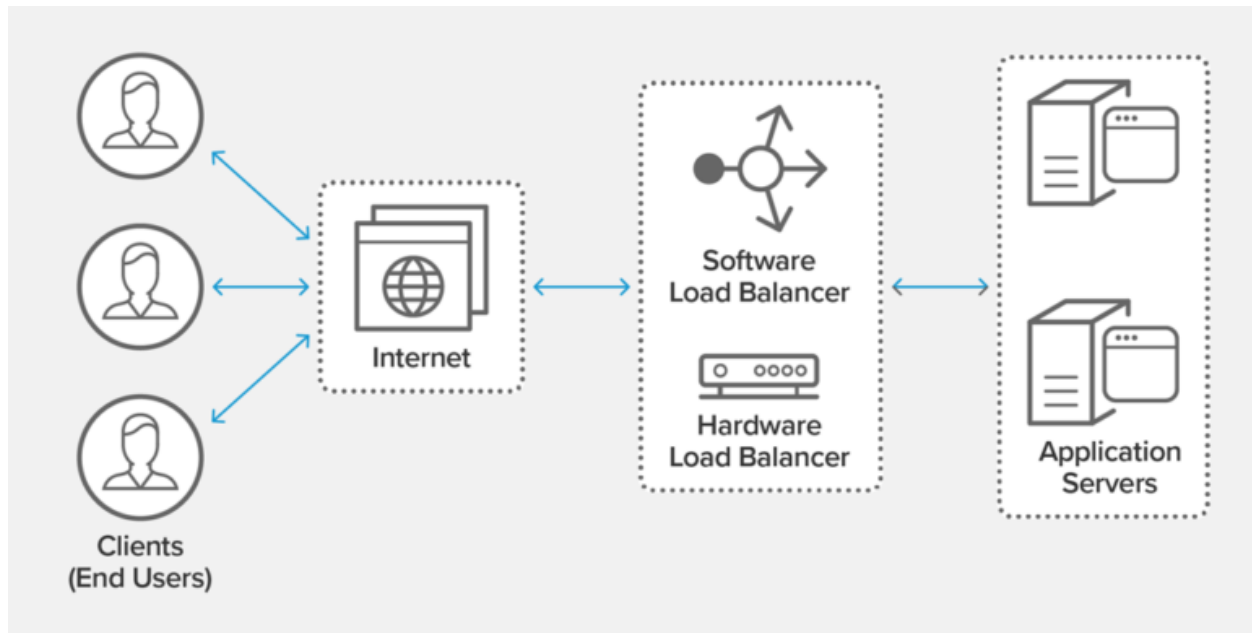


# Load Balancer

## Round Robin Method



### Introduction

Load balancers monitor the information flow between a server and a client computer (PC, laptop, tablet or smartphone). On-premises, in a data centre, or in the public cloud, the server may be. Physical or virtualized servers are also possible. The load balancer assists servers in efficiently moving data, optimising the use of application delivery resources, and avoiding server overloads.

### Load Balancing

Load balancing is the process of evenly distributing incoming network traffic among a group of backend servers, often referred to as a server farm or server pool.

Modern high-traffic websites must handle hundreds of thousands, if not millions, of concurrent user or client requests while returning accurate text, images, video, or application data in a timely and consistent manner. In order to save money, scale to

---

meet these high volumes, modern computing best practice generally requires adding more servers.

A load balancer sits in front of your servers, acting as a "traffic cop," routing client requests through all servers capable of fulfilling those requests in a way that maximises speed and power usage while ensuring that no single server is overworked, potentially degrading performance. The load balancer redirects traffic to the remaining online servers if a single server goes down. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

In this manner, a load balancer performs the following functions:

- Distributes client requests or network load efficiently across multiple servers
- Ensures high availability and reliability by sending requests only to servers that are online
- Provides the flexibility to add or subtract servers as demand dictates

## **Load Balancing Algorithms**

Different load balancing algorithms have different benefits; the load balancing approach you choose is determined by your requirements:

- **Round Robin** – A basic method of load balancing servers is used in this technique. To provide the same services, several identical servers are used. They all have the same internet domain name, but they all have different IP addresses. A DNS server keeps track of all IP addresses and their domain names. When a request is made for the IP address associated with a domain name, the address IP address associated with the domain name is received, the address is returned in a rotating sequential manner.
- **Least Connections** – A new request is sent to the server that has the fewest active client connections. Each server's relative processing ability is taken into account when deciding which one has the fewest connections.

