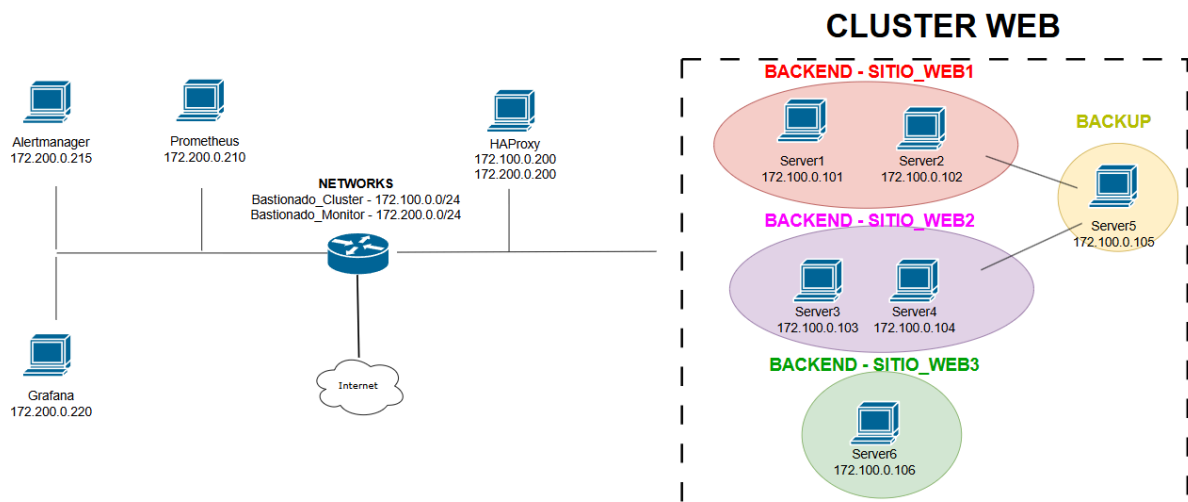


Bastionado de HAProxy

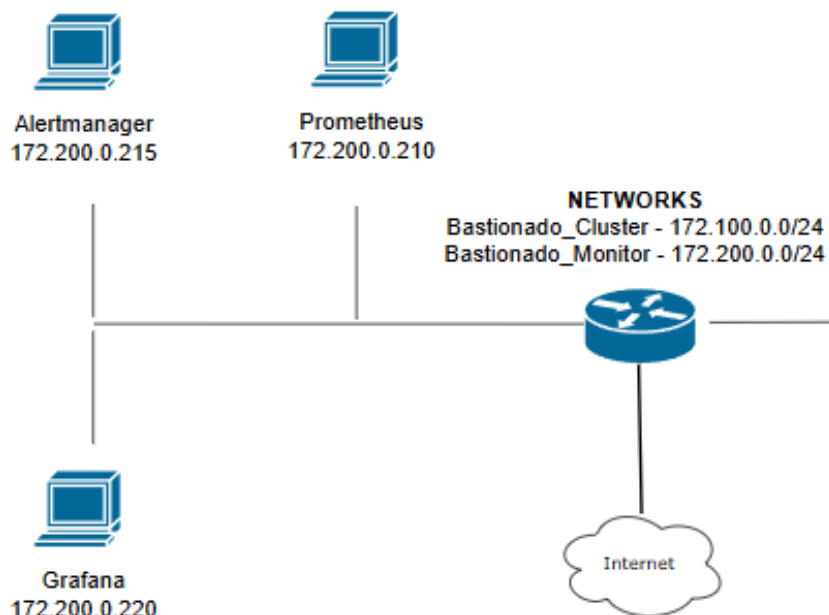
Pablo Martínez de la Iglesia

1. Escenario

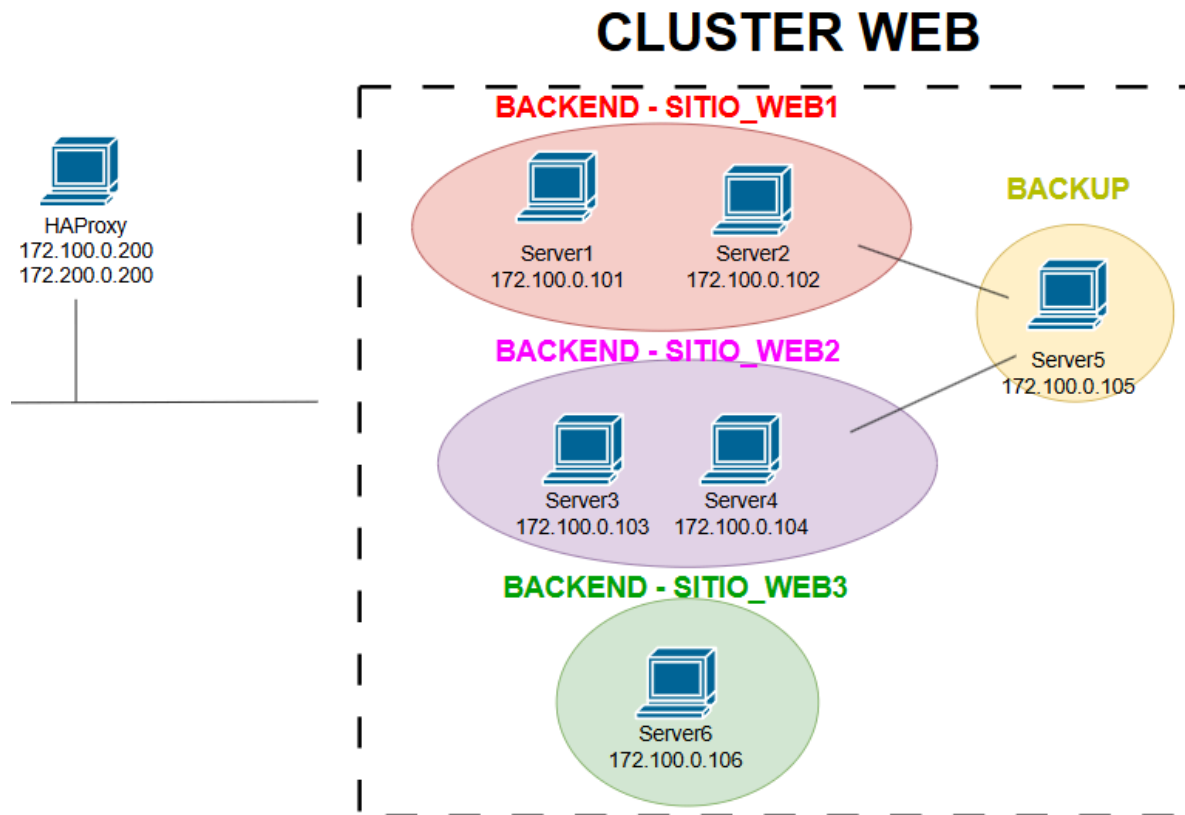
1.1 Esquema



Rede Bastionado_Monitor



Network Bastionado_Cluster



(NOTA: HAProxy pertenece ás dúas redes pero para deixar o esquema simple está no lado de cluster.)

1.2 Obxectivo

En este escenario vamos a centrarnos no HAProxy: que podemos facer con el e algunhas maneiras de facelo mais seguro. É moi importante prestar atención á securización do Haproxy para manter a dispoñibilidade dos servizos detrás del xa que é, no caso deste escenario, un punto crítico da nosa rede e aínda que non fose un SPOF seguen sendo puntos moi importantes dentro dunha rede.

2. Contedores

Este escenario está completamente montado usando Docker. Os contedores docker corren en [Alpine](#), unha distribución de Linux moi lixeira que nos permite facer o escenario o menos pesado posible. Como consecuencia, o sistema operativo é algo incómodo de usar.

Alpine usa ash en vez de bash, unha shell mínima e sinxela. Ao estar centrada na lixeireza, esta distribución tampouco ten comandos comúns que podemos atopar en outros Linux como nano, ip ou systemctl.

Neste escenario úsase a versión 3.21.3 de Alpine.

2.1 Cluster web

O cluster web consta de 6 contedores divididos en tres backends:

Sitio_web1

server1 - 172.100.0.101

server2 - 172.100.0.102

server5 (backup) - 172.100.0.105

Sitio_web2

server3 - 172.100.0.103

server4 - 172.100.0.104

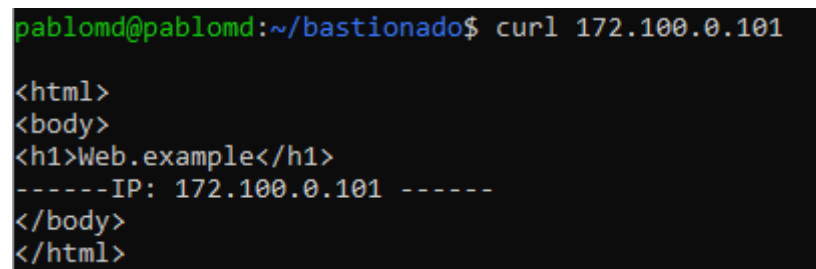
server5 (backup) - 172.100.0.105

Sitio_web3

server6 (https) - 172.100.0.106

Estes contedores corren Apache2 como servizo web. Este apache ten unha pequena modificación para usar arquivos .php antes ca .html por defecto, desta forma podemos pasarlle un arquivo index.php para que nos saque a IP do contedor e comprobar que o balanceo ou a división dos backends nos funciona correctamente.

Exemplo:



```
pablomd@pablomd:~/bastionado$ curl 172.100.0.101
<html>
<body>
<h1>Web.example</h1>
-----IP: 172.100.0.101 -----
</body>
</html>
```

Tamén vamos a copiar dentro do contedor un arquivo chamado **health.html** o cal o HAProxy usará para monitorar se o contedor está activo ou baixado.

2.2 HAProxy

[HAProxy](#) é un software gratuito e de código aberto usado principalmente como balanceador de carga ou proxy inverso.

O contedor HAProxy é o noso punto de interese neste escenario. Corre, por suposto, HAProxy; pero tamén corre rsyslog e fail2ban. Estas dúas tecnoloxías deberíanse lanzar en un servidor separado, servindo como servidor central de logs. Neste escenario lanzámolas xunto ao HAProxy xa que é o noso único punto de interese e simplemente quero ensinar o seu funcionamento.

Este contedor tamén ten instalado supervisor, un sistema de cliente/servidor que nos permite monitorar e xestionar procesos.

Na dirección 172.100.0.200:9999 temos a páxina de estadísticas:

State	Description	Name	Action
running	pid 7, uptime 0:02:32	fail2ban	Restart Stop Clear Log Tail -f Stdout Tail -f Stderr
running	pid 8, uptime 0:02:32	haproxy	Restart Stop Clear Log Tail -f Stdout Tail -f Stderr
running	pid 9, uptime 0:02:32	rsyslog	Restart Stop Clear Log Tail -f Stdout Tail -f Stderr

Con supervisor levantamos os outros tres servizos.

A principal funcionalidade do Rsyslog é poder gardar logs do HAProxy xa que este non loggea por defecto. Estes logs serán leídos polo Fail2ban e buscará líneas que concorden co seu filtro feito con expresións regulares.

Fail2ban está configurado para banear durante 60s e ler ata 120s atrás no log. Cando 3 entradas da mesma IP concorden co filtro do f2b este creará unha regra iptable baneando a IP. O filtro é o seguinte:

```
#failregex = ^%(__prefix_line)s<HOST>.*<STATS>.*\d* 400
```

E fará match coas lineas que saen cando alguén se intenta loggear no /stats do HAProxy (veremolo mais adiante) con credenciais incorrectas.

Por último, toca falar do HAProxy.

O HAProxy está configurado para ter 4 frontends:

- O frontend principal
- O frontend https
- O frontend stats
- O frontend prometheus

Vamos a comezar explicando o frontend principal:

```
#-----  
# main frontend which proxys to the backends  
#-----  
frontend cluster_frontend  
    bind 172.100.0.200:80  
  
    #ACLs  
    acl host_web1 hdr(host) -i www.web1.example  
    acl host_web2 hdr(host) -i www.web2.example  
    acl health path_beg -i /check  
    acl too_fast sc0_http_req_rate gt 10  
  
    #Redirect  
    http-request set-path /health.html if health  
  
    #Deny  
    http-request deny if { path /admin }  
  
    #Logging  
    http-request set-log-level silent unless { path /admin }  
  
    #Rate limit  
    stick-table type ip size 100k expire 1m store http_req_rate(1s)  
    http-request track-sc0 src  
    http-request deny deny_status 429 if too_fast  
  
    #Backends  
    use_backend sitio_web1 if host_web1  
    use_backend sitio_web2 if host_web2  
    default_backend sitio_web1
```

Primeiro temos en cal ip:porto o frontend vai escoitar.

Despois temos as ACLs (Listas de Control de Acceso). Estas ACLs serven para controlar as peticións do cliente. Nós temos 4: A primeira e a segunda con practicamente idénticas e buscan na cabeceira da petición se existe a cadea “www.web1/web2.example” ignorando maiúsculas/minúsculas.

