PROJECTS

Setting Up a Web Server on Debian

Building a real LAMP-style stack from scratch on Debian 12 Bookworm: planning the stack, Apache & Nginx, MySQL/MariaDB/PostgreSQL, PHP, SSL/HTTPS, firewall & security basics, and XAMPP as a local development alternative.

7 Chapters | Philip Osztromok | Generated with Claude

CHAPTER 1: OVERVIEW & PLANNING

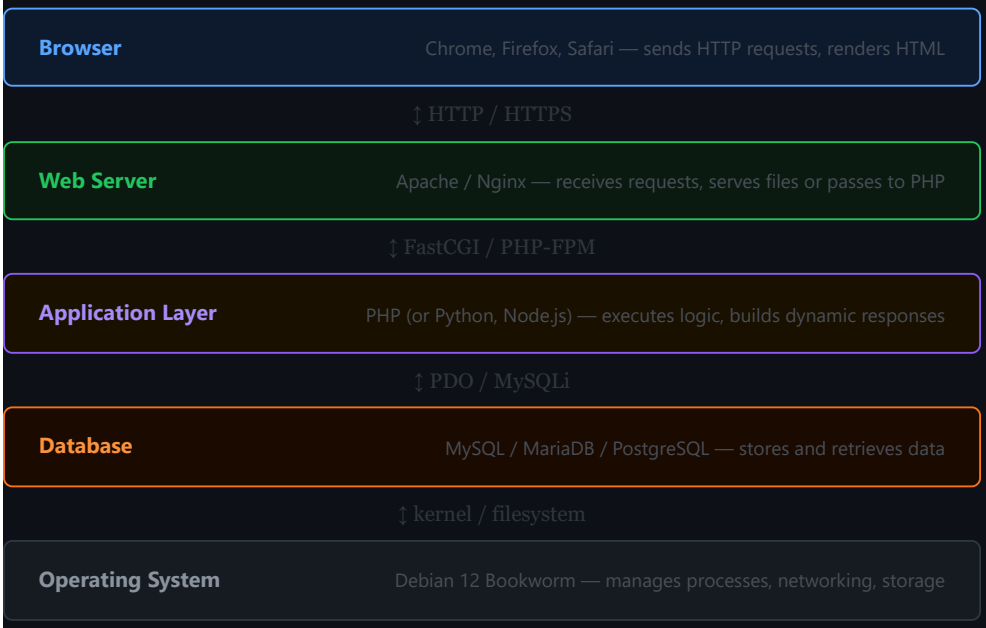
Chapter 1 — Overview & Planning

Before installing anything, it pays to understand what a web server stack actually is, how the components talk to each other, and which options are available at each layer. Debian 12 Bookworm is an excellent platform for a web server — stable, well-supported until 2028, and with first-class packages for every component covered in this series.

This chapter gives you the mental model you need before the hands-on installation begins in Chapter 2. Even if you've run web servers before, the component comparison tables are worth keeping as a reference.

What Is a Web Stack?

A web stack is a collection of software layers that work together to serve web pages to visitors. Each layer has a distinct responsibility, and they communicate in a well-defined sequence every time a browser makes a request.



When a visitor loads a page, the browser sends an HTTP request to the web server. If the request is for a static file (an image, CSS, JavaScript) the web server sends it directly. If it's for a dynamic page (a PHP script), the web server hands the request to PHP, which runs the script, queries the database if needed, builds the HTML response, and sends it back up the chain to the browser.

> **LAMP and LEMP** are the two traditional acronyms for this stack on Linux. **LAMP** = Linux + Apache + MySQL + PHP. **LEMP** = Linux + (E)Nginx + MySQL + PHP. This series covers both web server options so

> you can make an informed choice.

Component Options at Each Layer

Web Server — Apache vs Nginx

Feature	Apache	Nginx
.htaccess per-directory config	✓ Yes — config in each folder	✗ No — all config in server files
PHP handling	mod_php (embedded) or PHP-FPM	PHP-FPM only (external process)
Static file performance	Good	Excellent — event-driven, low memory
Concurrent connections	Good (worker/event MPM)	Excellent — handles thousands with ease
Configuration style	Declarative + .htaccess overrides	Block-based, all in one place
Documentation & community	Vast — decades of examples online	Large and growing
Reverse proxy / load balancing	Supported (mod_proxy)	First-class feature
Best for	Shared hosting, PHP apps with .htaccess, WordPress	High-traffic sites, microservices, static sites
Debian package	`apache2`	`nginx`

Recommendation for this site: Apache is the better match for Philip's Learning Blog — it supports `.htaccess` out of the box, which is what the existing site uses, and it's the most widely documented option for PHP/MySQL setups. Chapter 2 covers both in full.

Database — MySQL vs MariaDB vs PostgreSQL

Feature	MySQL 8	MariaDB 10.x	PostgreSQL 15
Drop-in MySQL replacement	✓ (it is MySQL)	✓ Mostly yes	✗ Different SQL dialect
PHP PDO support	✓ pdo_mysql	✓ pdo_mysql	✓ pdo_pgsql
Performance (reads)	Excellent	Excellent	Good (better for complex queries)
JSON support	Strong	Good	Excellent (JSONB)
Fully open source	Community Edition only	✓ Yes	✓ Yes

Feature	MySQL 8	MariaDB 10.x	PostgreSQL 15
Debian Bookworm package	MySQL APT repo required	`mariadb-server` (in main repo)	`postgresql` (in main repo)
Best for	MySQL-specific apps, full MySQL compatibility	WordPress, general PHP apps, MySQL migration	Complex queries, data integrity, GIS

Note: MariaDB is available directly from Debian's main repository with no extra setup — just `apt install mariadb-server`. MySQL requires adding the MySQL APT repository first. For most PHP-based sites (including this one) MariaDB is a seamless and simpler choice.

Application Layer — PHP versions

Debian 12 Bookworm ships **PHP 8.2** in its main repository — a modern, well-supported version. PHP 8.3 is available via the `sury.org` third-party repository if you need it. This series uses PHP 8.2 from the Debian repo unless otherwise specified.

Version	Source	Security Support	Notes
PHP 8.2	Debian main repo	Until Dec 2026	Recommended — no extra repo needed
PHP 8.3	sury.org PPA	Until Nov 2027	Latest stable — needs extra repo
PHP 8.1	sury.org PPA	Until Dec 2025	Avoid for new installs

Which Stack Should You Choose?

Running a PHP site (WordPress, Laravel, custom PHP)

LAMP — Apache + MariaDB + PHP 8.2. The most documented combination with the best ecosystem support.

High-traffic site or reverse proxy needed

LEMP — Nginx + MariaDB + PHP-FPM. Nginx handles thousands of simultaneous connections with lower memory than Apache.

Just want something running quickly for testing

XAMPP — one installer, everything included. Covered in Chapter 7. Not recommended for production.

Philip's Learning Blog (this site)

Apache + MySQL (or MariaDB) + PHP 8.2. The existing .htaccess config, PDO queries, and admin interface are all built for this combination.

Prerequisites

Before starting Chapter 2, make sure the following are in place on your Debian 12 Bookworm machine:

1. System is up to date

Terminal

```
$ sudo apt update && sudo apt full-upgrade -y
Reading package lists... Done
Building dependency tree... Done
...
```

2. Essential tools installed

Terminal

```
$ sudo apt install -y curl wget gnupg2 ca-certificates lsb-release apt-transport-https
Reading package lists... Done
...
```

3. Know your machine's IP address

Terminal

```
$ ip addr show | grep "inet " | grep -v 127.0.0.1
    inet 192.168.1.50/24 brd 192.168.1.255 scope global eth0

# You'll use this IP to test the web server from another machine on the network
```

4. sudo access confirmed

Terminal

```
$ sudo whoami
root
# If this returns "root" you have the access you need
```

```
# If philip is not in the sudo group: su -c "usermod -aG sudo philip" root
```

About the installation media error: If you see errors about a CD-ROM source when running `apt update`, open `/etc/apt/sources.list` with `sudo vi /etc/apt/sources.list` and comment out any line beginning with `deb cdrom://` by adding a `#` at the start. Save and re-run `sudo apt update`.

What This Series Covers

1	Overview & Planning	Stack anatomy, component choices, comparison tables, prerequisites — this chapter
2	Web Server	Apache and Nginx — install, configure, virtual hosts, testing
3	Database	MySQL, MariaDB, PostgreSQL — install, harden, create users and databases
4	PHP	Install PHP 8.2, configure php.ini, essential extensions, test with a live script
5	SSL / HTTPS	Let's Encrypt with Certbot for public domains, self-signed certs for local dev
6	Firewall & Security	UFW rules, fail2ban, file permissions, hiding version information
7	XAMPP	All-in-one alternative — install, configure, pros and cons vs the manual stack

Next: Chapter 2 — Web Server

Chapter 2 covers the full installation and configuration of both Apache and Nginx on Debian 12 Bookworm — including virtual hosts, document roots, enabling/disabling sites, and verifying the server is running correctly. You only need to install one, but both are documented so you can make the right choice for your setup.

CHAPTER 2: WEB SERVER - APACHE & NGINX

Chapter 2 — Web Server

The web server is the front door of your stack — it listens for incoming HTTP requests and decides what to do with them. This chapter covers both Apache and Nginx in full: installation, directory structure, configuration, virtual hosts, and essential management commands. You only need to install one; both are documented here so the chapter serves as a complete reference regardless of which you choose.

> **Which should I pick?** For Philip's Learning Blog — Apache. The existing site uses `.htaccess` files and was built around Apache conventions. For a new high-traffic site or a reverse proxy in front of another service — Nginx. The rest of this chapter documents both equally.

Apache

⬡ APACHE HTTP SERVER

Installation

Terminal

```
$ sudo apt update
$ sudo apt install -y apache2
Reading package lists... Done
Setting up apache2 (2.4.57-2) ...

# Apache starts automatically after install. Verify:
$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled)
     Active: active (running) since Tue 2026-06-17 10:00:00 BST

# Test from the server itself
$ curl -s http://localhost | grep -o "<title>.*</title>"
<title>Apache2 Debian Default Page: It works</title>
```

Open `http://YOUR_SERVER_IP` in a browser — you should see the Apache2 Debian Default Page. That page lives at `/var/www/html/index.html` and confirms Apache is serving files.

