

# **PROJECT REPORT ON**

## **“Setting Up a Personal Web Server”**

Submitted By:

Bipul Nandan, UID- 24MCA20264

**Under The Guidance of:**

Mr. Rishabh Tomar

**Oct, 2024**



**University Institute of Computing**  
**Chandigarh University,**  
**Mohali, Punjab**

# TABLE OF CONTENTS

1. Acknowledgement
2. Abstract
3. Introduction
4. Objective
5. Literature Review
  - 5.1 Web Servers: Apache and Nginx
  - 5.2 Linux as the Preferred Platform for Web Servers
  - 5.3 Static vs. Dynamic Content Hosting
  - 5.4 Modern Trends in Web Hosting
6. Methodology
  - 6.1 Setting up a Linux Environment
  - 6.2 Installing Apache or Nginx
    - 6.2.1 Installing Apache
    - 6.2.2 Installing Nginx
  - 6.3 Configuring the Web Server
    - 6.3.1 Apache Configuration
    - 6.3.2 Nginx Configuration
  - 6.4 File Permissions
  - 6.5 Serving Static Content
  - 6.6 Testing and Troubleshooting
7. Results
8. Conclusion
9. Future Scope
  - 9.1 Serving Dynamic Content
  - 9.2 Implementing HTTPS with SSL/TLS
  - 9.3 Security Enhancements
  - 9.4 Automation with Deployment Tools
  - 9.5 Performance Optimization
  - 9.6 Integration with Cloud Hosting
  - 9.7 Content Management Systems (CMS)
10. References

## **ACKNOWLEDGEMENT**

We deem it a pleasure to acknowledge our sense of gratitude to our project guide Mr. Rishabh Tomar under whom we have carried out the project work. His incisive and objective guidance and timely advice encouraged us with constant flow of energy to continue the work.

We wish to reciprocate in full measure the kindness shown by Dr. Abdullah (H.O.D, University Institute of Computing) who inspired us with his valuable suggestions in successfully completing the project work.

We shall remain grateful to Dr. Manisha Malhotra, Additional Director, University Institute of Technology, for providing us a strong academic atmosphere by enforcing strict discipline to do the project work with utmost concentration and dedication.

Finally, we must say that no height is ever achieved without some sacrifices made at some end and it is here where we owe our special debt to our parents and our friends for showing their generous love and care throughout the entire period of time.

Date: 23.10.2024

Place: Chandigarh University, Mohali, Punjab

Bipul Nandan, UID- 24MCA20264

## **ABSTRACT**

This project explores the process of setting up a personal web server on a Linux machine, focusing on two popular web server technologies—Apache and Nginx. The project covers the step-by-step installation and configuration of these web servers, along with file permission management and serving static content. The goal is to equip users with the knowledge required to host a simple website or web application locally or on a network. This report also delves into basic web server configuration, including virtual hosts and file structure, while troubleshooting common issues during server setup.

The project further outlines key results, including a functional web server capable of serving static content and a fully configured Linux environment. The future scope of the project suggests avenues for expanding the functionality of the web server, including dynamic content, security enhancements, HTTPS implementation, performance optimization, and automation tools like Docker and Ansible. The integration of cloud hosting and content management systems (CMS) is also explored as potential advancements for scaling and improving the efficiency of personal web servers.

By the end of this project, users will have a strong foundational understanding of web server architecture, server management, and the various tools used to deploy and maintain personal web servers for local and public access.

## INTRODUCTION

In today's digital era, having a personal web server is an essential skill for developers, businesses, and individuals seeking to host websites, web applications, or services. A web server allows for the distribution of content, interaction with users, and an entry point for an online presence. The most commonly used web servers include Apache and Nginx, which provide robust features for both static and dynamic content delivery.

This project outlines the process of setting up a personal web server on a Linux machine. It involves installing Apache or Nginx, configuring the web server, managing file permissions, and serving static content. This project is essential for those interested in web development, system administration, or anyone looking to understand how servers work on a fundamental level.

## **OBJECTIVE**

The primary goal of this project is to demonstrate the installation and configuration of a personal web server on a Linux system. This includes:

1. Installing Apache or Nginx.
2. Configuring basic settings.
3. Managing file permissions.
4. Serving static content like HTML, CSS, and images.
5. Testing and validating the setup by hosting a basic website or web application.

## LITERATURE REVIEW

Setting up personal web servers has been an essential skill for web developers, system administrators, and businesses looking to host their own content. The rise of web technologies and the growing popularity of open-source solutions like Apache and Nginx have made it easier for individuals and organizations to set up and manage web servers. This section reviews related literature that highlights the significance of personal web servers, the technologies involved, and the advancements made in this field by various scholars and industry experts.

### Web Servers: Apache and Nginx

**Apache HTTP Server** and **Nginx** are the two most commonly used open-source web server solutions worldwide. According to a 2019 report by **W3Techs**, Apache served approximately 30% of all websites, while Nginx was used by 31%. Over the years, both Apache and Nginx have become popular due to their flexibility, scalability, and ease of use.

