222026556 - B Leech

Data Communications PM2 - 2025/04/15

# Data Communications: Linux-based, load-balanced Web application

## Project overview

### Problem statement

Single-server web applications that experience a lot of traffic regularly have response-time issues or crashes due to overload and a single point of failure, leading to website downtime, which is terrible, especially for e-commerce sites. Down-time breaks clients' trust, as the website will be seen as unreliable.

### Solution

The solution involves creating a simple web application with multiple servers, with a load balancer in the middle that will distribute the requests evenly amongst the servers, as well as the potential integration of a cloud service for backup and to scale the infrastructure. This will improve the system's availability, reliability, and performance. Load balancing ensures that incoming requests are evenly distributed, and multiple nodes handle the request load in parallel to ensure no server is overloaded while another is sleeping, improving the application's response time and throughput.

### Key benefits of this system

1. High availability: Continuous operation during server failures
2. Improved performance: Parallel request processing
3. Scalability: Easily add nodes during increased traffic
4. Fault tolerance: Automatic failover mechanisms
5. Reduced downtime: Maintenance without service interruption

## System architecture

- Nodes 1, 2, and 3 will be the web application and three servers running in a Docker container. At least two servers must run to demonstrate the project. A mini web application will be created. Potentially, an FTP to store files on a database, to be able to download, or some base functionality to demonstrate the project. All the servers will be configured exactly the same, as the servers on load balancing needs to be the same, so that users cant notice a difference, however for this project, each server will have a different primary color for demonstration purposes, functionality will be the same throughout.
- Node 4 will be the load balancer, distributing the incoming traffic across the application servers

using algorithms such as Round Robin, least connection, IP hashing, or weighted loading. The load balancer will run in VirtualBox.
- Node 5 will be the database server storing the application's data. It will be a MySQL Database running in VirtualBox.
- Node 6 will be a cache server to cache frequently accessed data to improve performance and reduce the load on servers, e.g., Redis or Memcached.
- Node 7 will be the Cloud integration to host parts of the infrastructure and improve scalability, as well as for potential backups. E.g., AWS EC2 or RDS.

### Node diagram

Below is a self-generated diagram using Mermaid to demonstrate how the nodes communicate visually. However, this diagram excludes the cache server and the cloud integration.

### Technical specifications

| Node | Role | Technology Stack | Hosting Method | Purpose |
|---|---|---|---|---|
| 1 - 3 | Application Servers | NodeJS | Docker Container | Host Identical web apps |
| 4 | Load Balancer | Nginx | VirtualBox | Distribute web traffic |
| 5 | Database | MySQL | VirtualBox | Central data storage |

# Implementation guide

## 1. Core Setup

### 1.1 App Servers (Node 1 - 3)

Created a basic Express web server using NodeJS. The server is copied 3 times to have three servers running on three different ports. Ideally, the servers have to be configured exactly the same, as the user should not be able to notice that there are other servers. For demonstration purposes, each server will have a different color to make it visually clear that the servers are different.

| Server | Color |
|---|---|
| Server 1 | Red |
| Server 2 | Blue |
| Server 3 | Yellow |

Each server must run on a different port and display the same frontend hence only one frontend file is created. The front end can be any UI that will communicate with the database to demonstrate the connections; in this case, a simple registration and login page for user management and a files page to upload, download, and manage files. FTP integration is used to upload and download files using a user authentication system.

Down below is a figure illustrating the structure of how my servers are set up, along with the front end. When compiling the docker files, you must make sure to copy the frontend files in as well and mention them in all the docker files. I had this issue with compiling, and the servers crashed due to the inability to locate the front-end components to send in a response.

A docker file is compiled to run these files. In the main folder is a **`docker-compose.yml`** file with the configurations

In the terminal run **`docker-compose up -d`** to start the servers and

You can then navigate to a browser and locate the server URL to view it. E.g. localhost:3000.

Use **`docker-compose down`** to stop the servers.

### 1.2 Load Balancer

As stated, the load balancer runs on VirtualBox on an Ubuntu server. To set up Nginx, you need to follow the following steps.

1. Install the balancer **`sudo apt install nginx`**
2. **`sudo systemctl status nginx`** (To view status and confirm installation)
3. To start and run nginx, use the following commands. **`sudo systemctl start nginx`**, **`sudo systemctl enable nginx`**.
4. Next, it has to be configured; open the config file **`sudo nano /etc/nginx/nginx.conf`** Ideal is to create a copy of the file first to be able to restore if a mistake is made, as undo is not an option.
5. Add the following information to configure it

```
http {
    upstream app_servers {
		# least_conn;
        List all servers with their addresses.
        e.g. "server 127.0.0.1:3000"
    }
    server {
        listen 80;
        location / {
            proxy_pass http://app_servers;
			proxy_set_header Host $host;
			proxy_set_header X-Real-IP $remote_addr;
			proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        }
    }
}
```

1. Test the configuration file. It will say if there are any syntax errors **`sudo nginx -t`**
2. If the test is successful, restart nginx to start using it. **`sudo systemctl restart nginx`**

* If localhost for nginx does not work, use the device where the servers run on its IP Address.

