
# Relatório - Balanceamento de carga em Servidores Web com HAProxy e Keepalived

Disponibilidade e Desempenho

2021 - 2022

**Bruno Teixeira**
a2019100036@isec.pt

Licenciatura de Engenharia Informática
21 de janeiro de 2021

# Conteúdo

# Lista de Figuras

# Capítulo 1

# Introdução

Este trabalho tem como objetivo estudar o balanceamento de carga e o *failover* em servidores *web* com o HAProxy e o KeepAlived.
O principal foco é, criar alguns cenários com possíveis falhas, analisar esses cenários, descobrir os pontos fracos e tentar sempre minimizar o *downtime*.

Para alcançar este objetivo foram então criadas algumas experiências de modo a perceber como é possível criar uma infraestrutura segura e com um *downtime* reduzido ou até mesmo nulo aplicando ao mesmo tempo o conceito de balanceamento de carga.

# Capítulo 2

# Desenvolvimento

## 2.1 Descrição das tecnologias usadas

## 2.2 Flask, MariaDB e Galera Cluster

O Flask é um *micro-framework* do *Python* destinado principalmente para pequenas aplicações com requisitos mais simples. O mesmo funciona bastante bem com bases de dados tendo sido este o escolhido para o desenvolvimento da aplicação *web* para este trabalho.
Para que a aplicação fosse *stateful* foi então usada uma base de dados em MariaDB, uma vez que a mesma tem uma comunidade enorme na Internet, tornando-a bastante simples de ser utilizada.

O *Galera Cluster* é um *cluster* virtual para MariaDB que permite a replicação entre diferentes bases de dados, mantendo assim uma disponibilidade e desempenho alto uma vez que existe mais do que uma base de dados para responder a diferentes *querys* dos clientes, sendo que todas as mudanças feitas numa base de dados são replicadas para a outra.

## 2.3 HAProxy

O HAProxy é um serviço Linux que garante o balanceamento de carga para HTTP e TCP. Na prática, o mesmo recebe as conexões dos utilizadores e atua como *proxy*, criando um canal de comunicação entre o utilizador e um dos *webservers*.

O HAProxy funciona em dois modos diferentes, HTTP ou TCP.

- Quando opera em TCP dizemos que é um *Proxy* de camada 4 (OSI)
  - Quando o HAProxy opera neste modo, o mesmo apenas tem acesso a qual IP e Porto o cliente está a tentar aceder, não conseguindo assim ver a informação trocada entre de ambas as partes.
- Quando opera em HTTP dizemos que é um *Proxy* de camada 7 (OSI)
  - Quando o HAProxy opera neste modo, o mesmo tem acesso a toda a informação, logo estamos a confiar no mesmo para ter acesso a esses dados, dados que transitam de um lado para o outro.

## 2.4 KeepAlived

O objetivo principal do KeepAlived é fornecer instalações simples para existir balanceamento de carga e alta disponibilidade (a alta disponibilidade é alcançada pelo protocolo VRRP) para sistemas baseados em Linux. O Keepalived agrega então um conjunto de servidores de balanceamento de carga (HAProxy), e consoante a saúde dos mesmos, ele toma decisões sobre pelo qual deverá passar a operacionalização.

## 2.5 Ambiente para as experiências

Para serem feitas algumas experiências foi criado um ambiente com várias máquinas virtuais, estando estas agregadas a um virtualizador ESXi, ou seja, todas as máquinas estão na mesma LAN.

- **webserver01** - 192.168.1.180
- **webserver02** - 192.168.1.181
- **mariadb01** - 192.168.1.182
- **mariadb02** - 192.168.1.186
- **haproxy01** - 192.168.1.183
- **haproxy02** - 192.168.1.184
- **haproxy03** - 192.168.1.185
- **haproxy04** - 192.168.1.187

### 2.5.1 Aplicação Web

Como descrito anteriormente, foi criada uma aplicação em Flask. Esta aplicação funciona como uma espécie de lista de compras em que o utilizador depois de fazer o *login*, consegue adicionar e eliminar produtos do seu cesto.


Figura 2.1: Index - Aplicação Web

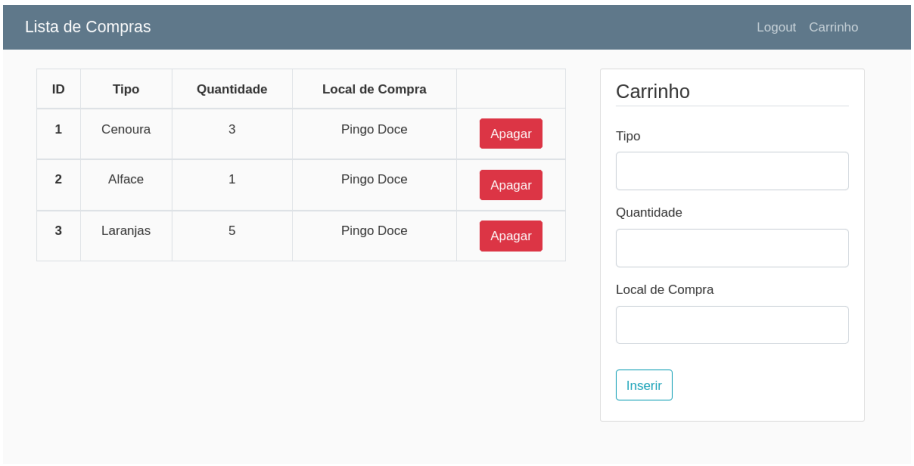



Figura 2.2: Carrinho - Aplicação Web

### 2.5.2 Configuração do HAProxy

No ficheiro de configuração do HAProxy (localizado em ***/etc/haproxy/haproxy.cfg***) existem 5 secções, sendo que estas definem como é que o servidor se comporta, quais são as definições por omissão, e como é que o cliente faz pedidos e recebe respostas.

- **global**
  - Nesta primeira secção estão definidas as medidas em que o processo vai operar, sendo estas medidas de um nível mais baixo, ou seja, relacionadas com o sistema operativo.
- **defaults**
  - Esta secção não é obrigatória, no entanto permite reduzir a duplicação de comandos, uma vez que as configurações feitas aqui são aplicadas na secção ***frontend*** e ***backend***.
- **listen**
  - Aqui podemos combinar o **frontend** e **backend** ao mesmo tempo. Isto é útil, pois é aqui feito o redirecionamento para o ***endpoint*** de estatisticas.
- **frontend**
  - Nesta secção definimos como é que os pedidos dos utilizadores irão ser encaminhados para o **backend**.
- **backend**
  - Aqui definimos os ***webservers*** que vão operar na infraestrutura, definindo tambem o algoritmo de ***load balancing*** a ser utilizado

```
# O comando "stats socket" ativa a API do HAProxy sendo assim possível gerar um "endpoint" com todas as estatísticas do proxy e dos servidores web
global
        stats socket /run/haproxy/admin.sock mode 660 level admin

# Neste caso usamos o modo "http"
defaults
        mode http

# Configuração referente ao "endpoint" de estatisticas
# Por motivos de simplicidade o "auth" está sem  encriptação
listen stats
       bind 192.168.1.183:9999
       stats enable
       stats hide-version
       stats refresh 10s
       stats uri /stats
       stats auth haproxy:haproxy

# "http-in" é apenas um nome para o frontend
# Qualquer pessoa que se conecte a este servidor irá ser redirecionado para os "webservers" e fazemos isso usando "default_backend <nome do backend>"
frontend http-in
         bind 192.168.1.183:80
         default_backend webservers

# Damos o mesmo nome ao backend que demos no "default_backend"
# No "balance" escolhemos o algoritmo que quisermos utilizador na atribuição de "webservers" aos clientes (round-robin ou leastcoon)
# Fazemos um "check" com um intervalo de 500 milisegundos, se falhar 3 vezes retira esse "webserver" do grupo
backend webservers
        balance roundrobin
        server webserver01 192.168.1.180:5000 check inter 500 fall 3
        server webserver02 192.168.1.181:5000 check inter 500 fall 3
```

Figura 2.3: Configuração - HAProxy

Conforme foi configurado o HAProxy, ao acedermos a ***http://192.168.1.183:9999/stats*** conseguimos visualizar uma página *web* com várias estatísticas sobre os *webservers*.

