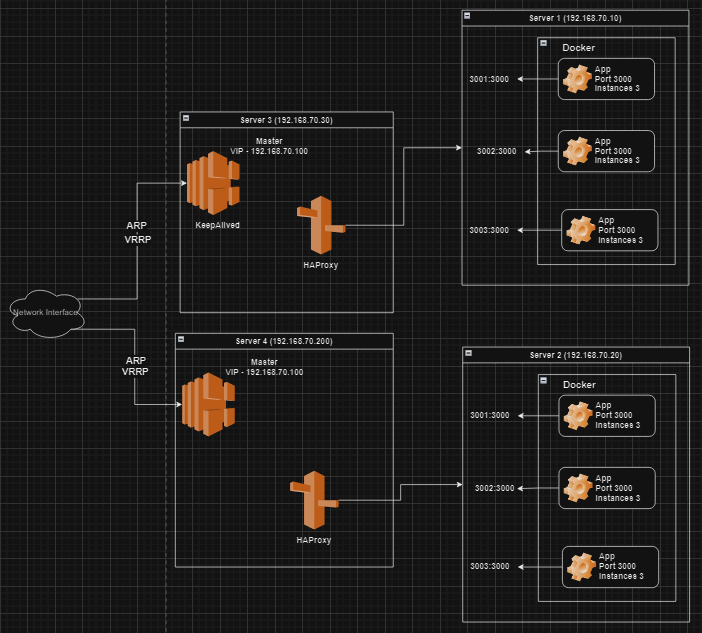
Setup Description



1. Install Docker in server 1 & 2

sudo yum check-update

curl -fsSL https://get.docker.com/ | sh

sudo systemctl start docker

sudo systemctl status docker

2. Install Docker Compose in sevrer 1 & 2

sudo curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

docker compose version

3. copy the docker compose file to run the servers in server1 and server 2 and up the services

a. create websrv folder and initalize npm

cd websrv

npm init -y

b. install express

npm i express

c. create index.js file and copy below

const express = require("express")

const os = require("os")

const app = express()

app.get("/", (req,res) => {

res.send(process.env.serverid)

})

app.listen(3000, () => {

console.log("listening to port 3000")

})

d. create Dockerfile and copy below code

FROM node:18

# Create app directory

WORKDIR /app

# Install app dependencies

# A wildcard is used to ensure both package.json AND package-lock.json are copied

# where available (npm@5+)

COPY package\*.json ./

RUN npm install

# If you are building your code for production

# RUN npm ci --omit=dev

# Bundle app source

COPY index.js .

EXPOSE 3000

CMD [ "node", "index.js" ]

e. come outside the websrv folder and create docker-compose.yml file and copy below code

services:

web-srv1:

build: ./websvr

restart: always

environment:

- serverid=web-srv1

deploy:

mode: replicated

replicas: 1

web-srv2:

build: ./websvr

restart: always

environment:

- serverid=web-srv2

deploy:

mode: replicated

replicas: 1

web-srv3:

build: ./websvr

restart: always

environment:

- serverid=web-srv3

deploy:

mode: replicated

replicas: 1

f. Run docker comopse up -d to run the complete setup

2. setup haproxy loadbalancer in server3 and server4

a. sudo yum update -y

b. sudo yum install haproxy

c. update the haproxy config in /etc/haproxy/haproxy.cfg

**global**

**maxconn 50000**

**log /dev/log local0**

**defaults**

**timeout connect 10s**

**timeout client 30s**

**timeout server 30s**

**log global**

**mode http**

**option httplog**

**maxconn 3000**

**frontend mysite**

**bind \*:80**

**default\_backend web\_servers**

**backend web\_servers**

**balance roundrobin**

**default-server check maxconn 20**

**server server1 192.168.70.10:3001**

**server server2 192.168.70.10:3002**

**server server3 192.168.70.10:3002**

d. run command to start haproxy

systemctl start haproxy

systemctl status haproxy

3. Setup keepalived in both server3 and server4 and assign a virtual ip

a. install keepalived

yum install keepalived -y

b. update the config for keepalived in file /etc/keepalived/keepalived.conf

vrrp\_instance VI\_1 {

    state MASTER                            # Set as MASTER or BACKUP on each server

    interface eth0                          # Network interface to monitor

    virtual\_router\_id 51                    # Unique ID for the virtual router

    priority 100                            # Higher value means higher priority on MASTER