- 
- **Least Time** – Sends requests to the server selected by a formula that combines the fastest response time and fewest active connections. Exclusive to NGINX Plus.
  - **Hash** – Requests are distributed according to a key you specify, such as the client IP address or the request URL. If the collection of upstream servers changes, NGINX Plus will add a consistent hash to reduce load redistribution.
  - **IP Hash** – The IP address of the client is used to determine which server receives the request.
  - **Random with Two Choices** – The Least Connections algorithm selects two servers at random and sends the request to the one that is chosen (or for NGINX Plus the Least Time algorithm, if so configured).

## Session Persistence

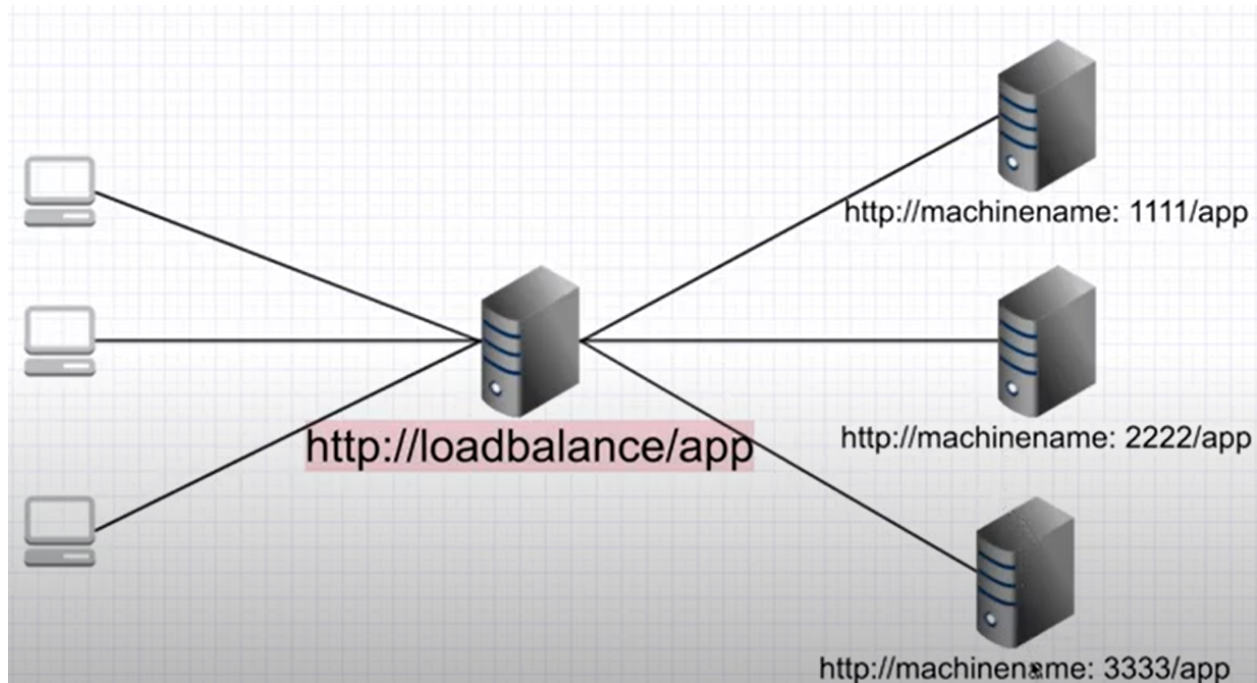
The browser often saves information about a user's session locally. In a shopping cart programme, for example, objects in a user's cart can be stored at the browser level before the user is ready to buy them. Changing which server the client sends requests to in the middle of a shopping session will result in performance problems or transaction failure. It is appropriate in certain circumstances that all requests from a client are sent to the same server for the duration of the session. This is known as *session persistence*. Session persistence is handled by the best load balancers. Another scenario in which session persistence is used is where an upstream server caches information requested by a user to improve performance. When switching servers, the information will have to be fetched a second time, resulting in output inefficiencies.

