# IT Project Apache 2.4 -Webserver Betrieb

## Introduction

A web server is used to serve websites on the internet. A webserver finds the content from the server and delivers it to the web. The challenging job of a web server is to deliver the requested content to many different web users at the same time. A web server processes files in different languages such as Java, PHP, Python etc.

Apache HTTP Server is an open-source web server. It is used to handle processing and delivery between web pages or resources from a website to clients such as web browsers. Apache is compatible with various Operating System, including Windows, MacOS, Linux and Unix etc. Apache serves different types of web content using the standard HTTP and HTTPS protocols. This makes Apache compatible with a majority of web browsers and web clients. By employing virtual hosting, Apache can host multiple websites on a single server. It associates different domain names or IP addresses with specific websites. The functionality of Apache can be extended using modules. There are also various security features available in Apache to protect the web servers and applications from threats. For example, Secure Sockets Layer (SSL) provides support for encrypted communication. Apache is very stable, good in performance and can handle large number of concurrent connections. It is suitable for high-traffic websites. Apache generates access and error logs which allows the administrators to monitor and analyse the activities on the server and troubleshoot issues efficiently.

## Objective

The primary objective of this project is to design, deploy and configure a productive internet service using the Apache 2.4 web server on the FreeBSD operating system. The project aims to provide a comprehensive understanding of the Apache 2.4 web server, encompassing its functionalities, features, and capabilities. Step by step process of setting up a webserver will be followed to achieve the desired results. The web server must offer a service accessible via an IP address on a specific port number. Apache 2.4 web server will be monitored using runit supervising tool. Runit is a simple but efficient init system which ensures high availability and reliability of services. Apache server will be configured and customized according to the requirements. The critical aspect of the project is the secure integration of X.509 certificates to enable encrypted communication through SSL/TLS. Another aspect of this project is log monitoring. Access and error logs will be monitored and rotated according to the requirements.

## Pre-requisites

Following are the tools and the requirements that are essential to initiate the project,

#### Internet Access

Internet access is necessary during this project in order to download the required tools/packages for the webserver to function. A webserver must also be able to provide services using an internet connection.

#### VirtualBox

Oracle VirtualBox 7.0 (Version 7.0.8 r156879 (Qt5.15.2)) has been installed on Windows 10.

This version has been downloaded from the website of Oracle <https://www.virtualbox.org/> and is the latest available version.

#### Operating System

FreeBSD Operating System is used as the hosting platform for the Apache web server. FreeBSD is downloaded from the website <https://www.freebsd.org/> as an .iso file and has been installed in Virtual Box with minimum base memory and processor hardware requirements.

FreeBSD Installation Guide: https://www.freebsd.org/doc/handbook/install.html

#### Root Access in FreeBSD

To install and configure the necessary software packages and to make changes in the system, administrative access (root privileges) is required.

#### Testing Environment

A testing environment will be ideal for the project tasks to be carried out before implementing the results on the server.

## Introduction to FreeBSD

FreeBSD is an open-source Unix-like OS that finds its origin in Berkeley Software Distribution (BSD) Unix. It was developed at the University of California, Berkeley and shares its roots with some other famous BSD-based operating systems. It is famous for its stability, security and high performance which makes it ideal for web servers. FreeBSD is ideal for internet or Intranet services as it provides robust network services under the heaviest loads and uses memory in an efficient manner while maintaining much better response times for thousands of simultaneous user processes. It is free of charge and comes with the source code. FreeBSD focuses on performance, networking, and storage with ease of administration and comprehensive documentation to fully utilize the potential of a computer.

### Features of FreeBSD

FreeBSD offers a lot of useful features to provide high performance, networking, and storage. It also offers dynamic functionality for system administration and documentation. We will discuss some features here to understand the FreeBSD Operating System better:

#### Kernels

The kernel is the core of the FreeBSD operating system. It is responsible for managing memory, enforcing controls, disk access, networking and much more. A kernel is the lowest layer above the CPU. FreeBSD has a monolithic kernel design, which means that most of the operating system’s functionalities are contained within a single, large kernel image. FreeBSD kernel provides a set of system call interfaces that allow user-level applications to interact with kernel-level services. To communicate with hardware devices, device drivers are also included in FreeBSD kernel.