A terceira ACL comproba se o path da URL comeza por /check e a cuarta simplemente trackea as peticións http dun orixe, neste caso serán IPs orixe, e cando pasen de 10 esta saltará.

Pasamos ao seguinte bloque, o de redireccionamento. Esta liña simplemente indica que se cambie o path da petición http se a ACL health salta.

O seguinte bloque é o de denegación, se a petición contén /admin será bloqueada.

Despois temos as regras de logging de este frontend. Colocámolo a silent, é dicir, que non garde ningún log; a non ser que a petición teña /admin.

No seguinte bloque temos un limitador de peticións. A primeira liña garda o número de peticións http por segundo nunha táboa, xunto coa ip do cliente. Esta táboa expira ao minuto.

Despois decímoslle ao HAProxy que faga tracking por IPs orixe e por último denegamos as peticións se a ACL too_fast salta, é dicir; se se fan mais de 10 requests en un segundo.

Por último temos os backends. Aquí chamamos ás dúas ACLs que nos quedan e dirixímolos aos seus respectivos backends. Temos tamén unha liña para indicar que backend usar por defecto

O seguinte backend é mais sinxelo:

```
#-----  
# frontend https  
#-----  
  
frontend https  
    bind 172.100.0.200:443 ssl crt /etc/haproxy/certs/haproxy.pem  
    http-request redirect scheme https unless { ssl_fc }  
    default_backend sitio_web3
```

Este backend usámolo para probar as funcionalidades https do HAProxy. Coa primeira liña indicámos que ten que escoitar no porto 443 e decímoslle que use un certificado SSL indicando a ruta ao mesmo. A segunda liña redirixe peticións http a https e a terceira indica que backend usar por defecto.

Os últimos dous backends son relacionados con estadísticas e monitoreo

```
#-----  
# stat frontends  
#-----  
  
frontend stats  
    bind 172.100.0.200:8181  
    stats enable  
    stats uri /stats  
    stats refresh 10s  
    stats auth administrator:abc123.  
  
frontend prometheus  
    bind 172.200.0.200:9090  
    mode http  
    http-request use-service prometheus-exporter  
    no log
```

Por un lado temos o propio frontend de monitoreo que ten o HAProxy, o cal mencionamos cando falamos de fail2ban, e despois temos un frontend que lle envía estadísticas ao servidor prometheus.

Por último, temos os backends

```
#-----  
# round robin balancing between the various backends  
#-----  
backend sitio_web1  
    balance roundrobin  
    option httpchk GET /health.html  
    http-check expect string ON  
    server server1 172.100.0.101:80 check  
    server server2 172.100.0.102:80 check  
    server server5 172.100.0.105:80 backup check  
  
backend sitio_web2  
    balance roundrobin  
    option httpchk  
    http-check expect status 200  
    server server3 172.100.0.103:80 check weight 3  
    server server4 172.100.0.104:80 check weight 1  
    server server5 172.100.0.105:80 backup check  
  
backend sitio_web3  
    balance roundrobin  
    server server6 172.100.0.106:80 check
```

O primeiro backend usa roundrobin normal, as peticións serán repartidas equitativamente entre todos os servidores do backend (excepto o servidor de backup), e ten un pequeno sistema de monitoreo propio do HAProxy. O balanceador fai unha petición intentando obter o arquivo /health.html dos servidores. Se non o atopa o balanceador asumirá que o servidor está caído/non operativo, se o atopa pasará á seguinte comprobación: O balanceador espera atopar a cadea ON dentro do arquivo.

Esto permítenos ter un servidor encendido pero non operativo, solo teremos que remover ou modificar o contido do arquivo.

O segundo backend usa weighed roundrobin, é dicir; repartirá as peticións en base ao peso repartido. Neste caso enviará tres peticións ao servidor3 e só unha ao servidor4. Este backend usa outra forma de monitoreo, simplemente envía unha petición e espera obter estado 200 (petición exitosa).

Estes dous backends usan o servidor5 como backup. Este servidor será usado se tódos os demais servidores do backend están caídos.

O último backend é o backend https.

2.3 Prometheus

[Prometheus](#) é un software de alertas e monitoreo de sistemas de código aberto. Neste escenario está activo no contedor “Prometheus” e estará apuntando a dúas ips diferentes, a do frontend HAProxy “Prometheus” onde recollerá datos dos frontends, backends, peticións, códigos de error, etc. Tamén vai estar apuntando ao contedor “Alertmanager” onde se recollerán todas as alertas que sexan activadas.