## NGINX

NGINX is an open source web server, reverse proxy, caching, load balancing, video streaming, and other applications. It began as a web server with the aim of providing maximum efficiency and stability. NGINX can also act as an email proxy server (IMAP, POP3, and SMTP), as well as a reverse proxy and load balancer for HTTP, TCP, and UDP clients, in addition to being an HTTP server.

---

## Round Robin Load balancer using Nginx



### Creating http web application:

We have created a web application using python which shows us the process id from where it is served from. We will be running this application in different ports.

### Code:

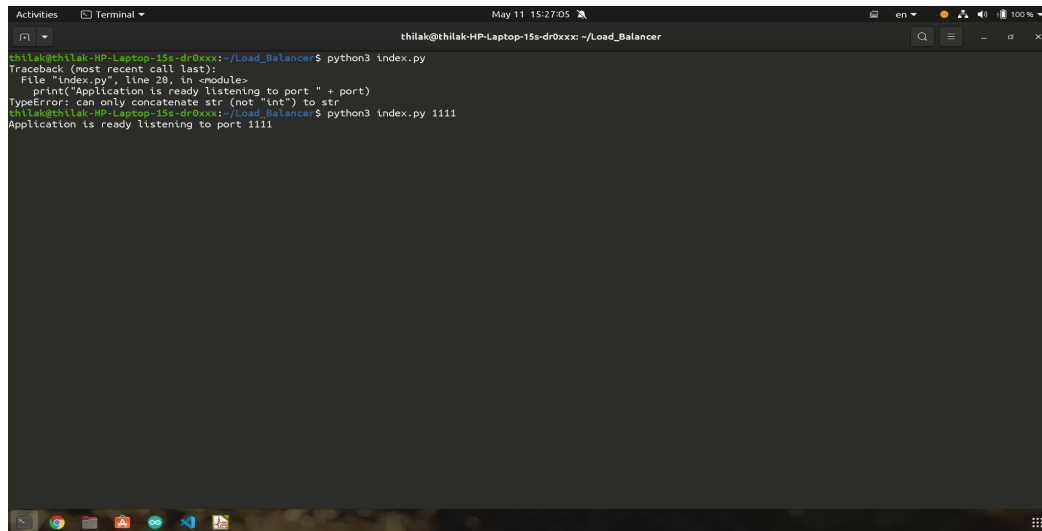
```
import tornado.web
import tornado.ioloop
import sys
import os

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("served from " + str(os.getpid()))

if __name__ == "__main__":
    app = tornado.web.Application([
        (r"/basic", MainHandler),
    ])
    port=8882
```

