**IN YOUR AWS CONSOLE**

***00. Beginning***

\* What is HAProxy ?

- Open-source load balancing software written in C

- Event-based (?)

- SOLE PURPOSE : To run a reverse proxy

- Fast and lightweight ... small memory and CPU requirements

- 20k connections/sec out of the box (over 100k/sec tuned)

- No major bugs since 2002

***01. Security Groups and Ports***

\* We don't really need HTTPS for this demo

**ON \*EACH\* WEB SERVER EC2 INSTANCE**

***01. Set up nginx as the web server***

\* /srv/www ... will serve as our base directory for web content

- Usually the base directory for content is /var/www

- Not sure why we had to change this

\* www-data:www-data is the user

- Needs to have access to /srv/www and all of its files

***02. Prepare your web page content***

\* /etc/nginx/sites-enabled/default ... the usual default location

- Not sure why we had to change this

- Open default BEFORE removing it (?)

- To compare its content to our new "default" /sites-enabled/tutorial

***03. Test your Nginx setup***

\* "Reload" will preserve any existing connections

- "Start" and "Restart" will destroy any existing connections

**ON THE LOAD BALANCER EC2 INSTANCE (Pt 2)**

***01. Configure Load Balancer***

**defaults**

***log global***

***mode http***

***option httplog***

- By default, the log output format sucks (source, destination, instance name)

- This turns each log line into a much richer format, including:

\* HTTP request, session status, captured cookies

***option dontlognull***

- Turns off logging for sessions in which no data were exchange

- Helps to ignore all the logs from successful health checks

***contimeout 5000***

- max time to wait for a connection attpemt to a web server

***clitimeout 5000***

- applies when the client needs to acknowledge or send data

***srvtimeout 5000***

- applies when the server needs to acknowledge or send data

- client and server timeout values should be identical

***errorfile 400 /etc/haproxy/errors/400.http***

------------------

***listen webfarm 0.0.0.0:80***

- Contains info for both the frontend and the backend

- "webfarm" is just a name for identifying an application

- HAProxy will listen for all traffic coming across Port 80

***mode http***

- Gives us HTTP-specific options

\* Choosing different backends based on the URL in the HTTP request

- "How does the PC in Europe know which web page you want ?"

\* But... we would need to negotiate/decrypt any SSL traffic on HAProxy

BEFORE parsing the HTTP headers

- Best to do handle SSL Termination on your web servers

***stats enable***

- Enables the connection stats paage for monitoring

***stats uri /haproxy?stats***

- URI for Web Interface (use public address)

\* Which app servers are up/down

\* Possible to generate results in .csv to feed into other programs

***balance roundrobin***

- RoundRobin : the next app server in the list is used

- LeastConn : the app server w/ the least connections is used

- Source : Sticky session using a has of the client's IP address

\* BAD!! Creates hot and cold spots

- URI : Like "Source" except it uses the URL requested

\* Good for serving static assets (images, css, js)

***option httpclose***

- Not totally sure yet... Something to do w/ HAProxy not support keep-alive

***option forwardfor***

- HAProxy works in revers-proxy mode... So its servers see its IP Address

- The HTTP header "X-Forwarded-For" can be added by HAProxy to its servers

- This will be the last header in the header list

- Make sure your web server knows how to handle this from HAProxy

\* Nginx => Add real\_ip\_header proxy\_protocol to NGINX config file

***server web1 172.30.1.80:80 check***

- Declares a backend server

- "web1" is the name for this web server that will appear in logs and alerts

- “check” option enables health checks

-------------------

***maxconn***

- # of connections HAProxy will accept at once (set high as possible)

- each connection uses 33 KB of memory

- default value is 2000

***option dontlog-normal***

- Will only log errors, timeouts and other issues

- Nginx will log normal request in its access logs

***option splice-auto***

- Enables TCP splicing if needed

- A Linux enhancement that speeds up packet handling for TCP conn's

***timeout connect|client|server 5000ms***

- Timeouts for different connections (in milliseconds)

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

***Layer X - API's***

***Layer 5 - Application (HTTP, SMTP, SSH, etc.)***

- HTTP is an application-layer protocol

- How does the PC in Europe know which web page you want? Application (Layer 5) takes care of this. HTTP for web... SMTP, IMAP, and POP3 for email clients... IRC for chat... SSH, Telnet, RDP for remote administration

***Layer 4 - Transport (TCP, UDP)***

- TCP is a transport-layer protocol

- You're talking to a PC in Europe, but which of its programs does your message need to go to? Transport (Layer 4) takes care of this.

- TCP reorders packets, resubmits lost packets, etc.

***Layer 3 - Network (IPv4, IPv6)***

- Where the magic of the Internet happens

- You can talk to a PC in Europe w/o knowing where it is

- Routes handle directing your traffic

***Layer 2 - Link (Ethernet, 802.11a/b/g/n)***

- Devices that share a physical communications medium

- How your computer talks to your home router

***Layer 1 - Physical***

- Electrical signals (or light or radio) transmit information

**Next Steps:**

\* Simulate Network Load (To test different algorithms)

\* Ways to forward the client's IP Address to web servers

- Add real\_ip\_header proxy\_protocol to NGINX config file !

\* Failover Load Balancer (using KeepAliveD)

\* Add a Database

\* Linux Tunings

\* Using In-Memory DB's

- Session Management

- API Authentication/Authorization

\* SSL Termination

\* Add high-level documentation