Báo cáo thực thực tập công ty an ninh mạng viettel: Load balancing

Bùi Hoàng Dũng

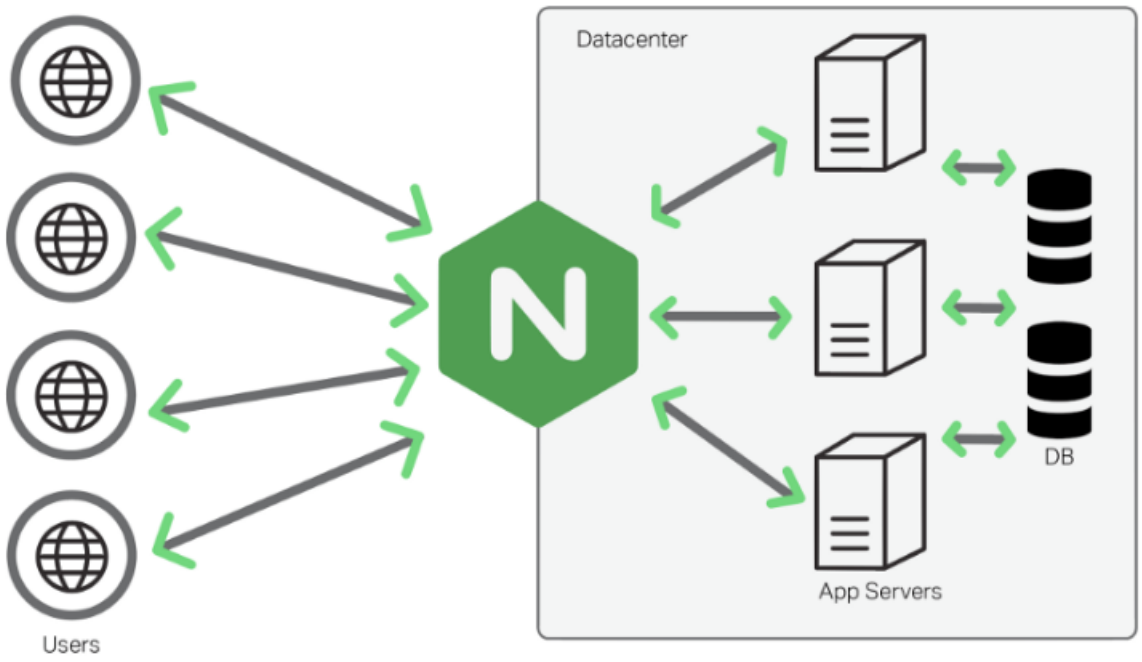
February 2024

Mục lục

1 Webserver

1.1 Nginx

Nginx là một máy chủ Web Server mã nguồn mở. Dự án Nginx được phát hành và sử dụng như một web server được thiết kế hướng đến mục đích cải thiện tối đa hiệu năng và sự ổn định. Nginx còn là 1 trong số ít những máy chủ được viết để giải quyết vấn đề C10K. Hiểu đơn giản thì do các máy chủ web truyền thống xử lý các yêu cầu dựa trên luồng (thread), tức là mỗi khi máy chủ web nhận được 1 yêu cầu mới thì nó sẽ tạo ra 1 luồng mới để xử lý yêu cầu này, và cứ thế khi số lượng các yêu cầu gửi đến máy chủ web ngày càng nhiều thì số lượng các luồng xử lý này trong máy chủ sẽ ngày càng tăng. Điều này dẫn đến việc thiếu hụt tài nguyên cấp cho các luồng xử lý trên. Còn với Nginx, nó không dựa trên luồng để xử lý yêu cầu mà nó sử dụng 1 kiến trúc bất đồng bộ hướng sự kiện linh hoạt. Kiến trúc này sử dụng ít, nhưng quan trọng hơn, là lượng bộ nhớ có thể dự đoán được khi hoạt động.



Hình 1: Nginx

Sau đây ta sẽ đi vào chi tiết cấu trúc của Nginx và Nginx làm thế nào để có thể handle được hàng nghìn requests, connection.

1.2 Lab for webserver

Các kiến thức cần lưu ý khi cấu hình một webserver sử dụng nginx:
Sau đây là một cấu hình cơ bản trong một file config của nginx:

```
events {
}
http {
include mime.types;
server {
```

```
listen 80;
server_name 192.168.74.132;
root /demo/demo;
}
}
```

Trong đó các trường trong file config trên được định nghĩa như sau:

- **listen**: để diễn tả webserver được host ở port nào. Ở đây webserver được host ở port 80.
- **server_name**: diễn tả địa chỉ IP của webserver
- **root** là chỉ đường dẫn đến thư mục chứa các nội dung của website

Tiếp theo là **Location block**. Location block diễn tả cách xử lý các yêu cầu HTTP tương ứng với một đường dẫn cụ thể, cho phép ta có thể áp dụng những cài đặt cụ thể cho các phần khác nhau của trang web hoặc ứng dụng. Thông thường location sẽ được sử dụng trong cấu hình server block. Với các option khác nhau khi sử dụng location ta sẽ có những ý nghĩa khác nhau. Cụ thể thể được định nghĩa trong file sau:

```
events {}

http {

  include mime.types;

  server {

    listen 80;
    server_name 167.99.93.26;

    root /sites/demo;

    # Preferential Prefix match
    location ^~ /Greet2 {
      return 200 'Hello from NGINX "/greet" location.';
    }

    # # Exact match
    # location = /greet {
    #   return 200 'Hello from NGINX "/greet" location - EXACT MATCH.';
    # }

    # # REGEX match - case sensitive
    # location ~ /greet[0-9] {
    #   return 200 'Hello from NGINX "/greet" location - REGEX MATCH.';
    # }
```

