

# Apache2 Web Server & User Management in an LXC Container

This guide will show you step-by-step how to install the Apache2 web server, create users with different permissions, and configure Apache aliases in an already set up Ubuntu LXC container ([apache110](#)).

Before we begin, we log into the Proxmox server via the web interface and then go to the console of container 110 through Proxmox. Now we log in as pdal with the previously set password.

```
apache110 login: pdal
Password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.12-11-pve x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

## 1. Installing Apache2

### Introduction to Apache2

**Apache2** is a widely used **web server** that receives HTTP requests from clients (e.g., web browsers) and delivers content based on them. It's part of the **open-source software family** and is used on many operating systems, especially Linux.

### Key Features of Apache2

- **Website Delivery:** Provides HTML, PHP, or other web content via HTTP/HTTPS.
- **Web Application Management:** Allows hosting dynamic web applications, often in conjunction with PHP, Python, or other server-side languages.
- **Virtual Hosts:** Supports multiple websites on one server, each with its own domain name and web directory.
- **Security and Configuration:** Offers modules for authentication, access control, encryption (SSL/TLS), and logging.

Apache2 is used to make **websites and web applications accessible to users**. It processes requests, delivers the corresponding files, and can dynamically generate content, for example, by integrating with PHP.

### A Quick Comparison to Nginx

- **Architecture:** Apache2, in its default configuration, uses a **process-based or thread-based approach**, while Nginx uses an **event-driven, asynchronous model**.
- **Performance:** Nginx is often **more resource-efficient and faster** with static content under high traffic, while Apache2 offers more flexibility with dynamic content.
- **Configuration:** Apache2 relies heavily on modular, per-directory configuration (.htaccess), while Nginx uses centralized configuration files.
- **Use Cases:** Apache2 is well-suited for classic PHP web applications, while Nginx is often used as a **reverse proxy** or for highly scalable web services.

## Update Package Lists

```
sudo apt update
```

This command updates the package information on the system.

```
pdal@apache110:~$ sudo apt update
[sudo] password for pdal:
Sorry, try again.
[sudo] password for pdal:
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1158 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1092 kB]
Fetched 2502 kB in 3s (898 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
pdal@apache110:~$
```

---

## Install Apache2

```
sudo apt install apache2 -y
```

```
pdal@apache110:~$ sudo apt install apache2
Reading package lists... done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1t64 libaprutil1t64 libaprutil1
  libldap-common libldap2 liblua5.4-0 libperl5.38t64 libperl5.38t64 libperl5.38t64
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec
  libtap-harness-archive-perl
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils libap
  libgdbm-compat4t64 libldap-common libldap2 liblua5.4
0 upgraded, 19 newly installed, 0 to remove and 0 not
Need to get 11.4 MB of archives.
After this operation, 61.5 MB of additional disk space
Do you want to continue? [Y/n] y
```

This command installs the Apache2 web server and starts it automatically. Check the status:

```
systemctl status apache2
```

## Default Web Directory

After installation, the **default web directory** (Document Root) is located here:

```
/var/www/html
```

This directory contains the default website ([index.html](#)). It can be accessed at <http://<server-ip>>.

## The Meaning of a Web Directory

The web directory is the storage location for files that Apache2 delivers via [HTTP](#).

Example:

- File: </var/www/html/test.html>
- Accessible in the browser: <http://<server-ip>/test.html>

## Managing HTML Documents on the Apache Server

### Creating Directly in the Document Root

Small HTML documents or simple test pages can be created directly on the server:

- Write, save, and close the file in the console (e.g., Proxmox->LXC->Console or with SSH).
- The page is then accessible in the browser at <http://<server-ip>/meineseite.html>.

```
sudo nano /var/www/html/meineseite.html
```

## For Larger Projects or Multiple Files

For more extensive websites or projects, it's more efficient to develop the files locally and upload them to the server via **FTP/SFTP**:

1. Install an FTP client (e.g., FileZilla, WinSCP).
2. Connect to the server:
  - Host: <Server-IP>
  - Username and password of the server account
  - Port: 22 for SFTP (secure)
3. Select the local project directory.
4. Upload the files to the server directory </var/www/html/>.
5. Check permissions and adjust if necessary:

```
sudo chown -R www-data:www-data /var/www/html/mein_projekt
sudo chmod -R 755 /var/www/html/mein_projekt
```

The website will be available in the browser after uploading and setting the permissions.

Example content:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  content="width=device-width, initial-scale=1">
  <title>Apache Test Page</title>
  <meta name="description" content="A W3C-compliant example document to verify the
Apache configuration">
</head>
<body>
  <header>
    <h1>Apache Test Page works!</h1>
  </header>
  <div>
    <p>This page serves as a simple function test of the web server.</p>
  </div>
</body>
</html>
```

```
<h2>Server Information</h2>
<ul>
  <li>Document Root: <code>/var/www/html</code></li>
  <li>HTTP Status: OK</li>
</ul>
</div>
</body>
</html>
```

Save the file and close the editor.

