Công ty An ninh mạng Viettel

Báo cáo về LoadBalancing

Sinh viên: Nguyễn Đan Trường

*MỤC LỤC*

[Chương 1. Webserver 3](#_Toc156489129)

[1. Nginx 3](#_Toc156489130)

[2. Haproxy 13](#_Toc156489131)

[Chương 2. Tìm hiểu về Load Balancer 25](#_Toc156489132)

[1. Khái niệm 25](#_Toc156489133)

[2. So sánh load balancer layer 4 và layer 7 26](#_Toc156489134)

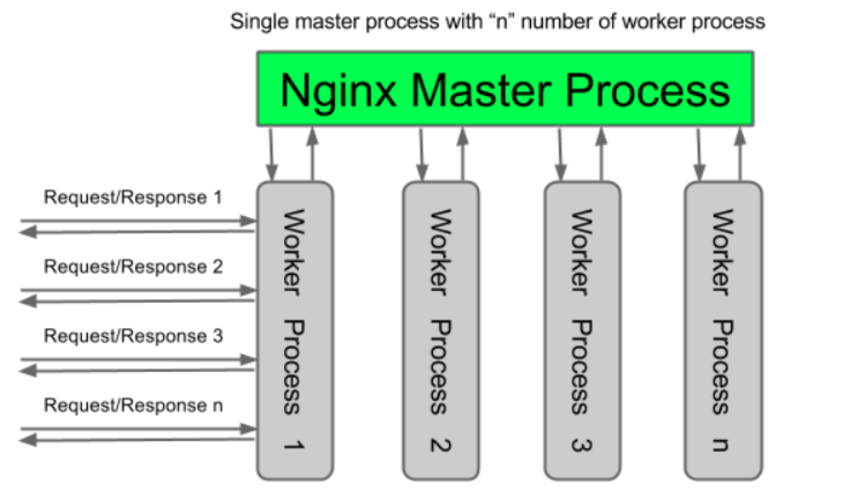
# Webserver

## Nginx

Nginx là một máy chủ proxy ngược mã nguồn mở (open source reverse proxy server) sử dụng phổ biến giao thức HTTP, HTTPS, SMTP, POP3 và IMAP.

Các máy chủ web truyền thống tạo một luồng duy nhất cho mọi yêu cầu, nhưng Nginx không hoạt động theo cách đó. Nginx thực hiện với kiến trúc hướng sự kiện không đồng bộ. Điều đó có nghĩa là các luồng tương tự được quản lý theo một worker process và mỗi worker process chứa các đơn vị nhỏ hơn gọi là worker connection. Toàn bộ các đơn vị này sau đó chịu trách nhiệm xử lý các luồng yêu cầu. Worker connection cung cấp các yêu cầu cho worker process, cũng sẽ gửi nó đến master process. Cuối cùng, master process cung cấp kết quả của những yêu cầu đó.

Một worker connection có thể xử lý tới 1024 yêu cầu tương tự. Do đó, Nginx có thể xử lý hàng ngàn yêu cầu mà không gặp bất kỳ khó khăn nào.



worker\_processes 2; (xác định số tiến trình mà nginx sẽ sử dụng để xử lý các yêu cầu.)

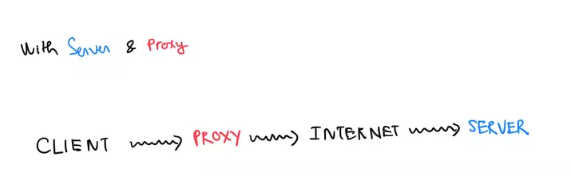
worker\_connections 1024; (xác định số lượng kết nối mà mỗi worker-process có thể xử lý đồng thời)

Các tính năng của Nginx:

* **Webserver**: nginx thường được sử dụng làm webserver phục vụ các tệp tĩnh như HTML, CSS, JS, hình ảnh và các tài nguyên tĩnh khác.
* **Proxy Server**: nginx thường được sử dụng làm reverse proxy để chuyển hướng yêu cầu từ client đến các máy chủ ứng dụng đằng sau nó
* **Load Balancer**: nginx có thể hoạt động như một load balancer để phân phối công việc đều giữa nhiều máy chủ, cải thiện khả năng chịu tải và đồng thời giảm rủi ro mất dịch vụ

**So sánh với forward proxy**

Forward Proxy hoặc có thể gọi tắt là Proxy là một "thực thể trung gian" nằm giữa client và internet.

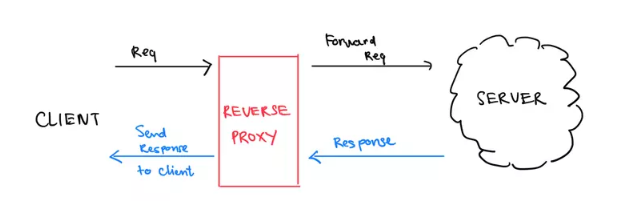
****

Mục đích chính của forward proxy là:

* Hạn chế người dùng truy cập một số trang web nhất định
* Chỉ cho phép client truy cập đến một phần tài nguyên của site thay vì toàn bộ
* Log lại các hoạt động trên internet của người dùng…

**Reverse Proxy**

Reverse Proxy là một "thực thể trung gian" nằm giữa internet và server

****

Mục đích chính của Reverse Proxy là:

* Ẩn server khỏi clinet khi không muốn public server
* Trong trường hợp hệ thống của bạn có nhiều hơn 1 server thì Reverse Proxy có thể hoạt động như một load balancer với mục đích phân tải.

**Hiểu về Driective và Context trong Nginx**

Trong cấu hình Nginx có 2 thuật ngữ quan trọng là directive và context

Mọi thứ trong file config Nginx, đều là **directive**. **Directive** gồm có 2 loại là:

* Simple Directive
* Block Directive

Simple Directive bao gồm **directive name** và **tham số**, ngăn cách nhau bởi dấu cách và kết thúc bằng dấu chấm phẩy. Ví dụ với file config trên, simple directive là:

* **listen 80;**: lắng nghe trên cổng 80.
* **server\_name nginx-tutorial.test;** đặt tên web server là nginx-tutorial.test.
* **return 200 "Hello World!\n";**: trả về status code là 200 và với nội dung: Hello World!.