    advert\_int 1                            # Advertisement interval in seconds

    authentication {

        auth\_type PASS                       # Authentication type (PASS or AH)

        auth\_pass PASSWORD                    # Authentication password

    }

    virtual\_ipaddress {

        192.168.70.100           # Virtual IP and subnet mask

    }

}

vrrp\_instance VI\_2 {

    state BACKUP                            # Set as MASTER or BACKUP on each server

    interface eth0                          # Network interface to monitor

    virtual\_router\_id 52                    # Unique ID for the virtual router

    priority 101                            # Higher value means higher priority on MASTER

    advert\_int 1                            # Advertisement interval in seconds

    authentication {

        auth\_type PASS                       # Authentication type (PASS or AH)

        auth\_pass PASSWORD                    # Authentication password

    }

    virtual\_ipaddress {

        192.168.70.100           # Virtual IP and subnet mask

    }

}

c. systemctl start keepalived

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

HAProxy Conifg --

**global**

**maxconn 50000**

**log /dev/log local0**

**user haproxy**

**group haproxy**

**stats socket /run/haproxy/admin.sock user haproxy group haproxy mode 660 level admin**

**nbproc 2**

**nbthread 4**

**cpu-map auto:1/1-4 0-3**

**ssl-default-bind-ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256**

**ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets**

/\*

> The maxconn setting limits the maximum number of connections that HAProxy will accept. Its purpose is to protect your load balancer from running out of memory.

> The log setting ensures that warnings emitted during startup and issues that arise during runtime get logged to syslog. It also logs requests as they come through. You can target the traditional UNIX socket where Syslog or journald, listen, /dev/log, or specify a remote rsyslog server so that log data is preserved externally to your load balancing server. Set a Syslog facility, which is typically local0, which is a facility categorized for custom use. Note that in order to read the logs, you will need to configure any of the syslog daemons, or journald, to write them to a file.

> The user and group lines tell HAProxy to drop privileges after initialization. Linux requires processes to be root in order to listen on ports below 1024. You’ll also typically want your TLS private keys to be readable only by root as well.

> The stats socket line enables the Runtime API, which you can use to dynamically disable servers and health checks, change the load balancing weights of servers, and pull other useful levers.

> The nbproc and nbthread settings specify the number of processes and threads, respectively, that HAProxy should spawn on startup. This can increase the efficiency of your load balancer. However, each process created by nbproc has its own stats, stick tables, health checks, and so on. Threads created with nbthread, on the other hand, share them.

> The ssl-default-bind-ciphers setting enumerates the SSL and TLS ciphers that every bind directive will use by default. It can be overridden with a more specific setting by adding the bind directive’s ciphers parameter. It takes a list of cipher suites in order of preference. HAProxy will select the first one listed that the client also supports, unless the prefer-client-ciphers option is enabled.

> The ssl-default-bind-options setting configures SSL/TLS options such as ssl-min-ver to disable support for older protocols.

\*/

**defaults**

**timeout connect 10s**

**timeout client 30s**

**timeout server 30s**

**log global**

**mode http**

**option httplog**

**maxconn 3000**

/\*

> The timeout connect setting configures the time that HAProxy will wait for a TCP connection to a backend server to be established. The “s” suffix denotes seconds. Without any suffix, the time is assumed to be in milliseconds. The timeout client setting measures inactivity during periods that we would expect the client to be speaking, or in other words sending TCP segments. The timeout server setting measures inactivity when we’d expect the backend server to be speaking. When a timeout expires, the connection is closed. Having sensible timeouts reduces the risk of deadlocked processes tying up a connections that could otherwise be reused.

When operating HAProxy in TCP mode, which is set with mode tcp, timeout server should be the same as timeout client.

> The log global setting is a way of telling each subsequent frontend to use the log setting that you defined in the global section. This isn’t required for logging, as new log lines can be added here or in each frontend.