Directory Structure

```
/etc/apache2/
├── apache2.conf          # Main config — global settings
├── ports.conf            # Which ports Apache listens on (80, 443)
├── sites-available/      # All virtual host config files live here
│   ├── 000-default.conf  # Default site (port 80)
│   └── default-ssl.conf  # Default HTTPS site (disabled by default)
├── sites-enabled/        # Symlinks to active sites in sites-available
│   └── 000-default.conf  # → ../sites-available/000-default.conf
├── mods-available/       # All available modules (.load + .conf pairs)
├── mods-enabled/         # Symlinks to active modules
└── conf-available/       # Additional config snippets

/var/www/html/            # Default document root
/var/log/apache2/
├── access.log            # Every request logged here
└── error.log             # Errors and warnings
```

> **sites-available vs sites-enabled:** Apache uses a symlink pattern — config files live in `sites-available/` and a symlink in `sites-enabled/` activates them. Never edit files in `sites-enabled/` directly. Use `a2ensite` / `a2dissite` to manage the symlinks automatically.

Useful Modules to Enable

Terminal — enable essential modules

```
# mod_rewrite — required for clean URLs and .htaccess rewrites
$ sudo a2enmod rewrite

# mod_headers — send custom HTTP headers (security headers, CORS)
$ sudo a2enmod headers

# mod_ssl — HTTPS support (needed for Chapter 5)
$ sudo a2enmod ssl
```

```
# mod_deflate — gzip compression for faster page loads
$ sudo a2enmod deflate

# Apply all module changes
$ sudo systemctl restart apache2

# Verify a module is enabled
$ apache2ctl -M | grep rewrite
 rewrite_module (shared)
```

Configuring the Default Site

The default virtual host config controls what Apache serves when no other virtual host matches. Edit it to point to your actual site files:

Terminal

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

```
# /etc/apache2/sites-available/000-default.conf
<VirtualHost *:80>
    ServerAdmin   webmaster@osztromok.com
    ServerName    osztromok.com
    ServerAlias   www.osztromok.com
    DocumentRoot  /var/www/html

    <Directory /var/www/html>
        Options       -Indexes +FollowSymLinks
        AllowOverride All       # Allows .htaccess to work
        Require       all granted
    </Directory>

    ErrorLog  ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Terminal — test config and reload

```
# Always test config syntax before reloading — catches typos
$ sudo apache2ctl configtest
```

```
Syntax OK

$ sudo systemctl reload apache2
```

Creating a Named Virtual Host

If you want to host multiple sites on one machine, or simply keep your config tidy, create a dedicated virtual host file for each site:

Terminal

```
# Create the document root for the new site
$ sudo mkdir -p /var/www/osztromok.com
$ sudo chown -R $USER:$USER /var/www/osztromok.com

# Create the virtual host config
$ sudo vi /etc/apache2/sites-available/osztromok.com.conf
```

```
# /etc/apache2/sites-available/osztromok.com.conf
<VirtualHost *:80>
    ServerName    osztromok.com
    ServerAlias   www.osztromok.com
    DocumentRoot  /var/www/osztromok.com

    <Directory /var/www/osztromok.com>
        Options       -Indexes +FollowSymLinks
        AllowOverride All
        Require       all granted
    </Directory>

    ErrorLog  ${APACHE_LOG_DIR}/osztromok.com-error.log
    CustomLog ${APACHE_LOG_DIR}/osztromok.com-access.log combined
</VirtualHost>
```

Terminal — enable the new site

```
# Enable the new site (creates the symlink in sites-enabled/)
$ sudo a2ensite osztromok.com.conf
Enabling site osztromok.com.
To activate the new configuration, you need to run:
```

```
  systemctl reload apache2

# Disable the default site if this is your only site
$ sudo a2dissite 000-default.conf

$ sudo apache2ctl configtest && sudo systemctl reload apache2
Syntax OK
```

Apache Management Commands

Task	Command
Start Apache	`sudo systemctl start apache2`
Stop Apache	`sudo systemctl stop apache2`
Restart (full restart)	`sudo systemctl restart apache2`
Reload (no downtime)	`sudo systemctl reload apache2`
Enable at boot	`sudo systemctl enable apache2`
Test config syntax	`sudo apache2ctl configtest`
Enable a site	`sudo a2ensite sitename.conf`
Disable a site	`sudo a2dissite sitename.conf`
Enable a module	`sudo a2enmod modulename`
Disable a module	`sudo a2dismod modulename`
List enabled modules	`apache2ctl -M`
View error log (live)	`sudo tail -f /var/log/apache2/error.log`
View access log (live)	`sudo tail -f /var/log/apache2/access.log`
Check Apache version	`apache2 -v`

Nginx

NGINX

> **Important:** Apache and Nginx both listen on port 80 by default. You cannot run both at the same time unless you change one of them to a different port. Stop Apache first if it is running: `sudo`

```
systemctl stop apache2 && sudo systemctl disable apache2
```

Installation

Terminal

```
$ sudo apt update
$ sudo apt install -y nginx
Setting up nginx (1.22.1-9) ...

# Nginx starts automatically. Verify:
$ sudo systemctl status nginx
● nginx.service - A high performance web server
     Active: active (running)

$ curl -s http://localhost | grep -o "<title>.*</title>"
<title>Welcome to nginx!</title>
```

Directory Structure

```
/etc/nginx/
├── nginx.conf          # Main config — worker processes, logging, includes
├── sites-available/    # All server block configs live here
│   └── default          # Default server block
├── sites-enabled/      # Symlinks to active configs
│   └── default          # → ../sites-available/default
├── conf.d/             # Additional config snippets (auto-loaded)
├── modules-available/
└── modules-enabled/

/var/www/html/          # Default document root (same as Apache)
/var/log/nginx/
├── access.log
└── error.log
```

No .htaccess in Nginx. Nginx does not read `.htaccess` files at all — all configuration lives in the server block files under `/etc/nginx/sites-available/`. Rewrite rules that live in `.htaccess` for Apache need to be translated into Nginx `rewrite` or `try_files` directives.

Configuring the Default Server Block

Terminal

```
$ sudo vi /etc/nginx/sites-available/default
```

```
# /etc/nginx/sites-available/default
server {
    listen      80;
    listen      [::]:80;
    server_name osztromok.com www.osztromok.com;

    root  /var/www/html;
    index index.php index.html index.htm;

    # Try the request URI, then as a directory, then fall back to index.php
    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    # Pass PHP requests to PHP-FPM (configured in Chapter 4)
    location ~ \.php$ {
        include         snippets/fastcgi-php.conf;
        fastcgi_pass    unix:/run/php/php8.2-fpm.sock;
    }

    # Deny access to .htaccess files (they do nothing in Nginx
    # but shouldn't be publicly readable)
    location ~ /\.ht {
        deny all;
    }

    error_log  /var/log/nginx/osztromok.com-error.log;
    access_log /var/log/nginx/osztromok.com-access.log;
}
```

Creating a Named Server Block

Terminal

```
# Create a dedicated config file for the site
$ sudo vi /etc/nginx/sites-available/osztromok.com
```

```
# Enable it with a symlink
$ sudo ln -s /etc/nginx/sites-available/osztromok.com /etc/nginx/sites-enabled/

# Disable the default block if this is your only site
$ sudo rm /etc/nginx/sites-enabled/default

# Test config syntax — always do this before reloading
$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful

$ sudo systemctl reload nginx
```

Nginx Management Commands

Task	Command
Start Nginx	`sudo systemctl start nginx`
Stop Nginx	`sudo systemctl stop nginx`
Restart (full restart)	`sudo systemctl restart nginx`
Reload (no downtime)	`sudo systemctl reload nginx`
Enable at boot	`sudo systemctl enable nginx`
Test config syntax	`sudo nginx -t`
Enable a site	`sudo ln -s /etc/nginx/sites-available/name /etc/nginx/sites-enabled/`
Disable a site	`sudo rm /etc/nginx/sites-enabled/name`
View error log (live)	`sudo tail -f /var/log/nginx/error.log`
View access log (live)	`sudo tail -f /var/log/nginx/access.log`
Check Nginx version	`nginx -v`

Side-by-Side Quick Reference

Task	Apache	Nginx
Test config	`apache2ctl configtest`	`nginx -t`

Task	Apache	Nginx
Reload config	`systemctl reload apache2`	`systemctl reload nginx`
Enable a site	`a2ensite name.conf`	`ln -s sites-available/name sites-enabled/`
Disable a site	`a2dissite name.conf`	`rm sites-enabled/name`
Enable a module	`a2enmod modulename`	`(edit nginx.conf / install package)`
Config location	`/etc/apache2/`	`/etc/nginx/`
Default doc root	`/var/www/html/`	`/var/www/html/`
Error log	`/var/log/apache2/error.log`	`/var/log/nginx/error.log`
.htaccess support	`Yes (AllowOverride All)`	`No`
PHP integration	`mod_php or PHP-FPM`	`PHP-FPM only`

Placing your site files: Both web servers default to `/var/www/html/` as the document root. For a real site, either copy or symlink your files there, or update the `DocumentRoot` / `root` directive in the virtual host config to point to wherever your files actually live. Ensure the web server user (`www-data` on Debian) can read the files: `sudo chown -R www-data:www-data /var/www/html`

Next: Chapter 3 — Database

Chapter 3 covers MySQL, MariaDB, and PostgreSQL — installation on Debian 12, running the security hardening script, creating databases and users, and connecting from PHP. All three are documented; for Philip's Learning Blog the existing `setup_database.sql` script is ready to run straight after installation.

CHAPTER 3: DATABASE - MYSQL, MARIADB & POSTGRESQL

Chapter 3 — Database

Every dynamic website needs somewhere to store data — user accounts, content, configuration. This chapter covers all three major database options available on Debian 12 Bookworm: MySQL, MariaDB, and PostgreSQL. Each section covers installation, initial security hardening, creating a database and user, and verifying the connection. For Philip's Learning Blog the `setup_database.sql` script generated earlier is ready to run immediately after installation.

> **Which should I install?** For a new PHP-based site on Debian, **MariaDB** is the simplest choice — it's in Debian's main repository (no extra repo needed), is a true drop-in replacement for MySQL, and is fully compatible with the `setup_database.sql` and all PDO queries in the existing codebase. MySQL requires adding a third-party repository first. PostgreSQL is the right choice if you need advanced query features, but requires adjusting the PHP connection setup.

MariaDB

MARIADB

Installation

Terminal

```
$ sudo apt update
$ sudo apt install -y mariadb-server
Setting up mariadb-server (1:10.11.6-0+deb12u1) ...