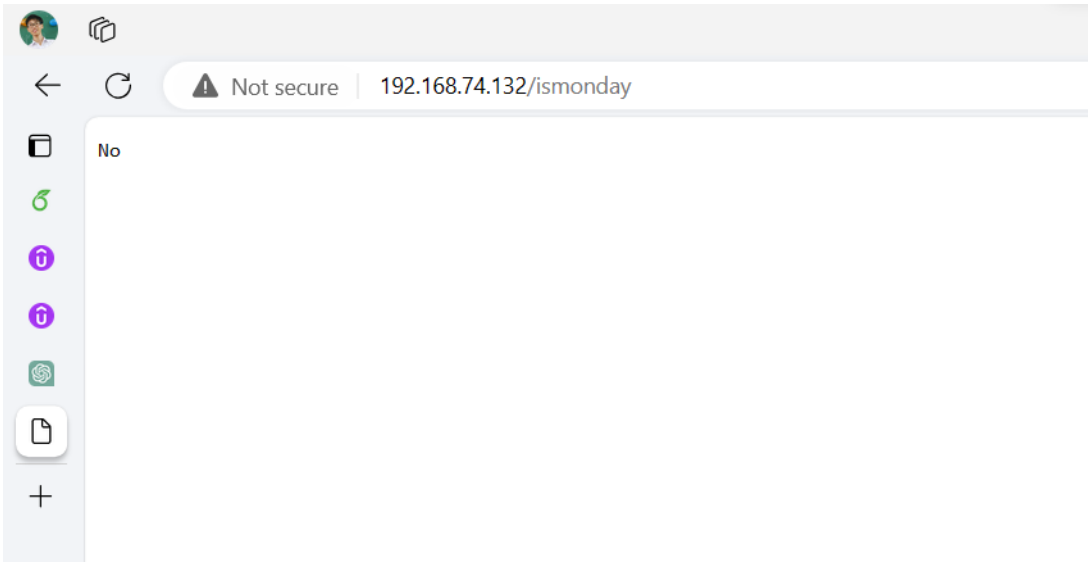
```
    # REGEX match - case insensitive
    location ~* /greet[0-9] {
      return 200 'Hello from NGINX "/greet" location - REGEX MATCH INSENSITIVE.';
    }
  }
}
```

Mỗi option ở trên đều có một ý nghĩa khác nhau. Đối với **location /greet** sẽ thể hiện uri là prefix match nghĩa là sẽ check URI trùng với những location có tiền tố tương ứng. Đoạn return 200 ý là để chỉ một đoạn text được trả về với đường dẫn tương ứng ở bên trên. Lưu ý 200 ở đây có nghĩa là một response code với mỗi loại response code thì sẽ có những chức năng khác nhau

Variables: Trong tệp cấu hình của Nginx, bạn có thể sử dụng các biến để định nghĩa các giá trị động và tái sử dụng chúng trong nhiều phần của cấu hình. Các biến trong cấu hình Nginx thường được đặt trong các khối http, server, hoặc location. Dưới đây là cách bạn có thể sử dụng biến trong tệp cấu hình Nginx:

```
events {}

http {

  include mime.types;

  server {

    listen 80;
    server_name 192.168.74.132;

    root /demo/demo;

    set $mon 'No';

    # Check if weekend
    if ( $date_local ~ 'Monday' ) {
      set $mon 'Yes';
    }

    location /is_monday {

      return 200 $mon;
    }
  }
}
```

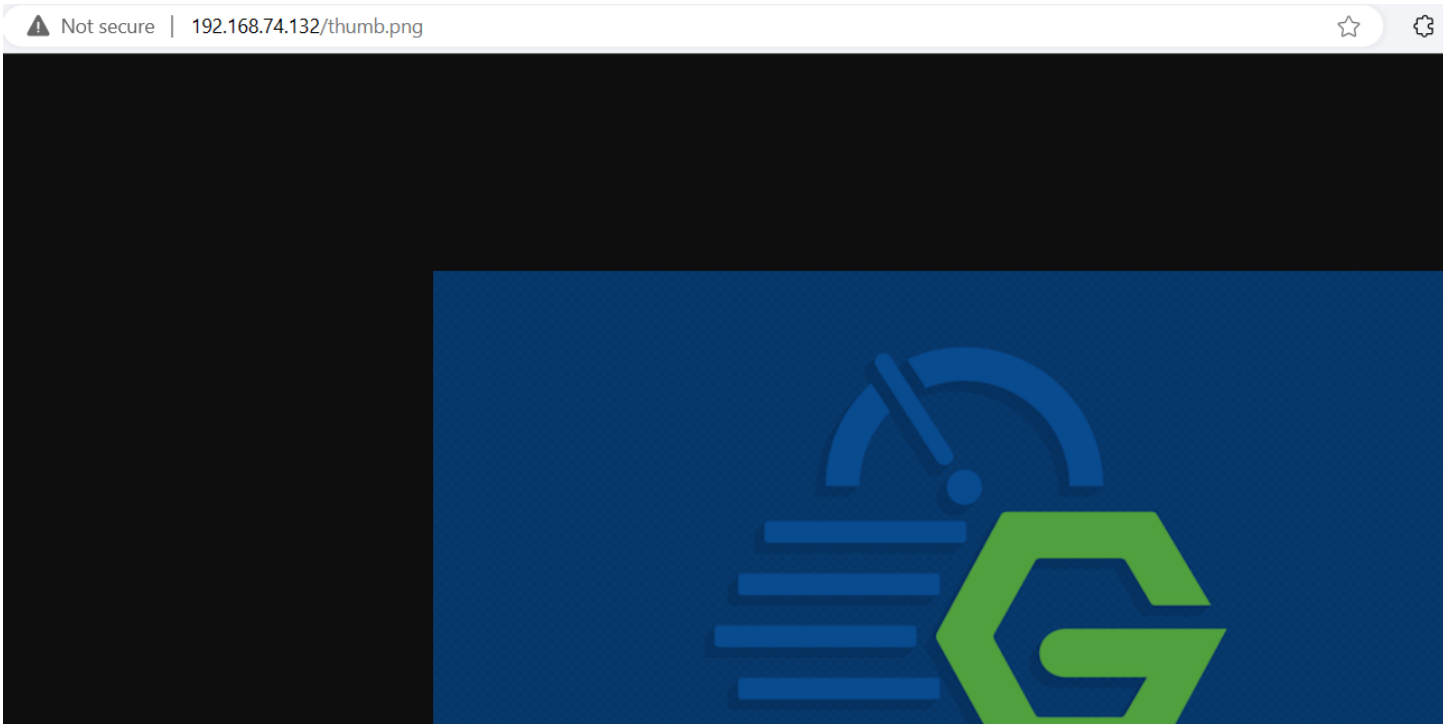
Ở đây, em thực hiện việc đặt một biến và thực hiện kiểm tra xem là ngày hôm nay có phải là thứ hai là không. Thực hiện chạy và thu được kết quả như sau:



Hình 2: Variable lab

Rewrite vàvà Redirect: Trong tệp cấu hình của Nginx, redirect và rewrite là hai chỉ thị quan trọng để điều hướng yêu cầu và xử lý URL. Redirect là quá trình chuyển hướng một yêu cầu từ một URL hoặc địa chỉ web đến một URL khác. Chuyển hướng có thể được thực hiện với mã trạng thái HTTP, thường là 301 (Moved Permanently) hoặc 302. Sau đây là một file config để test tính năng redirect:

```
events {
}
http {
include mime.types;
server {
listen 80;
server_name 192.168.74.132;
root /demo/demo;
location /logo{
return 307 /thumb.png; # redirect từ /logo sang /thumb.png
}
}
}
```



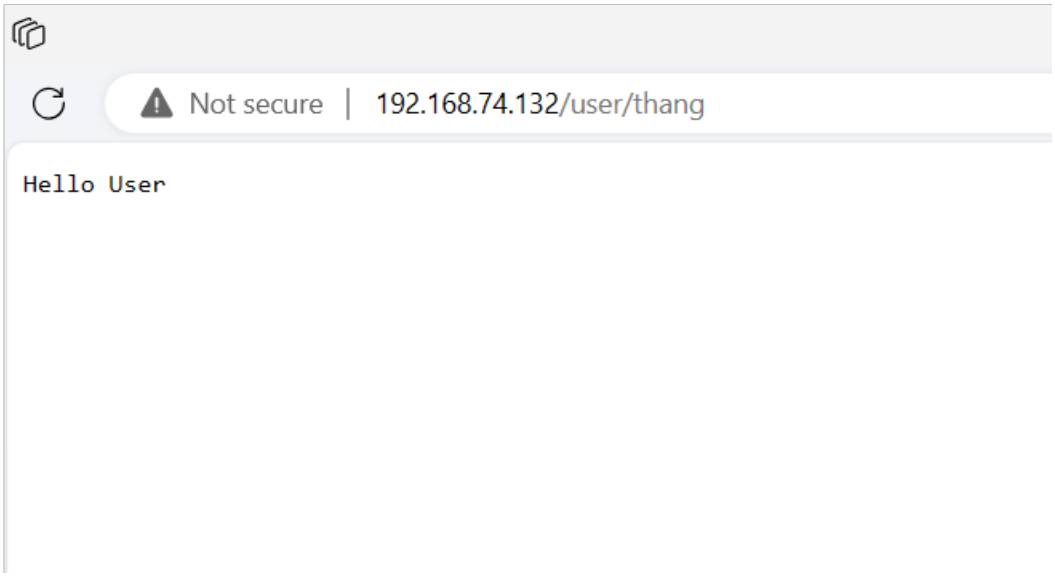
Hình 3: Redirect

Còn rewrite là quá trình chuyển đổi hay sửa đổi một URL trước khi nó được xử lý hoặc chuyển tiếp đến máy chủ. Rewrite thường được sử dụng để điều chỉnh cấu trúc URL hoặc để xử lý các yêu cầu theo các quy tắc cụ thể. Sau đây là một file config cụ thể để test tính năng rewrite. Lưu ý khi sử dụng rewrite thì Nginx sẽ thực hiện re-evaluated như một request mới :

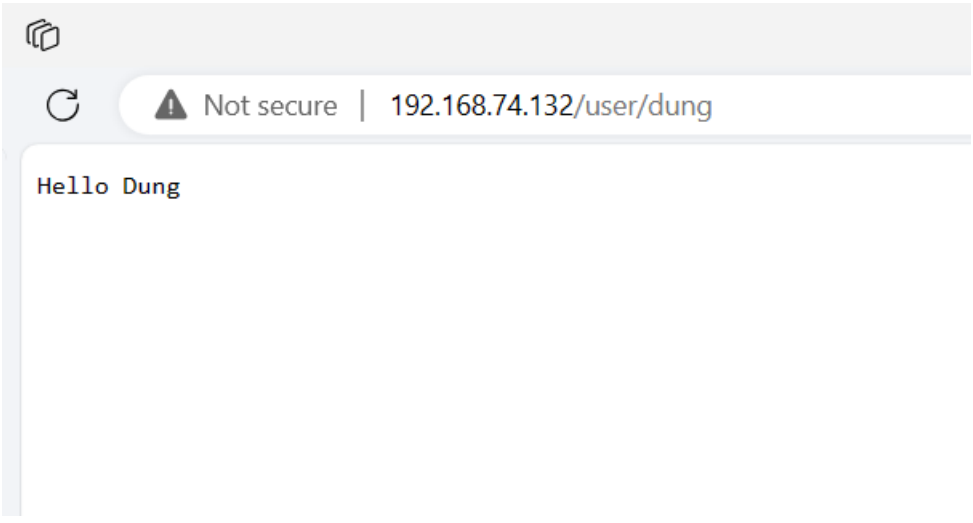
```
events {}

http {

  include mime.types;

  server {

    listen 80;
    server_name 167.99.93.26;

    root /sites/demo;

    rewrite ^/user/(\w+) /greet/$1 last;
    rewrite ^/greet/john /thumb.png;

    location /greet {

      return 200 "Hello User";
    }
```

```
    location = /greet/john {
      return 200 "Hello John";
    }
  }
}
```

Thực hiện kiểm tra kết quả:



Hình 4: /user/thang



Hình 5: /user/dung

Try files and named location: là một chỉ thị được sử dụng để thử một loạt các tệp và thư mục để xác định tệp nào hoặc thư mục nào sẽ được sử dụng cho một yêu cầu cụ thể. Điều này thường được sử dụng trong các tình huống xử lý URL hoặc khi cần xác định tệp cụ thể để phục vụ. Try files sẽ tìm xem trong những file hay đường dẫn được định nghĩa xem có tồn tại hay không và ngay khi tệp hoặc đường dẫn đó xuất hiện thì sẽ thực hiện response với tệp và đường dẫn được tồn tại và sẽ không xét những tệp hoặc đường dẫn ở đằng sau. Lưu ý chỉ tệp hoặc đường dẫn cuối cùng được định nghĩa mới thực hiện quá trình rewrite:

```
events {}

http {

  include mime.types;

  server {

    listen 80;
    server_name 192.168.74.132;

    root /demo/demo;
    try_files /cat.png /greet;

    location /greet {

      return 200 "Hello User";
    }

  }
}
```

Trong ví dụ của file config trên thì try_files sẽ thực hiện tìm lần lượt cat.png và /greet và trả về những object tương ứng nếu object đó tồn tại.

