Hosting My Website: A Step-by-Step Guide

French version:

https://samueldecarnelle.com/projects/hosting-portfolio/hosting-my-portfolio-fr.pdf

1. Setting Up the Website and Virtual Machine

I began by developing my personal portfolio using **HTML, CSS, JavaScript**, and **PHP** to create both static and dynamic content.

To host the site, I used my **Proxmox hypervisor** to create a **virtual machine (VM)** running **Ubuntu Server**. This VM would serve as the foundation of my self-hosted web server.

2. Domain Name and DNS Configuration

I registered my domain name:
`samueldecarnelle.com`

To make the website accessible from anywhere on the internet:

- I contacted my **ISP** and requested a **static public IP address**.
- I then configured the **DNS A record** to point the domain to this static IP, effectively routing traffic to my home server.

3. Installing Web Server Components

Once the Ubuntu Server VM was ready, I installed the essential components:

```
sudo apt update
sudo apt install apache2 php mysql-server -y
```

- **Apache2** serves the website content.
- **PHP** handles the server-side scripting.

- **MySQL** manages databases.

I also secured MySQL by:

- Disabling **remote root login**
- Enforcing **strong passwords**
- Running `mysql_secure_installation`

4. Securing the Server with UFW (Firewall)

To protect the server from unauthorized access, I set up **UFW** (Uncomplicated Firewall), which is easy to manage:

```
sudo ufw enable
sudo ufw allow OpenSSH
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw status verbose
```

This configuration ensures only essential ports (SSH, HTTP, HTTPS) are open.

5. Configuring Port Forwarding on My Router

Since the server is inside my home network (behind NAT), I configured **port forwarding** on my router:

- **Port 80** → Internal IP of VM (for HTTP)
- **Port 443** → Internal IP of VM (for HTTPS)

This allows external requests to reach the internal web server.

6. Enabling HTTPS with Let's Encrypt (Certbot)

To secure traffic with **SSL/TLS**, I used **Certbot**:

```
sudo apt install certbot python3-certbot-apache -y
sudo certbot --apache
```

This automatically installed and configured an SSL certificate, enabling secure **HTTPS** access to the website.

7. Enhancing SSH Security with Fail2Ban

Fail2Ban protects against brute-force attacks by monitoring logs and banning IPs that repeatedly fail authentication.

So, I install Fail2Ban:

```
sudo apt install fail2ban -y
sudo nano
```

And I add theses configurations inside the `/etc/fail2ban/jail.local file`:

```
[sshd]
enabled = true
port = ssh
logpath = /var/log/auth.log
maxretry = 5
bantime = 30m
findtime = 30m
```

Then I apply the changes using:

```
sudo systemctl restart fail2ban
sudo systemctl enable fail2ban
```

Finaly I check the status using:

```
sudo fail2ban-client status sshd
```

8. Securing Web Applications with ModSecurity (WAF)

To filter malicious HTTP requests, I installed **ModSecurity**:

```
sudo apt install libapache2-mod-security2 -y
sudo a2enmod security2
sudo systemctl restart apache2
```

Then I change “SecRuleEngine DetectionOnly” to “SecRuleEngine On” inside my /etc/modsecurity/modsecurity.conf file.

Then restart Apache and test the WAF:

```
sudo systemctl restart apache2
curl -A "sqlmap" http://localhost
```

It returns **403 Forbidden** so ModSecurity is working.

9. Configuring Nginx as a Reverse Proxy

To enhance performance, I set up **Nginx** as a reverse proxy for Apache (which now listens on port 8080):

Step 1: Change Apache Port

```
sudo nano /etc/apache2/ports.conf
```

I change the “Listen 80” port to “Listen 8080” inside the

I edit the virtual host config file:

```
sudo nano /etc/apache2/sites-available/000-default.conf
```

Update `<VirtualHost *:80>`

to `<VirtualHost *:8080>`,

Then I restart Apache:

```
sudo systemctl restart apache2
```

Step 2: Set Up Nginx

```
sudo apt install nginx -y
```

I add theses line to the `/etc/nginx/sites-available/portfolio`:

```
server {
    listen 80;
    server_name samueldecarnelle.com;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

I enable the config:

```
sudo ln -s /etc/nginx/sites-available/portfolio /etc/nginx/sites-
enabled/
sudo rm /etc/nginx/sites-enabled/default
```

```
sudo systemctl restart nginx
```

10. Enforcing HTTPS Redirection in Nginx

To redirect all HTTP traffic to HTTPS, I edit the same Nginx config file:

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

I add above the current server block:

```
server {
    listen 80;
    server_name samueldecarnelle.com;
    return 301 https://$host$request_uri;
}
```

Restart Nginx:

```
sudo systemctl restart nginx
```

11. Monitoring and Log Checking Commands

Regular log checks are essential for spotting issues and detecting potential threats. Here's a list of key commands I use to monitor the system:

Apache Logs

```
sudo tail -f /var/log/apache2/access.log
sudo tail -f /var/log/apache2/error.log
```

Nginx Logs

```
sudo tail -f /var/log/nginx/access.log
sudo tail -f /var/log/nginx/error.log
```

Authentication Logs

```
sudo tail -f /var/log/auth.log
```

Fail2Ban Status

```
sudo fail2ban-client status sshd
```

12. My Update/Upgrade script to keep my server updated

I wrote a Bash script to auto update and upgrade my webserver, so it's always updated.

Script:

```
#!/bin/bash

LOG_FILE="/home/sam/update_upgrade.log"

echo "================================================" | tee -a $LOG_FILE

echo "Update started at: $(date '+%Y-%m-%d %H:%M:%S')" | tee -a $LOG_FILE

echo "================================================" | tee -a $LOG_FILE

echo "Starting update and upgrade process..." | tee -a $LOG_FILE

sudo apt update -y | tee -a $LOG_FILE

sudo apt upgrade -y | tee -a $LOG_FILE

sudo apt autoremove -y | tee -a $LOG_FILE

sudo apt autoclean -y | tee -a $LOG_FILE
```

```
echo "Update and upgrade completed successfully!" | tee -a $LOG_FILE

echo "================================================" | tee -a $LOG_FILE

echo "Update complete at: $(date '+%Y-%m-%d %H:%M:%S')" | tee -a $LOG_FILE

echo "================================================" | tee -a $LOG_FILE
```

Automation

Then I scheduled it to run daily using Cron.

Conclusion

By carefully configuring Apache, Nginx, UFW, and implementing additional layers of protection like Fail2Ban and ModSecurity, I've successfully built a self-hosted web server that is both secure and reliable.

My website is now accessible from anywhere using my custom domain, with traffic fully encrypted via HTTPS. It's actively protected against brute-force attempts and common web threats, and I can monitor its behavior in real time through system logs.

With this setup in place, my portfolio is not just online, it's stable, secure, and ready for real-world use.