Block directive tương tự với simple directive ngoại trừ việc thay vì kết thúc bằng dấu chấm phẩy thì block directive kết với cặp ngoặc nhọn **{ }**. Block directive có thể chứa những directive khác ở trong nó, được gọi là **context**. Ví dụ như **events**, **http**,... Có 4 context chính trong Nginx:

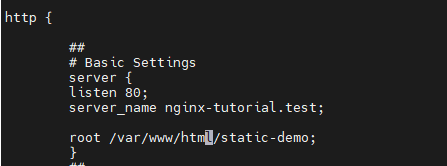
* **events { }**: Dùng để thiết lập cấu hình toàn cục. Và trong mỗi 1 file config, sẽ chỉ có duy nhất một context **events**.
* **http{ }**: Dùng để cấu hình Nginx xử lý các request HTTP và HTTPS. Và cũng như **events**, mỗi 1 file config sẽ chỉ có một **http**.
* **server{}**: Nằm trong context **http**, được sử dụng để cấu hình cho từng virtual host. Có thể có nhiều context **server** trong context **http**.
* **main**: Context **main** chính là file cấu hình.

Vd:

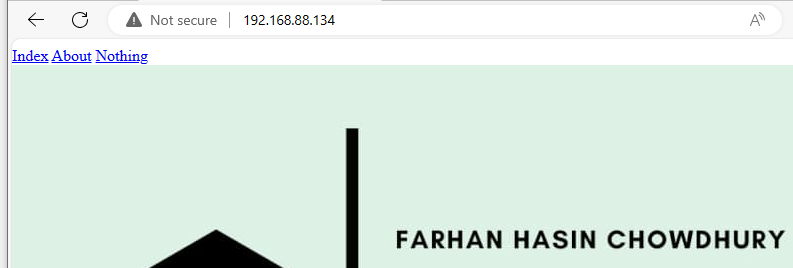
|  |
| --- |
| *events {*  *}*  *http {*  *server {*  *listen 80;*  *server\_name nginx-tutorial.test;*  *return 200 "Hello, this response from port 80!\n";*  *}*  *server {*  *listen 8080;*  *server\_name nginx-tutorial.test;*  *return 200 "Hello, this response from port 8080!\n";*  *}*  *}* |

**Lab webserver**

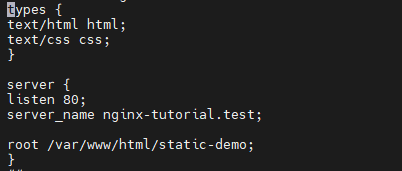
Tạo một virtual host trên Nginx lắng nghe trên port 80

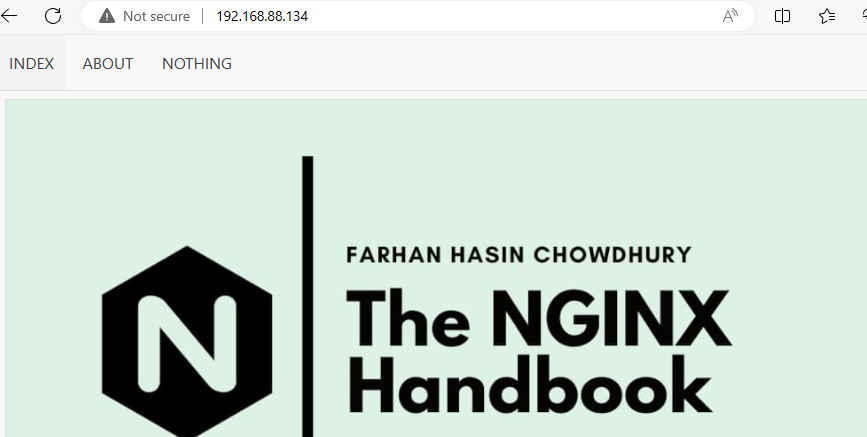


Reload lại nginx và truy cập vào trang web nhưng thấy file css chưa hoạt động



Thêm cấu hình sau để đọc file css:

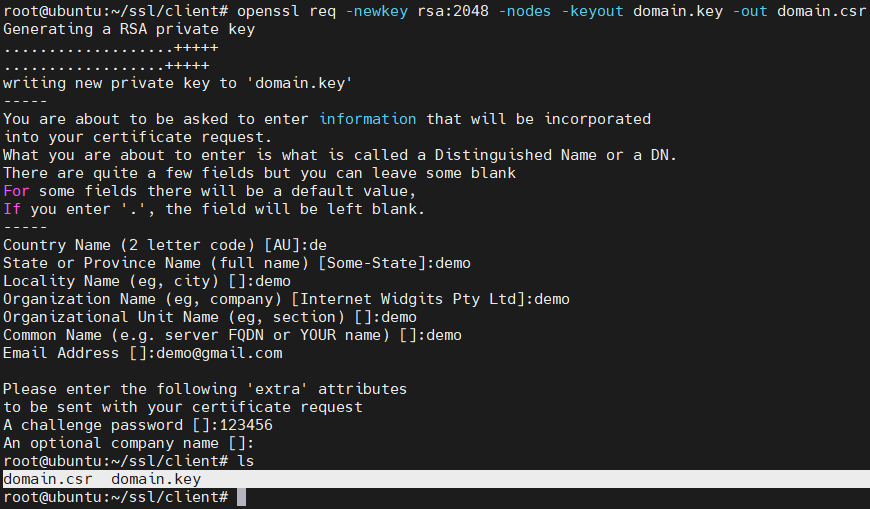




**Lab cert ssl**

Nếu chưa có private key thì có thể sử dụng options -newkey

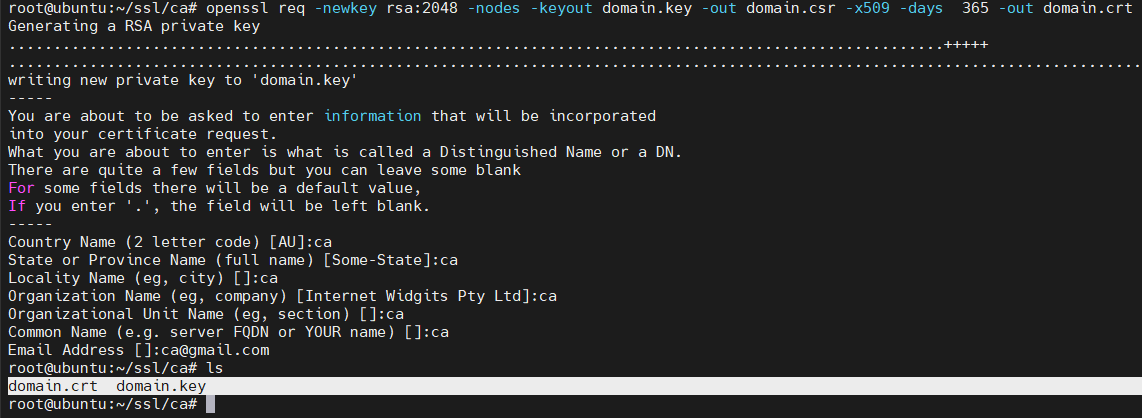
# openssl req -newkey rsa:2048 -nodes -keyout domain.key -out domain.csr



- Sau khi đã có 2 file .key và .csr thì cần gửi request lên CA để yêu cầu chứng thực (tạo các file của CA)

Tạo file crt và key cho CA để ký csr cho client

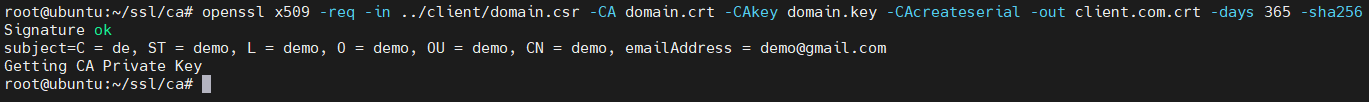
*openssl req -newkey rsa:2048 -nodes -keyout domain.key -out domain.csr -x509 -days 365 -out domain.crt*



- gửi request yêu cầu chứng thực

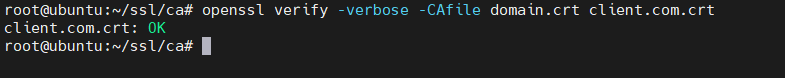
*openssl -x509 -req -in domain.csr(file request của client) -CA domain.crt(file của CA) -CAkey domain.key(Key của CA) -CAcreatserial -out client.com.crt -days 365 -sha256*

*openssl x509 -req -in domain.csr -CA domain.crt -CAkey domain.key -CAcreateserial -out client.com.crt -days 365 -sha256*