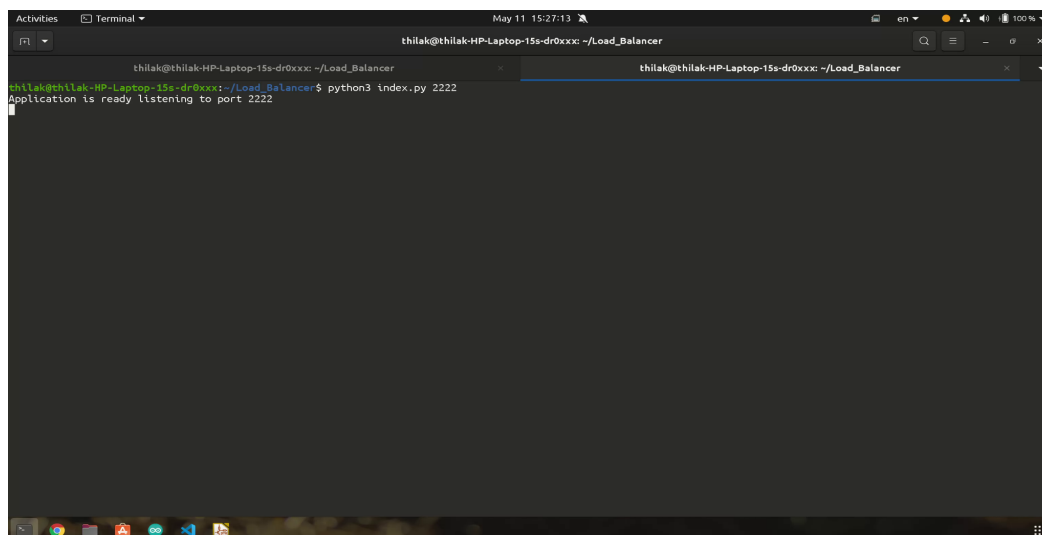
```
if (sys.argv.__len__() > 1):  
    port = sys.argv[1]  
  
app.listen(port)  
print("Application is ready listening to " + port)  
tornado.ioloop.IOLoop.current().start()
```

**application taking input from port 1111,2222:**



A terminal window titled 'thilak@thilak-HP-Laptop-15s-dr0xxx: ~/Load\_Balancer' showing the execution of a Python script. The first command is 'python3 index.py', which results in a traceback error: 'TypeError: can only concatenate str (not "int") to str' at line 20. The second command is 'python3 index.py 1111', which successfully outputs 'Application is ready listening to port 1111'.

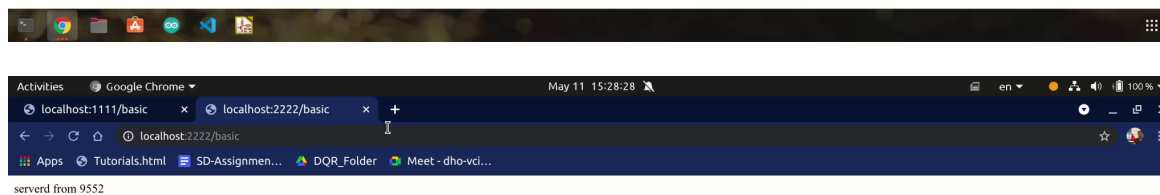
```
thilak@thilak-HP-Laptop-15s-dr0xxx:~/Load_Balancer$ python3 index.py  
Traceback (most recent call last):  
  File "index.py", line 20, in <module>  
    print("Application is ready listening to port " + port)  
TypeError: can only concatenate str (not "int") to str  
thilak@thilak-HP-Laptop-15s-dr0xxx:~/Load_Balancer$ python3 index.py 1111  
Application is ready listening to port 1111
```