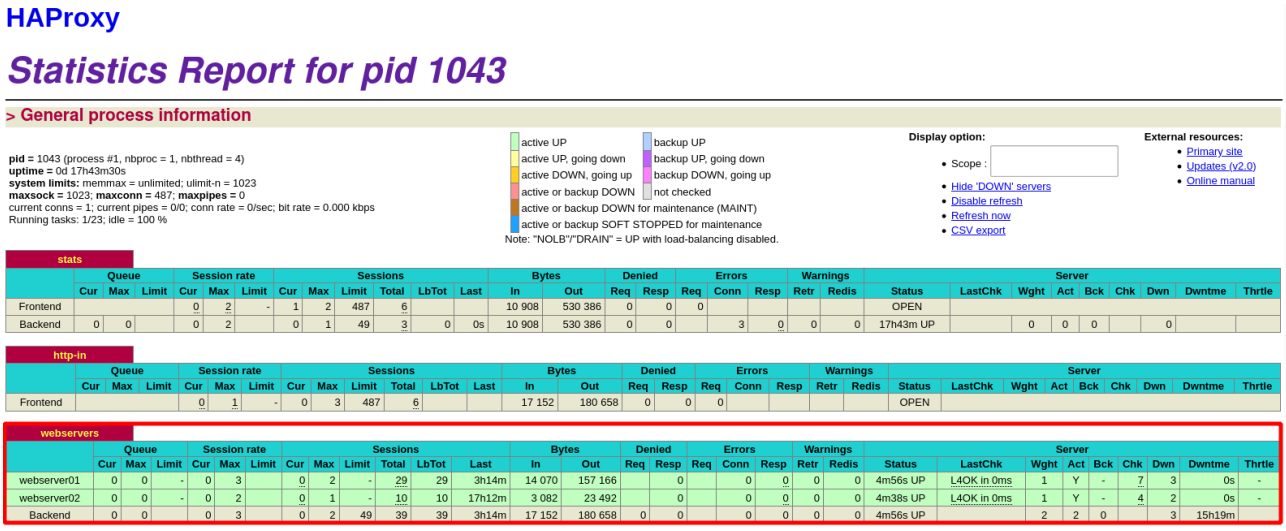
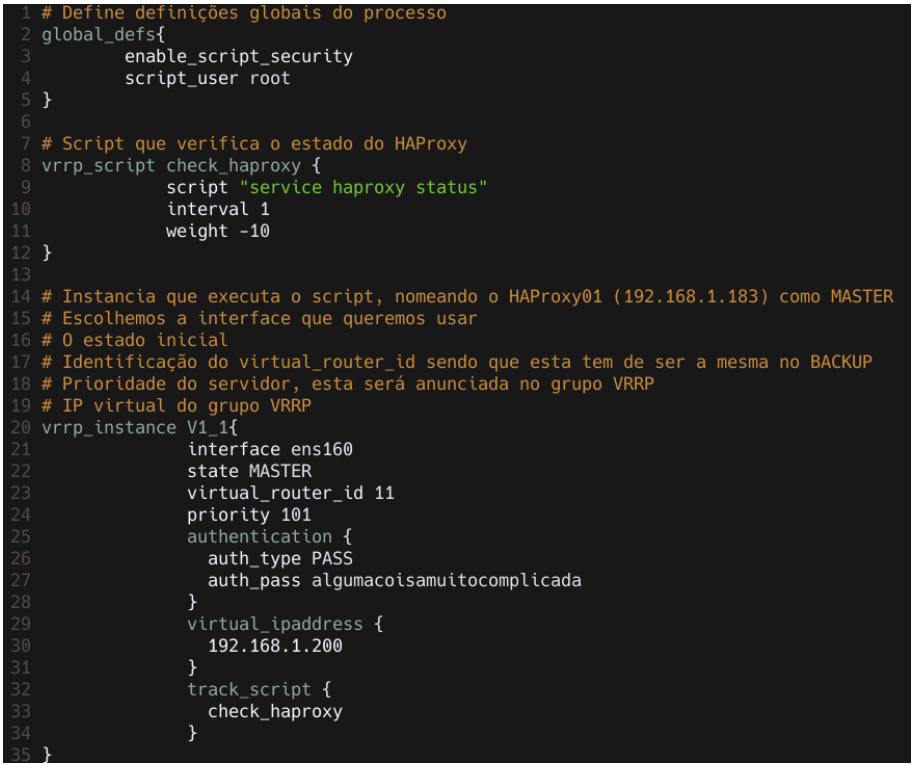



Figura 2.4: Estatísticas - HAProxy

### 2.5.3 Configuração do KeepAlived

No ficheiro de configuração do Keepalived (localizado em */etc/keepalived/keepalived.conf*), foi criado um **vrrp_script** com o intuito de verificar, com um intervalo de 2 em 2 milisegundos, se o haproxy está a funcionar corretamente. Se o mesmo não estiver a funcionar, o peso dele diminui em 10 reduzindo assim a sua prioridade tornando o BACKUP num MASTER.
Depois disto, foi criado um **vrrp_instance** que define uma instância individual do protocolo VRRP com alguns atributos.



```
# Define definições globais do processo
global_defs{
        enable_script_security
        script_user root
}

# Script que verifica o estado do HAProxy
vrrp_script check_haproxy {
            script "service haproxy status"
            interval 1
            weight -10
}

# Instancia que executa o script, nomeando o HAProxy01 (192.168.1.183) como MASTER
# Escolhemos a interface que queremos usar
# O estado inicial
# Identificação do virtual_router_id sendo que esta tem de ser a mesma no BACKUP
# Prioridade do servidor, esta será anunciada no grupo VRRP
# IP virtual do grupo VRRP
vrrp_instance V1_1{
              interface ens160
              state MASTER
              virtual_router_id 11
              priority 101
              authentication {
                auth_type PASS
                auth_pass algumacoisamuitocomplicada
              }
              virtual_ipaddress {
                192.168.1.200
              }
              track_script {
                check_haproxy
              }
}
```

Figura 2.5: Configuração - Keepalived

O mesmo foi feito para o segundo servidor de HAProxy, no entanto foi alterada a prioridade e o estado para definir que este seria o BACKUP.

Foi tambem necessário alterar o Ip, para onde faziamos ***bind*** inicialmente (**Ip do servidor HAProxy**), para o novo IP virtual (**192.168.1.200**) na secção de ***frontend*** do ficheiro de configuração do HAProxy.

Por fim foi preciso colocar ***net.ipv4.ip_nonlocal_bind=1*** no ficheiro */etc/sysctl.conf* uma vez que no segundo servidor de HAProxy o IP virtual ainda não está ativo (só fica ativo quando esse for o MASTER), logo não é possível iniciar o *bind*.

Esta configuração do KeepAlived apenas foi feita a partir da segunda experiência, uma vez que na primeira ainda não é usado o mesmo para mostrar os riscos que isso tem.

## 2.6 Experiências

### 2.6.1 Experiência 01

Nesta experiência foi usado um servidor de balanceamento de carga (HAProxy), dois *webservers* e uma base de dados externa (MariaDB). Foi feita uma captura no HAProxy para perceber o que acontecia quando o utilizador fazia um pedido ao mesmo.

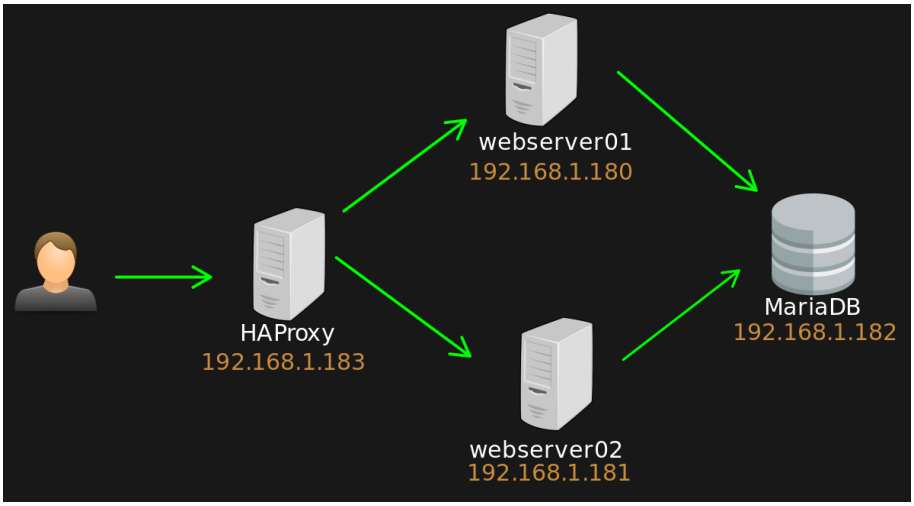



Figura 2.6: Esquema - Experiência 01

**Resultado**

Conseguiu-se perceber que o cliente (**192.168.1.123**) envia um *HTTP GET Request* diretamente ao servidor HAProxy (**192.168.1.183**). De seguida, o servidor HAProxy faz um *HTTP GET Request* ao *webserver* disponível, que neste caso foi o *webserver01* (**192.168.1.180**). O *webserver01* responde com o *HTTP status code 200*, mostrando que a comunicação ocorreu sem falhas, sendo feito depois um redirecionamento do HAProxy para o cliente. Imediatamente a seguir foi feito outro pedido pelo mesmo cliente, no entanto consegue-se perceber que, como está a ser usar o algoritmo **round-robin**, quem respondeu foi o *webserver02*.