# MariaDB starts automatically. Verify:
$ sudo systemctl status mariadb
● mariadb.service - MariaDB Database Server
     Active: active (running)

$ mariadb --version
mariadb  Ver 15.1 Distrib 10.11.6-MariaDB, for debian-linux-gnu (x86_64)
```

Security Hardening

Always run the security script immediately after installation. It removes anonymous users, disables remote root login, removes the test database, and sets a root password.

Terminal — security hardening

```
$ sudo mariadb-secure-installation

Enter current password for root (enter for none): [press Enter]
Switch to unix_socket authentication [Y/n]: n
Change the root password? [Y/n]: Y
New password: [enter a strong root password]
Re-enter new password: [repeat]
Remove anonymous users? [Y/n]: Y
Disallow root login remotely? [Y/n]: Y
Remove test database and access to it? [Y/n]: Y
Reload privilege tables now? [Y/n]: Y
All done!
```

Creating a Database and User

Terminal — connect as root

```
$ sudo mariadb -u root -p
Enter password:
Welcome to the MariaDB monitor.
MariaDB [(none)]>
```

```
-- Run these inside the MariaDB prompt

-- Create the database
CREATE DATABASE learning_blog
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;

-- Create the application user
CREATE USER 'philip'@'localhost' IDENTIFIED BY 'AsT@1sAd3mon';

-- Grant full access to the database
GRANT ALL PRIVILEGES ON learning_blog.* TO 'philip'@'localhost';
```

```
FLUSH PRIVILEGES;
EXIT;
```

> **For Philip's Learning Blog:** The full `setup_database.sql` script already contains these statements plus all the CREATE TABLE definitions. Run it directly instead: `sudo mariadb -u root -p < setup_database.sql`

Verify the Connection

Terminal

```
# Connect as the application user to confirm it works
$ mariadb -u philip -p learning_blog
Enter password:
MariaDB [learning_blog]> SHOW TABLES;
+------------------------+
| Tables_in_learning_blog |
+------------------------+
| admin_users             |
| page_content            |
| pages                   |
| subjects                |
| subtopics               |
+------------------------+
MariaDB [learning_blog]> EXIT;
```

MariaDB Management Commands

Task	Command
Start MariaDB	`sudo systemctl start mariadb`
Stop MariaDB	`sudo systemctl stop mariadb`
Restart MariaDB	`sudo systemctl restart mariadb`
Enable at boot	`sudo systemctl enable mariadb`
Connect as root	`sudo mariadb -u root -p`
Connect as user	`mariadb -u philip -p dbname`
Run a SQL file	`mariadb -u root -p < file.sql`

Task	Command
Dump a database	`mariadb-dump -u root -p learning_blog > backup.sql`
Check version	`mariadb --version`

MySQL

MYSQL 8

> **MySQL requires a third-party repository.** It is not in Debian's main repo. You must add the official MySQL APT repository before installing. Choose the **Bookworm** package when prompted.

Adding the MySQL Repository

Terminal — add MySQL APT repo

```
# Download the MySQL APT repository config package
$ wget https://dev.mysql.com/get/mysql-apt-config_0.8.29-1_all.deb

# Install it — a dialog will appear asking which MySQL version to use
# Select "MySQL 8.0" then "Ok"
$ sudo dpkg -i mysql-apt-config_0.8.29-1_all.deb

$ sudo apt update
```

Installation

Terminal

```
$ sudo apt install -y mysql-server
Setting up mysql-server (8.0.36-1debian12) ...

# Verify:
$ sudo systemctl status mysql
● mysql.service - MySQL Community Server
     Active: active (running)

$ mysql --version
```

```
mysql  Ver 8.0.36 for Linux on x86_64 (MySQL Community Server - GPL)
```

Security Hardening

Terminal

```
$ sudo mysql_secure_installation

VALIDATE PASSWORD component — would you like to setup? Y
Password validation policy: 1  (MEDIUM — recommended)
New root password: [enter strong password]
Remove anonymous users? Y
Disallow root login remotely? Y
Remove test database? Y
Reload privilege tables? Y
All done!
```

Creating a Database and User

Terminal

```
$ sudo mysql -u root -p
mysql>
```

```
-- MySQL 8 uses caching_sha2_password by default.
-- Specify mysql_native_password for compatibility with older PHP drivers.

CREATE DATABASE learning_blog
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;

CREATE USER 'philip'@'localhost'
    IDENTIFIED WITH mysql_native_password
    BY 'AsT@1sAd3mon';

GRANT ALL PRIVILEGES ON learning_blog.* TO 'philip'@'localhost';

FLUSH PRIVILEGES;
EXIT;
```

> **mysql_native_password vs caching_sha2_password:** MySQL 8 defaults to `caching_sha2_password` which requires an SSL connection or RSA key exchange. Specifying `mysql_native_password` avoids "Authentication plugin not supported" errors with PHP PDO over a standard local socket connection.

MySQL Management Commands

Task	Command
Start MySQL	`sudo systemctl start mysql`
Stop MySQL	`sudo systemctl stop mysql`
Restart MySQL	`sudo systemctl restart mysql`
Enable at boot	`sudo systemctl enable mysql`
Connect as root	`sudo mysql -u root -p`
Connect as user	`mysql -u philip -p dbname`
Run a SQL file	`mysql -u root -p < file.sql`
Dump a database	`mysqldump -u root -p learning_blog > backup.sql`
Check version	`mysql --version`

PostgreSQL

⬡ POSTGRESQL 15

Installation

Terminal

```
# PostgreSQL 15 is in Debian's main repo — no extra repository needed
$ sudo apt update
$ sudo apt install -y postgresql postgresql-contrib
Setting up postgresql-15 (15.6-0+deb12u1) ...

$ sudo systemctl status postgresql
● postgresql.service - PostgreSQL RDBMS
     Active: active (running)
```

```
$ psql --version
psql (PostgreSQL) 15.6 (Debian 15.6-0+deb12u1)
```

How PostgreSQL Authentication Works

PostgreSQL uses **peer authentication** by default for local connections — it maps the Linux system user to a database user of the same name. The database superuser is `postgres`, matching the `postgres` Linux user created during installation. To connect, you switch to that user first:

Terminal — connect as postgres superuser

```
$ sudo -u postgres psql
psql (15.6)
Type "help" for help.
postgres=#
```

Creating a Database and User

```
-- Inside the psql prompt

-- Create the application user with a password
CREATE USER philip WITH PASSWORD 'AsT@1sAd3mon';

-- Create the database owned by that user
CREATE DATABASE learning_blog
    OWNER philip
    ENCODING 'UTF8'
    LC_COLLATE 'en_GB.UTF-8'
    LC_CTYPE   'en_GB.UTF-8'
    TEMPLATE   template0;

-- Grant all privileges
GRANT ALL PRIVILEGES ON DATABASE learning_blog TO philip;

\q
```

Allowing Password Authentication

To connect with a username and password (as PHP will), you need to allow `md5` or `scram-sha-256` authentication in `pg_hba.conf`:

Terminal

```
$ sudo vi /etc/postgresql/15/main/pg_hba.conf
# Find the line for local connections and change "peer" to "md5":
# TYPE  DATABASE  USER  ADDRESS  METHOD
local   all       all           md5    ← change from "peer"

$ sudo systemctl restart postgresql

# Now connect with password authentication
$ psql -U philip -d learning_blog -W
Password:
psql (15.6)
learning_blog=>
```

PHP PDO Connection String for PostgreSQL

PostgreSQL uses a different PDO driver than MySQL/MariaDB. Update `config.php` if switching to PostgreSQL:

```php
// config.php — PostgreSQL version
define('DB_HOST',    'localhost');
define('DB_NAME',    'learning_blog');
define('DB_USER',    'philip');
define('DB_PASS',    'AsT@1sAd3mon');