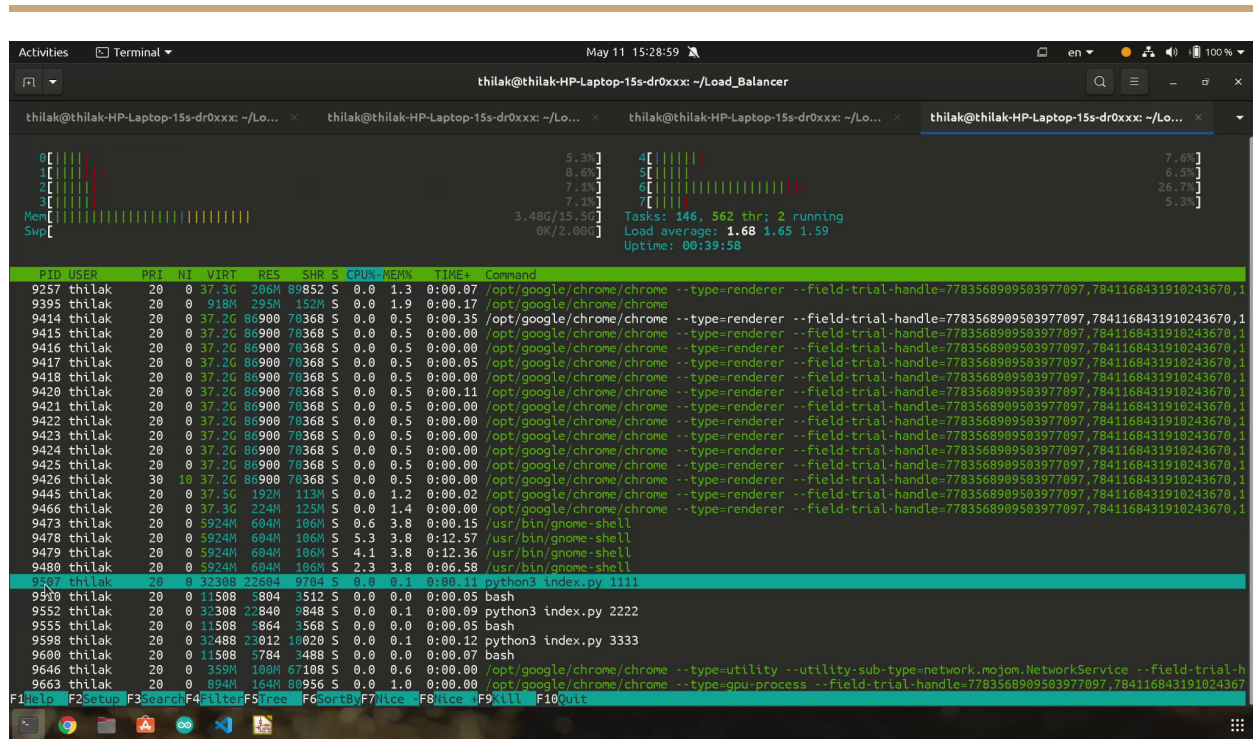
A terminal window titled 'thilak@thilak-HP-Laptop-15s-dr0xxx: ~/Load\_Balancer' showing the execution of the Python script with the command 'python3 index.py 2222'. The output is 'Application is ready listening to port 2222'.

```
thilak@thilak-HP-Laptop-15s-dr0xxx:~/Load_Balancer$ python3 index.py 2222  
Application is ready listening to port 2222
```

**Output:**



Here we have run the application in two different ports simultaneously which will act as servers and access them in chrome it is showing the process id served from and by refreshing the site it will not change the process id which means not changing the server using load balancer will be distributing the load across the server.