Logging: Với logging thì Nginx cung cấp 2 file log đó là access-log và error-log. Mỗi loại đều có mục đích và nhiệm vụ riêng: access-log là để ghi log với những truy cập, còn error-log là ghi những log về lỗi cấu hình hay lỗi kết nối. Sau đây là một ví dụ về logging trong nginx:

```
events {}

http {

  include mime.types;

  server {

    listen 80;
    server_name 192.168.74.132;
```

```
    root /demo/demo;

    location /secure {

      # Add context specific log
      access_log /var/log/nginx/secure.access.log;

      # Disable logs for context
      #access_log off;

      return 200 "Welcome to secure area.";
    }

  }
}
```

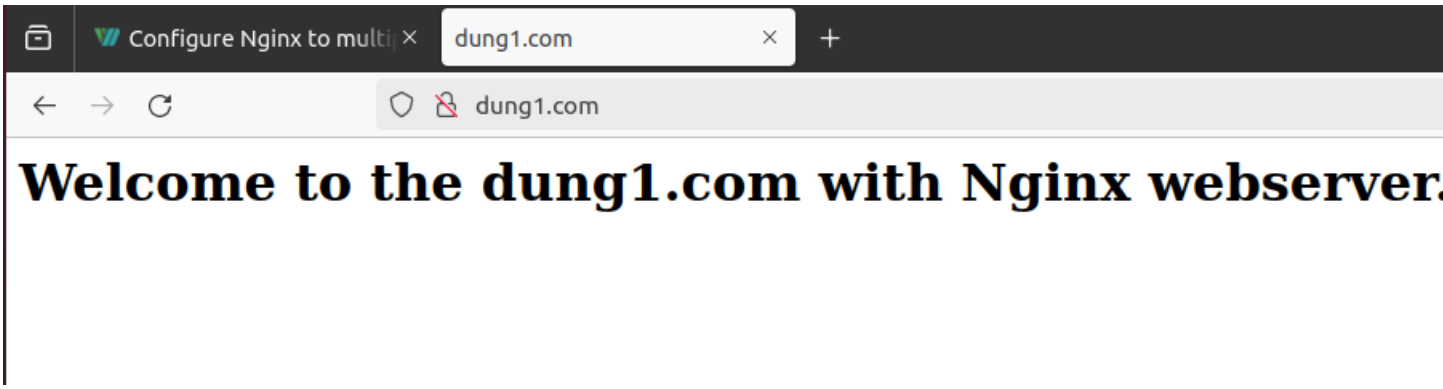
Đoạn config trên thực hiện ghi log cho đường dẫn vào một file log riêng khác 2 file log mặc định của nginx.

Buffer và timeout: Buffering là quá trình một process hay một Nginx worker thực hiện quá trình đọc và ghi dữ liệu lên RAM hoặc nếu dữ liệu quá lớn thì sẽ thực hiện ghi lên disk. Ví dụ khi một webserver nhận một request HTTP TCP ở port 80 Webserver sẽ thực hiện ghi nó nên RAM để response lại request đó thì webserver sẽ thực hiện đọc một file HTML từ disk và ghi nó lại lên RAM và thực hiện transfer dữ liệu cho client. Dưới đây là một số setup về buffer và timeout trong nginx:

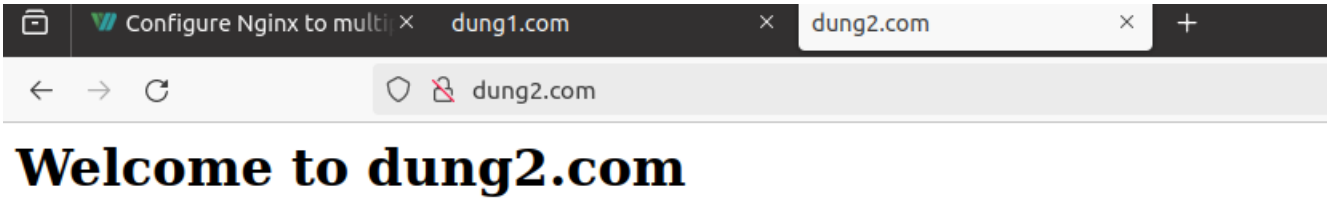
```
 client_body_buffer_size 10K;
  client_max_body_size 8m;

  # Buffer size for Headers
  client_header_buffer_size 1k;

  # Max time to receive client headers/body
  client_body_timeout 12;
  client_header_timeout 12;

  # Max time to keep a connection open for
  keepalive_timeout 15;

  # Max time for the client accept/receive a response
  send_timeout 10;

  # Skip buffering for static files
  sendfile on;

  # Optimise sendfile packets
  tcp_nopush on;
```

Thực hiện deploy 2 website trên nginx cùng một host với 2 domain khác nhau:

```
events {}
http {
  include mime.types;
  server {
    listen 80;
    root /var/www/html/dung1.com;
    index index.html;
    server_name dung1.com;
    location / {
       try_files $uri $uri/ =404;
    }
  }
  server {
    listen 80;
    root /var/www/html/dung2.com;
    index index.html;
    server_name dung2.com;
    location / {
       try_files $uri $uri/ =404;
   }
  }
}
```



Hình 6: dung1.com



Hình 7: dung2.com

1.3 Lab for cert ssl

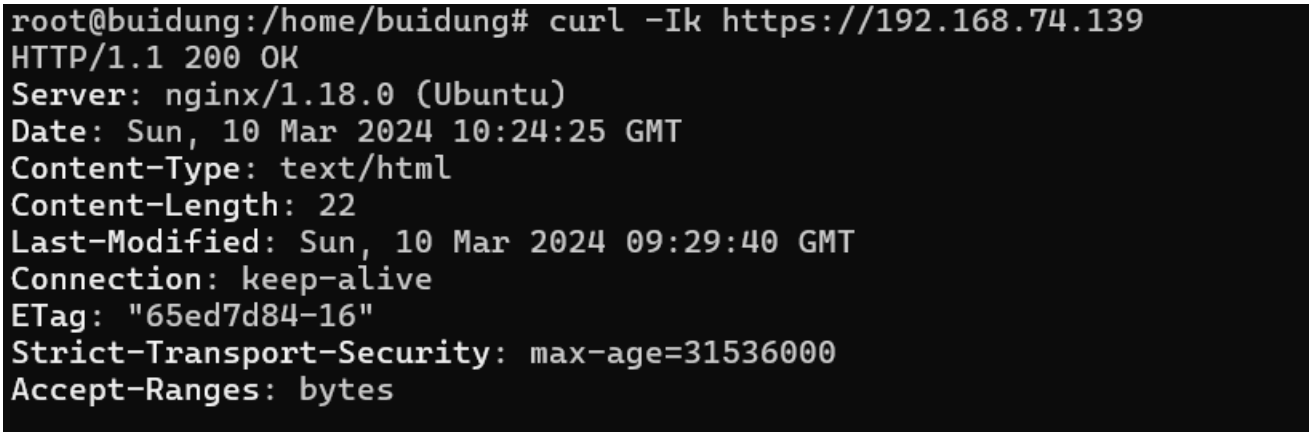
Thực hiện lab để config certficate cho trang web sử dụng HTTPS ở port 443:
Dầu tiên ta thực hiện tạo key và certificate cho website:

```
openssl req -x509 -days 10 -nodes -newkey rsa:2048 -keyout
/etc/nginx/ssl/self.key -out /etc/nginx/ssl/self.crt
```