// DSN changes from mysql: to pgsql:
$dsn = 'pgsql:host=' . DB_HOST . ';dbname=' . DB_NAME;
$pdo = new PDO($dsn, DB_USER, DB_PASS);
```

PostgreSQL Management Commands

Task	Command
Start PostgreSQL	`sudo systemctl start postgresql`
Stop PostgreSQL	`sudo systemctl stop postgresql`
Restart PostgreSQL	`sudo systemctl restart postgresql`

Task	Command
Enable at boot	`sudo systemctl enable postgresql`
Connect as superuser	`sudo -u postgres psql`
Connect as user	`psql -U philip -d learning_blog -W`
Run a SQL file	`psql -U philip -d learning_blog -f file.sql`
Dump a database	`pg_dump -U philip learning_blog > backup.sql`
List databases	`\l (inside psql)`
List tables	`\dt (inside psql)`
Check version	`psql --version`

Side-by-Side Quick Reference

Task	MariaDB / MySQL	PostgreSQL
Connect as superuser	`sudo mariadb -u root -p`	`sudo -u postgres psql`
Create database	`CREATE DATABASE name;`	`CREATE DATABASE name OWNER user;`
Create user	`CREATE USER 'u'@'localhost' IDENTIFIED BY 'pw';`	`CREATE USER u WITH PASSWORD 'pw';`
Grant access	`GRANT ALL ON db.* TO 'u'@'localhost';`	`GRANT ALL PRIVILEGES ON DATABASE db TO u;`
List databases	`SHOW DATABASES;`	`\l`
List tables	`SHOW TABLES;`	`\dt`
Dump database	`mysqldump / mariadb-dump -u root -p db > bk.sql`	`pg_dump -U user db > bk.sql`
Restore database	`mysql / mariadb -u root -p db < bk.sql`	`psql -U user -d db -f bk.sql`
PHP PDO prefix	`mysql:host=...;dbname=...`	`pgsql:host=...;dbname=...`

Task	MariaDB / MySQL	PostgreSQL
Config file	`/etc/mysql/mariadb.conf.d/`	`/etc/postgresql/15/main/`

Next: Chapter 4 — PHP

Chapter 4 covers installing PHP 8.2 on Debian 12, configuring `php.ini` for a production web server, installing the essential extensions your site needs (pdo_mysql, mbstring, curl, gd and more), and verifying everything works with a live test script.

CHAPTER 4: PHP

Chapter 4 — PHP

PHP is the glue between your web server and your database. This chapter covers two installation paths — the version that ships with Debian 12 Bookworm's main repository (PHP 8.2) and the newer PHP 8.3 available via the Sury third-party repository — along with the extensions you'll need for a real-world site, the key configuration settings to change from their defaults, and how to verify everything is working for both Apache and Nginx.

> **Which version?** PHP 8.2 (Debian main repo) is stable, well-supported, and the simplest to install — one command, no extra repository. PHP 8.3 adds typed class constants, readonly properties in cloned objects, and minor performance gains. For Philip's Learning Blog, **8.2 is the recommended choice**; upgrading to 8.3 later is straightforward via the Sury repo.

Option A — PHP 8.2 (Debian Main Repository)

Installation

PHP 8.2 is available directly from Debian's main repository — no GPG keys or third-party sources to configure. The package you need depends on your web server.

◆ PHP 8.2 · DEBIAN MAIN

For Apache (mod_php)

philip@debian — PHP 8.2 install (Apache)

```
philip@debian:~$ sudo apt install -y php php-cli php-mysql php-mbstring php-curl \
    php-gd php-xml php-zip php-intl php-bcmath
Reading package lists... Done
The following NEW packages will be installed:
  php php-cli php-common php-mysql php-mbstring ... (and dependencies)
Setting up php8.2 (8.2.x-1+deb12uN) ...
Setting up libapache2-mod-php8.2 (8.2.x-1+deb12uN) ...
```

```
philip@debian:~$ sudo systemctl restart apache2
```

For Nginx (PHP-FPM)

philip@debian — PHP 8.2 install (Nginx + FPM)

```
philip@debian:~$ sudo apt install -y php-fpm php-cli php-mysql php-mbstring php-curl \
    php-gd php-xml php-zip php-intl php-bcmath
Setting up php8.2-fpm (8.2.x-1+deb12uN) ...

philip@debian:~$ sudo systemctl enable --now php8.2-fpm
philip@debian:~$ sudo systemctl restart nginx
```

Option B — PHP 8.3 (Sury Repository)

The deb.sury.org repository is maintained by Ondřej Surý, a Debian Developer who maintains the official PHP packages. It is widely used in production and is the standard way to get PHP versions newer than what Debian ships.

◆ PHP 8.3 · SURY REPOSITORY

philip@debian — PHP 8.3 via Sury

```
philip@debian:~$ sudo apt install -y apt-transport-https lsb-release ca-certificates cur

philip@debian:~$ curl -fsSLo /tmp/debsuryorg-archive-keyring.deb \
    https://packages.sury.org/debsuryorg-archive-keyring.deb
philip@debian:~$ sudo dpkg -i /tmp/debsuryorg-archive-keyring.deb

philip@debian:~$ echo "deb https://packages.sury.org/php/ $(lsb_release -sc) main" \
    | sudo tee /etc/apt/sources.list.d/php.list
deb https://packages.sury.org/php/ bookworm main

philip@debian:~$ sudo apt update

philip@debian:~$ sudo apt install -y php8.3 php8.3-cli php8.3-fpm php8.3-mysql \
```

```
    php8.3-mbstring php8.3-curl php8.3-gd php8.3-xml php8.3-zip \
    php8.3-intl php8.3-bcmath
Setting up php8.3-fpm (8.3.x+1) ...

philip@debian:~$ sudo update-alternatives --set php /usr/bin/php8.3
```

> **Version pinning:** if both 8.2 and 8.3 packages are installed, `update-alternatives --set php /usr/bin/php8.3` sets which binary `php` calls on the command line. The web server module (mod_php or FPM) is controlled separately — make sure Apache/Nginx points at the FPM socket for your chosen version.

Essential Extensions

A bare PHP install will process scripts, but you'll need extensions for database access, image handling, and character encoding. Below are the extensions used by Philip's Learning Blog and other common PHP sites.

php-mysql ESSENTIAL
PDO MySQL driver and MySQLi — required for any MySQL / MariaDB connection.

php-mbstring ESSENTIAL
Multi-byte string functions — required by most frameworks. Without it, Unicode text handling breaks.

php-curl ESSENTIAL
HTTP requests from PHP — needed for any external API calls (Anthropic, payment gateways, etc.).

php-xml ESSENTIAL
XML/DOM parsing — required by Composer, PHPUnit, and many libraries.

php-gd USEFUL
Image creation and manipulation — resize thumbnails, add watermarks, generate CAPTCHA images.

php-zip USEFUL
Read/write ZIP archives — required by Composer to install packages from GitHub.

php-intl USEFUL
Internationalisation — locale-aware number/date formatting and transliteration.

php-bcmath USEFUL
Arbitrary precision maths — avoids floating-point rounding errors in financial calculations.

php-pgsql POSTGRESQL
PDO PostgreSQL driver — only needed if using PostgreSQL instead of MySQL/MariaDB.

php-redis OPTIONAL
Redis client — session storage, caching. Install Redis server separately first.

php-imagick OPTIONAL
ImageMagick bindings — more powerful than GD for image conversion and PDF thumbnails.

php-opcache PERFORMANCE
Caches compiled PHP bytecode in memory — enabled by default in Debian packages, speeds up every request.

Checking installed extensions: run `php -m` to list all loaded modules, or `php -m | grep -i mysql` to confirm a specific one. The output from `phpinfo()` in a browser shows the full picture including configuration values.

Configuration — php.ini

PHP uses separate `php.ini` files for CLI and for each SAPI (Apache mod_php and PHP-FPM have their own). The paths depend on the PHP version.

Context	php.ini path (PHP 8.2)	php.ini path (PHP 8.3)
Apache (mod_php)	`/etc/php/8.2/apache2/php.ini`	`/etc/php/8.3/apache2/php.ini`
PHP-FPM (Nginx)	`/etc/php/8.2/fpm/php.ini`	`/etc/php/8.3/fpm/php.ini`
CLI (command line)	`/etc/php/8.2/cli/php.ini`	`/etc/php/8.3/cli/php.ini`

Recommended Changes

Open the appropriate `php.ini` for your web server SAPI and adjust these values. The defaults are conservative — fine for shared hosting, too restrictive for a real development machine.

philip@debian — edit php.ini (Apache example)

```
philip@debian:~$ sudo vi /etc/php/8.2/apache2/php.ini
```

Setting	Default	Recommended	Why
`memory_limit`	128M	256M	Enough headroom for image processing and Composer installs.
`upload_max_filesize`	2M	20M	Allows uploading images and documents without hitting the limit.
`post_max_size`	8M	25M	Must be larger than `upload_max_filesize` — controls total POST body.
`max_execution_time`	30	60	Prevents slow scripts from hanging the server indefinitely.
`display_errors`	Off	On *(dev only)*	Show errors in the browser during development. Set to `Off` on production.

Setting	Default	Recommended	Why
`error_reporting`	E_ALL & ~E_DEPRECATED	E_ALL	Catch everything including deprecation notices during development.
`date.timezone`	(empty)	Europe/London	Avoids "It is not safe to rely on the system's timezone" warnings.

> **Production warning:** `display_errors = On` leaks internal paths and logic to the browser. Always set it to `Off` on a publicly accessible server, and log errors to a file instead using `log_errors = On` and `error_log = /var/log/php_errors.log`.

Connecting PHP to Apache

When you install `php` (or `libapache2-mod-php8.2`), the module is enabled automatically. You can verify this and ensure the right module is active:

philip@debian — Apache + mod_php

```
philip@debian:~$ apache2ctl -M | grep php
php8.2_module (shared)

philip@debian:~$ sudo systemctl restart apache2
```

No virtual host changes are needed — Apache handles `.php` files automatically via mod_php as long as the module is loaded.

Connecting PHP to Nginx (PHP-FPM)

Nginx does not embed a PHP interpreter. Instead it passes `.php` requests to a separate **PHP-FPM** (FastCGI Process Manager) process via a Unix socket. Your Nginx server block needs a `location ~ \.php$` block pointing at that socket.

Verify the FPM socket path

philip@debian — find FPM socket

```
philip@debian:~$ ls /run/php/
php8.2-fpm.pid  php8.2-fpm.sock
```

Update the Nginx server block

Add or confirm the PHP location block inside your `server {}` block. This is the block introduced in Chapter 2 for the Nginx virtual host.