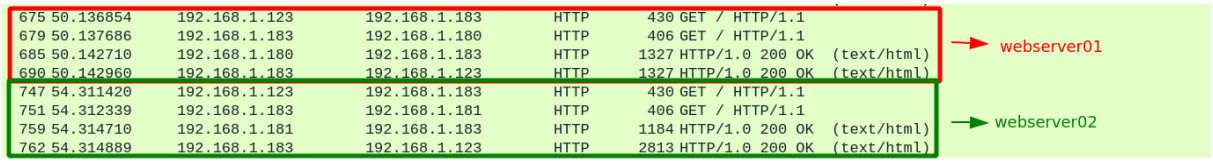



Figura 2.7: Wireshark - Experiência 01

**Problemas encontrados na Experiência 01**

É percetivel que a experiência feita anteriormente tem alguns problemas, como a existência de dois SPOFs (**Single Point of Failure**).

- Existe um SPOF no HAProxy.
- Existe um SPOF na Base de Dados.

Ou seja, caso o servidor de HAProxy ou a base de dados deixe de operar, o cliente deixa de ter comunicação com os *webservers*. Sabendo isto continuou-se com mais algumas experiências de modo a resolver estes problemas.

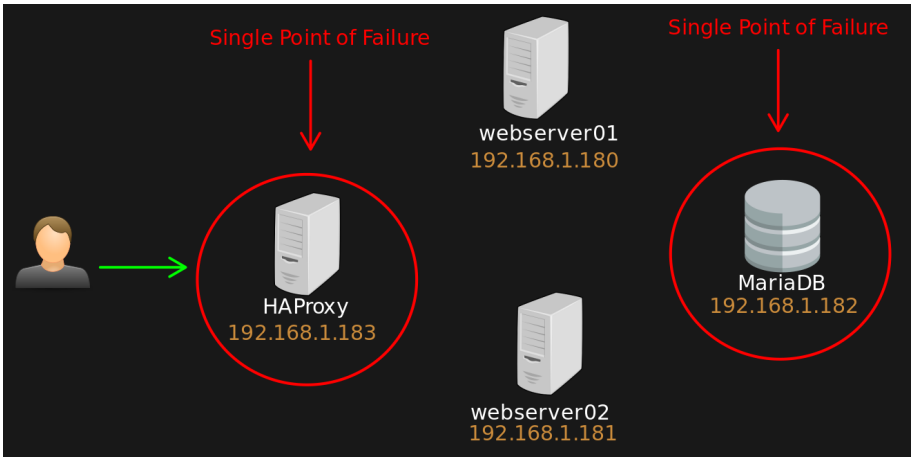



Figura 2.8: Single Point of Failure - Experiência 01

### 2.6.2 Experiência 02

Nesta segunda experiência apenas foi acrescentado mais um servidor de balanceamento de carga e o serviço de **KeepAlived** em ambos os servidores de HAProxy.
O objetivo da mesma era entender como seria feito o *fail-over* do **KeepAlived** e o que sucedia depois de um servidor *MASTER* tornar-se *BACKUP* e vice-versa.

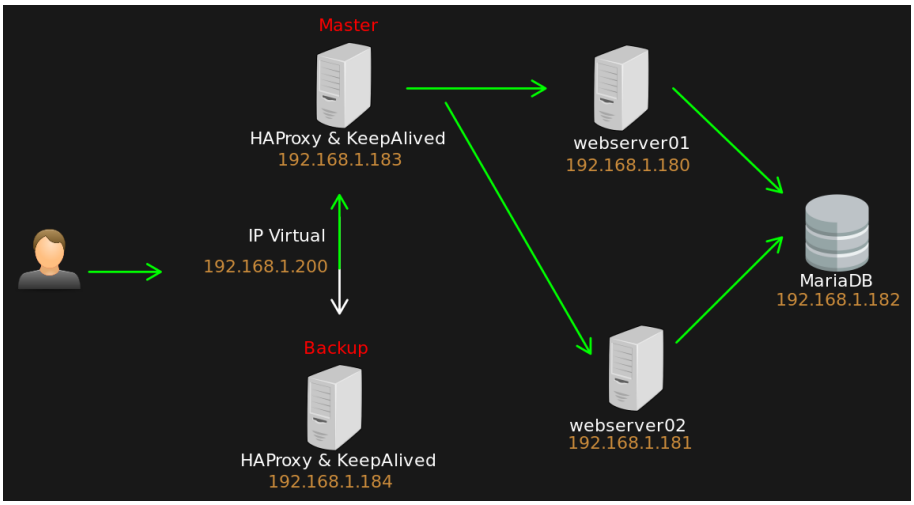



Figura 2.9: Esquema - Experiência 02

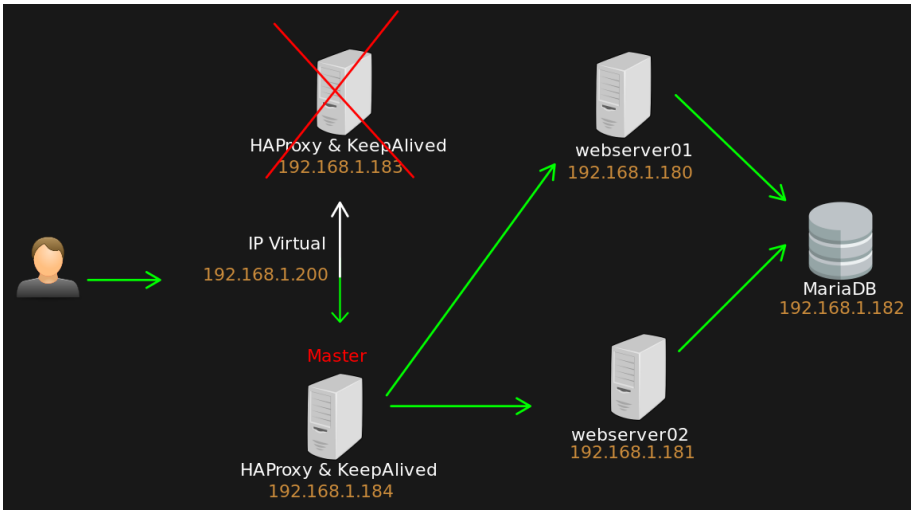



Figura 2.10: Esquema com *fail-over* - Experiência 02

**Resultado**

Foram feitas várias capturas, tanto no **HAProxy01(MASTER)** como no **HAProxy02(BACKUP)** usando o *wireshark*.
Com a captura no **HAProxy01(192.168.1.183)** percebe-se que o mesmo emite, de segundo em segundo, um *announcement* dizendo a sua prioridade, que neste caso é 101. Isto acontece porque no protocolo VRRP apenas o *MASTER* emite mensagens estando os outros *BACKUPs* à escuta desse aviso.

