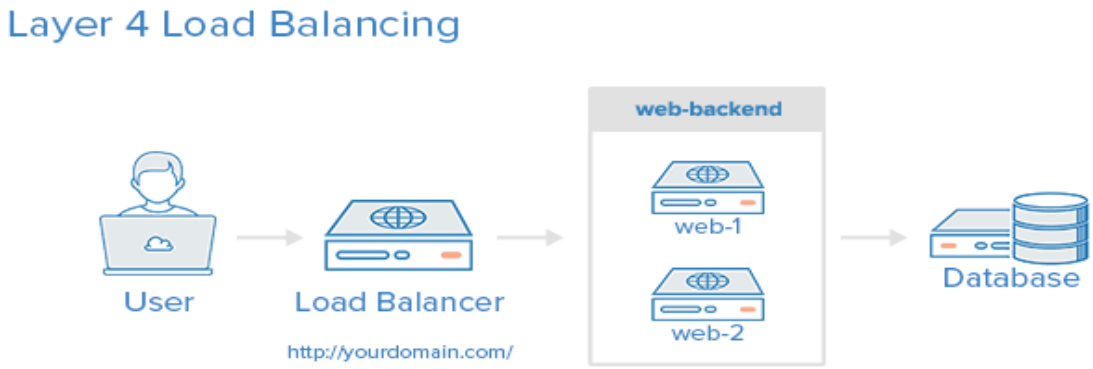


In this example, the user connects directly to your web server, at *yourdomain.com* and there is no load balancing. If your single web server goes down, the user will no longer be able to access your web server. Additionally, if many users are trying to access your server simultaneously and it is unable to handle the load, they may have a slow experience or they may not be able to connect at all.

Layer 4 Load Balancing

The simplest way to load balance network traffic to multiple servers is to use layer 4 (transport layer) load balancing. Load balancing this way will forward user traffic based on IP range and port (i.e. if a request comes in for *http://yourdomain.com/anything*, the traffic will be forwarded to the backend that handles all the requests for *yourdomain.com* on *port 80*). For more details on layer 4, check out the *TCP* subsection of our Introduction to Networking.

Here is a diagram of a simple example of layer 4 load balancing:

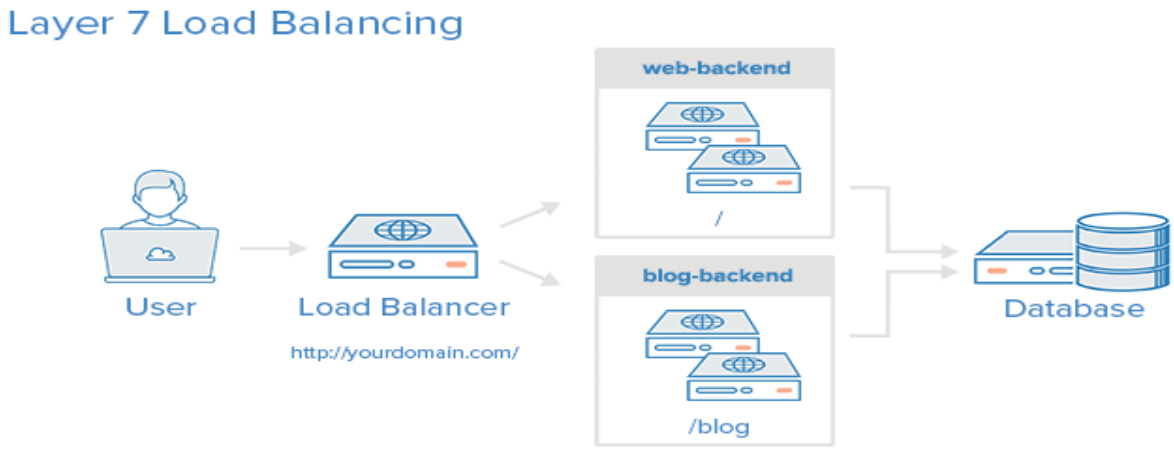


The user accesses the load balancer, which forwards the user's request to the *web-backend* group of backend servers. Whichever backend server is selected will respond directly to the user's request. Generally, all of the servers in the *web-backend* should be serving identical content--otherwise the user might receive inconsistent content. Note that both web servers connect to the same database server.

Layer 7 Load Balancing

Another, more complex way to load balance network traffic is to use layer 7 (application layer) load balancing. Using layer 7 allows the load balancer to forward requests to different backend servers based on the content of the user's request. This mode of load balancing allows you to run multiple web application servers under the same domain and port. For more details on layer 7, check out the *HTTP* subsection of our Introduction to Networking.