```
# /etc/nginx/sites-available/osztromok.com

server {
    listen 80;
    server_name osztromok.com www.osztromok.com;
    root /var/www/osztromok.com/public_html;
    index index.php index.html;

    location / {
        try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.2-fpm.sock;
    }

    location ~ /\.ht {
        deny all;
    }
}
```

philip@debian — reload Nginx

```
philip@debian:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful

philip@debian:~$ sudo systemctl reload nginx
```

Testing the PHP Installation

Step 1 — phpinfo() test page

Create a temporary test file in your web root to confirm PHP is executing:

philip@debian — create test page

```
philip@debian:~$ echo '<?php phpinfo(); ?>' | sudo tee /var/www/osztromok.com/public_html/inf
```

Visit `http://<your-server-ip>/info.php` in a browser. You should see the purple PHP information page listing your version, loaded extensions, and all configuration values.

> **Remove it immediately after testing.** The `phpinfo()` page exposes your server configuration to anyone who can reach it. `sudo rm /var/www/osztromok.com/public_html/info.php`

Step 2 — Command-line version check

philip@debian — CLI checks

```
philip@debian:~$ php --version
PHP 8.2.20 (cli) (built: Jun  4 2025 07:35:12) (NTS)
Copyright (c) The PHP Group

philip@debian:~$ php -m | grep -E 'mysql|mbstring|curl|gd|xml|zip'
curl
gd
mbstring
mysqlnd
pdo_mysql
SimpleXML
xml
zip
```

Step 3 — PHP-to-database connection test

This quick script confirms PHP can actually open a PDO connection to your MariaDB / MySQL database. Run it from the CLI — no browser needed.

```php
# /tmp/db_test.php  — delete after testing

<?php
$dsn  = 'mysql:host=localhost;dbname=learning_blog;charset=utf8mb4';
$user = 'philip';
$pass = 'AsT@1sAd3mon';

try {
    $pdo = new PDO($dsn, $user, $pass, [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    ]);
    $ver = $pdo->query('SELECT VERSION()')->fetchColumn();
    echo "Connected OK — database version: $ver\n";
} catch (PDOException $e) {
    echo "Connection failed: " . $e->getMessage() . "\n";
}
?>
```

philip@debian — run connection test

```
philip@debian:~$ php /tmp/db_test.php
Connected OK — database version: 10.11.6-MariaDB-2

philip@debian:~$ rm /tmp/db_test.php
```

Installing Composer

Composer is PHP's dependency manager — the equivalent of `npm` for Node. It's not strictly required to run Philip's existing site, but it's needed for any modern PHP framework (Laravel, Symfony, FastAPI-style microframeworks like Slim) or if you install libraries via `require` statements.

philip@debian — install Composer globally

```
philip@debian:~$ curl -sS https://getcomposer.org/installer | php
All settings correct for using Composer
Downloading...
```

```
Composer (version 2.x.x) successfully installed to: /home/philip/composer.phar

philip@debian:~$ sudo mv composer.phar /usr/local/bin/composer
philip@debian:~$ composer --version
Composer version 2.x.x  2025-xx-xx xx:xx:xx
```

PHP Management Commands

Task	Apache (mod_php)	Nginx (PHP-FPM)
Restart PHP after config change	`sudo systemctl restart apache2`	`sudo systemctl restart php8.2-fpm`
Check PHP-FPM status	`apache2ctl -M | grep php`	`sudo systemctl status php8.2-fpm`
View FPM error log	—	`sudo tail -f /var/log/php8.2-fpm.log`
List installed extensions	`php -m`	`php -m`
Install a new extension	`sudo apt install php-<name>`	`sudo apt install php8.2-<name>`
Confirm active php.ini	`php --ini`	`php --ini`
OPcache status	`php -r "echo opcache_get_status()['opcache_enabled'] ? 'enabled' : 'disabled';"`	

Quick-Verification Checklist

- **PHP version matches expectation** — `php --version` shows 8.2.x or 8.3.x.
- **Browser test page works** — `info.php` returns the PHP purple page, then removed.
- **All required extensions loaded** — `php -m` shows mysql, mbstring, curl, gd, xml, zip.
- **Database connection succeeds** — `db_test.php` prints "Connected OK", then removed.
- **OPcache enabled** — improves every page load with zero code changes.
- **php.ini tuned for environment** — memory, upload limits, timezone set, display_errors correct for dev/prod.
- **FPM socket active (Nginx only)** — `/run/php/php8.2-fpm.sock` exists, Nginx reload clean.

Next — Chapter 5: SSL / HTTPS. With PHP serving dynamic pages, the next step is encrypting traffic. Chapter 5 covers obtaining a free Let's Encrypt certificate with Certbot (for public-facing domains) and generating a self-signed certificate (for local development), with automatic HTTP → HTTPS redirect for both Apache and Nginx.

CHAPTER 5: SSL/HTTPS

Chapter 5 — SSL / HTTPS

Running a website over plain HTTP means every request and response — including login credentials — travels as readable text. SSL/TLS encrypts the connection, gives visitors the padlock icon in the browser, and is required for any site that handles passwords or personal data. This chapter covers two scenarios: obtaining a free, trusted certificate from **Let's Encrypt** for a publicly accessible domain, and generating a **self-signed certificate** for local development where no external CA is needed.

LET'S ENCRYPT + CERTBOT

Free, publicly trusted certificate. Browser shows a green padlock. Requires a real domain name pointed at your server and port 80 open to the internet. Auto-renews every 90 days. **Use this for any public-facing site.**

SELF-SIGNED CERTIFICATE

Generated locally in seconds. Browser shows a warning ("not trusted") because no CA has verified it. No domain or internet required. **Use this for local development only** — never for a site visitors will use.

Option A — Let's Encrypt with Certbot

Prerequisites: your domain (e.g. `osztromok.com`) must have its DNS A record pointing at this server's public IP address, and port 80 must be reachable from the internet. Certbot uses an HTTP challenge to verify you control the domain before issuing the certificate. If your server is behind Cloudflare Tunnel, see the note below.

◆ LET'S ENCRYPT · APACHE

Install Certbot

philip@debian — install Certbot for Apache

```
philip@debian:~$ sudo apt install -y certbot python3-certbot-apache
Setting up certbot (2.1.0-1) ...
Setting up python3-certbot-apache (2.1.0-1) ...
```

Obtain and install the certificate

philip@debian — run Certbot

```
philip@debian:~$ sudo certbot --apache -d osztromok.com -d www.osztromok.com
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Enter email address (used for urgent renewal and security notices):
 emubantam@gmail.com
Please read the Terms of Service at https://letsencrypt.org/documents/LE-SA...
(A)gree/(C)ancel: A
Would you be willing to share your email address with EFF? (Y)es/(N)o: N
Account registered.
Requesting a certificate for osztromok.com and www.osztromok.com
Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/osztromok.com/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/osztromok.com/privkey.pem
Please choose whether or not to redirect HTTP traffic to HTTPS:
1: No redirect
2: Redirect - Make all requests redirect to secure HTTPS access
2
Redirecting all traffic on port 80 to ssl in /etc/apache2/sites-enabled/osztromok.com.co
Congratulations! You have successfully enabled HTTPS on https://osztromok.com
```

Certbot edits your Apache virtual host automatically — adding the SSL directives and an HTTP → HTTPS redirect. It also enables `mod_ssl` and `mod_rewrite` if they weren't already active. You don't need to touch the virtual host file manually.

◆ LET'S ENCRYPT · NGINX

Install Certbot for Nginx

philip@debian — install Certbot for Nginx

```
philip@debian:~$ sudo apt install -y certbot python3-certbot-nginx
Setting up python3-certbot-nginx (2.1.0-1) ...

philip@debian:~$ sudo certbot --nginx -d osztromok.com -d www.osztromok.com
Requesting a certificate for osztromok.com and www.osztromok.com
Successfully received certificate.
```

```
Deploying certificate to VirtualHost /etc/nginx/sites-enabled/osztromok.com
Please choose whether or not to redirect HTTP traffic to HTTPS:
2
Congratulations! HTTPS enabled at https://osztromok.com
```

What Certbot Creates

After a successful run, Let's Encrypt certificates are stored in `/etc/letsencrypt/live/<domain>/`. These are symlinks pointing to the actual versioned files in `/etc/letsencrypt/archive/`.

File	Purpose	Used in config as
`fullchain.pem`	Your certificate + the Let's Encrypt intermediate chain	`SSLCertificateFile` / `ssl_certificate`
`privkey.pem`	Your private key — keep this secret	`SSLCertificateKeyFile` / `ssl_certificate_key`
`cert.pem`	Your certificate only (without chain) — rarely used directly	—
`chain.pem`	The intermediate chain only — rarely used directly	—

Manual HTTP → HTTPS Redirect

If you chose not to let Certbot add the redirect, or if you want to review exactly what it looks like, here are the configurations for both web servers:

Apache — manual redirect

```
# /etc/apache2/sites-available/osztromok.com.conf

# HTTP — redirect everything to HTTPS
<VirtualHost *:80>
    ServerName  osztromok.com
    ServerAlias www.osztromok.com
    Redirect permanent / https://osztromok.com/
</VirtualHost>

# HTTPS — the real site
```

```
<VirtualHost *:443>
    ServerName  osztromok.com
    ServerAlias www.osztromok.com
    DocumentRoot /var/www/osztromok.com/public_html

    SSLEngine on
    SSLCertificateFile    /etc/letsencrypt/live/osztromok.com/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/osztromok.com/privkey.pem

    <Directory /var/www/osztromok.com/public_html>
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

Nginx — manual redirect

```
# /etc/nginx/sites-available/osztromok.com

# HTTP — redirect to HTTPS
server {
    listen 80;
    server_name osztromok.com www.osztromok.com;
    return 301 https://$host$request_uri;
}

# HTTPS — the real site
server {
    listen 443 ssl;
    server_name osztromok.com www.osztromok.com;
    root /var/www/osztromok.com/public_html;
    index index.php index.html;

    ssl_certificate     /etc/letsencrypt/live/osztromok.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/osztromok.com/privkey.pem;
    include             /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam         /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.2-fpm.sock;
```

```
    }
}
```

Auto-Renewal

Let's Encrypt certificates expire after 90 days. Certbot installs a systemd timer that checks for renewal twice a day automatically — you don't need to do anything after the initial setup. You can verify the timer is running and test the renewal process with a dry run:

philip@debian — verify auto-renewal

```
# Check the renewal timer is active
philip@debian:~$ sudo systemctl status certbot.timer
● certbot.timer - Run certbot twice daily
     Loaded: loaded (/lib/systemd/system/certbot.timer; enabled)
     Active: active (waiting)
    Trigger: Mon 2026-06-18 00:00:00 UTC; 11h left

# Dry run — confirms renewal will work without actually renewing
philip@debian:~$ sudo certbot renew --dry-run
Simulating renewal of an existing certificate for osztromok.com
Congratulations, all simulated renewals succeeded.

# Check expiry of current certificate
philip@debian:~$ sudo certbot certificates
Found the following certs:
  Certificate Name: osztromok.com
  Domains: osztromok.com www.osztromok.com
  Expiry Date: 2026-09-15 (VALID: 89 days)
  Certificate Path: /etc/letsencrypt/live/osztromok.com/fullchain.pem
```

Option B — Self-Signed Certificate (Local Development)

When developing locally — no domain, no internet — a self-signed certificate lets you test HTTPS behaviour without Let's Encrypt. The browser will show a warning that the certificate is not trusted (because it isn't signed by a recognised CA), but you can click through and the connection is still encrypted.