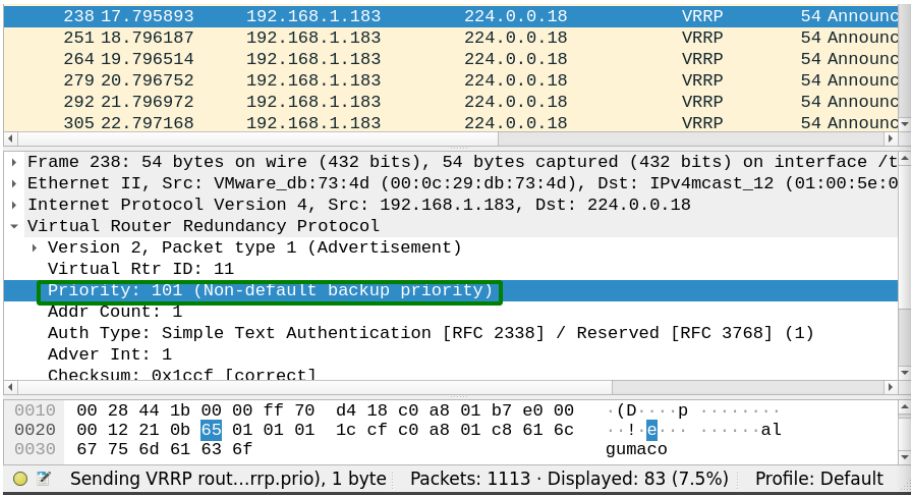



Figura 2.11: Wireshark no 192.168.1.183 - Experiência 02

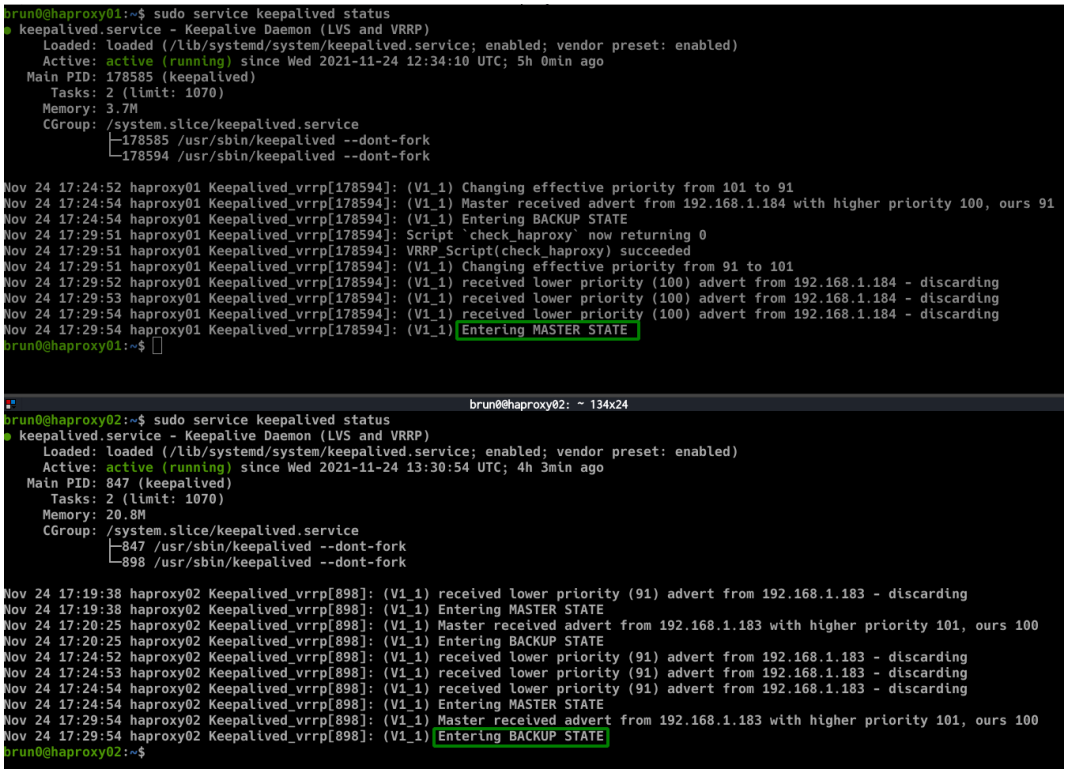



Figura 2.12: Estado inicial do KeepAlived em ambos os servidores - Experiência 02

Depois de desligar o serviço HAProxy do **HAProxy01(192.168.1.183)**, o mesmo fica com uma prioridade de 91 passando assim para o estado de *BACKUP* ao mesmo tempo que o **HAProxy02(192.168.1.184)** passa para o estado de *MASTER* uma vez que a sua prioridade é superior (100).

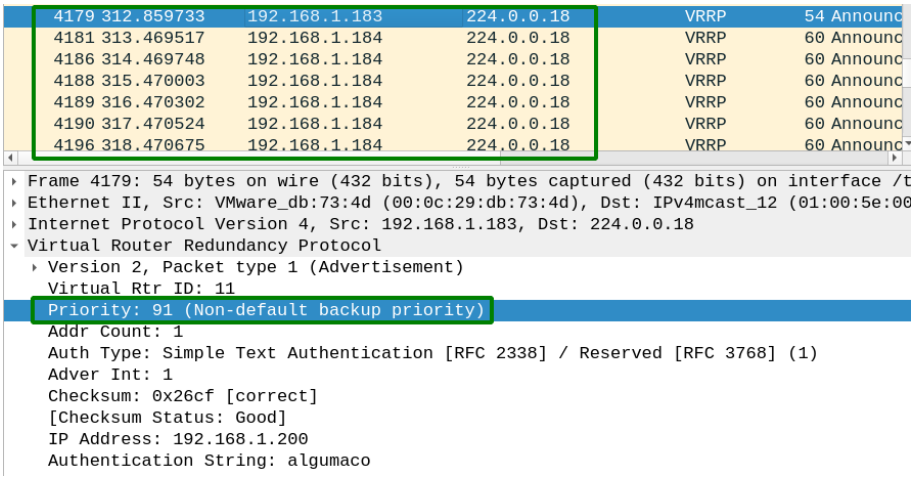



Figura 2.13: Wireshark no 192.168.1.183 com o HAProxy desligado - Experiência 02

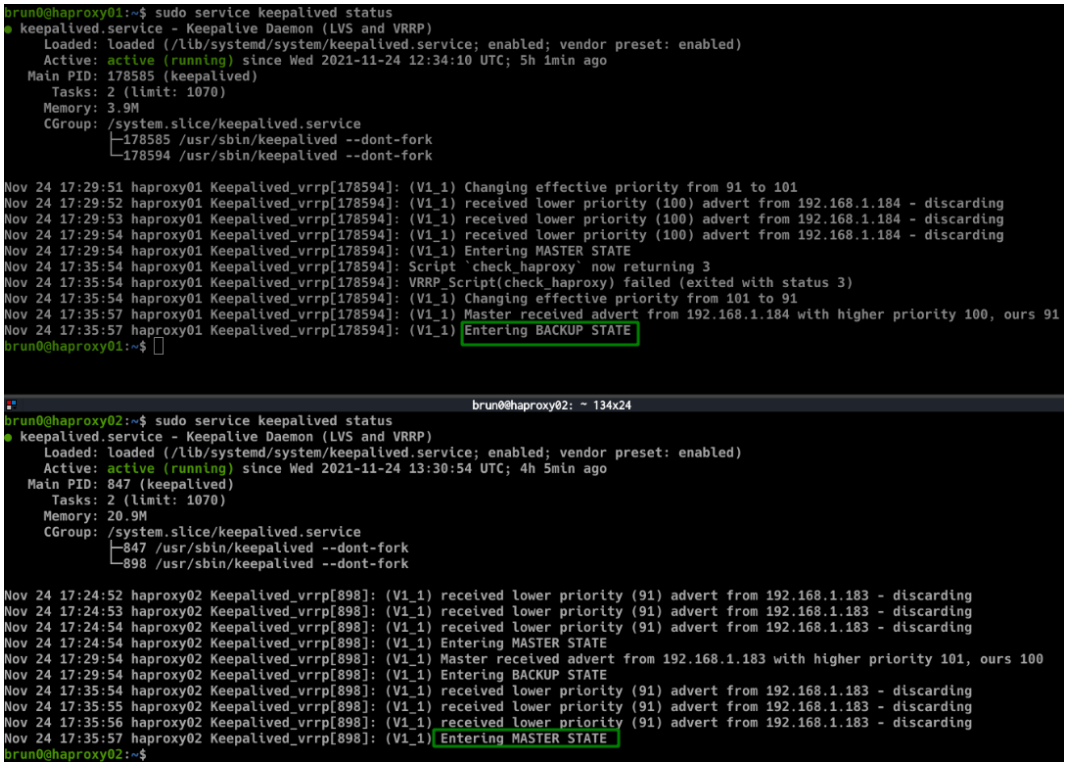



Figura 2.14: Estado atual do KeepAlived em ambos os servidores - Experiência 02

Para terminar, voltou-se a ativar o serviço haproxy no **HAProxy01(192.168.1.183)** tornando-se assim novamente *MASTER* uma vez que a preempção está ativa por omissão fazendo com que a sua prioridade volte a ser 101 como estava definida inicialmente.