Here is a diagram of a simple example of layer 7 load balancing:



In this example, if a user requests *yourdomain.com/blog*, they are forwarded to the *blog* backend, which is a set of servers that run a blog application. Other requests are forwarded to *web-backend*, which might be running another application. Both backends use the same database server, in this example.

A snippet of the example frontend configuration would look like this:

```
frontend http
  bind *:80
  mode http
```

acl url_blog path_beg /blog
use_backend blog-backend if url_blog

default_backend web-backend

This configures a frontend named *http*, which handles all incoming traffic on port 80.

acl url_blog path_beg /blog matches a request if the path of the user's request begins with */blog*.

use_backend blog-backend if url_blog uses the ACL to proxy the traffic to *blog-backend*.

default_backend web-backend specifies that all other traffic will be forwarded to *web-backend*.

****

Load Balancing Algorithms

****

The load balancing algorithm that is used determines which server, in a backend, will be selected when load balancing. HAProxy offers several options for algorithms. In addition to the load balancing algorithm, servers can be assigned a *weight* parameter to manipulate how frequently the server is selected, compared to other servers.

Because HAProxy provides so many load balancing algorithms, we will only describe a few of them here. See the HAProxy Configuration Manual for a complete list of algorithms.

A few of the commonly used algorithms are as follows:

roundrobin

Round Robin selects servers in turns. This is the default algorithm.

leastconn

Selects the server with the least number of connections--it is recommended for longer sessions. Servers in the same backend are also rotated in a round-robin fashion.

source

This selects which server to use based on a hash of the source IP i.e. your user's IP address. This is one method to ensure that a user will connect to the same server.

Sticky Sessions

Some applications require that a user continues to connect to the same backend server. This persistence is achieved through sticky sessions, using the *appsession* parameter in the backend that requires it.

Health Check

HAProxy uses health checks to determine if a backend server is available to process requests. This avoids having to manually remove a server from the backend if it becomes unavailable. The default health check is to try to establish a TCP connection to the server i.e. it checks if the backend server is listening on the configured IP address and port.

If a server fails a health check, and therefore is unable to serve requests, it is automatically disabled in the backend i.e. traffic will not be forwarded to it until it becomes healthy again. If all servers in a backend fail, the service will become unavailable until at least one of those backend servers becomes healthy again.

For certain types of backends, like database servers in certain situations, the default health check is insufficient to determine whether a server is still healthy.

Other Solutions

If you feel like HAProxy might be too complex for your needs, the following solutions may be a better fit:

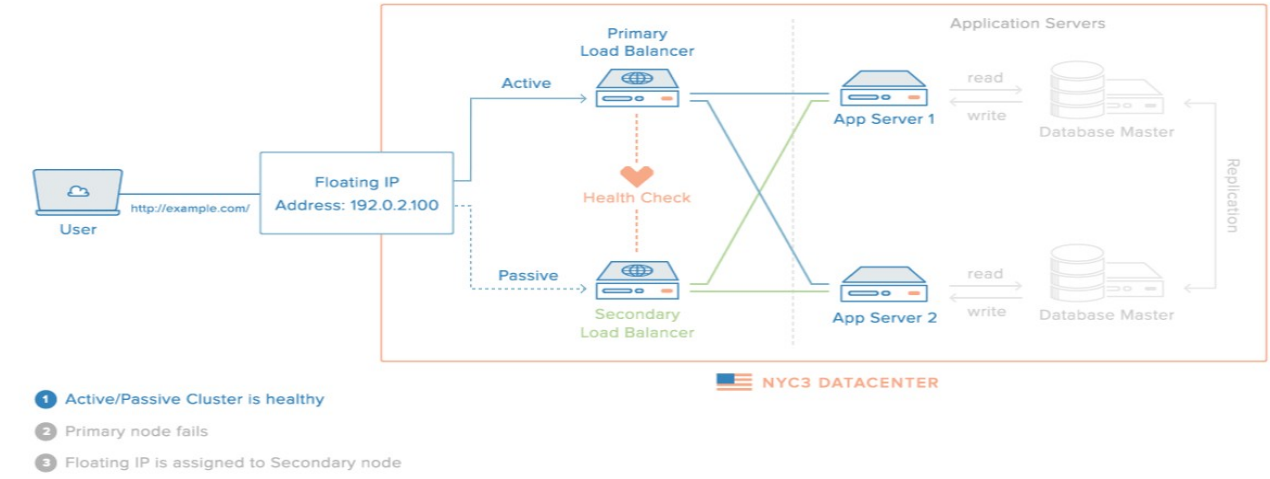
- Linux Virtual Servers (LVS) - A simple, fast layer 4 load balancer included in many Linux distributions
- Nginx - A fast and reliable web server that can also be used for proxy and load-balancing purposes. Nginx is often used in conjunction with HAProxy for its caching and compression capabilities

High Availability

The layer 4 and 7 load balancing setups described before both use a load balancer to direct traffic to one of many backend servers. However, your load balancer is a single point of failure in these setups; if it goes down or gets overwhelmed with requests, it can cause high latency or downtime for your service.

