**HA PROXY:**

* The goal behind ha proxy is to provide fast, easy and reliable solution for load balancing tcp and http-based applications.
* It has been designed to run on Linux, Solaris, bsd and AIX platforms and mostly on x86-64 bit platforms.

**KEY CONCEPTS:**

* **FRONTEND** = It dictates where and how the incoming traffic should be routed to machines behind ha proxy. They allow you to setup rules (acl’s) that will watch for specific URL syntax and route a user’s request as needed.

**frontend http-in**

**bind \*:80**

**default\_backend appX-backend**

* In the above example, we are only allowing port **80** for incoming traffic and redirecting all traffic to default backend called **appX-backend**.
* **BACKEND** = This is where you hosted your data and applications. These backend servers are where we route all traffic by specifying rules in frontend.
* Backend can be a single server (or) multiple servers. But in the context of load balancers, it should be at least 2 servers.

**backend appX-backend**

**balance roundrobin**

**server appX\_01 192.168.2.2:8080 check**

**server appX\_02 192.168.2.3:8080 check**

* In the above example, the backend is called **appx-backend** with two servers that are using round robin method to route traffic between them.
* **ACLs** = these are the backbone for complex configurations that contains multiple frontends and multiple backends and needed very precise routing. With acls you can rewrite and redirect all requests as needed.

**frontend http-in**

**bind \*:80**

**acl url\_appX path\_beg -i /appX/**

**use\_backend appX-backend if url\_appX**

**default\_backend appZ-backend**

* In the above example, if the incoming request is <http://example.com/appX>, route the traffic to backend appX and route all remaining traffic to appZ-backend.
* **ALGORITHMS** = ha proxy comes with multiple number of options when it comes to chosing the method in which you want requests to be serve you backend.

**ROUND** **ROBIN** = Each server used in a never ending line, starting from the first one until end of the list. The next request will go to the first one again. By default, ha proxy uses round robin method, if you don’t specify any.

**LEAST** **CONNECTION** = ha proxy determines which server has the least no of connections and sends the next request to that server. This algorithm is great for long lasting connections such as **LDAP** and **SQl** but not for **HTTP**.

* **Configuration file = /etc/haproxy/haproxy.cfg**
* **Binary = /usr/sbin/haproxy**
* **Init script = /etc/init.d/haproxY**

**CONFIGURATION FILE:**

* HA proxy is completely controlled by this file haproxy.cfg. this is where you build you frontend and backend servers**.**
* These are some of the settings that you find in haproxy.cfg.
* **Mode http =** By setting this option, ha proxy can now inspect http headers for all requests and modify and redirect each request. The other option is TCP, which allows ha proxy to route per ip and port ignoring all http headers.
* **Time Outs =** ha proxy has ability to set timeouts for various acpects of http/tcp protocols.

**Timeout connection =** time to wait for successful connection

**Timeout server =** maximum server side inactivity time**.**

**Timeout client =** maximum client side inactivity time**.**

* Like this, there are multiple number of timeouts as you can check in haproxy documentation.
* **Option httpd-server-close** = this option allows you to configure how the ha proxy handles connections from server side. By default, ha proxy runs in keep-alived mode, where connections are keep open and in idle state. But, by using this option, we can force close these open connections from consuming resources.
* **Rspadd x-forwaded-host =** the common use case of this is when hosting various applications behind ha proxy and you ensure that end users only sees responces from the ha proxy not from the machines behind it.
* **Active/Passive =** It is used when you need to all the traffic to be routed to a single until it goes offline. After it went offline, ha proxy automatically route traffic to 2nd machine.

**backend appZ-backend**

**server appZ\_01 192.168.2.2:8080 check**

**server appZ\_02 192.168.2.3:8080 check backup**

**INSTALLATION:**

* Prepare your backend servers with http installed.
* Go to your frontend server.
* **yum install haproxy** = To install ha proxy.
* After installing haproxy we need to enable logging for further debugging.
* Open main config file haproxy.cfg, uncomment this line

**log 127.0.0.1 local2**

* Now, we need to enable udp syslog reception in rsyslog.conf to separate haproxy logs from others.
* Open **/etc/rsyslog.conf**. Uncomment **ModLoad**, **UdpServerRun**. Our server will listen to port **514** to collect logs into syslog.
* Now create a file called **haproxy.conf** in **/etc/rsyslog.d/** dir to configure separate logs.
* After creating the file in rsyslog.d dir. Add this line in that file.