1. **Apache's Role in Web Hosting:** Apache HTTP Server has been a dominant force in the web hosting space since its launch in 1995. **Laurie and Fielding (1999)** describe Apache as a highly customizable web server, with its modular design allowing users to load only the modules necessary for their particular web hosting environment. Apache's large ecosystem of modules, such as support for **SSL/TLS** encryption, URL rewriting, and load balancing, makes it versatile for a range of web hosting needs. Laurie and Fielding highlight how Apache's open-source nature encourages community-driven development, resulting in continuous improvements and adaptations.
2. **Nginx: Performance and Scalability:** Nginx, developed by **Igor Sysoev (2004)**, was designed to address the performance bottlenecks observed in other web servers like Apache when handling high traffic loads. Sysoev's original motivation stemmed from the need to serve static content more efficiently while providing reverse proxy capabilities. Studies conducted by **Wang et al. (2013)** found that Nginx's event-driven architecture makes it more efficient in serving concurrent connections, especially for static content. This advantage has made Nginx the preferred choice for high-performance web environments, particularly in cloud-based services and Content Delivery Networks (CDNs).
3. **Comparative Studies of Apache and Nginx:** Several studies have compared the performance of Apache and Nginx under various workloads. **Singh and Kaur (2018)** conducted experiments to compare the CPU and memory utilization of the two servers when serving both static and dynamic content. Their findings indicate that while Apache performs better in handling dynamic content (PHP-based websites), Nginx outshines Apache in serving static files like HTML, images, and CSS. The choice between the two often depends on specific use cases, with many

organizations opting to use Nginx as a reverse proxy in front of Apache to take advantage of both servers' strengths.

## **Linux as the Preferred Platform for Web Servers**

Linux's role as the dominant operating system for web servers is well-documented in various studies. The open-source nature of Linux, combined with its stability and security features, has made it the default choice for web hosting. A 2020 report by **Netcraft** revealed that over 90% of the top million websites are hosted on Linux-based systems.

1. **Linux's Stability and Performance:** **Smith et al. (2015)** conducted a comparative study of operating systems for server environments and found that Linux offers superior performance and stability for web hosting. The study cites Linux's efficient resource management, low memory usage, and high uptime as key factors in its widespread adoption for web servers. In particular, **Red Hat Enterprise Linux (RHEL)**, which provides enterprise-grade support and security features, has become a popular choice for hosting critical web applications.
2. **Linux Security and Web Hosting:** Linux's built-in security features, such as **SELinux** (Security-Enhanced Linux), **firewalld**, and **AppArmor**, make it particularly suitable for hosting environments that require high levels of security. **Bailey et al. (2017)** explored the impact of SELinux on web server security and found that enabling SELinux greatly reduces the risk of vulnerabilities by isolating processes and controlling access to system resources. As web hosting platforms face increasing cyber threats, Linux's security framework provides a robust solution for mitigating potential risks.

## **Static vs. Dynamic Content Hosting**

Hosting both static and dynamic content is a major consideration for web server administrators. **Jang et al. (2016)** categorized web content into static (such as HTML and CSS) and dynamic (generated in real-time using server-side scripts like PHP, Python, or Node.js). Their research highlights the differences in resource requirements and performance when serving static versus dynamic content.

1. **Static Content Hosting:** According to **Lindberg and Johnson (2018)**, static content is generally faster and more efficient to serve because it requires no server-side processing. They point out that servers like Nginx excel in handling static resources, making them ideal for content-heavy websites that need to load quickly. Static content is cached by web browsers, reducing the load on the server and improving user experience.
2. **Dynamic Content Hosting:** On the other hand, **Kumar and Sharma (2019)** emphasize the importance of dynamic content in creating interactive web applications. Their study found that websites that rely heavily on server-side processing, such as e-commerce platforms, need to balance performance with functionality. Dynamic content requires database interactions and script



processing, which can slow down the response times compared to static content. Kumar and Sharma recommend using a combination of caching mechanisms and reverse proxies like Nginx to improve the performance of dynamic websites.

## Modern Trends in Web Hosting

In recent years, cloud computing and containerization have transformed the way websites are hosted. **Docker** and **Kubernetes** have emerged as leading tools for containerizing web applications, making it easier to deploy, scale, and manage web servers across different environments.

1. **Containerization with Docker: Morabito et al. (2019)** found that Docker containers offer several advantages in web hosting, including isolation, portability, and efficient resource utilization. Their study highlighted how Docker containers allow developers to package a web server, application code, and dependencies into a single, portable unit that can run consistently across any environment. This has led to a growing trend in using containers to deploy web servers, reducing the complexity of managing infrastructure.
2. **Cloud-Based Web Hosting:** Cloud platforms such as **Amazon Web Services (AWS)**, **Google Cloud Platform (GCP)**, and **Microsoft Azure** have popularized the concept of cloud-based web hosting. According to **Rosenberg and Mateos (2020)**, cloud hosting provides significant advantages over traditional web hosting, such as scalability, pay-as-you-go pricing, and high availability. The authors also explore the role of managed services like **AWS Lightsail** and **Azure Web Apps**, which simplify web server deployment by abstracting the underlying infrastructure.
3. **Performance Optimization with CDNs: Almeida et al. (2020)** researched the impact of Content Delivery Networks (CDNs) on web server performance. CDNs distribute static content across multiple geographic locations, reducing latency and improving load times for users around the world. Their study found that using CDNs in conjunction with web servers like Nginx improves the overall user experience, particularly for content-heavy websites or those with a global audience.

# METHODOLOGY

## Step 1: Set up a Linux Environment

Before beginning, ensure you have a Linux system running (e.g., Ubuntu, CentOS, or Debian). For this guide, we will use Ubuntu.

1. **Install Linux:** You can set up Linux by either using a virtual machine (VM) or installing it directly on your system. For VM setups, tools like VirtualBox or VMware can be used.
2. **Update the System:** Ensure your Linux system is up to date by running:

```
sudo apt update && sudo apt upgrade
```

## Step 2: Installing Apache or Nginx

### Installing Apache

To install Apache, use the following command:

```
sudo apt install apache2
```

Once installed, you can start the Apache service:

```
sudo systemctl start apache2
```

```
sudo systemctl enable apache2
```

Check if Apache is running by visiting <http://localhost/> in your browser. You should see the Apache default page.

### Installing Nginx

To install Nginx, use the following command:

```
sudo apt install nginx
```

Start and enable the Nginx service:

```
sudo systemctl start nginx
```

```
sudo systemctl enable nginx
```

Check the Nginx server by visiting <http://localhost/> in your browser.