> The mode setting defines whether HAProxy operates as a simple TCP proxy or if it’s able to inspect incoming traffic’s higher-level HTTP messages. The alternative to specifying mode http is to use mode tcp, which operates at the faster but less-aware level

> The maxconn setting limits the number of connections each frontend will accept and, by default, is set to 2000. If you want to allow more connections, you can increase it here up to your global maxconn.

> The option httplog setting, or more rarely option tcplog, tells HAProxy to use a more verbose log format when sending messages to Syslog. You will generally prefer option httplog over option tcplog in your defaults section because when HAProxy encounters a frontend that uses mode tcp, it will emit a warning and downgrade it to option tcplog anyway.

\*/

**frontend www.mysite.com**

**bind 10.0.0.3:80**

**bind 10.0.0.3:443 ssl crt /etc/ssl/certs/mysite.pem**

**http-request redirect scheme https unless { ssl\_fc }**

**use\_backend api\_servers if { path\_beg /api/ }**

**default\_backend web\_servers**

/\*

> A bind setting assigns a listener to a given IP address and port. The IP can be omitted to bind to all IP addresses on the server, and a port can be a single port, a range, or a comma-delimited list. You’ll often use the ssl and crt arguments to instruct HAProxy to manage SSL/TLS terminations rather than having your web servers doing that.

> A http-request redirect setting responds to the client that they should try a different URL.

> The use\_backend setting chooses a backend pool of servers to respond to incoming requests if a given condition is true. It is followed by an ACL statement, such as if path\_beg /api/, that allows HAProxy to select a specific backend based on some criteria, such as checking if the path begins with /api/.

> The default\_backend setting is found in nearly every frontend and gives the name of a backend to send traffic to if a use\_backend rule doesn’t send it elsewhere first. If a request isn’t routed by a use\_backend or default\_backend directive, HAProxy will return a 503 Service Unavailable error.

\*/

**backend web\_servers**

**balance roundrobin**

**cookie SERVERUSED insert indirect nocache**

**option httpchk HEAD /**

**default-server check maxconn 20**

**server server1 10.0.1.3:80 cookie server1**

**server server2 10.0.1.4:80 cookie server2**

/\*

> The balance setting controls how HAProxy will select the server to respond to the request if no persistence method overrides that selection. A persistence method might be to always send a particular client to the same server based on a cookie. Common load balancing values include roundrobin, which just picks the next server and starts over at the top of the list again, and leastconn, where HAProxy selects the server with the fewest active sessions.

> The cookie setting enables cookie-based persistence. It tells HAProxy to send a cookie named SERVERUSED to the client, and to associate it with the server's name that gave the initial response. This causes the client to continue speaking with that server for the duration of their session. Note that the server's name is set with a cookie argument on the server line.

> The option httpchk setting causes HAProxy to send Layer 7 (HTTP) health checks instead of Layer 4 (TCP) checks to your backend servers. Servers that don’t respond are not served any more requests. Whereas TCP checks succeed if they can make a connection to the backend server’s IP and port, HTTP health checks expect to get back a successful HTTP response.

By default, an HTTP health check makes a request to the root path, /, using the OPTIONS verb. However, arguments specified here can customize that. HAProxy will treat any check that gets a 2xx or 3xx response code to be successful, although this, too, can be customized with an http-check line.

> The default-server setting configures defaults for any server lines that follow, such as enabling health checks, max connections, etc

> The server setting is the heart of the backend. Its first argument is a name, followed by the IP address and port of the backend server. You can specify a domain name instead of an IP address. In that case, it will be resolved at startup or if you add a resolvers argument, it will be updated during runtime. If the DNS entry contains an SRV record, the port and weight will be filled in from it too. If the port isn’t specified, then HAProxy will use the same port that the client connected on, which is useful for randomly used ports such as for active-mode FTP.

Although we added option httpchk to set up HTTP-based health checking of our servers, each server must opt in to health checks by adding a check argument. This can be set on the server line

Every server line should have a maxconn setting that limits the maximum number of concurrent requests that the server will be given.

\*/