Hai dous arquivos xa configurados para probar tanto as logging rules como as alert rules. O arquivo de configuración é bastante sinxelo de entender. Ten un apartado para o Alertmanager no cal só lle teremos que colocar a ip obexctivo, ten os arquivos de regras que amosarei a continuación e ten outro target apuntando ao fronted “Prometheus” do HAProxy.

```
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets: ["172.200.0.215:9093"]

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  - "rules.yml"
  - "alerts.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "haproxy"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ["172.200.0.200:9090"]
    scrape_interval: 15s
```

En Rules.yml temos só unha regra de rexistro. Estas regras úsanse ou para facer queries que se soen repetir moito ou para despois ser usadas nas alertas. Esta regra simplemente recolle o ratio de peticións que chegan aos frontends cada 30s

```
groups:
  - name: haproxy.rules
    rules:
      - record: haproxy:http_requests:rate30s
        expr: rate(haproxy_frontend_http_requests_total[30s])
```

En Alerts.yml temos dúas alertas. A primeira usa a anterior regra de rexistro para comprobar o número de peticións cada 30s. Se estas exceden 6 ao longo de 30 segundos esta alerta saltará.

A segunda alerta comproba se o servidor de backend está en uso. Prometheus líase un pouco cos servidores de backend e ás veces a recolección da información do HAProxy non funciona correctamente polo que en vez de comprobar se o servidor de backend está activo, comprobamos se os servidores do sitio web1 están caídos.

Ao igual que a regra anterior, se están caídos por 30s esta regra será activada


```

groups:
- name: haproxy.alerts
  rules:
    # Moitas peticións
    - alert: HAProxyHighRequestRate
      expr: haproxy:http_requests:rate30s > 6
      for: 30s
      labels:
        severity: warning
      annotations:
        description: "HAProxy está recibindo {{ $value }} peticións cada 30s"

    # Servidor de Backup en uso
    - alert: HAProxyBackupServerInUse
      expr: haproxy_backend_active_servers{instance="172.200.0.200:9090", job="haproxy", proxy="sitio_web1"} == 0
      for: 30s
      labels:
        severity: warning
      annotations:
        description: "Servidor backend 'server5' para web1 en uso"

```

2.4 Alertmanager

O [Alertmanager](#) é un compoñente, xeralmente de Prometheus, usado para xestionar as alertas que, no noso caso, o Prometheus envía.

Neste escenario simplemente vemos que o prometheus envía as alertas e podemos xogar un pouco con elas. Simplemente está ahí para ver algo mais do Prometheus.

Falando do contedor, necesitamos pasarlle o parámetro "--cluster.listen-address = 127.0.0.1:100" porque neste escenario non nos importa moito o clustering de Alertmanager xa que só o usamos para comprobar as conexións co prometheus

2.5 Grafana

[Grafana](#) é un software que permite a visualización e formato de datos métricos. Neste escenario, ao igual que o Alertmanager, simplemente vemos rapidamente que soporta conexión con Prometheus e pode tamén usar as súas regras de rexistro.

A este contedor dámoslle un arquivo apuntando ao prometheus como fonte de datos para que esté rexistrado de forma automática.

3. Probas

Como levantar o escenario

O escenario usa un Makefile para poder levantar, baixar ou borrar o escenario:

make build -> constrúe o escenario

make up -> levanta e constrúe o escenario

make down -> baixa o escenario e borra os contedores

male remove -> baixa o escenario, borra os contedores e as imaxes destes.

Tests.sh

Hai un scrip bash que xa ten algunhas probas automatizadas. Podémolo executar con ./tests.sh e escoller a proba que queiramos. A lista é a seguinte:

---- Lista ----

1. Probar balanceado web1.example
2. Probar balanceado web2.example
3. Probar backup
4. Probar https
5. Peticións ao balanceador
6. Too many requests

URLs de interese

Páxina de stadísticas do propio haproxy

- 172.100.0.200:8181/stats - administrator/abc123.