◆ SELF-SIGNED · OPENSSL

Generate the certificate

philip@debian — generate self-signed cert

```
# Create a directory for local certs
philip@debian:~$ sudo mkdir -p /etc/ssl/local

# Generate a 2048-bit private key and self-signed certificate (valid 365 days)
philip@debian:~$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
    -keyout /etc/ssl/local/selfsigned.key \
    -out    /etc/ssl/local/selfsigned.crt \
    -subj "/C=GB/ST=England/L=Local/O=Dev/CN=localhost"
Generating a RSA private key
.........+++++
writing new private key to '/etc/ssl/local/selfsigned.key'

# Restrict key permissions
philip@debian:~$ sudo chmod 600 /etc/ssl/local/selfsigned.key
```

Configure Apache to use it

```
# /etc/apache2/sites-available/local-ssl.conf

<VirtualHost *:443>
    ServerName localhost
    DocumentRoot /var/www/html

    SSLEngine on
    SSLCertificateFile    /etc/ssl/local/selfsigned.crt
    SSLCertificateKeyFile /etc/ssl/local/selfsigned.key

    <Directory /var/www/html>
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

philip@debian — enable SSL site

```
philip@debian:~$ sudo a2enmod ssl
philip@debian:~$ sudo a2ensite local-ssl
philip@debian:~$ sudo apache2ctl configtest
Syntax OK
philip@debian:~$ sudo systemctl reload apache2
```

Configure Nginx to use it

```
# /etc/nginx/sites-available/local-ssl

server {
    listen 443 ssl;
    server_name localhost;
    root /var/www/html;
    index index.php index.html;

    ssl_certificate     /etc/ssl/local/selfsigned.crt;
    ssl_certificate_key /etc/ssl/local/selfsigned.key;

    location / {
        try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.2-fpm.sock;
    }
}
```

philip@debian — enable Nginx SSL site

```
philip@debian:~$ sudo ln -s /etc/nginx/sites-available/local-ssl /etc/nginx/sites-enable
philip@debian:~$ sudo nginx -t
nginx: configuration file syntax is ok
philip@debian:~$ sudo systemctl reload nginx
```

Cloudflare Tunnel — A Note

Philip's site uses a Cloudflare Tunnel to route traffic from the public internet to the home server. In this setup the SSL picture is slightly different:

- **Browser → Cloudflare:** always HTTPS — Cloudflare handles this with their own certificate. Visitors get the padlock regardless of what your server does.
- **Cloudflare → your server (tunnel):** traffic travels through the encrypted tunnel — you can run plain HTTP on the server itself if you wish, or add a self-signed cert for the server-side leg.
- **Let's Encrypt on a tunnelled server:** the HTTP-01 challenge Certbot uses requires port 80 to be directly reachable from the internet, which a Cloudflare Tunnel doesn't provide. You would need to use the DNS-01 challenge instead (`certbot certonly --manual --preferred-challenges dns`), which requires adding a TXT record in Cloudflare DNS.

> **Simplest approach with Cloudflare Tunnel:** set Cloudflare SSL/TLS mode to **Full** in the Cloudflare dashboard, generate a self-signed cert for your server (Option B above), and point the tunnel at `https://localhost`. Visitors get trusted HTTPS; your server gets an encrypted tunnel leg with a self-signed cert that Cloudflare accepts in Full mode.

Verify HTTPS is Working

philip@debian — verify certificate

```
# Check certificate details from the command line
philip@debian:~$ echo | openssl s_client -connect osztromok.com:443 2>/dev/null \
    | openssl x509 -noout -dates -subject
notBefore=Jun 17 00:00:00 2026 GMT
notAfter=Sep 15 00:00:00 2026 GMT
subject=CN = osztromok.com

# Check that HTTP redirects to HTTPS
philip@debian:~$ curl -I http://osztromok.com
HTTP/1.1 301 Moved Permanently
Location: https://osztromok.com/
```

SSL Checklist

- **Certificate obtained** — `sudo certbot certificates` shows domain name, expiry date, and cert path.
- **HTTPS loads correctly** — browser shows padlock, no mixed-content warnings.
- **HTTP redirects to HTTPS** — `curl -I http://<domain>` returns 301 pointing at `https://`.
- **Auto-renewal active** — `sudo systemctl status certbot.timer` shows *active (waiting)*.
- **Dry run passes** — `sudo certbot renew --dry-run` completes without errors.
- **mod_ssl enabled (Apache)** — `apache2ctl -M | grep ssl` shows *ssl_module*.
- **Private key permissions** — `ls -la /etc/letsencrypt/live/<domain>/` shows key readable only by root.

Next — Chapter 6: Firewall & Security Basics. With HTTPS in place, Chapter 6 hardens the rest of the server: UFW firewall rules, fail2ban brute-force protection, correct file system permissions on the web root, and stripping version information from HTTP response headers.

CHAPTER 6: FIREWALL & SECURITY BASICS

Chapter 6 — Firewall & Security Basics

A working web stack is also an attack surface. This chapter covers the essential hardening steps for a Debian 12 Bookworm server: controlling inbound traffic with UFW, banning brute-force attackers automatically with fail2ban, setting correct file system permissions on your web root, and stripping the information leaks that tell scanners exactly what software you're running.

> **Scope:** This chapter covers server-level hardening — firewall rules, process-level banning, filesystem permissions, and HTTP header hygiene. SSL/HTTPS (encrypting traffic) is covered in Chapter 5. Application-level security (SQL injection, XSS, CSRF, etc.) is a separate topic covered in the Securing Your Web Server course.

1. UFW — Uncomplicated Firewall

UFW is a frontend for `iptables` that makes managing firewall rules readable. It's not installed by default on Debian 12 but is in the main repository. The principle is simple: **deny everything by default, then explicitly allow only the ports you need.**

> **Allow SSH before enabling UFW.** If you enable UFW without allowing SSH first you will lock yourself out of a remote machine immediately. This must be the first rule you add.

Install and configure UFW

philip@debian — UFW setup

```
philip@debian:~$ sudo apt install -y ufw

philip@debian:~$ sudo ufw default deny incoming
Default incoming policy changed to 'deny'

philip@debian:~$ sudo ufw default allow outgoing
Default outgoing policy changed to 'allow'

# Allow SSH first — do this before enabling the firewall
philip@debian:~$ sudo ufw allow ssh
```

```
Rules updated
Rules updated (v6)

philip@debian:~$ sudo ufw allow http
philip@debian:~$ sudo ufw allow https

# Enable the firewall
philip@debian:~$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup

philip@debian:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW IN    Anywhere
80/tcp                     ALLOW IN    Anywhere
443/tcp                    ALLOW IN    Anywhere
```

Rule reference

Command	Effect	Action
`sudo ufw allow ssh`	Opens port 22 (SSH)	ALLOW
`sudo ufw allow http`	Opens port 80 (HTTP)	ALLOW
`sudo ufw allow https`	Opens port 443 (HTTPS)	ALLOW
`sudo ufw allow 3306`	Opens MySQL port — only if remote DB access needed	ALLOW
`sudo ufw deny 3306`	Blocks MySQL from the internet (correct default)	DENY
`sudo ufw allow from 192.168.1.0/24`	Allow all traffic from local LAN only	ALLOW
`sudo ufw allow from 192.168.1.50 to any port 22`	Allow SSH from one specific IP only	ALLOW
`sudo ufw delete allow http`	Remove a previously added rule	REMOVE
`sudo ufw disable`	Turn off firewall (all traffic passes through)	OFF
`sudo ufw reload`	Reload rules without disabling	RELOAD

> **MySQL firewall rule:** MariaDB and MySQL bind to `127.0.0.1` by default, so they're unreachable from outside even without a UFW rule. Only add a rule for port 3306 if you specifically need remote database

> access — and if you do, restrict it to a known IP address, not `Anywhere`.

2. fail2ban — Automatic Brute-Force Protection

fail2ban watches log files for repeated failed authentication attempts and temporarily bans the offending IP address using `iptables`. Without it, a server exposed to the internet will receive thousands of SSH and HTTP brute-force attempts per day from automated scanners.

Install fail2ban

philip@debian — install fail2ban

```
philip@debian:~$ sudo apt install -y fail2ban
Setting up fail2ban (1.0.2-2) ...

philip@debian:~$ sudo systemctl enable --now fail2ban
philip@debian:~$ sudo systemctl status fail2ban
● fail2ban.service - Fail2Ban Service
     Loaded: loaded (/lib/systemd/system/fail2ban.service; enabled)
     Active: active (running)
```

Configure a local jail

fail2ban reads its main config from `/etc/fail2ban/jail.conf`, but you must **never edit that file directly** — it gets overwritten on package updates. Instead, create a local override file:

philip@debian — create jail.local

```
philip@debian:~$ sudo vi /etc/fail2ban/jail.local
```

```
# /etc/fail2ban/jail.local

[DEFAULT]
# Ban for 1 hour after 5 failed attempts within 10 minutes
bantime  = 3600
findtime = 600
```