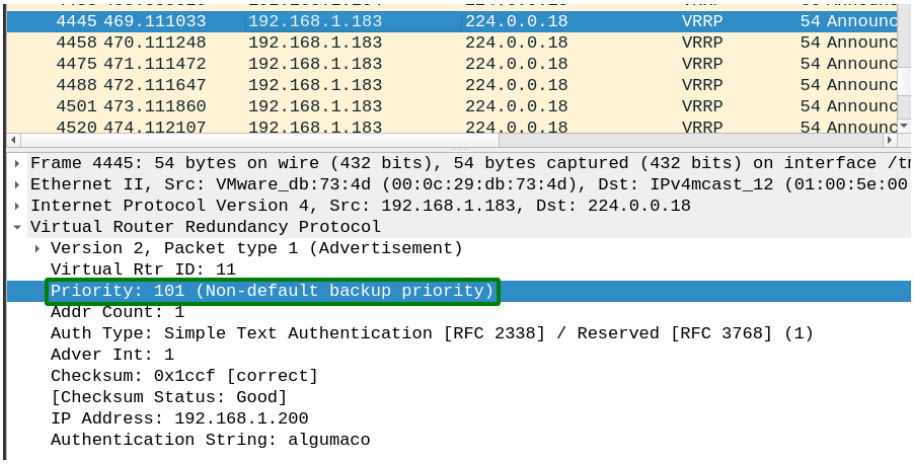



Figura 2.15: Wireshark no 192.168.1.183 com o HAProxy retomado - Experiência 02

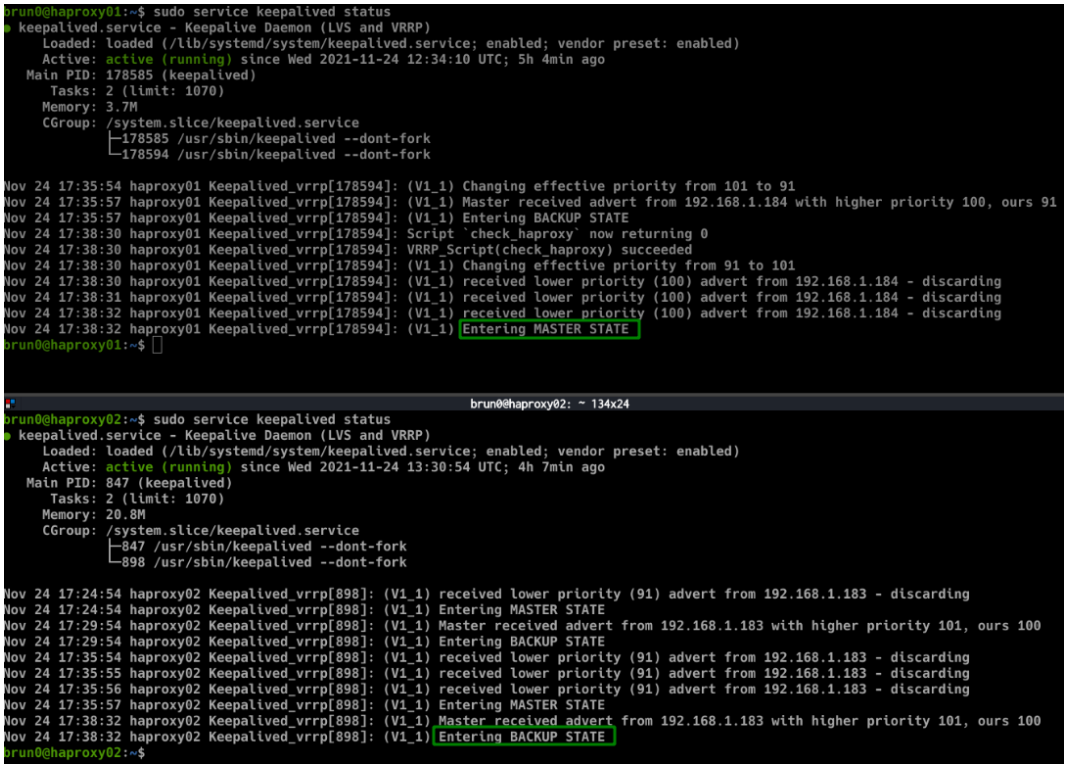



Figura 2.16: Estado final do KeepAlived - Experiência 02

### Problemas encontrados na Experiência 02

Com esta nova arquitetura, foi possível resolver um SPOF**(Single Point of Failure)** colocando mais um servidor de balanceamento de carga e acrescentando o serviço de **KeepAlived**, no entanto é notório que continua a existir um SPOF na base de dados.

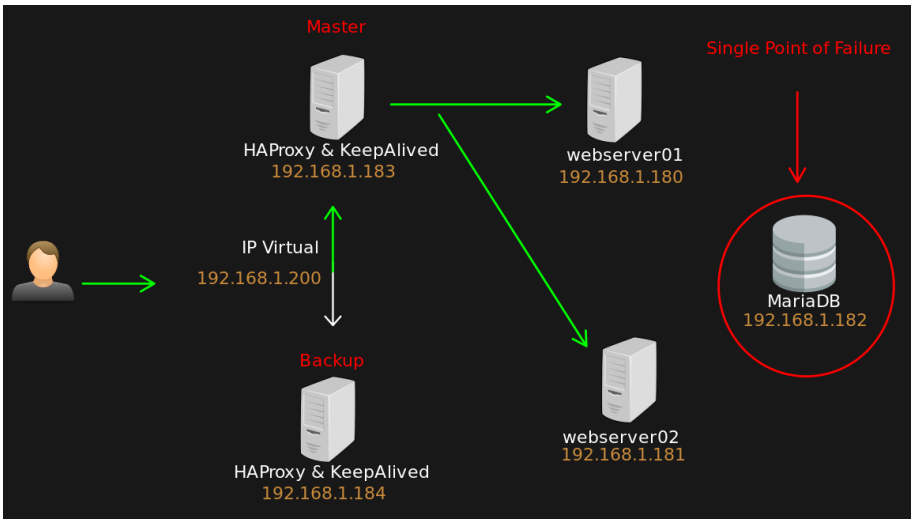



Figura 2.17: Single Point of Failure - Experiência 02

### 2.6.3 Experiência 03

Para resolver o SPOF encontrado na última experiência, foi preciso utilizar mais um servidor HA-Proxy(**192.168.185**) e mais uma base de dados(**192.168.1.186**). Esta base de dados agora vai-se juntar à base de dados (**192.168.1.182**) já existente fazendo assim um ***Galera Cluster***.

No primeiro servidor de base de dados (**192.168.1.182**) foi criado o ficheiro **galera.cnf** localizado em ***/etc/mysql/mariadb.conf.d/galera.cnf*** contendo todas as configurações necessárias para a configuração do *cluster*.

```
[mysqld]
bind-address=0.0.0.0
default_storage_engine=InnoDB
binlog_format=row
innodb_autoinc_lock_mode=2
wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so

 ;; IP da mariadb01 e da mariabd02
wsrep_cluster_address="gcomm://192.168.1.182,192.168.1.186"
wsrep_cluster_name="mariadb-galera-cluster"
wsrep_sst_method=rsync

;; IP do servidor local
wsrep_node_address="192.168.1.182"
wsrep_node_name="galera-db-01"
```

Figura 2.18: Configuração do *Galera Cluster* no mariadb01 - Experiência 03

No segundo servidor de base de dados (**192.168.1.186**) foi tambem criado o ficheiro **galera.cnf** com as configurações necessárias ao mesmo fazer parte do *cluster*.

```
[mysqld]
bind-address=0.0.0.0
default_storage_engine=InnoDB
binlog_format=row
innodb_autoinc_lock_mode=2
wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so

 ;; IP da mariadb01 e da mariabd02
wsrep_cluster_address="gcomm://192.168.1.182,192.168.1.186"
wsrep_cluster_name="mariadb-galera-cluster"
wsrep_sst_method=rsync

;; IP do servidor local
wsrep_node_address="192.168.1.186"
wsrep_node_name="galera-db-01"
```

Figura 2.19: Configuração do *Galera Cluster* no mariadb02 - Experiência 03

Depois de ambos os servidores de base de dados configurados, foi visto que o tamanho do *cluster* era de 2, para isso foi utilizado o comando ***sudo mysql -u root -p -e "show status like 'wsrep_cluster_size'"***.

```
brun0@mariadb01:~$ sudo mysql -u root -p -e "show status like 'wsrep_cluster_size'"
[sudo] password for brun0:
Enter password:
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 2     |
+--------------------+-------+
```

Figura 2.20: Tamanho do *Galera Cluster* - Experiência 03

De seguida o novo servidor de HAProxy (**192.168.1.185**) foi configurado de modo a receber e encaminhar tráfego dos clientes e das base de dados, fazendo assim um balanceamento de carga entre as bases de dados existentes no *Galera Cluster*.
Para isto, foi editado o ficheiro **haproxy.cfg** localizado em ***/etc/haproxy/haproxy.cfg***.

```
1  frontend mariadb_cluster_frontend
2      bind *:3306
3      mode tcp
4      option tcplog
5      default_backend galera_cluster_backend
6
7  ;; modo TCP
8  ;; utilizando roundrobin entre o mariabd01 e mariadb02
9  backend galera_cluster_backend
10     mode tcp
11     option tcpka
12     balance roundrobin
13     server mariadb01 192.168.1.182:3306  check weight 1
14     server mariadb02 192.168.1.186:3306  check weight 1
```

Figura 2.21: Configuração do HAProxy - Experiência 03

Por fim, foi necessário alterar o IP presente na variável de ambiente **DB_HOST**, em ambos os *webservers*, de **192.168.1.182** para o IP do HAProxy (**192.168.1.185**).