**local2.\* /var/log/haproxy.log**

* Go to main config file **/etc/haproxy/haproxy.cfg**.
* We have done with the logs, now we have to configure our frontend and backend servers.
* Go to end of the file, add your frontend and backend servers.

**frontend**

**bind \*:80**

**default\_backend APP**

* Now add backend servers below frontend configuration.

**backend APP**

**balance roundrobin**

**server HOSTNAME IP:PORT check**

**server HOSTNAME IP:PORT check**

* Place your server hostnames, ips and ports in the place of HOSTNAME, IP, PORT and save the file.
* Restart the haproxy server. Check the status whether its running (or) not.
* Go to browser, type your load balancer ip to see your backend servers site.
* It will load traffic based on roundrobin method.
* If you want to see the statistics of your backend and frontend servers.
* Go to **/etc/haproxy/haproxy.cfg**, add your status configuration.

**listen stats**

**bind \*:PORT**

**stats enable**

**stats hide-version**

**stats uri /stats**

**stats auth admin:admin@123**

* **PORT** = port for statistics report requests (your choice).
* **/stats** = url for statistics page.
* **Admin:admin@123** = user and password to access the statistics page.

**KEEPALIVED:**

* Ha proxy acts as a load balancer for our backend servers where **keepalived** acts as **high** **availability** solution for our servers.
* Keepalived is a routing software written in C. The main goal of this project is to provide simple, rodust facilities for load balancing and HA to linux systems and linux based infrastructures.
* What we have to do is to take 4 servers. Install ha proxy and keepalived on both servers.

**yum install haproxy && yum install keepalived –y**

* After installing 2 packages in both servers. We have to creating a floating ip**(VIP)** that can be moved between capable load balancers. This will be configured each to split traffic between 2 backend servers. If the primary load balancer goes down, all the traffic is redirected to 2nd load balancer automatically.
* Install httpd in another 2 servers and configure load balancing on both ha proxy servers with the http servers.
* Go to **/etc/keepalived/keepalived.conf,**  paste this content

**vrrp\_script chk\_haproxy {**

**script "killall -0 haproxy" # check the haproxy process**

**interval 2**

**weight 2**

**}**

**vrrp\_instance VI\_1 {**

**interface eth0**

**state MASTER**

**virtual\_router\_id 51**

**priority 101**

**virtual\_ipaddress {**

**}**

**track\_script {**

**chk\_haproxy**

**}**

**}**

* In the above configuration,
* It wil check the haproxy for every 2 seconds and add 2 points**(weight)** if its OK.
* In this, we are specifying the state of HA Proxy as **MASTER**.
* **Virtual\_router\_id** = the route id to route traffic.
* **Priority** = the priority of the server (master > backup).
* **Virtual** **ip** = the floating ip between master and backup.
* Go to backup server, copy the above configuration and do some modifications as per the requirement.
* For ex:

**The priority of backup server should be less than master server.**

**State should be backup.**

* Once we configured everything, start the **haproxy** and **keepalived** on **both** servers and check logs on both servers **(/var/log/messages).**
* In master server, it will show like this “ **ENTERING MASTER STATE**”.
* In backup server, it will show like this “**ENTERING BACKUP STATE**”.
* If the logs are shown like above, then you have configured keepalived successfully.
* The above logs, means that if master server goes down. All requests are routed to backup server and it acts as master server until master is up.
* Stop the haproxy in master server. Wait for few minutes and refresh your bowser you will see your content from backend. Check the logs im /var/log/messages in backup server, it will say **“ENTERING MASTER STATE”.** Which means backup is acting as master because master is down.