Sau khi tạo xong key cũng như certificate tiến hành thực hiện config file nginx.conf

```
events{}
http{
  server{
     listen 443 ssl;
     root /home/buidung/test;
     server_name 192.168.74.139;
     index text.html;
     ssl_certificate /etc/nginx/ssl/self.crt;
     ssl_certificate_key /etc/nginx/ssl/self.key;
     ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
     #optimize cipher suite
     ssl_prefer_server_ciphers on;
     ssl_ciphers ECDH+AESGCM:ECDH+AES256:ECDH+AES128:DH:3DES:!ADH:!AECDH:!MD5;
     ssl_dhparam /etc/nginx/ssl/dhparam.pem;
     # enables HSTS
     add_header Strict-Transport-Security "max-age=31536000" always;
     # SSL session
     ssl_session_cache shared:SSL:40m;
     ssl_session_timeout 4h;
     ssl_session_tickets on; #provide browser with a ticket

     location / {
         try_files $uri $uri/ =404;
     }
}
```

Sau khi hoàn thành file config tiến hành curl đến địa chỉ trên:



Hình 8: Curl

1.4 Lab for reverse proxy

Reverse proxy là một loại máy chủ hoạt động như một lớp trung gian giữa máy chủ và người dùng cuối. Reverse proxy đứng trước nhiều server để nhận request từ client . Trong kiến trúc mạng, nó được sử dụng để chuyển hướng các yêu cầu từ người dùng đến các máy chủ đích và trả lại các phản hồi từ máy chủ đến người dùng. Reverse proxy có thể chịu trách nhiệm cho việc che giấu thông tin về các máy chủ đích, làm cho chúng không thể tiếp cận trực tiếp từ bên ngoài. Điều này giúp bảo vệ máy chủ khỏi các cuộc tấn công trực tiếp. Ngoài ra, Reverse proxy có khả năng chuyển hướng yêu cầu từ người dùng đến các máy chủ đích dựa trên các quy tắc cấu hình. Điều này cho phép phân phối tải và cải thiện hiệu suất hệ thống bằng cách chia công việc giữa các máy chủ, có thể lưu trữ bản sao của các tài nguyên từ máy chủ đích để giảm độ trễ và tăng tốc độ truy cập cho người dùng khi cùng một tài nguyên được yêu cầu nhiều lần. Reverse proxy có thể giúp giảm gánh nặng cho máy chủ bằng cách thực hiện giải mã và mã hóa SSL/TLS. Điều này giúp cải thiện hiệu suất và quản lý chứng chỉ SSL/TLS.

Forward proxy là một loại máy chủ đứng trước nhiều client để gửi request của client vào internet đến server. Forward proxy nhận request từ client và response lại câu trả lời của server tới các client. Forward proxy có thể được sử dụng để che giấu địa chỉ IP của máy khách và bảo vệ sự ẩn danh của người dùng. Nó cũng có thể kiểm soát và lọc nội dung để ngăn chặn truy cập vào các trang web cụ thể hoặc loại nội dung. Forward proxy có khả năng kiểm soát quyền truy cập vào Internet, cho phép quản trị viên thiết lập các quy tắc và chính sách để giới hạn hoặc cho phép truy cập vào các tài nguyên cụ thể. Forward proxy có thể lưu trữ các bản sao của các tài nguyên được yêu cầu, giảm độ trễ và tiết kiệm băng thông bằng cách cung cấp nội dung lưu trữ (cache) cho các máy khách.

	Forward Proxy	Reverse Proxy
Purpose	Provides anonymity and caching to clients	Improves server performance, load balancing, and security
Location	Between the client and the internet	Between the internet and server
Visibility	The client is aware of the proxy	The server is not aware of the proxy
Configuration	The client must be configured to use proxy	Server must be configured to use proxy use
Use cases	Bypassing content filters, accessing restricted content	Load balancing, caching, SSL/TLS offloading, web application firewall
Examples	Squid, Proxy, Tor	Nginx, Apache, HAProxy

Hình 9: Reverse proxy vs Forward proxy

Sau đây em sẽ thực hiện một bài lab cấu hình nginx như một reverse proxy: Đầu tiên, tạo một webserver bằng python trong file server.py bởi câu lệnh sau:

```
from http.server import SimpleHTTPRequestHandler
from socketserver import TCPServer
port = 8000
host = "192.168.74.139"
handler = SimpleHTTPRequestHandler
httpd = TCPServer((host, port), handler)
print(f"Server đang lắng nghe tại http://{host}:{port}")
httpd.serve_forever()
```

Tiếp theo thực hiện config tiếp nginx.conf:

```
    events{}
http{
   server{
       listen 8888;
       server_name 192.168.74.139;
       location / {
          return 200 'Hello from Nginx\n';
       }
       location /python{
          proxy_set_header proxied nginx;
          proxy_pass 'http://192.168.74.139:8000/';
          include proxy_params;
       }
```