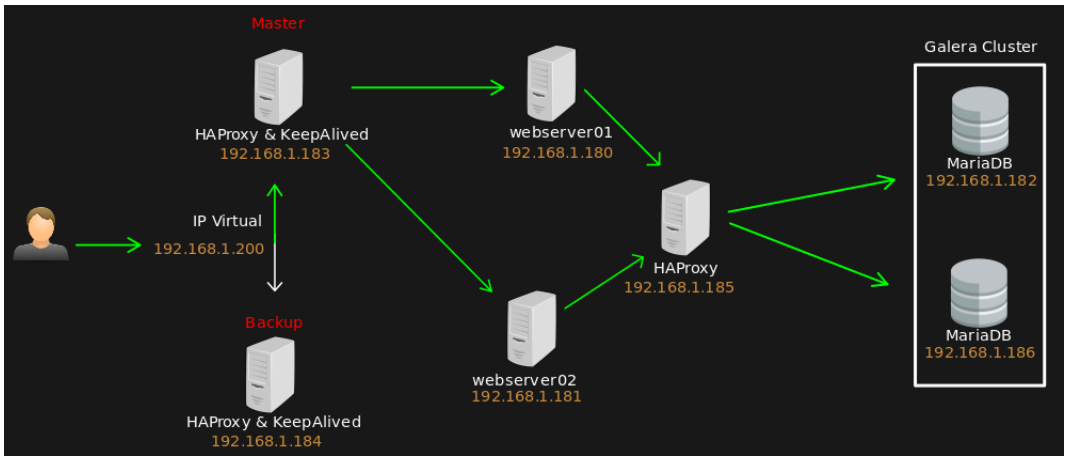



Figura 2.22: Esquema - Experiência 03

### Resultado

Foi realizada uma captura no HAProxy03 (**192.168.1.185**) usando o *wireshark* para perceber como é que o tráfego circulava na rede.

Inicialmente o *webserver01* emite um ***ARP Request*** para saber qual é o **MAC** do novo HAProxy (**192.168.1.185**), sendo que o HAProxy transmite o seu **MAC** e logo a seguir faz uma saudação (*Server Greeting*) à base de dados que neste caso foi à mariadb01 (**192.168.1.182**).
De seguida existe um *Login Request* e é percetivel que agora a comunicação entre *webserver* e base de dados, passa sempre pelo HAProxy.

| | | | | | | |
|---|---|---|---|---|---|---|
| 103 | 24.100226 | VMware_e8:2f:8e | Broadcast | ARP | 60 | Who has 192.168.1.185? Tell 192.168.1.181 |
| 104 | 24.100293 | VMware_68:10:76 | VMware_e8:2f:8e | ARP | 42 | 192.168.1.185 is at 00:0c:29:68:10:76 |
| 112 | 24.149549 | 192.168.1.182 | 192.168.1.185 | MySQL | 185 | Server Greeting proto=10 version=10.4.22-MariaDB-1:10.4.22+maria~focal-log |
| 114 | 24.149848 | 192.168.1.185 | 192.168.1.181 | MySQL | 185 | Server Greeting proto=10 version=10.4.22-MariaDB-1:10.4.22+maria~focal-log |
| 116 | 24.150565 | 192.168.1.181 | 192.168.1.185 | MySQL | 214 | Login Request user=brun0 db=carrinho |
| 118 | 24.150765 | 192.168.1.185 | 192.168.1.182 | MySQL | 214 | Login Request user=brun0 db=carrinho |
| 120 | 24.163095 | 192.168.1.182 | 192.168.1.185 | MySQL | 77 | Response OK |
| 122 | 24.163320 | 192.168.1.185 | 192.168.1.181 | MySQL | 77 | Response OK |
| 124 | 24.163698 | 192.168.1.181 | 192.168.1.185 | MySQL | 89 | Request Query |
| 126 | 24.163896 | 192.168.1.185 | 192.168.1.182 | MySQL | 89 | Request Query |
| 128 | 24.164471 | 192.168.1.182 | 192.168.1.185 | MySQL | 77 | Response OK |
| 130 | 24.164575 | 192.168.1.185 | 192.168.1.181 | MySQL | 77 | Response OK |

Figura 2.23: Primeira conexão - Experiência 03

Passado algum tempo, e já tendo sido feito o *login* na aplicação, foi atualizada a página do **carrinho** e mais uma vez aqui é notório o caminho da comunicação entre *webserver* e base de dados.
Desta vez o HAProxy encaminhou a *query* do cliente para a mariadb02 (**192.168.1.186**) e posteriormente esta respondeu ao HAProxy voltando a ser encaminhada essa resposta para o cliente.

| | | | | | | |
|---|---|---|---|---|---|---|
| 8862 | 2300.471856 | 192.168.1.181 | 192.168.1.185 | MySQL | 272 | Request Query |
| 8863 | 2300.472034 | 192.168.1.185 | 192.168.1.186 | MySQL | 272 | Request Query |
| 8864 | 2300.472863 | 192.168.1.186 | 192.168.1.185 | MySQL | 416 | Response |
| 8866 | 2300.472986 | 192.168.1.185 | 192.168.1.181 | MySQL | 416 | Response |

Figura 2.24: Atualizada a página da aplicação - Experiência 03

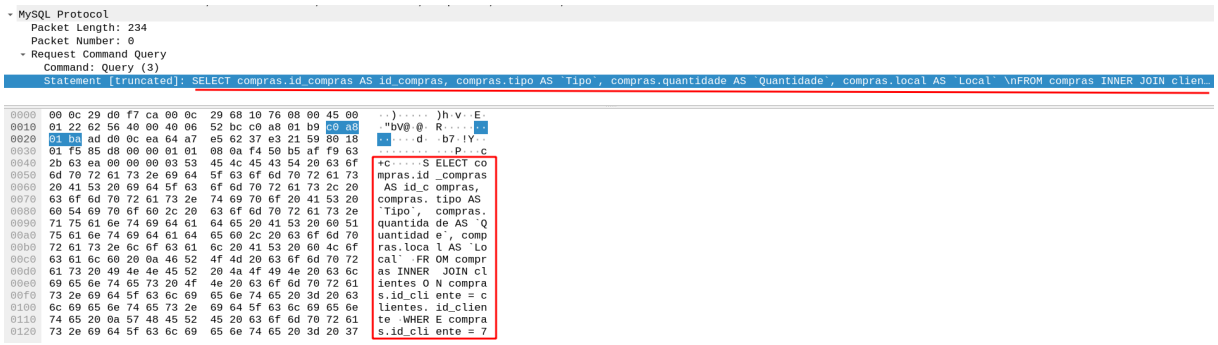



Figura 2.25: *Query* SQL depois de atualizada a página da aplicação - Experiência 03

### Problemas encontrados na Experiência 03

Com a implementação do novo HAProxy e do *Galera Cluster*, foi possível resolver um SPOF que a antiga experiência tinha, no entanto passou a haver outro SPOF, agora no servidor de HAProxy que gere o *Galera Cluster*.

