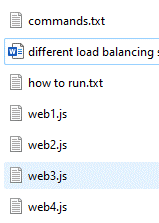
Table of Contents

* Concept.
* Installation.
* Testing files.
* Update configuration.
* The web files node script.
* Configuration, code of each algorithm.
  + Random load balancer.
  + Round Robin load balancer.
  + Weighted Round Robin load balancer.
* Usage of each algorithm (when to use).
  + Usage of random load balancer.
  + Usage of Round Robin load balancer.
  + Usage of Weighted Round Robin load balancer.
* Final notes (important to read).

[NGINX](https://www.nginx.com/resources/glossary/nginx/): is open-source software for web serving, reverse proxying, caching, load balancing, media streaming, and more. It started out as a web server designed for maximum performance and stability. In addition to its HTTP server capabilities, NGINX can also function as a proxy server for email (IMAP, POP3, and SMTP) and a reverse proxy and load balancer for HTTP, TCP, and UDP servers. [1]

[Nodejs:](https://nodejs.org/en/about) As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

figure 1

**installation:**

Download and unzip NGINX. Click [here](https://nginx.org/en/download.html) to download.

Download and Install Nodejs. click [here](https://nodejs.org/en/download) to download.

Testing Nodejs:

* Download the files from this link. click here to download.
* Unzip the folder.
* Make sure you have all these files in fig1.

In commands .txt, change the file paths to the corrected absolute path.  
which is where you installed NGINX and the current folder paths respectively.

A blue and white screen with black text

Description automatically generated

Figure 2

* Open terminal in the current folder and run this command[2]:

cmd < commands.txt

* links CTRL + click the links below to make sure everything works correctly.

<http://localhost/>

<http://localhost:2321/>

<http://localhost:2322/>

<http://localhost:2323/>

<http://localhost:2324/>

* Update the NGINX configuration file with the one provided by the link.
* ReRun cmd < commands.txt in cmd.

notice in the upgraded version of the NGINX configuration in fig2: defines an upstream group named myservers in NGINX. With random load balancing “NGINX will distribute incoming requests randomly among the listed servers”.

This way, each request will be sent to one of the servers in the upstream group in a random manner. useful when the load balancer receives many requests and needs to distribute them evenly to the nodes.

Web#.Js code. (Node.js script that creating the HTTP server)

A computer code with many black text

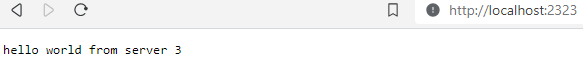
Description automatically generated with medium confidence

the server will listen for incoming requests only on the local machine, and the server will be accessible at <http://127.0.0.1> ports 2321, 2322, 2323, 2324  
  
you can access <http://127.0.0.1> usually also including the port <http://127.0.0.1:2321>, but depending on your load balancing algorithm you may not be able to access the port you want through the browser.

Further in the code, the server will print which server handled the request and append each request served separately in the same file(allservers.log). This is only for testing our load balancing algorithms.

Replace the nginx .config file with one in this link.

Till now you have completely fulfilled the random load balancing tech. now lets move to round robin.



What you should see on browser

A screenshot of a computer program

Description automatically generatedwhat you should see in cmd

Note: start /B node web1.js: will start the server in the background. “keeps cmd running”.When using “start /B” please make sure to stop NGINX when you are done using this command: nginx -s stop

A screenshot of a computer program

Description automatically generated

When trying using a used port.

Round Robin:

Open the config file and update the upstream myservers group:

upstream myservers {  
 server 127.0.0.1:2321 weight=1;  
 server 127.0.0.1:2322 weight=1;  
 server 127.0.0.1:2323 weight=1;  
 server 127.0.0.1:2324 weight=1;  
 }

on cmd run “cmd < commands.txt” command. You will get the messege on the right that the ports are being used. What to do?

A screenshot of a computer program

Description automatically generated

Reopen cmd. Navigate back to your web folder using cd command. Rerun “cmd < commands.txt”.

Configuring random load balancing. In the nginx config file.  
A screenshot of a computer program

Description automatically generated

Configuring round robin load balancing. In the nginx config file.

A number of numbers and letters

Description automatically generated with medium confidence

Configuring weighted round robin load balancing. In the nginx config file.

A screenshot of a computer code

Description automatically generated

- What is the difference between these algorithms?

- When to use random?

- When to use round robin?

- When to use weighted round robin?  
  
lets test each of them, and see who will serve a number of requests.

We will make requests using cmd and curl.

Type into cmd the below command for sending 10 requests at the same time then check the log file.

for /l %i in (1,1,10) do curl -I http://localhost:2777

if you don’t want to try it yourself, I have included the output in the screen shots, now lets discuss the output of the log file.

Random log file:

A screenshot of a computer

Description automatically generated

Round robin log file:

A screen shot of a computer

Description automatically generated

Weighted round robin:

A screenshot of a computer screen

Description automatically generated

Notice that random is completely random, while the round robin is circular request distribution evenly among servers, while weighted round robin will distribute the requests using the weight value (as the one with more weight will serve more requests).

We talked about the random and when to use it. What about the other two?

Will, you can see that from the round robin, if all the servers have the same capability of serving requests it will be used. But if for our example:

Server 1 serves 2 requests per milli second.  
Server 2 serves 4 requests per milli second.  
Server 3 serves 3 requests per milli second.  
Server 4 serves 1 request per milli second.

Then we will need a good load balancing algorithm to finish all the requests in the least amount of time.

If it happened that we used unweghted round robin algorithem for such case this will take 4 milli seconds to serve all requests but if we used weighted round robin instead, it will take 1 milli seconds to serve all requests. Which is 4 times the performance. That is how powerful a good and a fitting load balancer algorithm is.

Final notes:

* If the command cmd < commands.txt does not work for you then please copy paste the file contents on cmd and press enter (or paste it line by line).
* Use start /B instead of running a terminal per port for the server as this command will keep the port running in the background and you still can use the same terminal again instead of opening a new cmd terminal.
* This code is used on Windows OS version 11.
* Use the command for /l %i in (1,1,10) do curl -I <http://localhost:2777> for sending 10 requists in a single command instead of opening browser and keeping refreshing.
* Use my log code to log the served requests and by who served the requests instead of menually doing it.
* Click on this link to see the code.