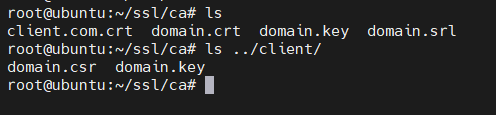


- verify thông tin Cert

openssl verify -verbose -CAfile ca.crt domain.crt



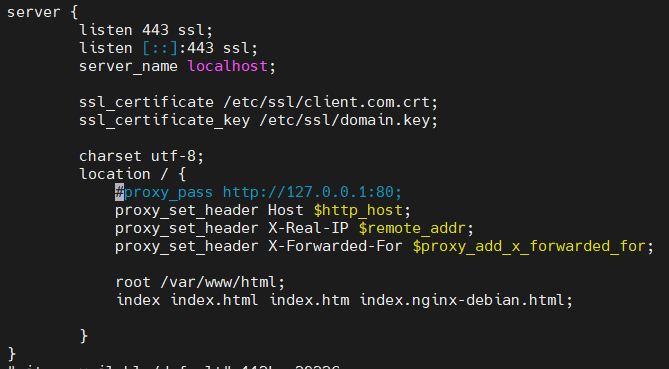
Cấu hình nginx với ssl



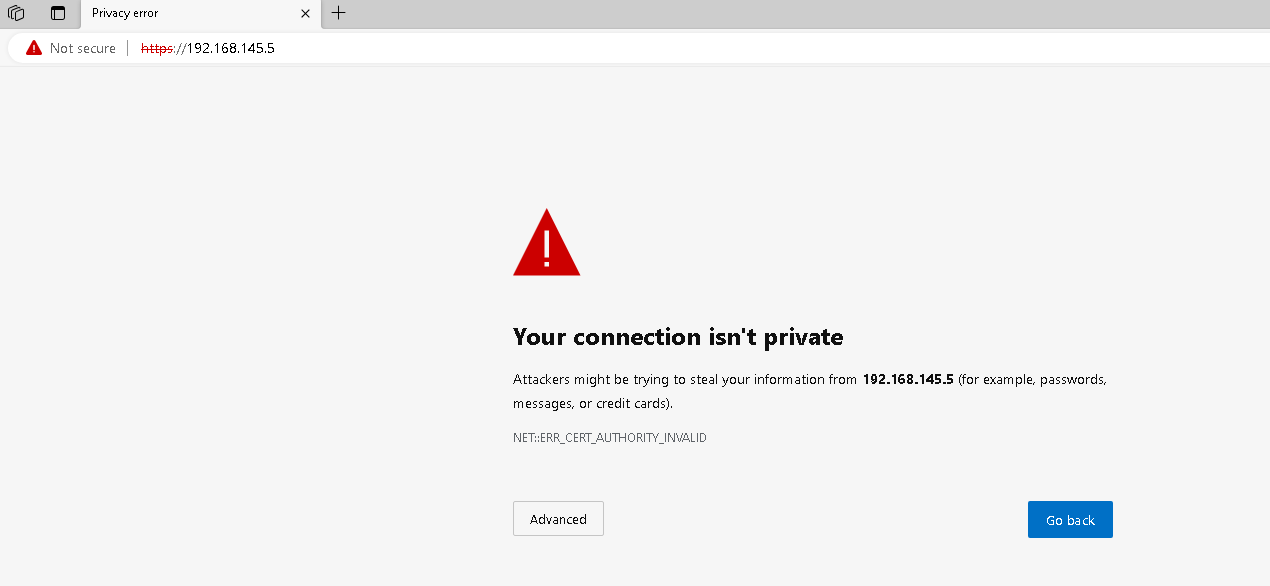
v

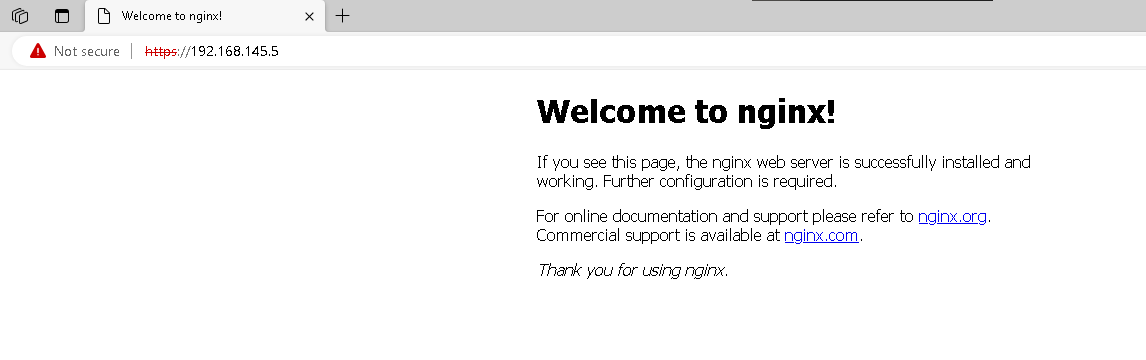
v





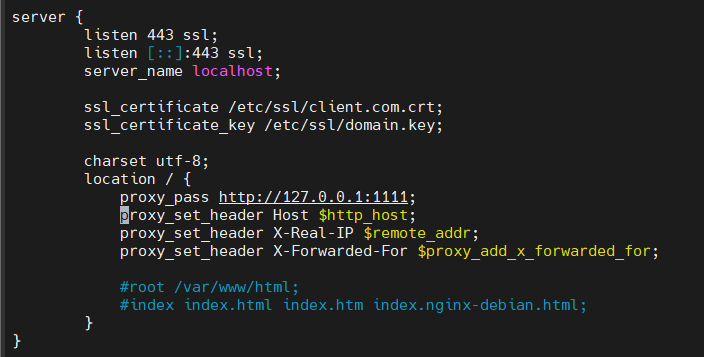
Kiểm tra kết nối:





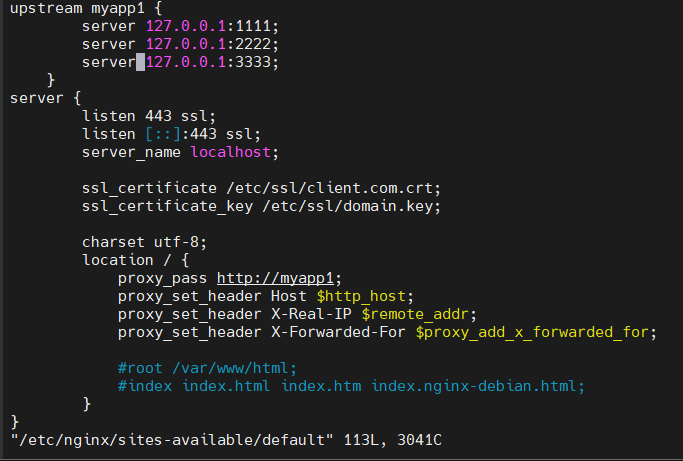
**Lab Reverse Proxy**

Cấu hình proxy\_pass http://127.0.0.1:1111;





Ngoài ra nginx cũng có thể đóng vai trò làm một loadbalancer









## Haproxy

HAProxy – High Availability Proxy là phần mềm cân bằng tải TCP/HTTP và giải pháp proxy mã nguồn mở phổ biến, có thể chạy trên Linux, Solaris và FreeBSD. Nó thường dùng để cải thiện hiệu suất và sự tin cậy của môi trường máy chủ bằng cách phân tán lưu lượng tải trên nhiều máy chủ.

Một số thuật ngữ

**ACL**

ACL được sử dụng để kiểm tra một số điều kiện và thực hiện hành động (chọn server hoặc block 1 request) dựa trên kết quả kiểm tra.

*Acl url\_blog path\_beg /blog*

ACL này sẽ phù hợp với đường dẫn request của người dùng bắt đầy bằng /blog.

**Backend**

Backend là một tập các servers nhận các requests được chuyển tiếp. Các backend được định nghĩa thuộc backend section của cấu hình HAProxy.

Dưới đây là một ví dụ về cấu hình backend, web-backend và blog-backend với hai webservers cho mỗi loại trên port 80:

|  |
| --- |
| backend web-backend  balance roundrobin  server web1 web1.yourdomain.com:80 check  server web2 web2.yourdomain.com:80 check    backend blog-backend  balance roundrobin  mode http  server blog1 blog1.yourdomain.com:80 check  server blog1 blog1.yourdomain.com:80 check |

**Frontend**

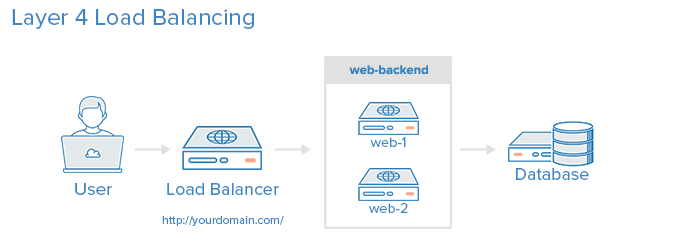
Frontend xác định các request sẽ được chuyển tiếp đến backend. Frontend được định nghĩa trong phần frontend section của cấu hình HAProxy. Định nghĩa của chúng bao gồm các thành phần sau:

* IP và port (10.1.1.7:80, \*:443)
* Các ACL
* Các quy tắc use\_backend, các quy tắc này xác định backend nào sẽ sử dụng, tùy thuộc vào điều kiện ACL thích hợp

**Phân loại Load Balancing**

**Layer 4 Load balancing**

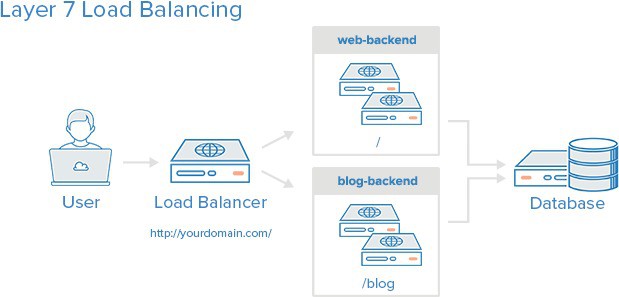
Cân bằng tải theo layer 4 sẽ chuyển tiếp lưu lượng truy cập của người dùng dựa trên IP và port (ví dụ: nếu có request đến <http://yourdomain.com/anything>, traffic sẽ được chuyển tiếp đến backend xử lý tất cả các request cgi youdomain.com trên port 80)



**Layer 7 Load Balancing**

Một các phức tạp hơn để cân bằng tải network traffic là sử dụng layer 7 load balancing. Sử dụng layer 7 cho phép bộ cân bằng tải chuyển tiếp các request đến backend server khác nhau dựa trên nội dung request của người dùng.

Chế độ cân bằng tải này cho phép chạy nhiều web appplication server trong cùng một domain và port.



Trong ví dụ này, nếu người dùng yêu cầu yourdomain.com/blog, họ được chuyển tiếp đến blog backend, là một tập hợp các servers chạy một ứng dụng blog. Các yêu cầu khác được chuyển tiếp tới web-backend, web-backend này có thể đang chạy một ứng dụng khác. Cả hai backend đều sử dụng cùng một máy chủ cơ sở dữ liệu.

Vd cấu hình frontend:

|  |
| --- |
| frontend http  bind \*:80  mode http  acl url\_blog path\_beg /blog  use\_backend blog-backend if url\_blog    default\_backend web-backend |

Ví dụ này cấu hình một frontend có tên là *http,* xử lý tất cả lưu lượng truy cập đến port 80

*acl url\_blog path\_beg /blog* khớp với request nếu đường dẫn request của người dùng bắt đầu bằng /blog

*use\_backend blog-backend if url\_blog* sử dụng ACL để ủy quyền lưu lượng truy cập vào blog-backend.

*default\_backend web-backend* chỉ định rằng tất cả traffic khác sẽ được chuyển tiếp đến web-backend.

**Thuật toán load balancing**

**round-robin**

Thuật toán round-robin thực hiện chọn servers theo lần lượt. Đây chính là thuật toán mặc định.

**leastconn**

Chọn server có số lượng kết nối ít nhất - được khuyến nghị cho các phiên dài hơn. Các servers trong cùng một backend cũng được xoay vòng theo kiểu round-robin.

**source**

Thuật toán này sẽ chọn server nào sẽ sử dụng dựa trên hàm băm của IP nguồn tức là địa chỉ IP của người dùng. Đây là một phương pháp để đảm bảo rằng người dùng sẽ kết nối với cùng một server.

**Sticky Sessions**

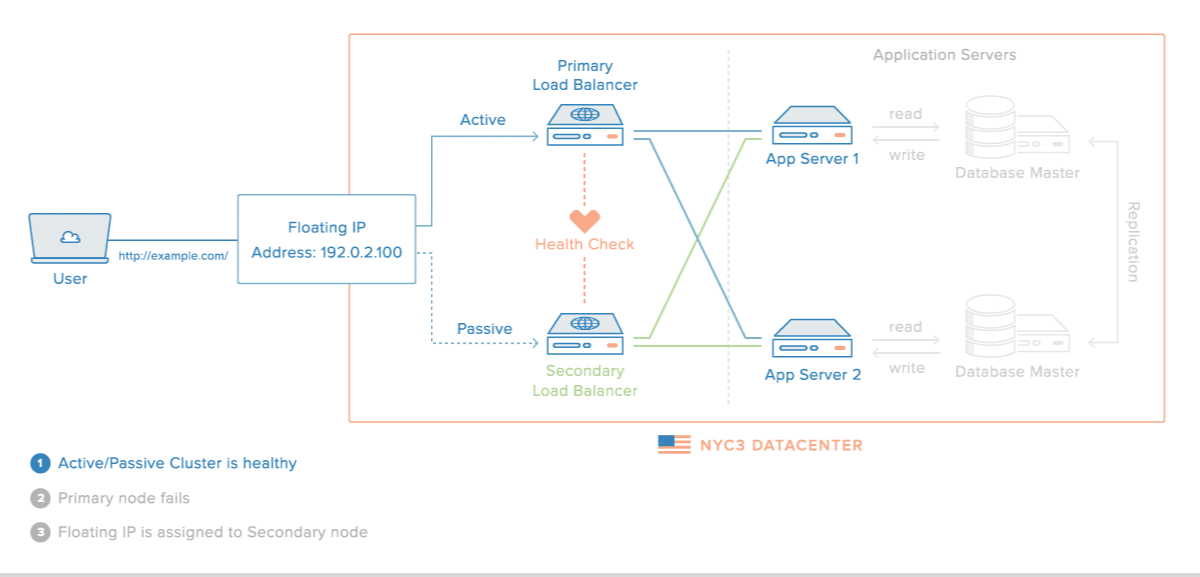
Một số ứng dụng yêu cầu người dùng tiếp tục kết nối với cùng một backend server. Sự tiếp tục kết nối này đạt được thông qua các phiên cố định, sử dụng tham số appsession trong backend yêu cầu nó.