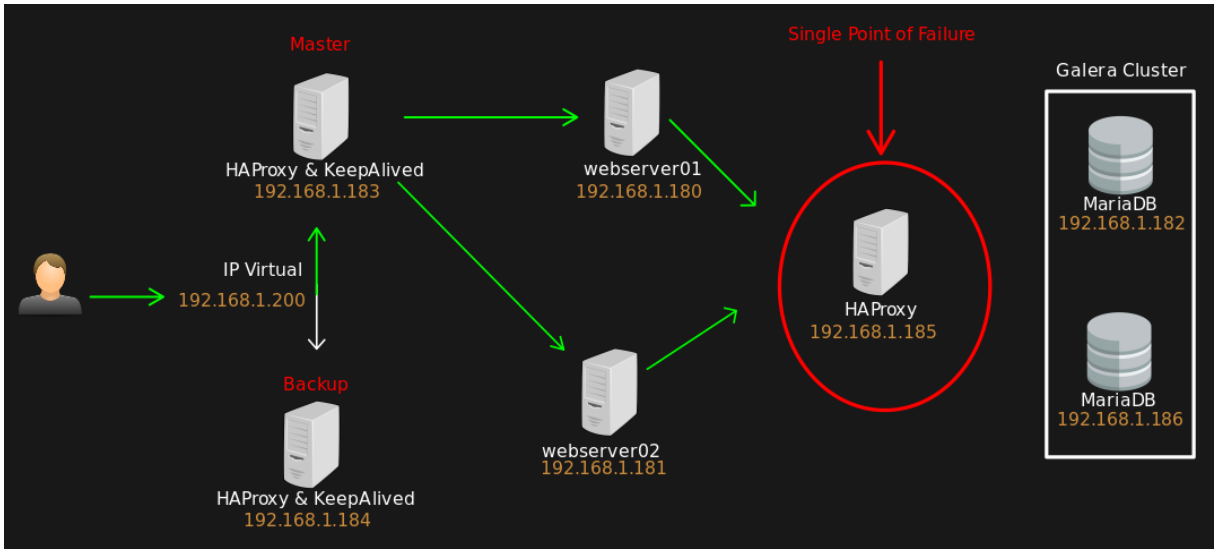



Figura 2.26: Single Point of Failure - Experiência 03

### 2.6.4 Experiência 04

Para resolver o SPOF encontrado na última experiência, foi preciso utilizar mais um servidor de HAProxy (**192.168.1.187**) e acrescentar o KeepAlived em ambos os servidores HAProxy que fazem o balanceamento de carga para o *Galera Cluster*.

```
brun0@haproxy03:~$ sudo cat /etc/haproxy/haproxy.cfg
frontend mariadb_cluster_frontend
    bind *:3306
    mode tcp
    option tcplog
    default_backend galera_cluster_backend

backend galera_cluster_backend
    mode tcp
    option tcpka
    balance roundrobin
    server mysql-01 192.168.1.182:3306  check weight 1
    server mysql-02 192.168.1.186:3306  check weight 1
```

```
brun0@haproxy04:~$ sudo cat /etc/haproxy/haproxy.cfg
frontend mariadb_cluster_frontend
    bind *:3306
    mode tcp
    option tcplog
    default_backend galera_cluster_backend

backend galera_cluster_backend
    mode tcp
    option tcpka
    balance roundrobin
    server mysql-01 192.168.1.182:3306  check weight 1
    server mysql-02 192.168.1.186:3306  check weight 1
```

Figura 2.27: Configuração do HAProxy - Experiência 04

A configuração do KeepAlived foi exatamente igual, no entanto para estes dois servidores de HAProxy foi utilizado o ip virtual **192.168.1.250**, tendo na mesma um ***vrrp_script*** que faz o controlo da "vida"dos servidores de HAProxy.

```
brun0@haproxy03:~$ sudo cat /etc/keepalived/keepalived.conf
global_defs{
        enable_script_security
        script_user root
}

vrrp_script check_haproxy {
            script "service haproxy status"
            interval 1
            weight -10
}

vrrp_instance V1_1{
              interface ens160
              state MASTER
              virtual_router_id 11
              priority 101
              authentication {
                auth_type PASS
                auth_pass algumacoisamuitocomplicada
              }
              virtual_ipaddress {
                192.168.1.250
              }
              track_script {
                check_haproxy
              }
}
```

```
brun0@haproxy04:~$ sudo cat /etc/keepalived/keepalived.conf
global_defs{
        enable_script_security
        script_user root
}

vrrp_script check_haproxy {
            script "service haproxy status"
            interval 2
            weight -10
}

vrrp_instance V1_1{
              interface ens160
              state BACKUP
              virtual_router_id 11
              priority 100
              authentication {
                auth_type PASS
                auth_pass algumacoisamuitocomplicada
              }
              virtual_ipaddress {
                192.168.1.250
              }
              track_script {
                check_haproxy
              }
}
```

Figura 2.28: Configuração do KeepAlived - Experiência 04

Por fim, foi necessário alterar o IP presente na variável de ambiente **DB_HOST**, em ambos os *webservers*, de **192.168.1.185** (IP do HAProxy03) para o IP virtual (**192.168.1.185**).

```
brun0@webserver01:~/repo/flask_app$ printenv | grep "DB_HOST"
DB_HOST=mysql+pymysql://brun0:toor@192.168.1.250:3306/carrinho
```

```
brun0@webserver02:~/repo/flask_app$ printenv | grep "DB_HOST"
DB_HOST=mysql+pymysql://brun0:toor@192.168.1.250:3306/carrinho
```

Figura 2.29: Variável de ambiente DB_HOST - Experiência 04

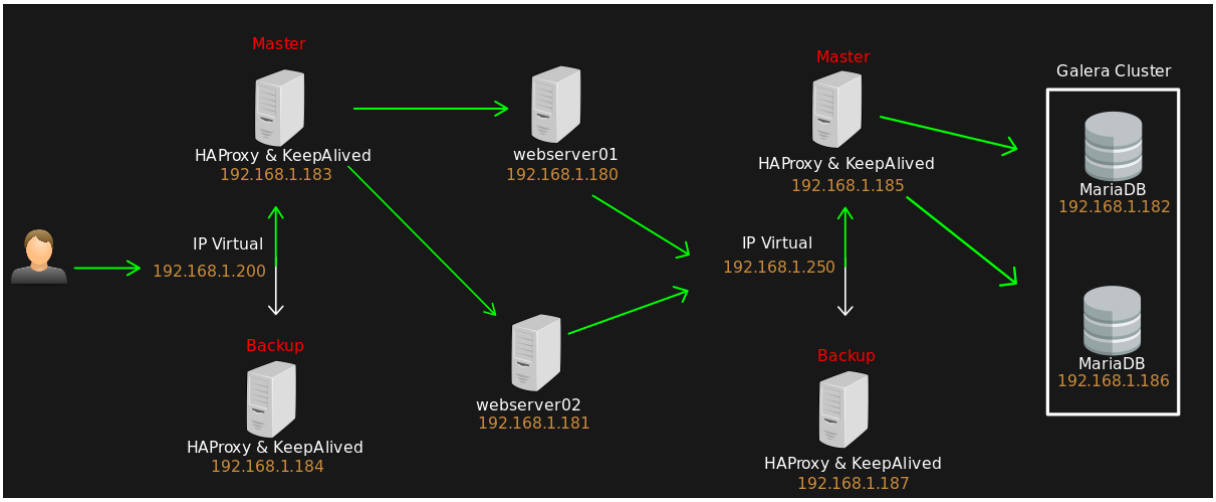



Figura 2.30: Esquema - Experiência 04

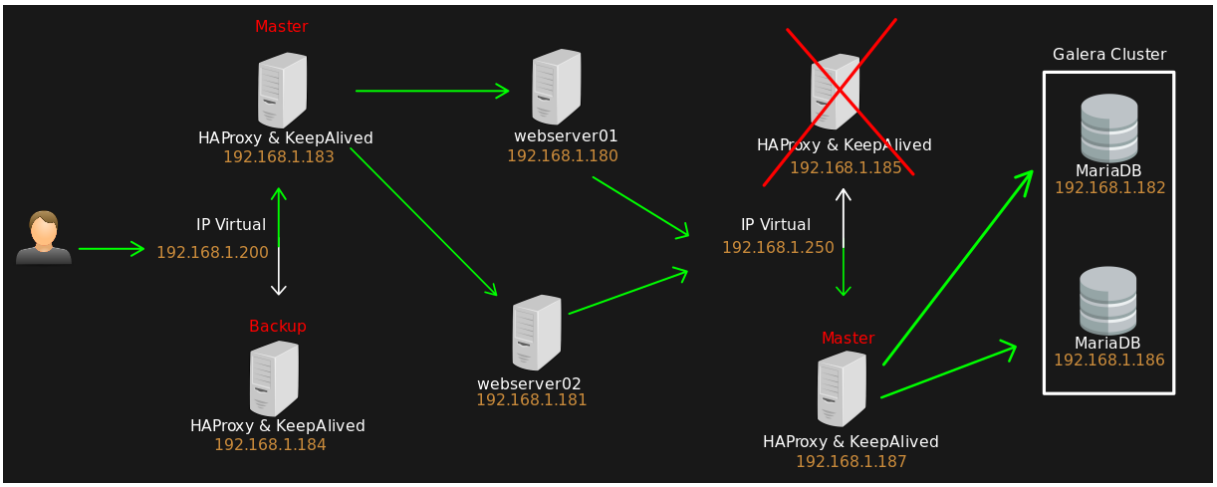



Figura 2.31: Esquema com *failover* - Experiência 04

**Resultado**

Foram feitas capturas, tanto no **HAProxy03**(**MASTER**) como no **HAProxy04**(**BACKUP**) usando o *wireshark*.
Como visto anteriormente, o HAProxy03(**192.168.1.185**) como é o *MASTER*, envia, de segundo em segundo, um *announcement* indicando a sua pioridade enquanto que o HAProxy04(**192.168.1.187**) está à escuta. Na captura do *wireshark* é possível tambem ver o *announcement* vindo do HAProxy01(**192.168.1.183**).