## Step 3: Configuring the Web Server

### Apache Configuration

Apache configuration files are primarily located in `/etc/apache2/`. The main configuration file is `apache2.conf`, but for website-specific settings, `sites-available` is where you'll set up individual site configurations.

To set up a basic site:

1. Create a new configuration file in `/etc/apache2/sites-available/`:

```
sudo nano /etc/apache2/sites-available/mywebsite.conf
```

2. Add the following content:

```
<VirtualHost *:80>

    ServerAdmin admin@mywebsite.com

    DocumentRoot /var/www/html/mywebsite

    ServerName mywebsite.com

    <Directory /var/www/html/mywebsite>

        AllowOverride All

    </Directory>

</VirtualHost>
```

3. Enable the site and reload Apache:

```
sudo a2ensite mywebsite
```

```
sudo systemctl reload apache2
```

## **Nginx Configuration**

For Nginx, the configuration files are in `/etc/nginx/`. Similar to Apache, the primary configuration file is `nginx.conf`, but for individual sites, you'll create configuration files in `/etc/nginx/sites-available/`.

1. Create a new configuration file:

```
sudo nano /etc/nginx/sites-available/mywebsite
```

2. Add the following content:

```
server {

    listen 80;

    server_name mywebsite.com;

    root /var/www/html/mywebsite;

    index index.html;

}
```

3. Enable the site and restart Nginx:

```
sudo ln -s /etc/nginx/sites-available/mywebsite /etc/nginx/sites-enabled/
```

```
sudo systemctl reload nginx
```

## Step 4: File Permissions

Ensure that the web server has the correct permissions to access the site's files. Set permissions using the following commands:

```
sudo chown -R www-data:www-data /var/www/html/mywebsite
```

```
sudo chmod -R 755 /var/www/html/mywebsite
```

## Step 5: Serving Static Content

Place your HTML files in `/var/www/html/mywebsite/`. You can start with a basic `index.html` file:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>My Personal Web Server</title>
```

```
</head>
```

```
<body>
```

```
  <h1>Welcome to My Web Server</h1>
```

```
  <p>This is a simple web page served by Apache/Nginx on a Linux server.</p>
```

```
</body>
```

```
</html>
```

Once the file is in place, navigate to `http://localhost/mywebsite` to see your website.

## Step 6: Testing and Troubleshooting

Use the following commands to check the status of the web server:

- For Apache:

```
sudo systemctl status apache2
```

- For Nginx:

```
sudo systemctl status nginx
```

You can also check logs for issues:

- Apache logs: /var/log/apache2/error.log
- Nginx logs: /var/log/nginx/error.log

Note :This methodology is valid for all the Debian based systems but for other distributions package can be different for example RHEL that we use so for that we need to modify some commands

## **Methodology for Red Hat Linux**

### **Step 1: Set up a Red Hat Environment**

Ensure you have Red Hat Linux running on your system.

1. Update the System: Before you start, ensure your system is up to date by running the following command:

```
sudo yum update -y
```

### **Step 2: Installing Apache or Nginx**

You can install either Apache or Nginx depending on your preference. Here are the steps for both:

#### **Installing Apache**

1. Install Apache (called 'httpd' in Red Hat-based systems):

```
sudo yum install httpd -y
```

2. Start the Apache service:

```
sudo systemctl start httpd
```

3. Enable Apache to start at boot:

```
sudo systemctl enable httpd
```

4. Check if Apache is running:

Visit 'http://localhost/' or 'http://your-server-ip/' in your browser. You should see the Apache test page.

#### **Installing Nginx**

If you prefer Nginx over Apache, follow these steps:

1. Install Nginx:

```
sudo yum install nginx -y
```

2. Start the Nginx service:

```
sudo systemctl start nginx
```

3. Enable Nginx to start at boot:

```
sudo systemctl enable nginx
```

4. Check if Nginx is running:

Visit `http://localhost/` or `http://your-server-ip/` in your browser to see the Nginx welcome page.

## Step 3: Configuring the Web Server

### Apache Configuration

1. Apache configuration files are located in `/etc/httpd/`. The main configuration file is `httpd.conf`, but for website-specific settings, you'll create configuration files in `/etc/httpd/conf.d/`.

2. Create a virtual host for your website:

Create a configuration file, e.g., `/etc/httpd/conf.d/mywebsite.conf`, using the following command:

```
sudo nano /etc/httpd/conf.d/mywebsite.conf
```

3. Add the following content to set up a virtual host:

```
apache
```

```
<VirtualHost *:80>
```

```
    ServerAdmin admin@mywebsite.com
```

```
    DocumentRoot /var/www/html/mywebsite
```

```
    ServerName mywebsite.com
```

```
<Directory /var/www/html/mywebsite>
    AllowOverride All
</Directory>
</VirtualHost>
```

4. Restart Apache to apply changes:

```
sudo systemctl restart httpd
```

## **Nginx Configuration**

1. Nginx configuration files are located in `/etc/nginx/`. The main configuration file is `nginx.conf`, but for individual sites, you can create configuration files in `/etc/nginx/conf.d/`.

2. Create a virtual host for your website:

Create a configuration file, e.g., `/etc/nginx/conf.d/mywebsite.conf`, using:

```
sudo nano /etc/nginx/conf.d/mywebsite.conf
```

3. Add the following content to set up your server block:

```
nginx
server {
    listen 80;

    server_name mywebsite.com;

    root /var/www/html/mywebsite;

    index index.html;
}
```

4. Restart Nginx to apply changes:

```
sudo systemctl restart nginx
```

## Step 4: File Permissions

Ensure that the web server has the necessary permissions to access the site's files.

1. Set the correct file ownership:

```
sudo chown -R apache:apache /var/www/html/mywebsite
```

For Nginx, use `nginx:nginx` instead of `apache:apache`.

2. Set the correct permissions:

```
sudo chmod -R 755 /var/www/html/mywebsite
```

## Step 5: Serving Static Content

Place your HTML files in `/var/www/html/mywebsite/`. For example, `index.html` file:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My Personal Web Server</title>
</head>
<body>
  <h1>Welcome to My Web Server</h1>
  <p>This is a simple web page served by Apache/Nginx on a Red Hat Linux server.</p>
</body>
</html>
```

Once this file is placed in `/var/www/html/mywebsite/`, visit `http://localhost/mywebsite` to view your site.