```
maxretry = 5

# Your own IP — never ban yourself
ignoreip = 127.0.0.1/8 ::1 192.168.1.0/24

# ── SSH protection ──────────────────────────────────────
[sshd]
enabled  = true
port     = ssh
logpath  = %(sshd_log)s
backend  = %(sshd_backend)s
maxretry = 3

# ── Apache: block rapid 404 scanning ────────────────────
[apache-noscript]
enabled  = true

[apache-overflows]
enabled  = true

# ── Apache: auth failures (htpasswd protected areas) ────
[apache-auth]
enabled  = true

# ── Nginx equivalents (uncomment if using Nginx) ────────
# [nginx-http-auth]
# enabled = true
#
# [nginx-noscript]
# enabled = true
```

philip@debian — reload and verify

```
philip@debian:~$ sudo systemctl restart fail2ban

philip@debian:~$ sudo fail2ban-client status
Status
|- Number of jail:      4
`- Jail list:   apache-auth, apache-noscript, apache-overflows, sshd

philip@debian:~$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  `- Total failed:     0
```

```
`- Actions
   |- Currently banned: 0
   `- Total banned:     0
```

Useful fail2ban commands

Command	What it does
`sudo fail2ban-client status`	List all active jails
`sudo fail2ban-client status sshd`	Show failed attempts and banned IPs for the SSH jail
`sudo fail2ban-client set sshd unbanip 1.2.3.4`	Manually unban an IP address
`sudo fail2ban-client banned`	List all currently banned IPs across all jails
`sudo tail -f /var/log/fail2ban.log`	Watch banning activity in real time

3. File System Permissions

Incorrect permissions on your web root are one of the most common security mistakes on self-hosted servers. The goal is: Apache/Nginx can read your files, PHP-FPM can write to specific upload directories, but the web server process can never write to your PHP source files.

The ownership model

WEB ROOT OWNER

`philip:www-data`

You own the files; the web server group can read them.

DIRECTORIES

`755`

Owner: rwx · Group: r-x · Other: r-x

PHP / HTML FILES

`644`

Owner: rw- · Group: r-- · Other: r--

UPLOAD / CACHE DIRECTORIES

`775 (www-data writable)`

Only directories PHP needs to write to. Not the whole web root.

> **Never use 777.** World-writable files let any process on the system modify your PHP source code. If your shared host or tutorial suggests 777, it's wrong. The correct permission is 775 on specific writable directories, and only when the web server group is `www-data`.

Applying correct permissions

philip@debian — set web root permissions

```
# Set ownership: you own everything, www-data is the group
philip@debian:~$ sudo chown -R philip:www-data /var/www/osztromok.com/public_html

# Directories: 755 — traversable by the web server, not writable
philip@debian:~$ sudo find /var/www/osztromok.com/public_html -type d -exec chmod 755 {} \;

# Files: 644 — readable by web server, writable only by you
philip@debian:~$ sudo find /var/www/osztromok.com/public_html -type f -exec chmod 644 {} \;

# Upload directory (if it exists): writable by www-data
philip@debian:~$ sudo chmod 775 /var/www/osztromok.com/public_html/uploads

# Verify
philip@debian:~$ ls -la /var/www/osztromok.com/public_html/
drwxr-xr-x  philip www-data  admin/
drwxrwxr-x  philip www-data  uploads/
-rw-r--r--  philip www-data  index.php
```

Configuration file protection

Your database credentials live in a PHP config file. That file should never be readable by anyone except you and should never be accessible via the browser.

philip@debian — protect config file

```
# Lock the config file so only you can read it
philip@debian:~$ chmod 600 /var/www/osztromok.com/public_html/includes/config.php

# Better: move config outside the web root entirely
philip@debian:~$ sudo mkdir -p /etc/osztromok
philip@debian:~$ sudo mv /var/www/osztromok.com/public_html/includes/config.php /etc/osztromo
philip@debian:~$ sudo chmod 640 /etc/osztromok/config.php
philip@debian:~$ sudo chown philip:www-data /etc/osztromok/config.php
```

> **Moving config outside the web root** means even if a misconfiguration causes Apache to serve raw PHP files as text, your database password is never exposed — it's in `/etc/osztromok/`, not inside `public_html`.

> Update the `require` path in your PHP files to `require '/etc/osztromok/config.php';`.

4. Hiding Server Information

By default, Apache and Nginx announce their version numbers in HTTP response headers and error pages. This tells automated scanners exactly which software to target. Removing this information is one of the fastest wins in server hardening.

Apache — hide version

philip@debian — Apache security.conf

```
philip@debian:~$ sudo vi /etc/apache2/conf-available/security.conf
```

Find and set (or add) these two directives:

```
# /etc/apache2/conf-available/security.conf

# Show only "Apache" — no version or OS info
ServerTokens Prod

# Suppress the "Apache/2.4.57 (Debian)" line in error pages
ServerSignature Off
```

philip@debian — enable config and reload

```
philip@debian:~$ sudo a2enconf security
philip@debian:~$ sudo systemctl reload apache2

# Verify: Server header should now show only "Apache"
philip@debian:~$ curl -I http://localhost | grep -i server
Server: Apache
```

Nginx — hide version

Add one directive to the `http {}` block in `nginx.conf`:

philip@debian — nginx.conf

```
philip@debian:~$ sudo vi /etc/nginx/nginx.conf
```

```
# Inside the http { } block in nginx.conf
http {
    server_tokens off;    # Hides "nginx/1.22.1" from Server header
    # ... rest of your config
}
```

philip@debian — reload nginx

```
philip@debian:~$ sudo nginx -t && sudo systemctl reload nginx
nginx: configuration file syntax is ok
nginx: configuration file test is successful

philip@debian:~$ curl -I http://localhost | grep -i server
Server: nginx
```

PHP — hide version from HTTP headers

PHP adds an `X-Powered-By: PHP/8.2.x` header to every response unless you turn it off. Edit `php.ini` for your web server SAPI:

```
# /etc/php/8.2/apache2/php.ini  (or /etc/php/8.2/fpm/php.ini for Nginx)

expose_php = Off
```

philip@debian — restart to apply

```
philip@debian:~$ sudo systemctl restart apache2
# or for Nginx:
philip@debian:~$ sudo systemctl restart php8.2-fpm

# Verify — X-Powered-By should be gone
philip@debian:~$ curl -I http://localhost | grep -i powered
```

```
(no output — header removed)
```

5. Additional Quick Wins

Directory Listing

Apache: add `Options -Indexes` to your virtual host. Nginx: remove `autoindex on` (off by default). Prevents browsing directories with no index file.

Hidden Files Exposed

Block `.git`, `.env`, and `.htaccess` files from being served. Apache: `Options FollowSymLinks` + `Require all denied` in a `<FilesMatch>` block.

Unused Apache Modules

Disable modules you don't need: `sudo a2dismod status autoindex`. Each loaded module is potential attack surface and adds memory overhead.

SSH Key Auth Only

Disable password-based SSH login. Set `PasswordAuthentication no` in `/etc/ssh/sshd_config` after confirming key login works.

Block directory listing and dot-files (Apache)

```
# Add inside your <VirtualHost> block or <Directory> directive

Options -Indexes FollowSymLinks

# Block access to hidden files (.git, .env, .htaccess, etc.)
<FilesMatch "(^\.|\.(env|git|sql|log|bak|swp)$)">
    Require all denied
</FilesMatch>
```

SSH hardening (sshd_config)

philip@debian — sshd_config

```
philip@debian:~$ sudo vi /etc/ssh/sshd_config
```

```
# /etc/ssh/sshd_config — key settings to review

# Only allow your user to log in via SSH
AllowUsers         philip
```

```
# Disable root login entirely
PermitRootLogin    no

# Disable password auth once SSH key is set up
PasswordAuthentication no

# Timeout idle sessions after 10 minutes
ClientAliveInterval 300
ClientAliveCountMax 2
```

philip@debian — reload SSH daemon

```
philip@debian:~$ sudo sshd -t
(no output = config is valid)

philip@debian:~$ sudo systemctl reload sshd
```

Confirm key-based login works before disabling password auth. Open a second SSH session and confirm you can log in with your key while the first session is still open. Only then reload sshd. If you lock yourself out of a remote machine you'll need physical console access to fix it.

Security Checklist

- **UFW enabled** — `sudo ufw status verbose` shows SSH, HTTP, HTTPS allowed, everything else denied.
- **MySQL/MariaDB port blocked** — port 3306 not open to the internet unless remote access is required.
- **fail2ban running** — `sudo fail2ban-client status` lists SSH and Apache/Nginx jails as active.
- **jail.local in place** — never edited `jail.conf` directly; custom settings survive package updates.
- **Web root ownership** — `philip:www-data`, directories 755, files 644, upload dirs 775 only where needed.
- **Config file protected** — `config.php` is either 640 or outside the web root entirely.
- **Server version hidden** — `ServerTokens Prod` / `server_tokens off` / `expose_php = Off` all set.
- **Directory listing disabled** — `Options -Indexes` prevents browsing empty directories.

- **Dot-files blocked** — `.env`, `.git`, `.sql` files cannot be fetched via the browser.
- **SSH hardened** — root login disabled, ideally password auth off and key auth confirmed working.

Next — Chapter 7: XAMPP. The final chapter covers XAMPP — the all-in-one Apache + MariaDB + PHP installer. It explains when XAMPP makes sense (local development, quick testing), how to install it on Debian, its directory structure, and the honest trade-offs versus the manual stack built in Chapters 1–6.

CHAPTER 7: XAMPP

Chapter 7 — XAMPP

All six previous chapters built a web server stack component by component — web server, database, PHP, SSL, firewall. XAMPP takes the opposite approach: one installer, everything bundled, running in minutes. This final chapter explains what XAMPP is, how to install it on Debian 12 Bookworm, what its directory structure looks like, and — critically — when it's the right tool and when it absolutely isn't.

> **Verdict up front:** XAMPP is excellent for local development and quick testing on a machine that isn't exposed to the internet. It is **not suitable for a public-facing production server.** If your Debian machine is hosting a live website, use the manual stack from Chapters 1–6.

What Is XAMPP?

XAMPP stands for **X** (cross-platform), **A**pache, **M**ariaDB, **P**HP, and **P**erl. It's an open-source package maintained by Apache Friends that bundles all of these components into a single self-contained installation under `/opt/lampp/`. It was originally designed to let developers run a local copy of a LAMP stack on Windows or macOS without touching their system packages — the Linux version is a secondary use case.

Everything XAMPP installs lives inside `/opt/lampp` and runs as its own processes. It does not interact with your system Apache, MariaDB, or PHP packages — they coexist independently, on different ports if needed.

Installing XAMPP on Debian 12

Step 1 — Download the installer

XAMPP is not in Debian's package repository. Download the Linux installer directly from the Apache Friends website. Choose the version that matches your PHP requirement — XAMPP bundles its own PHP version independent of the system.

philip@debian — download XAMPP installer