| | | | | | |
|---|---|---|---|---|---|
| 1657 | 399.422756 | 192.168.1.185 | 224.0.0.18 | VRRP | 54 Announcement (v2) |
| 1658 | 399.422938 | 192.168.1.183 | 224.0.0.18 | VRRP | 60 Announcement (v2) |
| 1662 | 400.423248 | 192.168.1.185 | 224.0.0.18 | VRRP | 54 Announcement (v2) |

Figura 2.32: *Announcements* do HAProxy01 e HAProxy03 - Experiência 04

Depois de desligar o serviço HAProxy do HAProxy03(**192.168.1.185**), o mesmo fica com uma prioridade de 91 passando assim para o estado de *BACKUP* ao mesmo tempo que o HAProxy04(**192.168.1.187**) passa para o estado de *MASTER* uma vez que a sua prioridade é superior (100).

```
brun0@haproxy03:~$ sudo service keepalived status
● keepalived.service - Keepalive Daemon (LVS and VRRP)
     Loaded: loaded (/lib/systemd/system/keepalived.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2021-12-11 17:57:36 UTC; 4h 53min ago
   Main PID: 10393 (keepalived)
      Tasks: 2 (limit: 2278)
     Memory: 4.1M
     CGroup: /system.slice/keepalived.service
             ├─10393 /usr/sbin/keepalived --dont-fork
             └─10404 /usr/sbin/keepalived --dont-fork

Dec 11 22:51:14 haproxy03 Keepalived_vrrp[10404]: (V1_1) Entering MASTER STATE
Dec 11 22:51:14 haproxy03 Keepalived_vrrp[10404]: (V1_1) Master received advert from 192.168
Dec 11 22:51:14 haproxy03 Keepalived_vrrp[10404]: (V1_1) Entering BACKUP STATE
Dec 11 22:51:15 haproxy03 Keepalived_vrrp[10404]: (V1_1) ip address associated with VRID 11
Dec 11 22:51:16 haproxy03 Keepalived_vrrp[10404]: (V1_1) ip address associated with VRID 11
Dec 11 22:51:17 haproxy03 Keepalived_vrrp[10404]: (V1_1) ip address associated with VRID 11
Dec 11 22:51:18 haproxy03 Keepalived_vrrp[10404]: (V1_1) ip address associated with VRID 11
Dec 11 22:51:19 haproxy03 Keepalived_vrrp[10404]: (V1_1) ip address associated with VRID 11
Dec 11 22:51:20 haproxy03 Keepalived_vrrp[10404]: (V1_1) ip address associated with VRID 11
Dec 11 22:51:21 haproxy03 Keepalived_vrrp[10404]: (V1_1) ip address associated with VRID 11
lines 1-20/20 (END)
```

```
brun0@haproxy04:~$ sudo service keepalived status
● keepalived.service - Keepalive Daemon (LVS and VRRP)
     Loaded: loaded (/lib/systemd/system/keepalived.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2021-12-11 17:57:19 UTC; 4h 53min ago
   Main PID: 27735 (keepalived)
      Tasks: 2 (limit: 1070)
     Memory: 4.1M
     CGroup: /system.slice/keepalived.service
             ├─27735 /usr/sbin/keepalived --dont-fork
             └─27745 /usr/sbin/keepalived --dont-fork

Dec 11 22:50:56 haproxy04 Keepalived_vrrp[27745]: (V1_1) Entering MASTER STATE
Dec 11 22:50:59 haproxy04 Keepalived_vrrp[27745]: (V1_1) ip address associated with VRID 11 not
Dec 11 22:50:59 haproxy04 Keepalived_vrrp[27745]: (V1_1) ip address associated with VRID 11 not
Dec 11 22:51:00 haproxy04 Keepalived_vrrp[27745]: (V1_1) ip address associated with VRID 11 not
Dec 11 22:51:01 haproxy04 Keepalived_vrrp[27745]: (V1_1) ip address associated with VRID 11 not
Dec 11 22:51:02 haproxy04 Keepalived_vrrp[27745]: (V1_1) ip address associated with VRID 11 not
Dec 11 22:51:05 haproxy04 Keepalived_vrrp[27745]: (V1_1) ip address associated with VRID 11 not
Dec 11 22:51:05 haproxy04 Keepalived_vrrp[27745]: (V1_1) ip address associated with VRID 11 not
Dec 11 22:51:06 haproxy04 Keepalived_vrrp[27745]: (V1_1) ip address associated with VRID 11 not
Dec 11 22:51:07 haproxy04 Keepalived_vrrp[27745]: (V1_1) ip address associated with VRID 11 not
lines 1-20/20 (END)
```

Figura 2.33: Troca de *MASTER* e *BACKUP* - Experiência 04

Neste momento continua a haver uma conexão normal do cliente para os *webservers*, sendo que os mesmos continuam a conseguir fazer *querys* às bases de dados como se nada tivesse acontecido.
Depois de ser reativado o serviço de HAProxy no HAProxy03, o mesmo passa novamente a ser *MASTER* visto que a sua prioridade volta a aumentar.

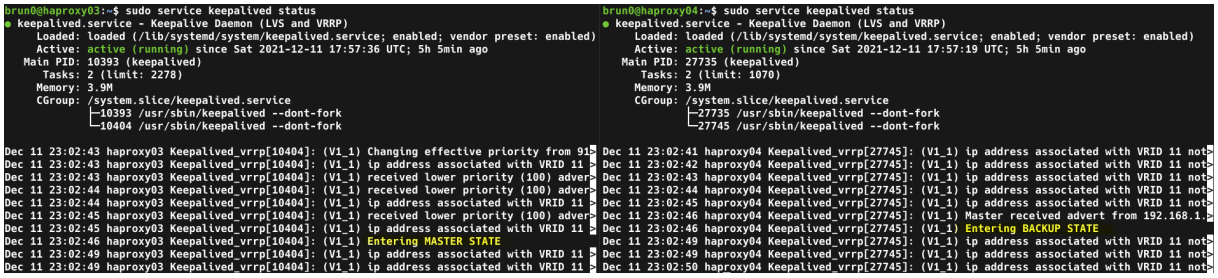



Figura 2.34: Reativado serviço HAProxy em HAProxy03 - Experiência 04

Por fim, podemos concluir que com esta arquitetura temos uma infraestrutura minimamente segura, capaz de ser disponível e de garantir desempenho. Tudo isto se deve aos vários *proxies* implementados assim como o *galera cluster* que oferece uma redundância enorme para um bom desempenho da infraestrutura.
A implementação final, não apresenta nenhum SPOF, cumprindo assim o objetivo proposto no inicio do trabalho.

# Capítulo 3

# Conclusão

Com a execução destas experiências foi possível perceber a importância que tem a disponibilidade e o desempenho numa infraestrutura que fornece serviços *web*.
Inicialmente existia um pequeno problema (**como resolver 2 SPOFs?**), porém com a resolução desses problemas, foram aparecendo outros que à primeira vista aparentavam ser pequenos, no entanto os mesmos prejudicavam a infraestrutura num todo, fazendo com que a mesma ficasse inoperacional.

A partir daqui foi percetível que antes de ser implementada e tornada operacional uma infraestrutura que fornece um serviço qualquer a mesma tem de ser bem planeada de modo a não ser preciso acrescentar muitos mais recursos aos inicialmente pensados.
Para isso é necessário criar um planeamento a pensar em todos os *single point of failures* que possam existir e como os conseguimos resolver, tornando assim a infraestrutura disponível e ao mesmo tempo com um bom desempenho fazendo uso do balanceamento de carga.

# Bibliografia


[1] Stack Overflow
https://stackoverflow.com/

[2] Mariadb e Galera Cluster
https://mariadb.com/kb/en/what-is-mariadb-galera-cluster/

[3] Flask
https://flask.palletsprojects.com/en/2.0.x/

[4] HAProxy
https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts

[5] Keepalived
https://www.redhat.com/sysadmin/keepalived-basics