## Creating configuration file with our own servers:

Here we created 3 servers on the same system with 3 different ports. By replacing the localhost with machine name we can create a load balancer for any http web application.

```
upstream pythonweb {

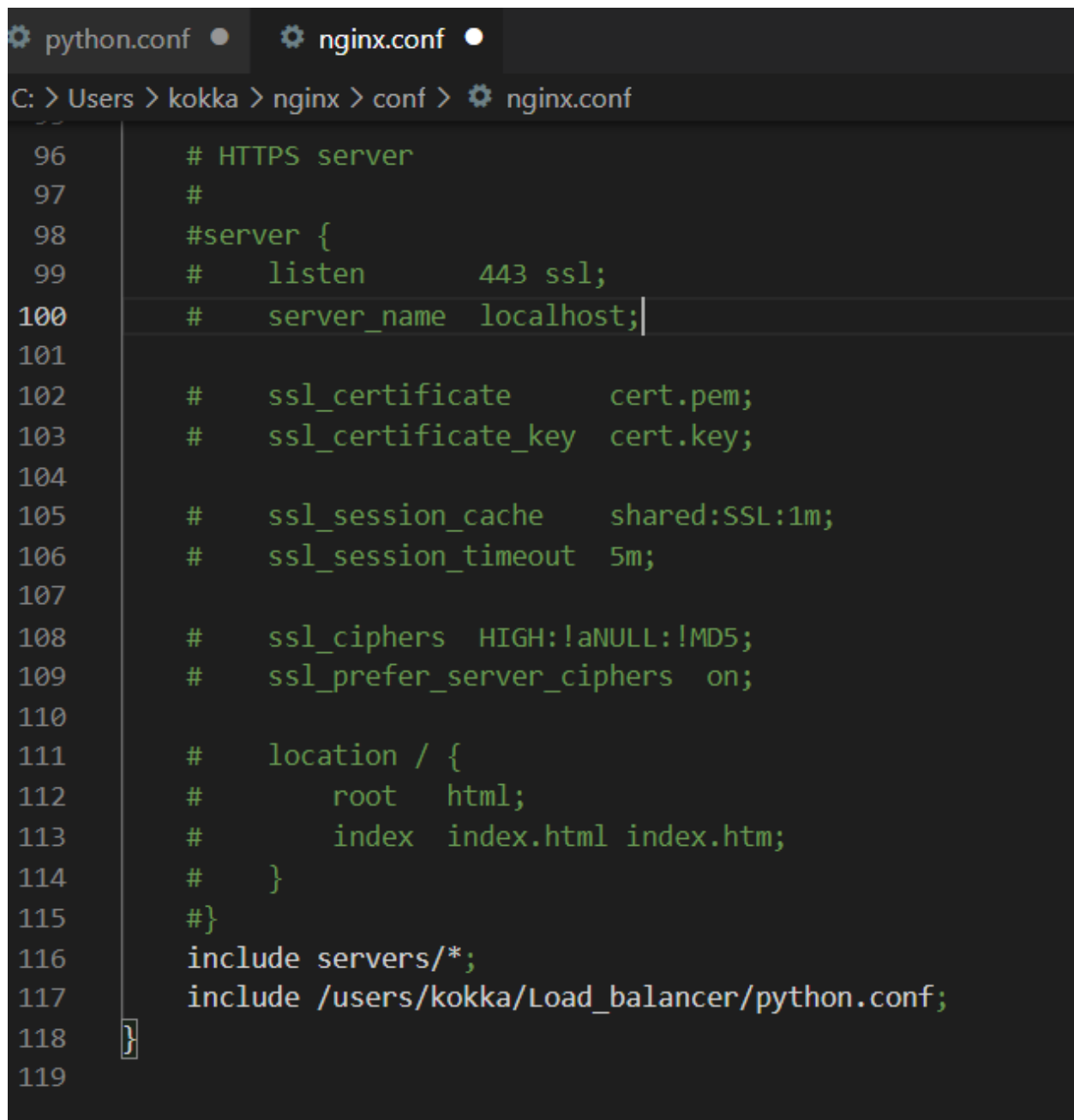
    server localhost:1111;
    server localhost:2222;
    server localhost:3333;
}

server {

    listen 80;

    location /basic {
        proxy_pass "http://pythonweb/basic";
    }
}
```

## Adding these servers to the nginx server:



```
python.conf ● nginx.conf ●
C: > Users > kokka > nginx > conf > nginx.conf

96 # HTTPS server
97 #
98 #server {
99 #     listen      443 ssl;
100 #     server_name  localhost;
101
102 #     ssl_certificate      cert.pem;
103 #     ssl_certificate_key  cert.key;
104
105 #     ssl_session_cache    shared:SSL:1m;
106 #     ssl_session_timeout  5m;
107
108 #     ssl_ciphers  HIGH:!aNULL:!MD5;
109 #     ssl_prefer_server_ciphers  on;
110
111 #     location / {
112 #         root      html;
113 #         index  index.html index.htm;
114 #     }
115 #}
116 include servers/*;
117 include /users/kokka/Load_balancer/python.conf;
118 }
119
```

So, I have changed the nginx configuration file by adding our own servers for the application using the configuration file we have created.



---

## Output:

**Accessing the application basic three times when the three servers are active and listening to ports 1111,2222,3333.**

### First time:



### Second time:



### Third time:



We can see here every time we refresh the site it will be served from a different server as we have done with the round robin method. It will be served from the next server and it will continue the loop irrespective of the connections to the server.

---

## **Conclusion**

In this report we explained the process of balancing the load of the servers using nginx by changing its configuration and adding the servers to it and here we have used the round robin algorithm for load balancing.