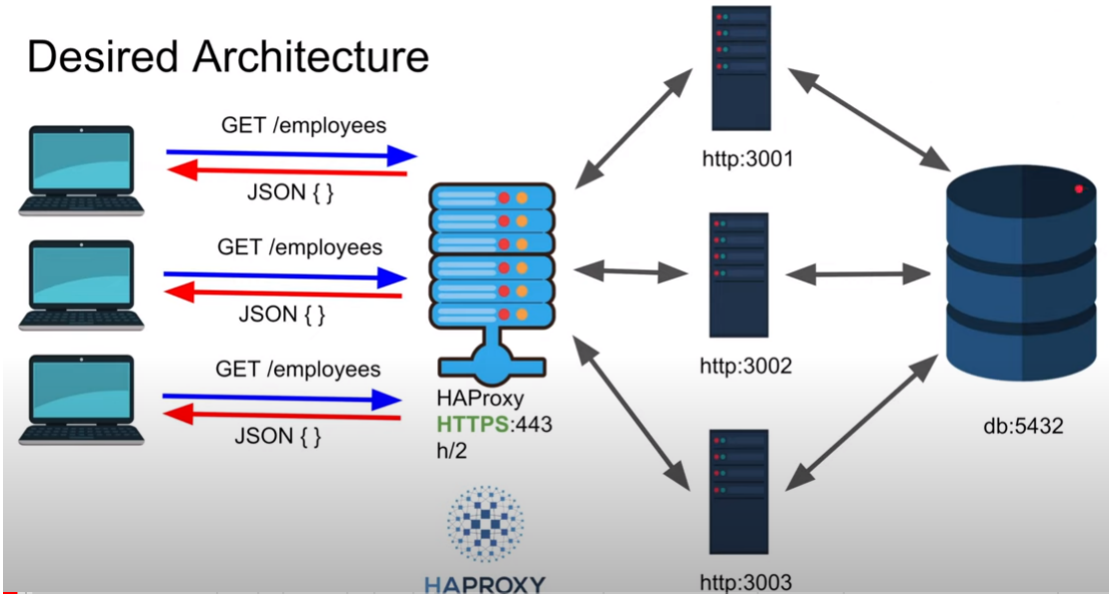
```
    }
}
```

2 HAproxy

2.1 HAproxy là gì

HAProxy (High Availability Proxy) là một phần mềm mã nguồn mở chuyên dụng cho việc cân bằng tải và chuyển hướng các yêu cầu trên mạng. Nó thường được sử dụng để tăng khả năng mở rộng và đảm bảo sẵn sàng cao cho các ứng dụng web và dịch vụ. HAproxy gồm 2 phần là backend và frontend:

- Front end: thực hiện handle các tác vụ ở trước haproxy. Bao gồm các nhiệm vụ như: ACL, binding, timeout client.
- Back end: thực hiện handle các tác vụ ở sau haproxy như là timeout server, timeout connection, thực hiện các thuật toán load balance.



Hình 10: HAproxy

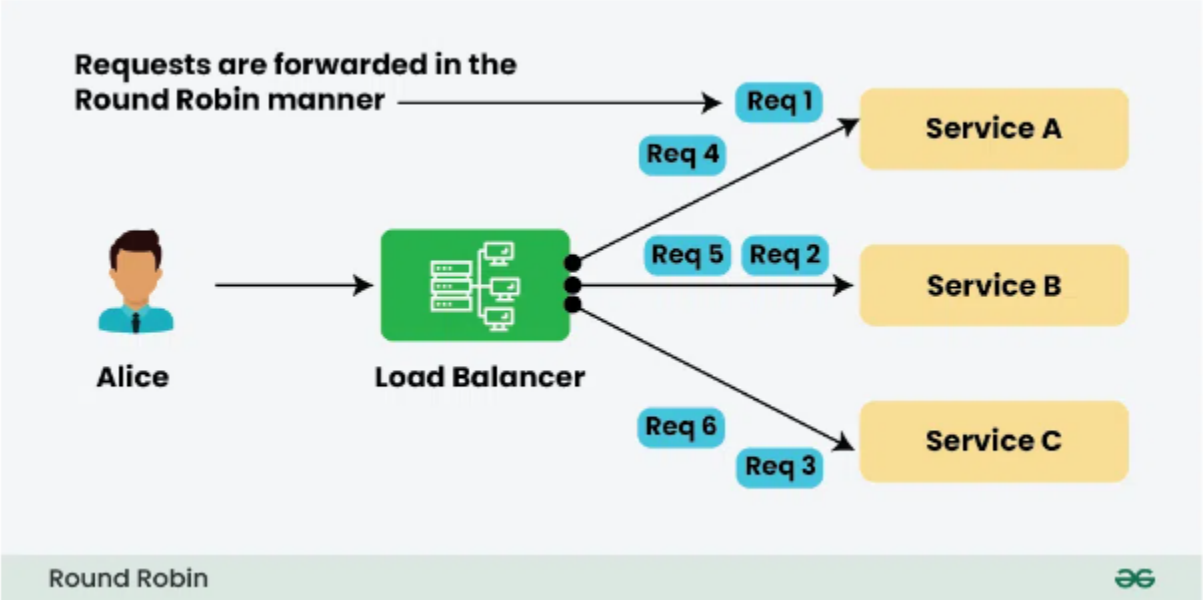
Load balancing là một thành phần trong hệ thống máy chủ và mạng, được sử dụng để phân phối công việc và tăng khả năng mở rộng của ứng dụng hoặc dịch vụ web. Chức năng chính của load balancer là phân phối lưu lượng truy cập từ người dùng hoặc thiết bị đến một nhóm các máy chủ hoặc các tài nguyên máy chủ sao cho mỗi máy chủ nhận được một lượng công việc tương đối cân đối. Load balancer thực hiện các nhiệm vụ như là: phân phối tải, duy trì hiệu suất, đảm bảo tính sẵn sàng của dịch vụ, bảo mật và thực hiện SSL,....

- Load balancing ở Layer 4 là một dạng của load balancing thực hiện ở tầng mạng của mô hình OSI (Open Systems Interconnection). Cụ thể, nó hoạt động ở tầng Transport (tầng 4) của mô hình này. Layer 4 load balancer thực hiện quyết định chia lưu lượng dựa trên các thông tin như địa chỉ IP nguồn, địa chỉ IP đích, cổng nguồn và cổng đích trong các gói tin dữ liệu. Nó không thể xem xét nội dung của các gói tin hay các thông tin ở tầng 7 (tầng ứng dụng) của mô hình OSI.

- Load balancing ở Layer 7 hoạt động ở tầng ứng dụng trong mô hình OSI. Layer 7 load balancer có khả năng hiểu và quyết định dựa trên thông tin ở tầng ứng dụng, bao gồm nội dung của gói tin dữ liệu. Điều này cho phép nó thực hiện các quyết định phân phối lưu lượng dựa trên thông tin chi tiết về ứng dụng, ví dụ như URI (Uniform Resource Identifier), header HTTP, cookie, hoặc thông tin khác liên quan đến ứng dụng.