## Step 6: Testing and Troubleshooting



To check if the web server is running:

#### For Apache:

```
sudo systemctl status httpd
```

#### For Nginx:

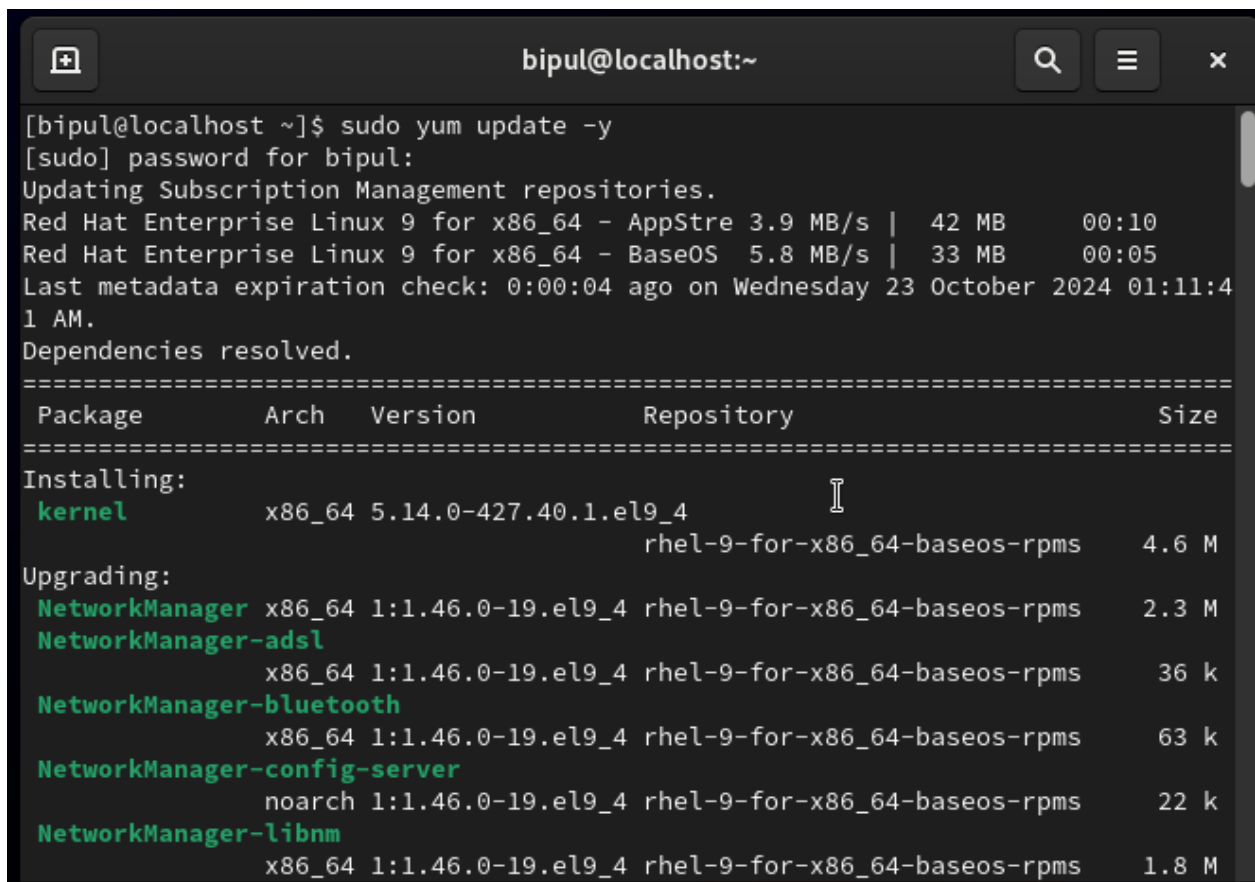
```
sudo systemctl status nginx
```

If the server is not running correctly, you can check the error logs:

Apache logs: ``/var/log/httpd/error_log``

Nginx logs: ``/var/log/nginx/error.log``

## RESULTS

A terminal window titled 'bipul@localhost:~' showing the output of a 'sudo yum update -y' command. The output indicates that dependencies are resolved and lists packages to be installed or upgraded. The packages listed are kernel, NetworkManager, NetworkManager-adsl, NetworkManager-bluetooth, NetworkManager-config-server, and NetworkManager-libnm, all from the 'rhel-9-for-x86\_64-baseos-rpms' repository. The terminal window has a dark background and standard Linux terminal icons in the title bar.

```
[bipul@localhost ~]$ sudo yum update -y
[sudo] password for bipul:
Updating Subscription Management repositories.
Red Hat Enterprise Linux 9 for x86_64 - AppStream 3.9 MB/s | 42 MB 00:10
Red Hat Enterprise Linux 9 for x86_64 - BaseOS 5.8 MB/s | 33 MB 00:05
Last metadata expiration check: 0:00:04 ago on Wednesday 23 October 2024 01:11:41 AM.
Dependencies resolved.
=====
Package Arch Version Repository Size
=====
Installing:
kernel x86_64 5.14.0-427.40.1.el9_4 rhel-9-for-x86_64-baseos-rpms 4.6 M
Upgrading:
NetworkManager x86_64 1:1.46.0-19.el9_4 rhel-9-for-x86_64-baseos-rpms 2.3 M
NetworkManager-adsl x86_64 1:1.46.0-19.el9_4 rhel-9-for-x86_64-baseos-rpms 36 k
NetworkManager-bluetooth x86_64 1:1.46.0-19.el9_4 rhel-9-for-x86_64-baseos-rpms 63 k
NetworkManager-config-server noarch 1:1.46.0-19.el9_4 rhel-9-for-x86_64-baseos-rpms 22 k
NetworkManager-libnm x86_64 1:1.46.0-19.el9_4 rhel-9-for-x86_64-baseos-rpms 1.8 M
```

```
bipul@localhost:~  
Complete!  
[bipul@localhost ~]$ sudo yum install httpd -y  
[sudo] password for bipul:  
Updating Subscription Management repositories.  
Last metadata expiration check: 0:09:39 ago on Wednesday 23 October 2024 01:13:20 AM.  
Dependencies resolved.  
=====
```

