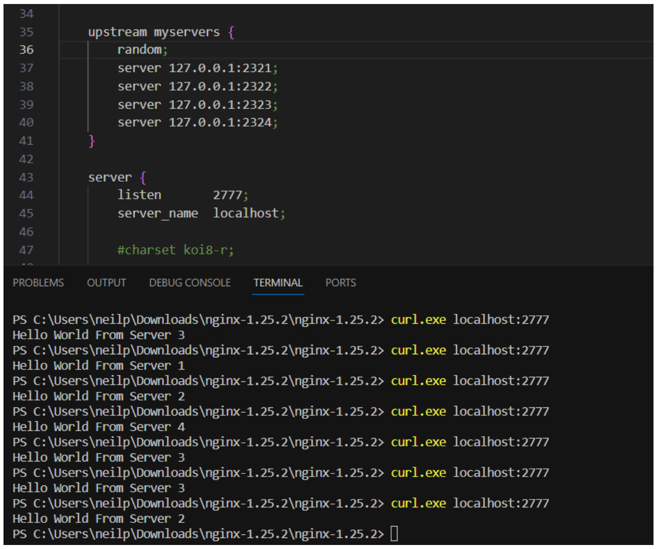
CS 457
Neil Patel

Experimental Report on NGINX Load Balancing

Random:

There are many different load balancing strategies that are supported by Nginx. One of these strategies is Random. With this strategy, the request is passed to any random server. When I implemented this, each server responded in a random order and a random number of times. I also used the least_conn parameter where two servers are randomly selected but the server that gets to respond is the server with the least active number of connections.

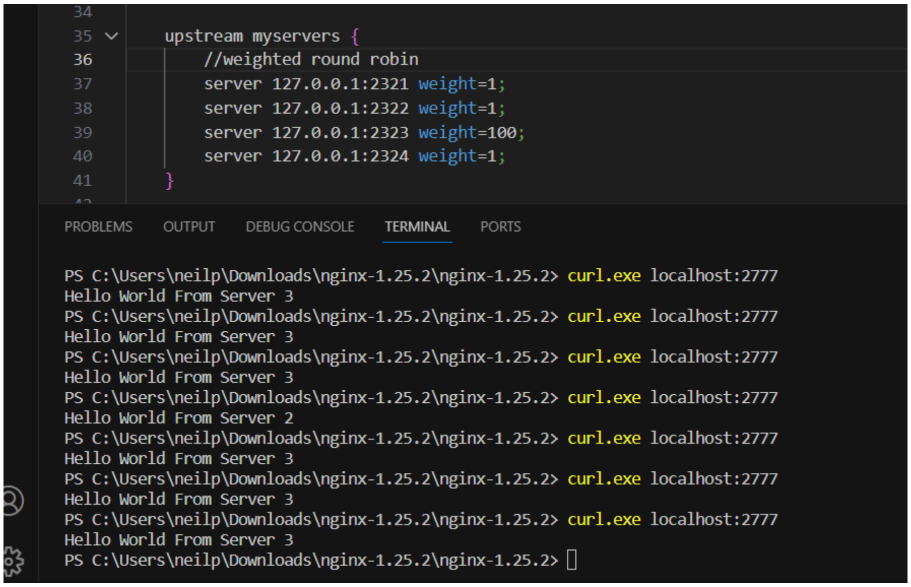


```

upstream myservers {
    random;
    server 127.0.0.1:2321;
    server 127.0.0.1:2322;
    server 127.0.0.1:2323;
    server 127.0.0.1:2324;
}

server {
    listen       2777;
    server_name  localhost;

    #charset koi8-r;
```

```
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 1
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 2
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 4
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 2
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2>
```

Round Robin:

Nginx also supports Round Robin. When implemented, each server responded in sequential order. Weights can be added to servers to make certain servers more likely to respond than others. Of my 4 servers, I gave server 3 a weight of 100 while the others received weights of 1. This led to servers 3 to be called upon more frequently than servers1,2, and 4.