**Health check**

**HAProxy** sử dụng health-check để xác định xem backend server có sẵn sàng để xử lý yêu cầu hay không. Điều này sẽ giúp bạn không phải xóa server khỏi backend một cách thủ công nếu server không khả dụng. Mặc định, health-check tức là cố gắng thiết lập kết nối TCP tới server, nghĩa là nó kiểm tra xem backend server có đang lắng nghe địa chỉ IP và port được cấu hình hay không.

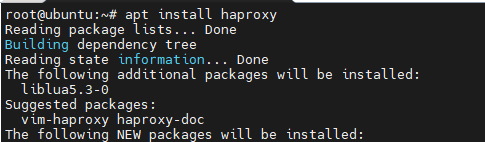
**High Availability**

**Thiết lập high availability (HA)** là một cơ sở hạ tầng không hề có một điểm lỗi nào. Bằng cách thêm dự phòng vào mỗi lớp của kiến trúc, HA sẽ ngăn được sự cố gặp phải khi một server duy nhất bị lỗi, giúp hệ thống không bị downtime. Bộ cân bằng tải tạo điều kiện thuận lợi cho dự phòng của backend layer (web/ app servers), nhưng nếu muốn đạt được độ khả dụng cao nhất, bạn cũng cần có bộ cân bằng tải dự phòng.

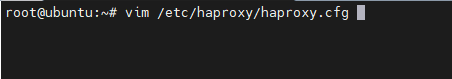


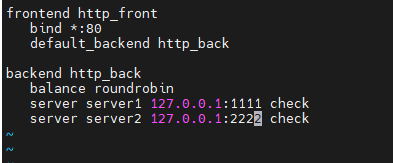
**Lab layer 4 load balancing**

Bước 1: install HAProxy

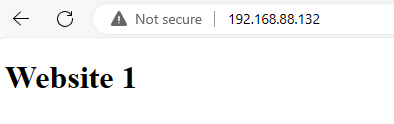


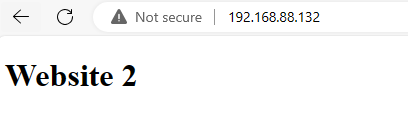
Bước 2: Configuring HAProxy





Bước 3: test kết nối

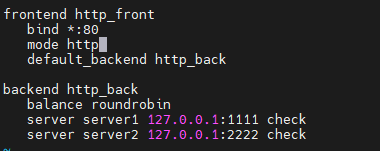




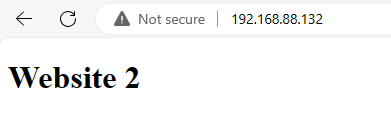
|  |
| --- |
| defaults  mode tcp  timeout client 10s  timeout connect 5s  timeout server 10s  timeout http-request 10s  frontend my\_frontend  bind 127.0.0.1:81, 127.0.0.1:82, 127.0.0.1:83  use\_backend first if { dst\_port = 81 }  use\_backend second if { dst\_port = 82 }  default\_backend third  backend first  server server1 127.0.0.1:8001  backend second  server server2 127.0.0.1:8002  backend third  server server3 127.0.0.1:8003 |

**Lab layer 7 load balancing**

* Config cơ bản cho layer 7 với 2 backend server

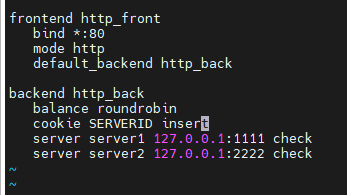


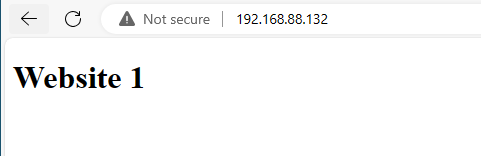




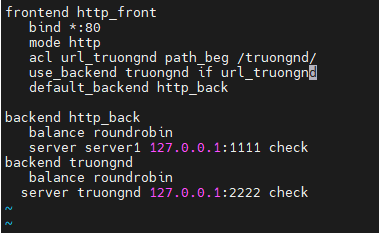
* Config layer 7 load balancing với cookie-based session

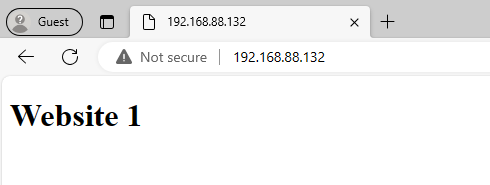
Cấu hình này sử dụng cookie để đảm bảo rằng máy khách luôn kết nối với cùng một backend.





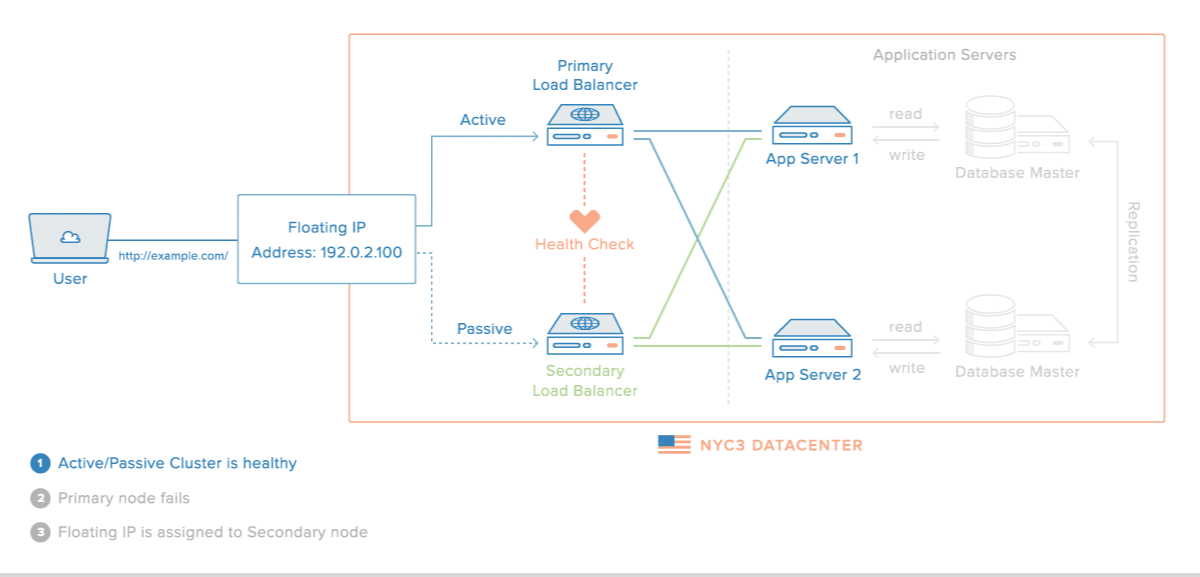
* Config layer 7 load balancing với url:







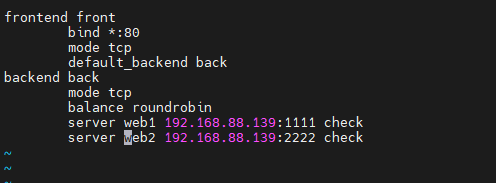
**Lab high availability**



Bài toàn: có 2 load balancer (active/passsive) khi active mất thì tự động chuyển passsive thành active và check dịch vụ vẫn còn

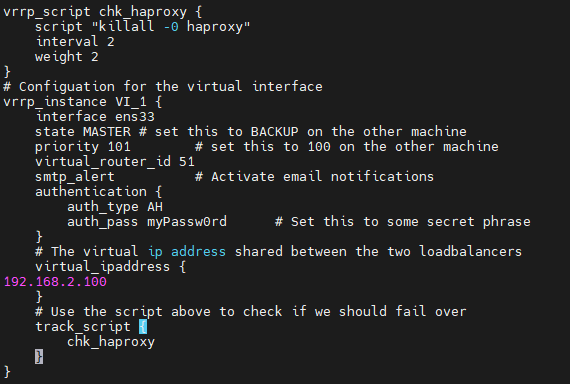
Bước 1: cấu hình load balancer bằng haproxy cho 2 server: 192.168.88.139 và 192.168.88.140:





Bước 2: cài đặt keepalive cho active (192.168.99.139)





vrrp\_script chk\_haproxy:

script "killall -0 haproxy": Sử dụng lệnh "killall -0 haproxy" để kiểm tra xem tiến trình HAProxy có đang hoạt động không. Lệnh này không thực sự giết tiến trình mà chỉ kiểm tra xem nó có tồn tại hay không.

interval 2: Kiểm tra trạng thái của HAProxy mỗi 2 giây một lần.

weight 2: Gán trọng số là 2 cho kiểm tra này, có nghĩa là nó quan trọng và ảnh hưởng đến quyết định chuyển giao.

2. vrrp\_instance VI\_1:

interface ens33: Xác định interface sẽ tham gia vào cụm VRRP, trong trường hợp này là "ens33".

state BACKUP: Định rằng máy chủ này sẽ ở trạng thái "BACKUP" trong cụm VRRP. Máy chủ còn lại (trên máy tính khác) sẽ được cấu hình là "MASTER".

priority 100: Đánh giá mức ưu tiên của máy chủ trong cụm VRRP. Trong trường hợp này, máy chủ này có mức ưu tiên là 100. Máy chủ còn lại thường được cấu hình với mức ưu tiên cao hơn để trở thành "MASTER".

virtual\_router\_id 51: Xác định một ID duy nhất cho cụm VRRP, trong trường hợp này là 51.

smtp\_alert: Kích hoạt thông báo qua email nếu có sự kiện chuyển giao trạng thái VRRP.

authentication: Xác thực cho VRRP.

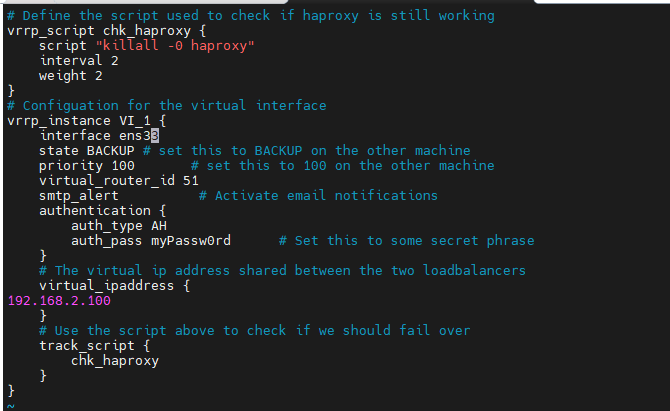
auth\_type AH: Sử dụng phương thức xác thực "AH" (Authentication Header).

auth\_pass myPassw0rd: Mật khẩu được sử dụng để xác thực giữa các thành viên trong cụm VRRP.

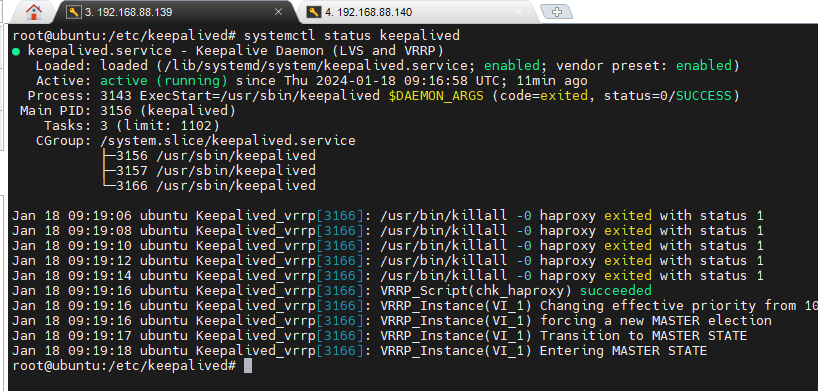
virtual\_ipaddress: Địa chỉ IP ảo được chia sẻ giữa hai máy chủ load balancer.