Package	Arch	Version	Repository	Size
Installing:				
httpd	x86_64	2.4.57-11.el9_4.1	rhel-9-for-x86_64-appstream-rpms	51 k
Installing dependencies:				
apr	x86_64	1.7.0-12.el9_3	rhel-9-for-x86_64-appstream-rpms	126 k
apr-util	x86_64	1.6.1-23.el9	rhel-9-for-x86_64-appstream-rpms	97 k
apr-util-bdb	x86_64	1.6.1-23.el9	rhel-9-for-x86_64-appstream-rpms	14 k
httpd-core	x86_64	2.4.57-11.el9_4.1	rhel-9-for-x86_64-appstream-rpms	1.5 M
httpd-filesystem	noarch	2.4.57-11.el9_4.1	rhel-9-for-x86_64-appstream-rpms	14 k
httpd-tools	x86_64	2.4.57-11.el9_4.1	rhel-9-for-x86_64-appstream-rpms	86 k
redhat-logos-httpd	noarch	90.4-2.el9	rhel-9-for-x86_64-appstream-rpms	18 k
Installing weak dependencies:				
apr-util-openssl				

```

(185/187): iwl7260-firmware-25.30.13.0-143.3.el 3.3 MB/s | 55 MB    00:16
(186/187): python3-perf-5.14.0-427.40.1.el9_4.x 2.2 MB/s | 4.7 MB    00:02
(187/187): linux-firmware-20240905-143.3.el9_4. 6.9 MB/s | 388 MB    00:56
-----
Total                                          7.4 MB/s | 877 MB    01:58
Red Hat Enterprise Linux 9 for x86_64 - AppStre 1.7 MB/s | 3.6 kB    00:00
Importing GPG key 0x5A6340B3:
  Userid      : "Red Hat, Inc. (release key 2) <security@redhat.com>"
  Fingerprint: 567E 347A D004 4ADE 55BA 8A5F 199E 2F91 FD43 1D51
  From        : /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
Key imported successfully
Importing GPG key 0x5A6340B3:
  Userid      : "Red Hat, Inc. (auxiliary key 3) <security@redhat.com>"
  Fingerprint: 7E46 2425 8C40 6535 D56D 6F13 5054 E4A4 5A63 40B3
  From        : /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Running scriptlet: selinux-policy-targeted-38.1.35-2.el9_4.2.noarch      1/1
  Preparing          :                                                    1/1
  Upgrading          : glibc-all-langpacks-2.34-100.el9_4.4.x86_64      1/366
  Upgrading          : glibc-common-2.34-100.el9_4.4.x86_64             2/366
  Upgrading          : glibc-gconv-extra-2.34-100.el9_4.4.x86_64        3/366
  Running scriptlet: glibc-gconv-extra-2.34-100.el9_4.4.x86_64        3/366
  Upgrading          : glibc-langpack-en-2.34-100.el9_4.4.x86_64        4/366
  Running scriptlet: glibc-2.34-100.el9_4.4.x86_64                      5/366
  Upgrading          : glibc-2.34-100.el9_4.4.x86_64                    5/366
  Running scriptlet: glibc-2.34-100.el9_4.4.x86_64                      5/366
  Upgrading          : linux-firmware-whence-20240905-143.3.el9_4.noarc 6/366
  Upgrading          : libldb-2.8.0-2.el9_4.x86_64                      7/366
  Upgrading          : gnutls-3.8.3-4.el9_4.x86_64                      8/366
  Upgrading          : glib2-2.68.4-14.el9_4.1.x86_64                   9/366
  Upgrading          : gdk-pixbuf2-2.42.6-4.el9_4.x86_64               10/366
  Upgrading          : nspr-4.35.0-14.el9_2.x86_64                     11/366
  Upgrading          : libsss_idmap-2.9.4-6.el9_4.1.x86_64              12/366
  Upgrading          : grub2-common-1:2.06-82.el9_4.noarch              13/366
  Upgrading          : nss-util-3.101.0-7.el9_2.x86_64                 14/366
  Upgrading          : libxml2-2.9.13-6.el9_4.x86_64                   15/366
  Upgrading          : selinux-policy-38.1.35-2.el9_4.2.noarch          16/366