Estadísticas enviadas ao prometheus

- 172.200.0.200:9090

Páxina do supervisord

- 172.100.0.200:9999

Páxina do prometheus

- 172.200.0.210:9090/graph

Páxina do alertmanager

- 172.200.0.215:9093

Páxina do grafana

- 172.200.0.220:3000

Fail2ban e rsyslog

Para comprobar que o fail2ban funciona conectámonos á máquina haproxy con:

```
$ docker exec -it haproxy ash
```

e usaremos:

```
/ # tail -fn 10 /var/log/haproxy.log
```

Desta forma tamén podemos comprobar que o rsyslog e as opcións de logging feitas no HAProxy funcionan correctamente.

Se nos diriximos á páxina /admin ou facemos un curl deberíanos aparecer unha liña como esta nos logs

```
2025-04-13T18:21:09+00:00 localhost haproxy[8]: 172.100.0.1:58758
[13/Apr/2025:18:21:09.521] cluster_frontend cluster_frontend/<NOSRV> 0/-1/-1/-1/0 403
192 -- PR-- 1/1/0/0/0 0/0 "GET /admin HTTP/1.1"
```

Con isto comprobamos que o rsyslog e haproxy loggean correctamente. Agora vamos a facer un curl a /stats catro veces. Á cuarta debería darnos un error de conexión

```
pablomd@pablomd:~/bastionado$ curl 172.100.0.200:8181/stats
<html><body><h1>401 Unauthorized</h1>
You need a valid user and password to access this content.
</body></html>
pablomd@pablomd:~/bastionado$ curl 172.100.0.200:8181/stats
<html><body><h1>401 Unauthorized</h1>
You need a valid user and password to access this content.
</body></html>
pablomd@pablomd:~/bastionado$ curl 172.100.0.200:8181/stats
<html><body><h1>401 Unauthorized</h1>
You need a valid user and password to access this content.
</body></html>
pablomd@pablomd:~/bastionado$ curl 172.100.0.200:8181/stats
curl: (7) Failed to connect to 172.100.0.200 port 8181 after 0 ms: Couldn't connect to server
```

Se listamos as regras iptables vemos que o fail2ban nos creou regras de baneo.

```
/ # iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
f2b-haproxy-http-auth  tcp  --  anywhere              anywhere             multiport dports 0:65535

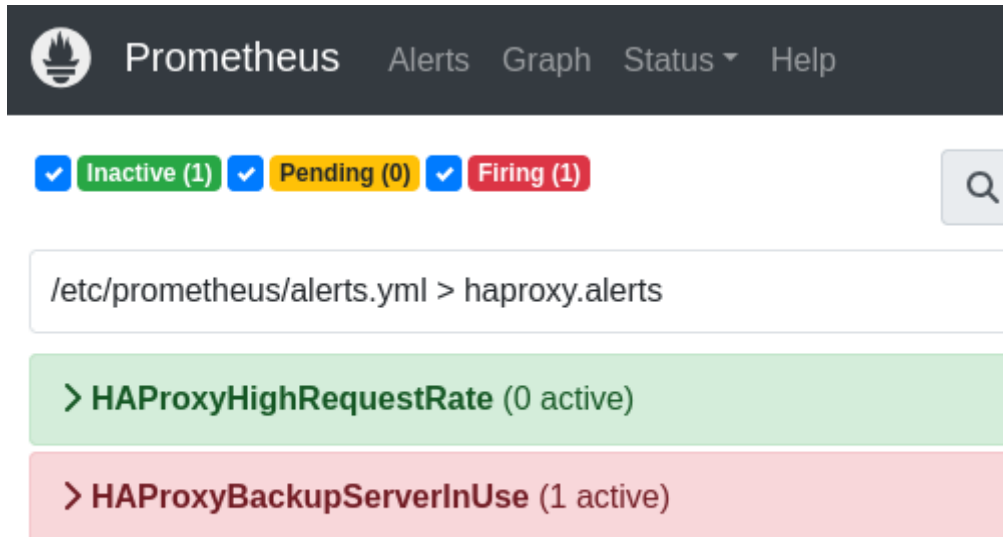
Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination

Chain f2b-haproxy-http-auth (1 references)
target     prot opt source                destination
REJECT     all  --  syn-172-100-000-001.res.spectrum.com anywhere             reject-with icmp-port-unreachable
RETURN     all  --  anywhere              anywhere
```

Alertas de prometheus

Se nos diriximos ao apartado de alertas do Prometheus podemos ver as dúas alertas. Neste caso vamos a facer saltar a alerta de “Servidor backend en uso”. Despois de 30s a alerta pasará de “pending” a “firing”



Podemos conectarnos ao Alertmanager para ver que o envío de alertas foi feito exitosamente.