A *high availability* (HA) setup is an infrastructure without a single point of failure. It prevents a single server failure from being a downtime event by adding redundancy to every layer of your architecture. A load balancer facilitates redundancy for the backend layer (web/app servers), but for a true high availability setup, you need to have redundant load balancers as well.

Here is a diagram of a basic high availability setup:



In this example, you have multiple load balancers (one active and one or more passive) behind a static IP address that can be remapped from one server to another. When a user accesses your website, the request goes through the external IP address to the active load balancer. If that load balancer fails, your failover mechanism will detect it and automatically reassign the IP address to one of the passive servers. There are a number of different ways to implement an active/passive HA setup. To learn more, read [this section of How To Use Floating IPs](#).

****

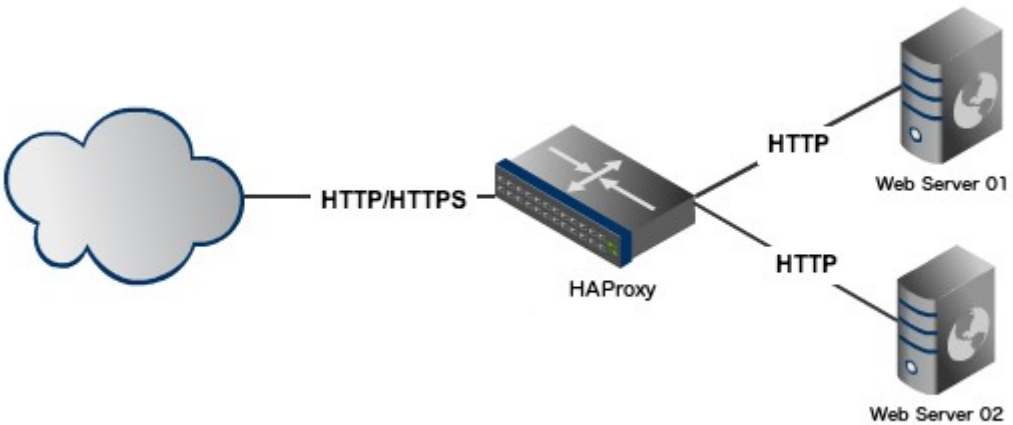
Installation

****

 # apt-get install software-properties-common
 # add-apt-repository ppa:vbernat/haproxy-1.6

 # apt-get update
 # apt-get install haproxy

links ::
https://haproxy.debian.net/#?distribution=Ubuntu&release=trusty&version=1.6



We need to enable HAProxy to be started by the init script /etc/default/haproxy. Set ENABLED option to 1 as:

nano /etc/default/haproxy

ENABLED=1

edit /etc/haproxy/haproxy.cfg

nano /etc/haproxy/haproxy.cfg

We'll create our own haproxy.cfg file. Using your favorite text editor create the /etc/haproxy/haproxy.cfg file as:

```
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin
        stats timeout 30s
        user haproxy
        group haproxy
        daemon

        # Default SSL material locations
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private

        # Default ciphers to use on SSL-enabled listening sockets.
        # For more information, see ciphers(1SSL). This list is from:
        #  https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
        ssl-default-bind-ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:E
CDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES:RSA+3DES:!aNULL:!MD5:!DSS
        ssl-default-bind-options no-sslv3

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
      timeout connect 5000
      timeout client  50000
      timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
```

```
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

listen webui
        bind *:8888
        stats enable
        stats uri /
        mode http
        stats auth admin:admin

frontend http
        bind *:80
        mode http

#       acl url_biz path_beg /bizruntime/public_html
#       use_backend biz-backend if url_biz

          acl bad_ip src 192.168.1.25
        use_backend bad_guy if bad_ip

        default_backend servers

backend servers
          balance roundrobin
        mode http
        server server3 192.168.1.32:80 check

backend biz-backend
        balance roundrobin
        mode http
        server server3 192.168.1.33:80 check

backend bad_guy
        mode http
        server server3 192.168.1.33:80 check
```

Explanation:

1) global
2) defaults
3) listen
4) frontend
5) backend

1) global

The log directive mentions a syslog server to which log messages will be sent.
The maxconn directive specifies the number of concurrent connections on the front-end. The default value is 2000 and should be tuned according to your system's configuration.
The user and group directives changes the HAProxy process to the specified user/group. These shouldn't be changed.