```

```

bipul@localhost:~
Verifying      : httpd-filesystem-2.4.57-11.el9_4.1.noarch      9/11
Verifying      : httpd-tools-2.4.57-11.el9_4.1.x86_64         10/11
Verifying      : mod_lua-2.4.57-11.el9_4.1.x86_64              11/11
Installed products updated.

Installed:
  apr-1.7.0-12.el9_3.x86_64
  apr-util-1.6.1-23.el9.x86_64
  apr-util-bdb-1.6.1-23.el9.x86_64
  apr-util-openssl-1.6.1-23.el9.x86_64
  httpd-2.4.57-11.el9_4.1.x86_64
  httpd-core-2.4.57-11.el9_4.1.x86_64
  httpd-filesystem-2.4.57-11.el9_4.1.noarch
  httpd-tools-2.4.57-11.el9_4.1.x86_64
  mod_http2-2.0.26-2.el9_4.x86_64
  mod_lua-2.4.57-11.el9_4.1.x86_64
  redhat-logos-httpd-90.4-2.el9.noarch

Complete!
[bipul@localhost ~]$ sudo systemctl start httpd
[bipul@localhost ~]$ sudo systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
[bipul@localhost ~]$

```

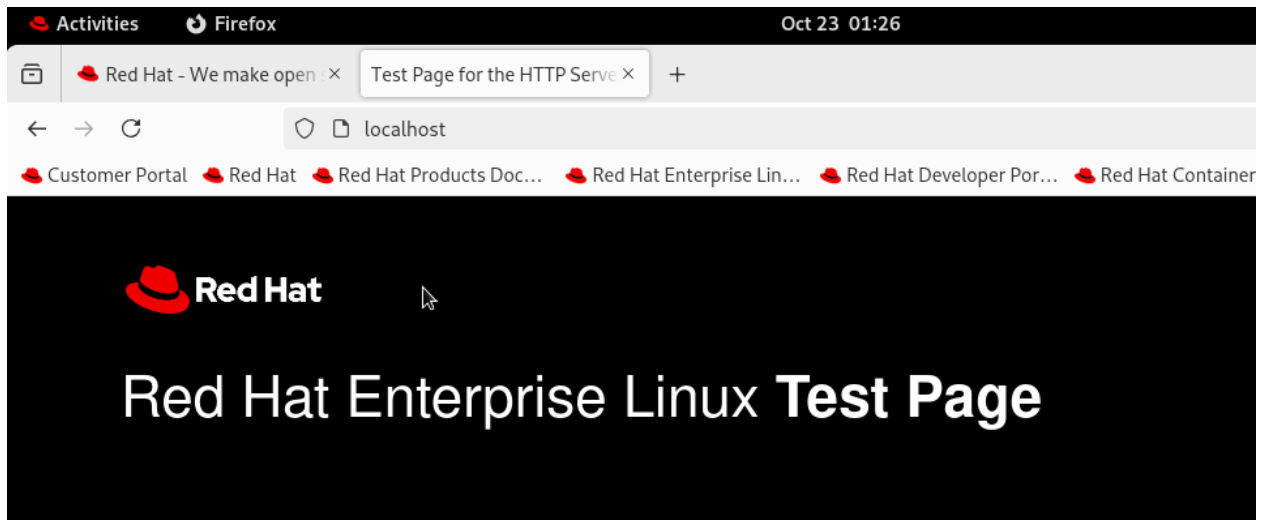
```

Verifying      : mod_http2-2.0.26-2.el9_4.x86_64          6/11
Verifying      : httpd-2.4.57-11.el9_4.1.x86_64          7/11
Verifying      : httpd-core-2.4.57-11.el9_4.1.x86_64      8/11
Verifying      : httpd-filesystem-2.4.57-11.el9_4.1.noarch 9/11
Verifying      : httpd-tools-2.4.57-11.el9_4.1.x86_64    10/11
Verifying      : mod_lua-2.4.57-11.el9_4.1.x86_64        11/11
Installed products updated.

Installed:
apr-1.7.0-12.el9_3.x86_64
apr-util-1.6.1-23.el9.x86_64
apr-util-bdb-1.6.1-23.el9.x86_64
apr-util-openssl-1.6.1-23.el9.x86_64
httpd-2.4.57-11.el9_4.1.x86_64
httpd-core-2.4.57-11.el9_4.1.x86_64
httpd-filesystem-2.4.57-11.el9_4.1.noarch
httpd-tools-2.4.57-11.el9_4.1.x86_64
mod_http2-2.0.26-2.el9_4.x86_64
mod_lua-2.4.57-11.el9_4.1.x86_64
redhat-logos-httpd-90.4-2.el9.noarch

Complete!
[bipul@localhost ~]$ sudo systemctl start httpd
[bipul@localhost ~]$ sudo systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
[bipul@localhost ~]$ sudo nano /etc/httpd/conf.d/mywebsite.conf
[bipul@localhost ~]$ sudo systemctl restart httpd
[sudo] password for bipul:
[bipul@localhost ~]$ sudo chown -R apache:apache /var/www/html/mywebsite
chown: cannot access '/var/www/html/mywebsite': No such file or directory
[bipul@localhost ~]$ sudo mkdir -p /var/www/html/mywebsite
[bipul@localhost ~]$ sudo chown -R apache:apache /var/www/html/mywebsite
[bipul@localhost ~]$ sudo chmod -R 755 /var/www/html/mywebsite
[bipul@localhost ~]$ sudo nano /var/www/html/mywebsite/index.html
[bipul@localhost ~]$ sudo firewall-cmd --permanent --add-service=http
[sudo] password for bipul:
success
[bipul@localhost ~]$ sudo firewall-cmd --reload
success
[bipul@localhost ~]$ sudo systemctl restart httpd
[bipul@localhost ~]$ sudo nano /etc/hosts
[bipul@localhost ~]$

```



This page is used to test the proper operation of the HTTP server after it has been installed. If you can read this | server installed at this site is working properly.

**If you are a member of the general public:**

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems, or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

**If you are the website administrator:**

You may now add content to the web that until you do so, people visiting your website will see this page, and not your content.