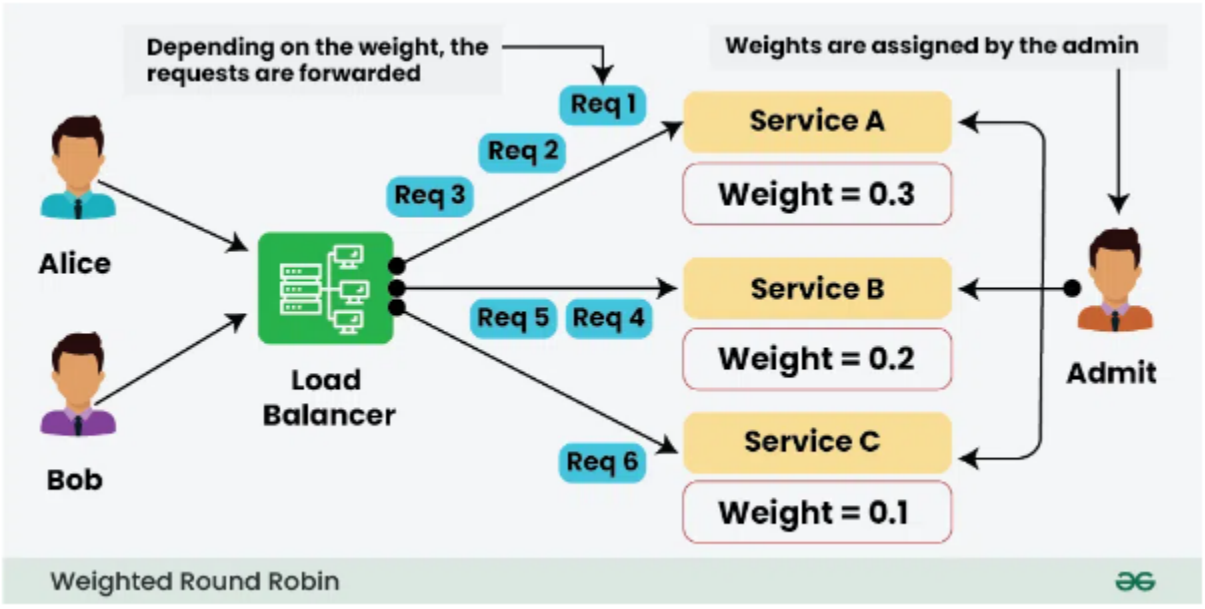
Các thuật toán load balancing:

- **Round robin** là một thuật toán thực hiện load balacing bằng cách rotate các request đến từng server/service một. Tuy nhiên thuật toán không xem xét đến việc xem tải của các server nên có thể gây quá tải nếu tải của một server nào đó đã gần như quá tải từ trước



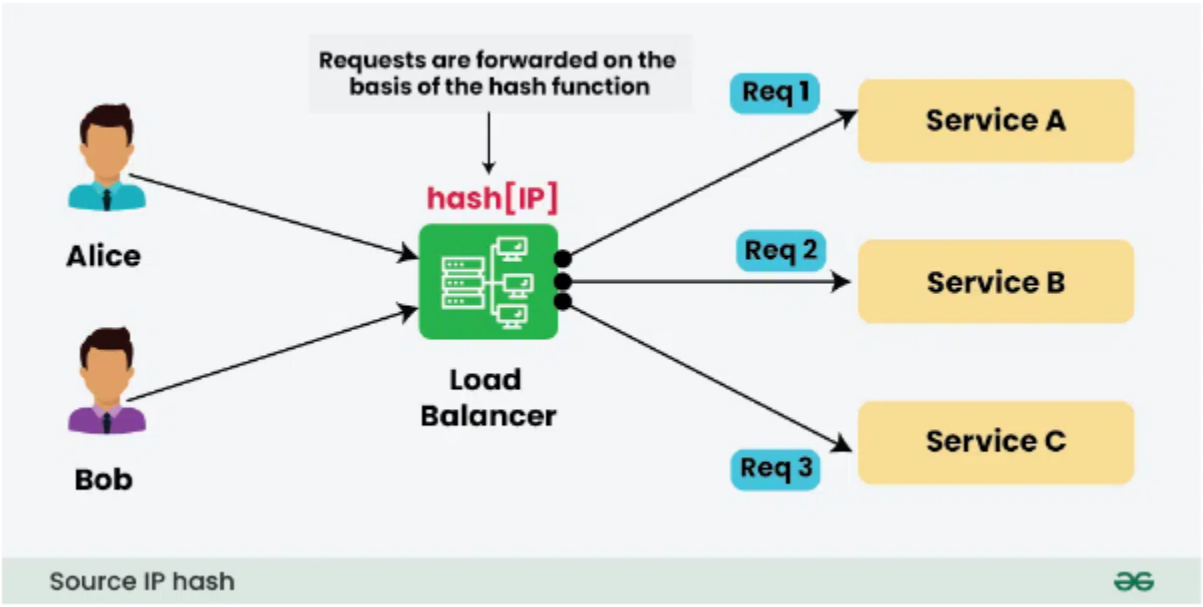
Hình 11: round robin algorithms

- **Weighted Round Robin** giống như round robin weighted round robin cũng là một thuật toán load balacing tĩnh với tư tưởng giống với round robin. Tuy nhiên thuật toán này có điểm khác là các resource sẽ được đánh điểm weighted. Dựa vào số điểm này mà request sẽ được phân phối tới các server. Server nào mà có điểm cao thì sẽ được nhận nhiều request. Nếu server mà đạt đến giới hạn nhận request thì sẽ reject các request nhận vào.



Hình 12: Weighted round robin

- Source IP hash load balacing: là một thuật toán được sử dụng trong network load balacing để phân phối các request đến với một danh sách các server dựa trên giá trị hash của source IP address. Mục đích của thuật toán là đảm bảm rằng những requests đến từ cùng một source IP address sẽ chuyển đến cùng một server.



Hình 13: Ip hash

Ngoài ra cũng còn một số thuật toán dynamic load balacing như là: Least Connection Method Load Balacing(dựa vào số connection đang có), Least Response Time Method Load Balacing (dựa vào thời gian response).