**Testing** Open in the browser:

```
http://<server-ip/meinseite.html>
```

The message "**Apache Test Page works!**" should appear, confirming that the web directory is correctly configured.

---

## 2. Installing and Testing PHP

### Introduction

PHP (Hypertext Preprocessor) is a server-side scripting language specifically designed for web development. It allows you to create dynamic websites that can generate content based on user input, databases, or external systems.

#### **Advantages over static HTML pages:**

- Content can be updated automatically (e.g., current news, user profiles, statistics).
- Enables interaction with databases (e.g., MySQL, PostgreSQL).
- Supports sessions and forms to provide user logins and personalized content.
- Reduces maintenance effort, as content can be managed centrally and delivered dynamically.

### Install PHP and Required Modules

```
sudo apt update
sudo apt install php libapache2-mod-php php-cli -y
```

This installs PHP, the Apache2 PHP module, and the PHP CLI.

---

### Reload Apache

```
sudo systemctl reload apache2
```

## Managing PHP Documents on the Apache Server

### Creating a PHP Document Directly in the Document Root

Small PHP files or simple test scripts can be created directly on the server:

```
sudo nano /var/www/html/phpinfo.php
```

Example content:

```
<?php  
phpinfo();  
?>
```

- Save and close the file.
- The PHP page is then accessible in the browser at <http://<server-ip>/phpinfo.php>.

**Testing** Open in the browser:

```
http://<server-ip>/phpinfo.php
```

The PHP info page should appear, displaying information about the PHP installation.

### For Larger PHP Projects or Multiple Files

For more extensive PHP projects, it's more efficient to develop the files locally and upload them to the server via **FTP/SFTP**:

1. Install an FTP client (e.g., FileZilla, WinSCP).
2. Connect to the server:
  - Host: <Server-IP>
  - Username and password of the server account
  - Port: 22 for SFTP (secure)
3. Select the local project directory.
4. Upload the files to the server directory </var/www/html/>.
5. Check permissions and adjust if necessary:

```
sudo chown -R www-data:www-data /var/www/html/mein_projekt  
sudo chmod -R 755 /var/www/html/mein_projekt
```

The PHP web application will be available in the browser after uploading and setting the permissions.

## 3. Create Groups (Optional)

On a Linux server, user groups are used to centrally manage permissions for multiple users. Groups simplify the management of file and directory access and determine which users can perform certain actions without having to assign rights to each user individually.

Why create groups on an Apache2 server?

### 1. **stud** Group

- This group can be created for students or developers who need to access specific web directories or files, especially when multiple users need access.
- Members of this group can then be granted read or write permissions on web content without individual user adjustments.
- Example: If `/var/www/html/my_website` belongs to the **stud** group, all group members can make changes to the files, while other users have only read permissions or no access at all.

### 2. **apacherestart** Group

- This group is used to control the permission to restart or reload Apache.
- Not every user should be able to restart or reload the web server, as this affects live operations.
- By creating the group and adding authorized users to it, management can be done securely and in a controlled manner.
- Example: Members of the **apacherestart** group can restart the server via `sudo systemctl reload apache2` or `sudo systemctl restart apache2` without needing root privileges for all users.

Advantages of Group Management

- **Centralized Rights Management:** Permission changes only need to be made once for the group.
- **Security:** Only specific users receive privileged rights (e.g., restarting the web server).
- **Flexibility:** New users can be easily added to the appropriate group without manually adjusting file permissions.

Note: The commands

```
sudo groupadd stud  
sudo groupadd apacherestart
```

create the groups. Error messages can be ignored if the groups already exist.

```
pdal@apache110:~$ sudo groupadd stud  
[sudo] password for pdal:  
pdal@apache110:~$ sudo groupadd apacherestart
```

## 4. Sudo Rights for Apache Services

### Why set up sudo rights for Apache services?

The Apache web server is typically run under a **non-privileged user** ([www-data](#)) to increase security. However, actions such as **restarting or reloading** the Apache service require **root privileges**, as they directly affect system services.