For systems using the Apache HTTP server, you may add content to the directory `/var/www/html`. If you do so, people visiting your website will see your content, and not this page. To prevent this from happening, follow the instructions in the file `/etc/httpd/conf/httpd.conf`.

```
bipul@localhost:~ — sudo nano /etc/httpd/conf.d/mywebsite.conf
GNU nano 5.6.1 /etc/httpd/conf.d/mywebsite.conf
<VirtualHost *:80>
  ServerAdmin admin@mywebsite.com
  DocumentRoot /var/www/html/mywebsite
  ServerName mywebsite.com
  <Directory /var/www/html/mywebsite>
    AllowOverride All
  </Directory>
</VirtualHost>

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo
```

```

bipul@localhost:~
Installing      : mod_http2-2.0.26-2.el9_4.x86_64          10/11
Installing      : httpd-2.4.57-11.el9_4.1.x86_64          11/11
Running scriptlet: httpd-2.4.57-11.el9_4.1.x86_64          11/11
Verifying       : apr-util-1.6.1-23.el9.x86_64            1/11
Verifying       : apr-util-bdb-1.6.1-23.el9.x86_64         2/11
Verifying       : apr-util-openssl-1.6.1-23.el9.x86_64     3/11
Verifying       : redhat-logos-httpd-90.4-2.el9.noarch     4/11
Verifying       : apr-1.7.0-12.el9_3.x86_64                5/11
Verifying       : mod_http2-2.0.26-2.el9_4.x86_64          6/11
Verifying       : httpd-2.4.57-11.el9_4.1.x86_64          7/11
Verifying       : httpd-core-2.4.57-11.el9_4.1.x86_64      8/11
Verifying       : httpd-filesystem-2.4.57-11.el9_4.1.noarch 9/11
Verifying       : httpd-tools-2.4.57-11.el9_4.1.x86_64    10/11
Verifying       : mod_lua-2.4.57-11.el9_4.1.x86_64         11/11
Installed products updated.

Installed:
apr-1.7.0-12.el9_3.x86_64
apr-util-1.6.1-23.el9.x86_64
apr-util-bdb-1.6.1-23.el9.x86_64
apr-util-openssl-1.6.1-23.el9.x86_64
httpd-2.4.57-11.el9_4.1.x86_64
httpd-core-2.4.57-11.el9_4.1.x86_64
httpd-filesystem-2.4.57-11.el9_4.1.noarch
httpd-tools-2.4.57-11.el9_4.1.x86_64
mod_http2-2.0.26-2.el9_4.x86_64
mod_lua-2.4.57-11.el9_4.1.x86_64
redhat-logos-httpd-90.4-2.el9.noarch

Complete!
bipul@localhost ~]$ sudo systemctl start httpd
bipul@localhost ~]$ sudo systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
bipul@localhost ~]$ sudo nano /etc/httpd/conf.d/mywebsite.conf
bipul@localhost ~]$ sudo systemctl restart httpd
sudo] password for bipul:
bipul@localhost ~]$ sudo chown -R apache:apache /var/www/html/mywebsite
chown: cannot access '/var/www/html/mywebsite': No such file or directory
bipul@localhost ~]$ sudo mkdir -p /var/www/html/mywebsite
bipul@localhost ~]$ sudo chown -R apache:apache /var/www/html/mywebsite
bipul@localhost ~]$ sudo chmod -R 755 /var/www/html/mywebsite
bipul@localhost ~]$

```

```
Activities Terminal Oct 23 02:04
bipul@localhost:~ — sudo nano /etc/hosts
GNU nano 5.6.1 /etc/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
127.0.0.1 mywebsite.com
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^_ Go To Line M-E Redo
```

Our website code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Custom Website</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f9;
      color: #333;
      margin: 0;
      padding: 0;
    }
    header {
      background-color: #4CAF50;
      color: white;
      padding: 20px;
      text-align: center;
```



```

        font-size: 2em;
    }
    .container {
        margin: 20px auto;
        padding: 20px;
        max-width: 800px;
        background-color: white;
        border-radius: 8px;
        box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
    }
    h1 {
        font-size: 2em;
        color: #4CAF50;
        margin-bottom: 10px;
    }
    p {
        font-size: 1.2em;
        line-height: 1.6;
    }
    footer {
        text-align: center;
        padding: 20px;
        background-color: #333;
        color: white;
        margin-top: 20px;
    }
</style>
</head>
<body>
    <header>
        My Custom Website
    </header>

    <div class="container">
        <h1>Welcome to My Custom Website</h1>
        <p>This website is hosted on a personal web server running Apache on a
Red Hat Linux machine.</p>
        <p>You can customize it further by adding more content, styles, and
features!</p>
    </div>