2.2 Lab for Haproxy

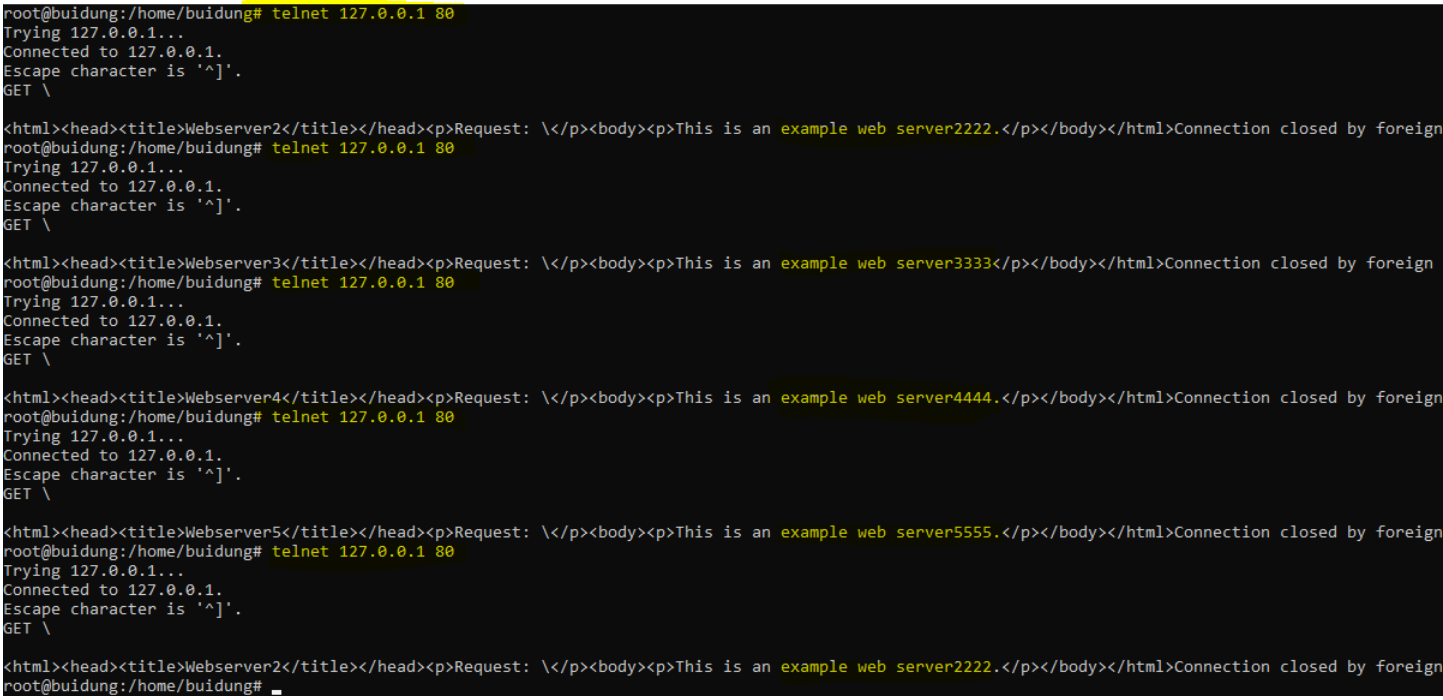
Thực hiện tạo 4 webserver bằng python như sau:

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import time
hostName = "127.0.0.1"
serverPort = 2222
class MyServer(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        self.wfile.write(bytes("<html><head><title>Webserver2</title></head>", "utf-8"))
        self.wfile.write(bytes("<p>Request: %s</p>" % self.path, "utf-8"))
        self.wfile.write(bytes("<body>", "utf-8"))
        self.wfile.write(bytes("<p>This is an example web server2222.</p>", "utf-8"))
        self.wfile.write(bytes("</body></html>", "utf-8"))
if __name__ == "__main__":
    webServer = HTTPServer((hostName, serverPort), MyServer)
    print("Server started http://%s:%s" % (hostName, serverPort))
    try:
        webServer.serve_forever()
    except KeyboardInterrupt:
        pass
    webServer.server_close()
    print("Server stopped.")
```

Những server sau thực hiện tương tự. Sau khi tạo xong 4 server lần lượt chạy các webserver vừa được tạo. Tiếp theo, thực hiện tạo một file config haproxy như sau:

```
frontend http80
  bind 127.0.0.1:80
  timeout client 60s
  default_backend allservers
backend allservers
  timeout connect 10s
  timeout server 100s
  server server2222 127.0.0.1:2222
  server server3333 127.0.0.1:3333
  server server4444 127.0.0.1:4444
  server server5555 127.0.0.1:5555
```

Chạy file config kia bằng cờ -f và thu được kết quả:



Hình 14: Telnet

Thực hiện config ACL cho mỗi URL đến

Ta có file config như sau:

```
frontend http80
  bind 127.0.0.1:80
  mode http
  acl app1 path_end -i /app1
  acl app2 path_end -i /app2
  http-request deny if { path -i -m beg /admin }
  use_backend app1Servers if app1
  use_backend app2Servers if app2
  timeout client 60s
  default_backend allservers
backend app1Servers
  mode http
  timeout connect 10s
  timeout server 10s
  server server2222 127.0.0.1:2222
  server server3333 127.0.0.1:3333
backend app2Servers
  mode http
  timeout connect 10s
  timeout server 10s
  server server4444 127.0.0.1:4444
  server server5555 127.0.0.1:5555
```

```
backend allservers
  mode http
  timeout connect 10s
  timeout server 100s
  server server2222 127.0.0.1:2222
  server server3333 127.0.0.1:3333
  server server4444 127.0.0.1:4444
  server server5555 127.0.0.1:5555
```

Thực hiện chạy file config và xem kết quả:

```
root@buidung:/home/buidung# curl http://127.0.0.1:80/app1
<html><head><title>Webserver2</title></head><p>Request: /app1</p><body><p>This is an examp
root@buidung:/home/buidung# curl http://127.0.0.1:80/app1
<html><head><title>Webserver3</title></head><p>Request: /app1</p><body><p>This is an examp
root@buidung:/home/buidung# curl http://127.0.0.1:80/app2
<html><head><title>Webserver4</title></head><p>Request: /app2</p><body><p>This is an examp
root@buidung:/home/buidung# curl http://127.0.0.1:80/app2
<html><head><title>Webserver5</title></head><p>Request: /app2</p><body><p>This is an examp
root@buidung:/home/buidung# curl http://127.0.0.1:80/app1
<html><head><title>Webserver2</title></head><p>Request: /app1</p><body><p>This is an examp
root@buidung:/home/buidung# cd proxy/
root@buidung:/home/buidung/proxy# nano test.conf
root@buidung:/home/buidung/proxy# curl http://127.0.0.1:80/admin
<html><body><h1>403 Forbidden</h1>
Request forbidden by administrative rules.
</body></html>
```

Hình 15: Kiểm tra kết quả