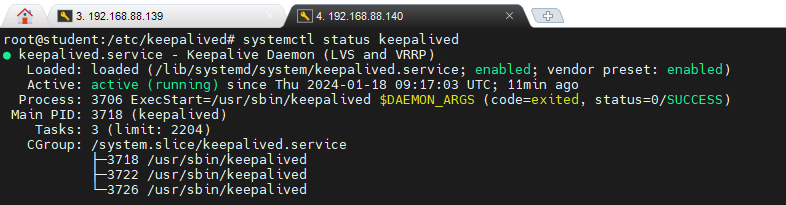
track\_script: Sử dụng script "chk\_haproxy" để kiểm tra xem máy chủ có nên chuyển giao trạng thái hay không.

Bước 3: Cấu hình keepalived cho passive (192.168.88.140)

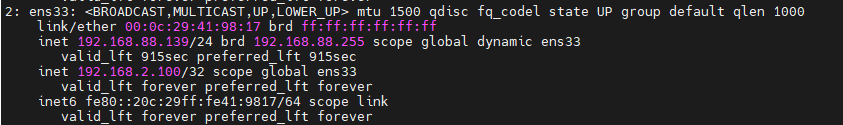


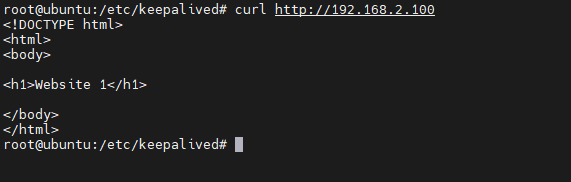
Bước 4: start service keepalived và test kết nối:



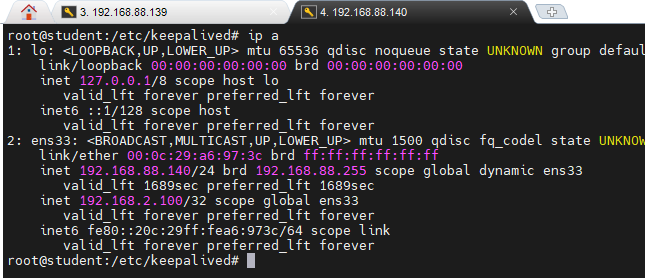


Trên máy active:

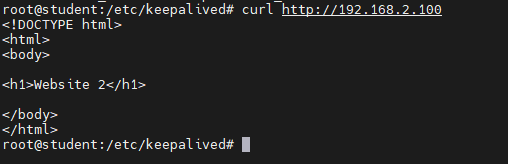




Thực hiện stop haproxy trên active và kiểm tra trên máy passive



Qua máy passive thấy đã có ip 192.168.2.100 và dịch vụ vẫn kết nối bình thường



# Tìm hiểu về Load Balancer

## Khái niệm

Load balancer là một thiết bị hoặc phần mềm có nhiệm vụ phân phối lưu lượng công việc đến một nhóm các máy chủ hoặc các tài nguyên máy chủ để tối ưu hóa hiệu suất và đảm bảo tính khả dụng của hệ thống. Chức năng chính của load balancer là phân chia công việc (workload) đều đặn giữa các máy chủ hoặc các phần của hạ tầng mạng, giảm tải cho từng máy chủ cụ thể và cung cấp tính khả dụng cao.

Công dụng của load balancer bao gồm:

**Cân bằng tải (Load Balancing):** Phân phối lưu lượng truy cập đến các máy chủ một cách cân đối, giúp tránh tình trạng quá tải cho một máy chủ cụ thể.

**Tăng tính khả dụng (High Availability):** Load balancer có thể chuyển hướng lưu lượng sang các máy chủ khác nếu một máy chủ gặp sự cố, giảm thiểu thời gian chết (downtime).

**Mở rộng linh hoạt (Scalability):** Load balancer giúp dễ dàng thêm máy chủ mới vào hạ tầng mà không làm ảnh hưởng đến tính ổn định của hệ thống.

**Bảo mật:** Nhiều load balancer cung cấp các tính năng bảo mật như tường lửa ứng dụng và chuyển đổi giao thức.

**Giảm độ trễ (Latency Reduction):** Bằng cách phân chia lưu lượng, load balancer có thể giảm độ trễ cho người dùng cuối và cải thiện trải nghiệm người dùng.

## So sánh load balancer layer 4 và layer 7

**Layer 4 Load Balancer:**

**Hoạt động ở tầng transport (Layer 4):**

* Chỉ xử lý thông tin tại tầng transport, chẳng hạn như địa chỉ IP và số cổng.

**Phân phối lưu lượng dựa trên thông tin tầng transport:**

* Quyết định chuyển hướng lưu lượng dựa trên địa chỉ IP và số cổng (TCP/UDP).

**Hiệu suất cao:**

* Thường đạt được hiệu suất cao hơn vì chỉ kiểm tra thông tin ở tầng transport mà không giải mã nội dung ở tầng ứng dụng.

**Ít phức tạp hơn**:

* Phù hợp cho các ứng dụng đơn giản với yêu cầu cân bằng tải dựa trên địa chỉ IP và số cổng.

**Layer 7 Load Balancer:**

**Hoạt động ở tầng ứng dụng (Layer 7):**

* Có thể xử lý thông tin ở mức ứng dụng, như HTTP header, URL, cookie.

**Phân phối lưu lượng dựa trên thông tin ứng dụng cụ thể:**

* Có khả năng đưa ra quyết định dựa trên nội dung cụ thể của gói tin, chẳng hạn như URL, header HTTP, hoặc cookie.

**Phức tạp hơn và yêu cầu tài nguyên cao hơn:**

* Yêu cầu xử lý nội dung ở tầng ứng dụng, nên thường phức tạp hơn và yêu cầu tài nguyên hệ thống cao hơn so với Layer 4.

**Thích hợp cho ứng dụng phức tạp:**

* Lợi ích khi cần cân bằng tải dựa trên thông tin chi tiết của ứng dụng, như phân phối tải cho các máy chủ web với nội dung động.

**Hiểu thông tin ứng dụng và có thể thực hiện các chính sách kiểm soát truy cập:**

* Có khả năng kiểm soát truy cập dựa trên thông tin ứng dụng, chẳng hạn như chặn hoặc chuyển hướng lưu lượng dựa trên URL, header HTTP, hoặc thông tin cookie.