To avoid giving every user root privileges, the [apacherestart](#) group is used in this case:

- Only members of this group are allowed to **restart or reload Apache services**.
- Permissions are controlled via [sudo](#), so authorized users get **temporarily elevated privileges** for these specific commands.
- This increases security, as only authorized users can perform critical actions, while other users still have limited access.

Advantage: No general root privileges are required; permission management is handled centrally via the [apacherestart](#) group.

We use [visudo](#) to set up sudo rights for specific users or user groups.

**Why visudo is used for sudo rights** [visudo](#) is a **secure tool for editing the sudo configuration file** [/etc/sudoers](#).

### Features and Advantages of [visudo](#):

- **Syntax Check:** [visudo](#) checks for correct file formatting upon saving. Errors in [/etc/sudoers](#) can otherwise prevent any sudo commands from being executed.
- **Locking Mechanism:** Only one user can edit the file at a time to prevent conflicts.
- **Secure Configuration:** Changes to user or group rights are reliably applied without compromising the system.

### Connection to the [apacherestart](#) group:

- When sudo rights are set up for the [apacherestart](#) group, the [/etc/sudoers](#) file is modified.
- [visudo](#) ensures that the new rule is correctly implemented, allowing group members to run Apache services with [sudo systemctl restart apache2](#) or [sudo systemctl reload apache2](#) without gaining root privileges for the entire system.

```
sudo visudo
```

```

# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/s

# This fixes CVE-2005-4890 and possibly breaks some versions of kdesu
# (#1011624, https://bugs.kde.org/show_bug.cgi?id=452532)
Defaults        use_pty

# This preserves proxy settings from user environments of root
# equivalent users (group sudo)
#Defaults:@sudo env_keep += "http_proxy https_proxy ftp_proxy all_proxy no_proxy"

# This allows running arbitrary commands, but so does ALL, and it means
# different sudoers have their choice of editor respected.
#Defaults:@sudo env_keep += "EDITOR"

# Completely harmless preservation of a user preference.
#Defaults:@sudo env_keep += "GREP_COLOR"

# While you shouldn't normally run git as root, you need to with etckeeper
#Defaults:@sudo env_keep += "GIT_AUTHOR_* GIT_COMMITTER_"

# Per-user preferences; root won't have sensible values for them.

```

Add the following lines at the end of the file:

```
%apacherestart ALL=(ALL) NOPASSWD: /bin/systemctl reload apache2
%apacherestart ALL=(ALL) NOPASSWD: /bin/systemctl restart apache2
```

```

# Allow members of group sudo to execute any command
@sudo    ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:

@include /etc/sudoers.d
%apacherestart ALL=(ALL) NOPASSWD: /bin/systemctl reload apache2
%apacherestart ALL=(ALL) NOPASSWD: /bin/systemctl restart apache2| [ Wrote 59 lines ]

```

Save with **Ctrl + O** followed by **Enter**. Now, close the editor with **Ctrl + X**.

## 5. Create User user1

In this step, a new user **user1** is created, assigned a home directory and the Bash shell, the password is set, and the user is added to both the **stud** group for web content access and the **apacherestart** group for restarting or reloading Apache services.

```

sudo useradd -m -d /home/user1 -s /bin/bash -g stud user1
echo "user1:mypassword" | sudo chpasswd
sudo usermod -aG apacherestart user1

```

```
pdal@apache110:~$ sudo useradd -m -d /home/user1 -s /bin/bash -g stud user1
[sudo] password for pdal:
pdal@apache110:~$ echo "user1:JadeHS20" | sudo chpasswd
pdal@apache110:~$ sudo usermod -aG apacherestart user1
pdal@apache110:~$ █
```

## 📁 6. Set Up a Web Directory for user1

In this step, a personal web directory is created for the user `user1`, ownership rights are set to `user1` and the `stud` group, and the access rights of the home directory are adjusted so the user can access their web content. The default user does not have write permissions to the standard Apache2 web directory `/var/www/html/`. For this reason, a web directory is created in the user's `/home` directory, which is then integrated into the Apache configuration.

```
sudo mkdir -p /home/user1/www
sudo chown user1:stud /home/user1/www
sudo chmod 755 /home/user1
```

```
pdal@apache110:~$ sudo mkdir -p /home/user1/www
pdal@apache110:~$ sudo chown user1:stud /home/user1/www
pdal@apache110:~$ sudo chmod 755 /home/user1
pdal@apache110:~$ █
```

## 🔗 7. Apache Alias Configuration for user1

### Goal

To set up an alias directory for a specific user (`user1`) so that the content of the `/home/user1/www` directory is accessible via a web browser.