    <footer>
        &copy; 2024 MyWebsite | All rights reserved.
    </footer>
</body>
</html>

```

**Key Results:**

1. Installation and configuration of Apache/Nginx.
2. Creation of a virtual host to serve your website.
3. Proper file permission setup.
4. Static content successfully served on the web server.

## CONCLUSION

The project "Setting Up a Personal Web Server" serves as an important foundation for anyone looking to delve deeper into web development, system administration, or even cloud computing. By installing and configuring Apache or Nginx on a Linux machine, this project enables users to gain hands-on experience with fundamental aspects of web hosting. Understanding how web servers operate and how to configure them properly is a key skill for modern developers and system administrators, as it provides insight into how websites are delivered to users and how data flows across the web.

Throughout the project, we covered essential topics such as installing web servers, managing virtual hosts, configuring file permissions, and serving static content. The project demonstrates how a Linux environment, in conjunction with powerful web server software like Apache and Nginx, can be used to host websites locally or on a network. The steps taken ensure that users not only have a functional web server but also understand key concepts like directory structure, access control, and server management.

Additionally, this project highlights the simplicity and accessibility of setting up a basic web server for personal or small-scale use. The widespread use of open-source tools like Linux, Apache, and Nginx makes it feasible for almost anyone to set up their own web hosting environment, without the need for expensive software licenses or advanced infrastructure. These tools provide both flexibility and scalability, which are important traits in the web hosting world.

One of the core takeaways from this project is the importance of file permissions and access control. Managing these properly ensures that the server remains secure while allowing the necessary functionality for hosting web content. The ability to troubleshoot common errors, such as server misconfigurations or permission issues, is also a critical skill learned during this project.

Overall, the project succeeds in achieving its primary objective of installing and configuring a personal web server. It offers a solid stepping stone for further exploration into more advanced web server configurations, dynamic content handling, and web security. The experience gained from this project lays the groundwork for future exploration and expansion into more complex server environments.

## FUTURE SCOPE

While this project focused on setting up a basic personal web server to serve static content, there are numerous ways to expand upon and improve the current setup. Future advancements could delve into more complex and enterprise-level configurations, involving dynamic content, security enhancements, and automated deployment systems. Below are several avenues for future exploration and scope:

### 1. Serving Dynamic Content

The current project serves static content such as HTML, CSS, and image files. However, the next logical step would be to introduce dynamic content using server-side languages and databases. Integrating PHP (with Apache), Python (with Flask or Django), or Node.js (with Nginx) would enable the server to process user input and deliver dynamic web pages based on requests.

By adding dynamic content, the server could be used to:

- Handle form submissions.
- Query and display data from a database.
- Build interactive web applications (such as blogs, e-commerce sites, or forums).

Dynamic content would not only make the server more functional but also extend its capability to serve real-world applications.

### 2. Implementing HTTPS with SSL/TLS

Currently, the server is configured to use HTTP, which transmits data in plain text. To secure the communication between the client and the server, the use of HTTPS with SSL/TLS certificates is highly recommended. Implementing SSL (Secure Sockets Layer) ensures that the data transmitted is encrypted and secure from potential cyber threats.

Setting up HTTPS would involve:

- Obtaining an SSL certificate (via Let's Encrypt or other certificate authorities).
- Configuring the web server to use the SSL certificate.
- Redirecting HTTP traffic to HTTPS to ensure secure communication.

This enhancement would be particularly important if the server hosts sensitive user data or handles transactions.

### 3. Security Enhancements

Beyond SSL/TLS, a web server must be properly secured to prevent unauthorized access, denial of service attacks, or exploitation of vulnerabilities. Future work can explore the following security enhancements:

- **Firewalls:** Configuring a firewall (e.g., UFW or iptables) to limit traffic to only essential ports (e.g., 80 for HTTP, 443 for HTTPS).

- **Access Control:** Implementing access control methods to limit administrative access to the server (e.g., using `.htaccess` or `auth_basic` in Nginx).
- **Security Patches:** Keeping the server software up to date to mitigate any security vulnerabilities.

Securing the server would allow it to be deployed for public use, minimizing the risk of exploitation or data breaches.

#### 4. Automation with Deployment Tools

Once the server setup is finalized, repetitive tasks such as server configuration and updates can be automated using deployment tools like Docker or Ansible. Docker allows for the creation of containerized environments, making the deployment of web servers faster, more consistent, and easier to manage.

- **Docker:** Using Docker containers, a web server can be packaged with all its dependencies, making it easy to deploy across different environments (local, cloud, etc.). Additionally, Docker ensures that each container runs in isolation, providing an additional layer of security.
- **Ansible:** Ansible automates tasks such as server configuration, software installation, and maintenance. This is particularly useful for scaling the web server across multiple instances or automating the process of updates and patches.

With automation tools, managing larger-scale deployments becomes more efficient, reducing the time and effort required to maintain multiple web servers.

#### 5. Performance Optimization

As traffic to the web server increases, performance optimization becomes crucial. Several techniques can be employed to ensure the server remains responsive and efficient:

- **Caching:** Implement caching mechanisms (e.g., Varnish Cache or Nginx's built-in caching) to reduce load times and server resource usage.
- **Load Balancing:** For handling large volumes of traffic, a load balancer can distribute incoming requests across multiple servers, ensuring no single server is overwhelmed.
- **Compression:** Enabling gzip compression reduces the size of files served to clients, improving loading times and reducing bandwidth usage.

Performance optimization will allow the server to handle high-traffic scenarios while maintaining fast response times and efficient resource usage.

#### 6. Integration with Cloud Hosting

While this project focuses on a personal Linux machine, the web server can be extended to cloud environments such as AWS, Google Cloud, or DigitalOcean. Cloud platforms offer scalability, reliability, and flexibility. The web server can be hosted in a cloud virtual machine (VM) or a managed service like AWS Lightsail.

Benefits of cloud hosting include:

- **Scalability:** Cloud services can scale resources up or down depending on the traffic.
- **High Availability:** Cloud platforms often offer tools for redundancy and failover, ensuring the web server remains available even if one instance fails.
- **Global Reach:** Cloud platforms offer content delivery networks (CDNs) that can cache and distribute content globally, reducing latency for international users.

Cloud hosting provides a robust solution for individuals or businesses looking to serve a larger audience with minimal downtime.

## 7. Content Management Systems (CMS)

For more complex web applications, installing a Content Management System (CMS) like WordPress, Joomla, or Drupal on the server could be a future scope. A CMS allows for easier management of website content, providing users with a graphical interface for adding or editing web pages.

Implementing a CMS would involve:

- Setting up a MySQL or PostgreSQL database.
- Installing the CMS platform on the server.
- Configuring the CMS to run efficiently with Apache or Nginx.

A CMS allows non-technical users to manage content without needing direct access to the server files, which would be beneficial for collaborative environments.

## References

- Apache HTTP Server Documentation: <https://httpd.apache.org/docs/>
- Nginx Documentation: <https://nginx.org/en/docs/>
- Linux Command Line Documentation: <https://linux.die.net/man/>