2) default

The above section has the default values. The option redispatch enables session redistribution in case of connection failures. So session stickness is overriden if a web server instance goes down. The retries directive sets the number of retries to perform on a web server instance after a connection failure.
The values to be modified are the various timeout directives. The contimeout option specifies the maximum time to wait for a connection attempt to a web server instance to succeed.
The clitimeout and srvtimeout apply when the client or server is expected to acknowledge or send data during the TCP process. HAProxy recommends setting the client and server timeouts to the same value.

3)

Above block contains configuration for both the frontend and backend. We are configuring HAProxy to listen on port 80 for webfarm which is just a name for identifying an application.
The stats directives enable the connection statistics page. This page can viewed with the URL mentioned in stats uri so in this case, it is http://192.168.205.15/haproxy?stats a demo of this page can be viewed here.
The balance directive specifies the load balancing algorithm to use. Algorithm options available are:

Round Robin (roundrobin),

Static Round Robin (static-rr),

Least Connections (leastconn),

Source (source),

URI (uri) and

URL parameter (url_param).

In the listen part, it added web interface for managing

4) and 5) can also added separately if those didnt add on listen part

4) the frontend server mentioned the haproxy server info and acl configuration

5) backend means the machines which needed to connected by clients

the acl rules can be used for many purpose,

eg:

Access Control List (ACL)

In relation to load balancing, ACLs are used to test some condition and perform an action (e.g. select a server, or block a request) based on the test result. Use of ACLs allows flexible network traffic forwarding based on a variety of factors like pattern-matching and the number of connections to a backend, for example.

Example of an ACL:

acl url_blog path_beg /blog

This ACL is matched if the path of a user's request begins with */blog*. This would match a request of *http://yourdomain.com/blog/blog-entry-1*, for example.

Links ::
https://www.haproxy.com/doc/aloha/7.0/haproxy/acls.html

for adding more haproxy server

added this in configuration file

at backend portion >>

stick-table type ip size 20k peers mypeers

at the bottom >>

```
peers mypeers
        peer haproxy 172.17.0.2:1024
        peer haproxy1 172.17.0.5:1024
```

so the entire configuration file looks like this,

```
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin
        stats timeout 30s
        user haproxy
        group haproxy
        daemon

        # Default SSL material locations
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private

        # Default ciphers to use on SSL-enabled listening sockets.
        # For more information, see ciphers(1SSL). This list is from:
        #  https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
        ssl-default-bind-ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:E
CDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES:RSA+3DES:!aNULL:!MD5:!DSS
        ssl-default-bind-options no-sslv3

defaults
        log     global
```

```
    mode   http
    option httplog
    option dontlognull
  timeout connect 5000
  timeout client  50000
  timeout server  50000
    errorfile 400 /etc/haproxy/errors/400.http
    errorfile 403 /etc/haproxy/errors/403.http
    errorfile 408 /etc/haproxy/errors/408.http
    errorfile 500 /etc/haproxy/errors/500.http
    errorfile 502 /etc/haproxy/errors/502.http
    errorfile 503 /etc/haproxy/errors/503.http
    errorfile 504 /etc/haproxy/errors/504.http

listen webui
  bind *:8888
  stats enable
  stats uri /
  mode http
  stats auth admin:admin

frontend http
  bind *:80
  mode http

    acl url_biz path_beg /biz
    use_backend biz-backend if url_biz

  default_backend servers

backend servers
    balance roundrobin
  stick-table type ip size 20k peers mypeers
    server server1 172.17.0.3:80 check
  server server2 172.17.0.4:80 check

backend biz-backend
    balance roundrobin
    mode http
    stick-table type ip size 20k peers mypeers
    server server3 172.17.0.6:80 check
    server server4 172.17.0.7:80 check

peers mypeers
  peer haproxy 172.17.0.2:1024
  peer haproxy1 172.17.0.5:1024
```

testing ::

add any domain name for the both haproxy server for accessing if one fall down.

nano /etc/hosts

172.17.0.2 ha.biz.com
172.17.0.5 ha.biz.com

so if 172.17.0.2 server fall down , the ha.biz.com is accessable with help of second haproxy server.

****

Links ::

videos >>
https://www.youtube.com/watch?v=mIOw4a34LCk

concepts >>
https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts

installation >>
https://haproxy.debian.net/#?distribution=Ubuntu&release=trusty&version=1.6

official documentation >>
http://cbonte.github.io/haproxy-dconv/

configuration >>
https://www.howtoforge.com/tutorial/ubuntu-load-balancer-haproxy/

acl rules >>
https://www.haproxy.com/doc/aloha/7.0/haproxy/acls.html

****