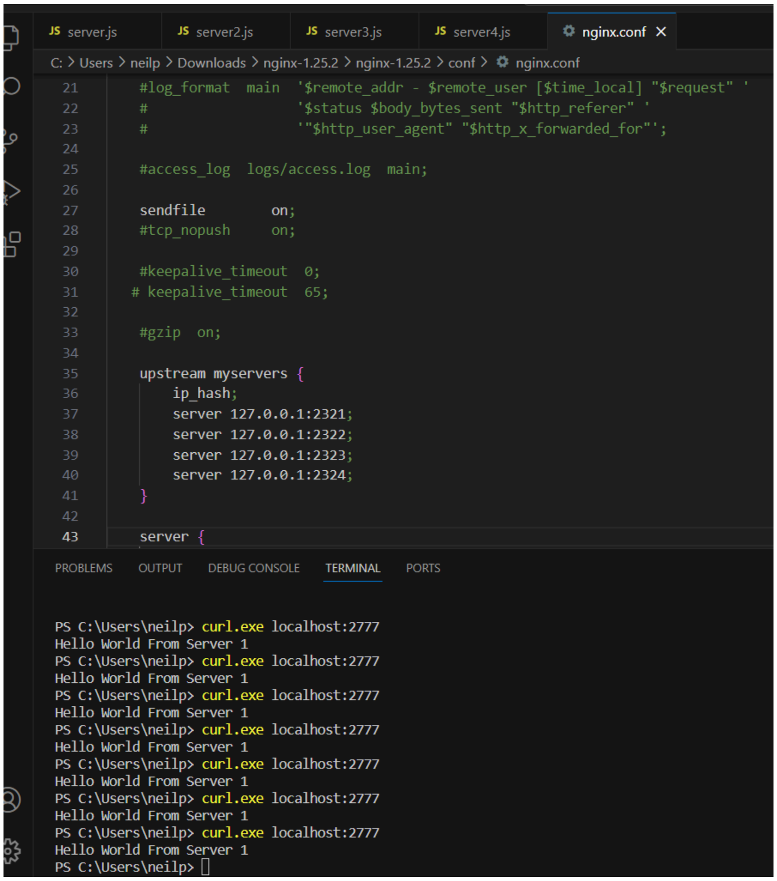


```

upstream myservers {
    //weighted round robin
    server 127.0.0.1:2321 weight=1;
    server 127.0.0.1:2322 weight=1;
    server 127.0.0.1:2323 weight=100;
    server 127.0.0.1:2324 weight=1;
}
```

```
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 2
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp\Downloads\nginx-1.25.2\nginx-1.25.2>
```

IP Hash:

Nginx even supports some more complicated load balancing strategies. One of which is IP Hash. With this method, IP addresses are given a server that it will communicate to, only receiving a different server if the assigned server is not operational. In my experiment, this led to me only having my requests passed to server 1.

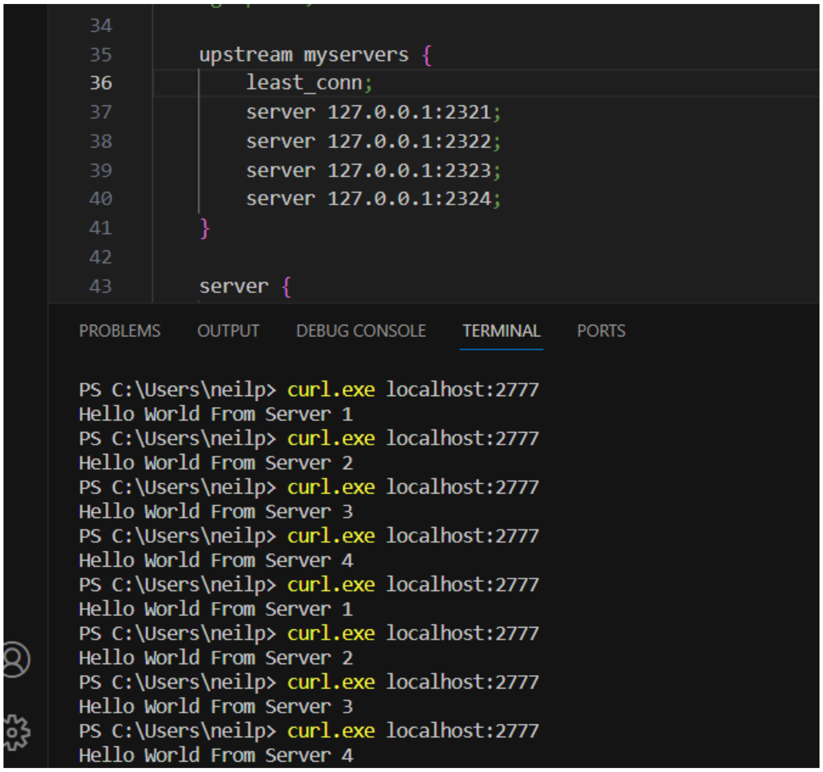


```
    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';

    #access_log  logs/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
   # keepalive_timeout  65;

    #gzip  on;

    upstream myservers {
        ip_hash;
        server 127.0.0.1:2321;
        server 127.0.0.1:2322;
        server 127.0.0.1:2323;
        server 127.0.0.1:2324;
    }

    server {
```

```
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 1
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 1
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 1
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 1
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 1
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 1
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 1
PS C:\Users\neilp> 
```

Least Connections:

Finally, Nginx supports Least Connections load balancing. This strategy seemed to be the most effective to balance loads between servers (of the free strategies). Least Connections operates by passing the request to a server with the lowest number of active connections of the available servers. The fact that the main goal of this strategy is to pass requests to the least active server at the time results in this strategy being effective in balancing the loads between servers to optimize their efficiency. I experimented with this balancing strategy by using curl to send many requests. The requests were usually not responded to by the same servers but to a different arrangement of them, verifying this load balancing technique.



```
upstream myservers {
    least_conn;
    server 127.0.0.1:2321;
    server 127.0.0.1:2322;
    server 127.0.0.1:2323;
    server 127.0.0.1:2324;
}

server {
```

```
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 1
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 2
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 4
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 1
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 2
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 3
PS C:\Users\neilp> curl.exe localhost:2777
Hello World From Server 4
```