Additionally, the load balancer stats can be viewed by adding the following code to the Nginx config file: below the location of the previously added code, add another location. Reload Nginx after adding this section. **`sudo systemctl reload nginx`**

```
location /nginx_status {
    stub_status on;
    allow 127.0.0.1 #if it doesn't work, use the server IP address
    deny all; # security to only allow this IP address range
}
```

To access the stats, ping the address in the terminal or browser. For example, in the terminal **`curl 127.0.0.1/nginx_status`** or in the browser **`127.0.0.1/nginx_status`**, the output will look like this.

```
Active connections: 3
server accepts handled requests
 36 36 68
Reading: 0 Writing: 3 Waiting: 0
```

To get the IP address in the console, install network diagnostic tools for any network operations on Ubuntu. **`sudo apt install net-tools`**. Then, run **`ifconfig`** to access the IP address.

#### Load balancing algorithms

Different algorithms or methods exist for the configuration of the load balancer. This solution uses the default Round Robin algorithm. Each algorithm has different outcomes and performance, as displayed below. It's up to the developer to choose which algorithm is best for their system. They will want the best possible method if the system has extreme traffic. Below is a table summarizing the different algorithms that can be used. To add it to Nginx, add the command shown in the table below, above the server's declaration, like the comment in the code snippet for the config file above. Except for the weighted algorithm, it gets added after the server declaration by giving the server a weight value.

| Algorithm | Command | Best used for |
|---|---|---|
| Round Robin | (default algorithm) | General use |
| Least connections | least_conn; | Uneven traffic loads |
| IP Hash | ip_hash; | Session persistence |
| Weighted | server … weight=X | Strong servers |

#### Potential troubleshooting for Nginx

- If the connection is refused, check the firewall to ensure it doesn't block the ports or IP address **`sudo ufw status`**. and verify that the system is running **`sudo systemctl status nginx`**.
- If Nginx is not Balancing the load. Ensure the configured servers are all running and check the config file for errors **`sudo nginx -t`**.

### 1.3 Database

After the setup of another Ubuntu server, the following steps can be followed to install the database.

#### Installation

1. First, constantly update the system. **`sudo apt update -y`**.
2. Install mySQL Server **`sudo apt install mysql-server`**.
3. For additional security, install mysql_secure_installation. This sets stringer security configurations, such as strong password requirements. **`sudo mysql_secure_installation`**. You can then decide which security to implement by following the prompts and setting the root password.
4. Start the Database server **`sudo systemctl start mysql`**, **`sudo systemctl enable mysql`**.

#### Setup

After starting the database, you can now log in and create your database **`mysql -u “username” -p`** enter and then type in password. If the password is correct but doesn't work, add sudo in front to raise privileges. If you did not set a password, use **`sudo mysql -u root`** and enter. Login must be successful on 1 of these attempts.

After successful login, if no password was set, set a password for security purposes.

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY "set-password";

FLUSH PRIVILEGES;
```

Create the database **`CREATE DATABASE db_name;`**

It is recommended that a new user be created, as one should not use the root account, and you can set permissions for that user. So, this new user will have all privileges for only the created database (use load_storage as an example).

```
CREATE USER 'user'@'%' IDENTIFIED BY "set-password";
GRANT ALL PRIVILEGES ON load_storage.* TO 'user'@'%';
FLUSH PRIVILEGES;
```

Lastly, create your database structure and tables using regular MySQL syntax that suits the database needed for your functionality. Once again, mine is file storage with user access. Example of one table structure.

```

CREATE TABLE USERS(
    user_id INT AUTO _INCREMENT,
    password VARCHAR(255) NOT NULL,
    user_email VARCHAR(150) NOT NULL UNIQUE,
    full_name VARCHAR(100) NOT NULL,
    PRIMARY KEY (user_id)
);
```

After tables have been created, the database should be ready to go.

**Potential troubleshooting for Nginx**

- If the connection is refused, check the firewall to ensure it doesn't block the ports or IP address **`sudo ufw status`**. verify that the system is running **`sudo systemctl status nginx`**.
- If you struggle to connect to the database, set the bind address to 0.0.0.0 in the config file and restart MySQL. However, this can be a security risk. **`sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf`**
- If you can't connect to the database, ensure the user has remote access privileges.

### 2. Security Implementation

## Maintenance

## Bibliography

1. Ghomi, E.J., Rahmani, A.M. and Qader, N.N., 2017. Load-balancing algorithms in cloud computing: A survey. Journal of Network and Computer Applications, 88, pp.50-71. (Accessed: 20 March 2025).
2. Kumar, P. and Kumar, R., 2019. Issues and challenges of load balancing techniques in cloud computing: A survey. ACM computing surveys (CSUR), 51(6), pp.1-35. (Accessed: 20 March 2025).
3. Mvula, M., 2024. The importance of Load balancing and its impact on system performance. Available at: https://www.linkedin.com/pulse/importance-load-balancing-its-impact-system-musenga-mvula-kpd1f/ (Accessed: 23 March 2025).
4. Neural Codex, 2025. What is load balancing? Why is it needed? Real-world examples. Available at: https://neuralcodex.com/what-is-load-balancing-why-is-it-needed-real-world-examples/ (Accessed: 23 March 2025).