```
# Check https://www.apachefriends.org for the current version number
# Example below uses 8.2.12 — substitute the actual current version

philip@debian:~$ cd /tmp
philip@debian:/tmp$ wget https://downloadsapachefriends.global.ssl.fastly.net/xampp-linux-x64
Saving to: 'xampp-linux-x64-8.2.12-0-installer.run'
... 156 MB/s eta 0s
'xampp-linux-x64-8.2.12-0-installer.run' saved [165823000/165823000]
```

Step 2 — Make it executable and run

philip@debian — run the installer

```
philip@debian:/tmp$ chmod +x xampp-linux-x64-8.2.12-0-installer.run
philip@debian:/tmp$ sudo ./xampp-linux-x64-8.2.12-0-installer.run
Welcome to the XAMPP Setup Wizard.
Select components to install:
  [x] XAMPP Developer Files
  [x] XAMPP Core Files (required)
  [x] phpMyAdmin
Installation Directory: /opt/lampp
Setup has finished installing XAMPP on your computer.
```

Headless install (no GUI): on a server without a desktop environment, add the `--mode text` flag to run the installer in the terminal:

`sudo ./xampp-linux-x64-8.2.12-0-installer.run --mode text`

Step 3 — Start and stop XAMPP

philip@debian — XAMPP control

```
philip@debian:~$ sudo /opt/lampp/lampp start
Starting XAMPP for Linux 8.2.12-0...
XAMPP: Starting Apache...ok.
XAMPP: Starting MySQL...ok.
XAMPP: Starting ProFTPD...ok.
```

```
philip@debian:~$ sudo /opt/lampp/lampp stop
Stopping XAMPP for Linux 8.2.12-0...
XAMPP: Stopping Apache...ok.
XAMPP: Stopping MySQL...ok.

philip@debian:~$ sudo /opt/lampp/lampp status
Apache is running.
MySQL is running.
ProFTPD is not running.
```

Step 4 — Access the XAMPP dashboard

Open a browser and navigate to `http://localhost`. You should see the XAMPP welcome page with links to phpMyAdmin and the dashboard. Your web files go in `/opt/lampp/htdocs/`.

Directory Structure

Everything lives under `/opt/lampp/`. This self-contained layout is intentional — it's easy to back up, easy to delete, and never touches system directories.

```
/opt/lampp/ ├── apache2/ Apache binaries and modules │ └── conf/ httpd.conf and extra/
├── etc/ php.ini and my.cnf live here │ ├── php.ini │ └── my.cnf ├── htdocs/ ← your web
files go here │ └── dashboard/ XAMPP welcome page ├── mysql/ MariaDB data directory │
└── data/ ├── php/ PHP binary and extensions ├── phpmyadmin/ phpMyAdmin (served at
/phpmyadmin) ├── logs/ Apache error.log and access.log └── lampp The control script
(start/stop/status)
```

> **Port conflicts:** if your system Apache is already running on port 80, XAMPP's Apache will fail to start. Either stop the system service first (`sudo systemctl stop apache2`) or configure XAMPP to listen on a different port by editing `/opt/lampp/etc/httpd.conf` — change `Listen 80` to `Listen 8080` and access via `http://localhost:8080`.

Initial Security Steps

Out of the box, XAMPP ships with its MariaDB root account having no password, phpMyAdmin accessible to anyone on the network, and an FTP server running by default. Run the XAMPP security

script immediately after installation:

philip@debian — XAMPP security wizard

```
philip@debian:~$ sudo /opt/lampp/lampp security
XAMPP: Quick security check...
XAMPP: MySQL is accessible via network. Recommended: configure network access.
XAMPP: The MySQL root user has no password set. Please change this.
Do you want to set a password for the MySQL root? [yes] yes
Password: ****
Do you want to allow access to phpMyAdmin from the network? [no] no
XAMPP: Security improvements complete.
```

> **XAMPP is not hardened for the internet.** The security script above is for local-only use. Even after running it, XAMPP lacks the UFW rules, fail2ban, and per-file permission model built in Chapters 5–6. For a production server, use the manual stack.

XAMPP vs Manual Stack — Honest Comparison

Factor	XAMPP	Manual Stack (Chs 1–6)
Setup time	**5–10 minutes**	1–2 hours (first time)
System integration	Self-contained in /opt/lampp — not managed by systemd, apt, or UFW	**Full systemd control, apt updates, UFW integration**
Security defaults	No root password, phpMyAdmin open, FTP running — must run security wizard	**mysql_secure_installation, UFW, fail2ban all covered in this course**
PHP version control	Tied to the version bundled with the XAMPP installer	**Install any PHP version via Debian or Sury repo; switch with update-alternatives**
Updates	Manual — download a new installer; no `apt upgrade`	`sudo apt upgrade` **patches everything**
Multiple sites	Awkward — one htdocs folder, no clean virtual-host management	**sites-available / sites-enabled per domain**
Production use	Not recommended — not designed or hardened for this	**The correct choice for any publicly accessible server**
phpMyAdmin	**Bundled and pre-configured**	Install separately via apt or Composer

Factor	XAMPP	Manual Stack (Chs 1–6)
Windows/macOS parity	**Identical experience across all platforms — good for team setups**	Linux-specific; developers on other OSes need their own setup
Learning value	Hides how the stack fits together	**You understand every component — easier to debug**
Uninstall	`sudo /opt/lampp/uninstall` **— clean removal**	Manual — `apt remove` each package

When to Use XAMPP vs the Manual Stack

USE XAMPP WHEN...

You need a **local dev environment up in 10 minutes** with no internet exposure. Testing a PHP script, evaluating a CMS, or demoing something to a client on a laptop. Cross-platform team where some members use Windows or macOS — XAMPP gives everyone the same setup. You want phpMyAdmin without any additional configuration.

USE THE MANUAL STACK WHEN...

The server is **publicly accessible** — even behind a Cloudflare Tunnel. You want automatic security updates via `apt upgrade`. You need to host multiple domains with separate virtual hosts. You're building a long-term setup you'll maintain and understand. This course's stack (Chapters 1–6) is the correct choice for Philip's Debian home server.

Useful XAMPP Commands

Command	What it does
`sudo /opt/lampp/lampp start`	Start Apache, MariaDB, ProFTPD
`sudo /opt/lampp/lampp stop`	Stop all XAMPP services
`sudo /opt/lampp/lampp restart`	Restart all services
`sudo /opt/lampp/lampp startapache`	Start Apache only
`sudo /opt/lampp/lampp startmysql`	Start MariaDB only
`sudo /opt/lampp/lampp status`	Show running/stopped state of each service
`sudo /opt/lampp/lampp security`	Interactive security hardening wizard
`sudo /opt/lampp/bin/php --version`	XAMPP's bundled PHP version (different from system PHP)
`sudo /opt/lampp/bin/mysql -u root -p`	Connect to XAMPP's MariaDB (separate from system MySQL)
`sudo /opt/lampp/uninstall`	Remove XAMPP completely

Autostarting XAMPP with systemd

If you want XAMPP to start automatically on boot (on a dedicated local development machine, not a production server), create a systemd unit file:

philip@debian — XAMPP systemd unit

```
philip@debian:~$ sudo vi /etc/systemd/system/xampp.service
```

```
# /etc/systemd/system/xampp.service

[Unit]
Description=XAMPP
After=network.target

[Service]
ExecStart=/opt/lampp/lampp start
ExecStop=/opt/lampp/lampp stop
Type=forking
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

philip@debian — enable autostart

```
philip@debian:~$ sudo systemctl daemon-reload
philip@debian:~$ sudo systemctl enable xampp
Created symlink /etc/systemd/system/multi-user.target.wants/xampp.service
```

Course Summary Checklist

Everything covered across the seven chapters — a complete reference for setting up a LAMP stack on Debian 12 Bookworm from scratch:

- **Ch 1 — Overview & Planning:** stack anatomy, component choices, prerequisites, IP address, essential tools.
- **Ch 2 — Web Server:** Apache (mod_php, virtual hosts, a2ensite) or Nginx (PHP-FPM, server blocks, try_files).
- **Ch 3 — Database:** MariaDB (recommended, no extra repo), MySQL 8 (APT repo), or PostgreSQL. mysql_secure_installation, user and database creation.
- **Ch 4 — PHP:** PHP 8.2 (Debian main) or 8.3 (Sury repo). Essential extensions, php.ini tuning, Apache mod_php vs Nginx FPM socket, connection testing.
- **Ch 5 — SSL/HTTPS:** Let's Encrypt via Certbot for public domains, self-signed certificates for local development, HTTP → HTTPS redirect.
- **Ch 6 — Firewall & Security:** UFW rules, fail2ban jail.local, filesystem permissions (755/644/775), server version hiding, SSH hardening.
- **Ch 7 — XAMPP:** Install, directory structure, security wizard, systemd autostart, and when to use it vs the manual stack.

Web Server Setup: Debian 12 Bookworm — Complete

All 7 chapters done · Manual LAMP stack built and hardened · XAMPP covered as the alternative

Ch 1 Overview | Ch 2 Web Server | Ch 3 Database | Ch 4 PHP | Ch 5 SSL/HTTPS | Ch 6 Firewall | Ch 7 XAMPP