### Creating a New Apache Configuration File

Open an editor of your choice (e.g., `nano`) and create the file:

```
sudo nano /etc/apache2/sites-available/stud-aliases.conf
```

Now, add the following content to the file.

```
<IfModule alias_module>
    Alias /user1 /home/user1/www
    <Directory /home/user1/www>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
</IfModule>
```

```
</Directory>
</IfModule>
```

Save with **CTRL+O**, confirm with Enter, and close the editor with **CTRL+X**.

```
webmin ALL=(ALL) NOPASSWD: /usr/bin/apt, /usr/bin/apt-get, /usr/bin/apt-cache, /bin/systemct
pdal@apache110:~$ sudo tee /etc/apache2/sites-available/stud-aliases.conf > /dev/null <<EOF
<IfModule alias_module>
Alias /user1 /home/user1/www
<Directory /home/user1/www>
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
</IfModule>
EOF
[sudo] password for pdal:
pdal@apache110:~$
```

## 🌀 8. Enable & Reload Apache Site

```
sudo a2ensite stud-aliases.conf
sudo systemctl reload apache2
```

```
pdal@apache110:~$ sudo a2ensite stud-aliases.conf
Enabling site stud-aliases.
To activate the new configuration, you need to run:
  systemctl reload apache2
pdal@apache110:~$ sudo systemctl re
reboot          reenable           reload
pdal@apache110:~$ sudo systemctl reload apache2.service
pdal@apache110:~$
```

## ☑ Result

- Apache2 is installed
- User user1 has a web directory under /home/user1/www accessible via:

```
http://<IP-des-Containers>/user1
```

user1 can reload Apache, e.g., with:

```
sudo systemctl reload apache2
```

## 💡 Tips for Testing Test web access:

```
curl http://localhost/user1
```

Apache status:

```
systemctl status apache2
```

## Sources

- "Apache HTTP Server Version 2.4 Documentation - Apache HTTP Server Version 2.4." Accessed: September 22, 2025. [Online]. Available at: [apache doc](#)
- "Access Control - Apache HTTP Server Version 2.4." Accessed: September 24, 2025. [Online]. Available at: [Apache2 Access Control](#)
- "Apache HTTP Server Tutorial: .htaccess files - Apache HTTP Server Version 2.4." Accessed: September 24, 2025. [Online]. Available at: [Apache2 htaccess](#)
- "Apache httpd Tutorial: Introduction to Server Side Includes - Apache HTTP Server Version 2.4." Accessed: September 24, 2025. [Online]. Available at: [Apache2 SSI](#)
- "Apache SSL/TLS Encryption - Apache HTTP Server Version 2.4." Accessed: September 24, 2025. [Online]. Available at: [Apache2 SSL/TLS](#)
- "Starting Apache - Apache HTTP Server Version 2.4." Accessed: September 24, 2025. [Online]. Available at: [Apache2 starten](#)
- "Apache Tutorial: Dynamic Content with CGI - Apache HTTP Server Version 2.4." Accessed: September 24, 2025. [Online]. Available at: [Apache2 CGI](#)
- "Authentication and Authorization - Apache HTTP Server Version 2.4." Accessed: September 24, 2025. [Online]. Available at: [Apache2 Authentication](#)
- "Server-Wide Configuration - Apache HTTP Server Version 2.4." Accessed: September 24, 2025. [Online]. Available at: [Apache2 Server Konfiguration](#)
- "Configuration Files - Apache HTTP Server Version 2.4." Accessed: September 24, 2025. [Online]. Available at: [Apache2 Konfigurationsdateien](#)
- "Getting Started - Apache HTTP Server Version 2.4." Accessed: September 24, 2025. [Online]. Available at: [Apache2 getting started](#)
- "Command Overview › Shell › Wiki › ubuntuusers.de." Accessed: August 20, 2025. [Online]. Available at: [Shell Commands](#)

---

## License

This work is licensed under the **Creative Commons Attribution - ShareAlike 4.0 International License**.

[To the license text on the Creative Commons website](#)