#### Boot Process of FreeBSD

The Boot Process of FreeBSD consists of series of steps, that start with the starting of a computer. We will discuss these steps briefly:

1. **BIOS:** When a computer is started, its firmware (BIOS) initializes the hardware components.
2. **Boot Loader:** Boot Loader of FreeBSD which is called **Loader** or **FreeBSD Boot Loader** by default, will be selected from the device.
3. **Kernel:** The boot loader will load the kernel (kernel or kernel.gz) from the selected device into the memory. Kernel is located in root directory of FreeBSD (/boot/kernel/kernel). After the kernel is loaded, it begins the initialization process. Its code will be executed from the memory.
4. **File System:** Kernel will mount the root file system which is located on a storage device.
5. **Init:** After the files system, kernel will execute the system’s initial user-space process **init** which is located at **/sbin/init**.
6. **System Services:** **init** process then reads the **/etc/rc.conf** to start the necessary system services. Other **rc** scripts in **/etc/rc.d** will also be executed if specified. Additionally, FreeBSD allows its users to customize the boot process by placing commands in **/etc/rc.local** file. Commands placed in this file will be executed during the boot process after all other system services are started.
7. **User Interaction:** Once the boot process is complete, user will be able to interact with the system through terminal.

#### Virtual Consoles

In standard FreeBSD, the system will boot into a command line login prompt. The first console that pops up is **ttyv0**, which indicates the **system console**. FreeBSD is a multiuser system, which is why every user account has a unique **username** and a separate **password** for each account. To switch between different consoles key combinations **Alt+F1** to **Alt+F8** have been reserved. For example, to switch to system console **ttyv0**, you have to use the combination **Alt + F1**. Similarly, to switch to the first virtual console **ttyv1** key combination **Alt +F2** needs to be used.

#### Accounts

Although FreeBSD allows multiple users to use the computer at the same time, only one user can use the computer while sitting in front of it. Whereas any number of users can log in to the system through the network. Each user needs an account to use the system. There are different types of accounts that perform different tasks. For example, system accounts, user accounts and the super user account.

**System user accounts** are used to run services such as DNS, mail, and web servers. Examples of such accounts are **daemon**, **www** etc. **User accounts** are assigned to real people. This type of account has certain information associated with it which is as follows:

* **Username:** It is unique for each user and is typed at the time of login.
* **Password:** Each account has its own password to login.
* **User ID** **(UID):** It is a number that uniquely identifies the user of FreeBSD system.
* **Group ID** **(GID):** It is also a number that uniquely identifies the primary group.
* **Password Change Time:** This option can be used to set the time of password expiry.
* **Account Expiration Time:** Time of expiry of an account can be set here.
* **User’s Full Name:** Full name of the user who is using this account.
* **Home Directory:** This is the user’s starting directory when the user logs in. Normally it is **/home/username** or **/usr/home/username**.
* **User Shell:** Shell provides the user’s default environment for interacting with the system.

**Superuser accounts,** which is the root account has the privilege to manage the system with no limitations. It must be used with much care as one slight change in the command or in any file can alter the system.

### Accounts Management

In FreeBSD different commands are used to manage the accounts. Here are some useful commands in user account management:

|  |  |
| --- | --- |
| **Commands** | **Summary** |
| adduser | The recommended command-line application for adding new users. |
| rmuser | The recommended command-line application for removing users. |
| chpass | A flexible tool for changing user database information. |
| passwd | The command-line tool to change user passwords. |
| pw | A powerful and flexible tool for modifying all aspects of user accounts. |
| bsdconfig | A system configuration utility with account management support. |

### Group Management

A **Group** is a collection of user accounts. It can be identified using a group name and **Group id (GID)**. The group name to group ID mapping is listed in **/etc/group**. This can be modified only by the super user account. **Group** allows multiple users to share common permissions for certain files and directories. If rights of a file or a directory have been assigned to a specific group, it means that all users of that particular group will have the same access rights to that file or directory.

#### Permissions

In FreeBSD, every file and directory have certain permissions that are associated to it. Those could be read (r), write (w) and execute (x) permissions. It is important to understand the permissions in FreeBSD in order to understand the file system of FreeBSD.

**Value Permission Directory Listing**

0 No read, no write, no execute ---

1 No read, no write, execute --x

2 No read, write, no execute -w-

3 No read, write, execute -wx

4 Read, no write, no execute r--

5 Read, no write, execute r-x

6 Read, write, no execute rw-

7 Read, write, execute rwx

To view the permission on the files or directories, command **ls -l** can be used. To Modify file permissions, we can use **chmod** command. **chmod** means change file modes. The syntax of the **chmod** command is as follows:

**chmod [who][action][permission] filename**

**[who]** represents the users, and could be u(User/Owner), g(Group), o(Others) or a(all),

**[action]** represents action to be taken. **+** is to add, **-** is to remove and **=** is to set.

**[permission]** represents the desired permission that is to be set. **r, w and x** will be used for this purpose.

**Examples:**

Grants execute permissions: **chmod +x filename**

Remove execute permission from all users: **chmod a-x filename**

#### Directory Structure of FreeBSD

Here is a brief look at the File System of FreeBSD.

**Directory Description**

/ Root directory of the file system.

/bin/ User utilities fundamental to both single-user and multi-user environments.

/boot/ Programs and configuration files used during operating system bootstrap.

/boot/defaults/ Default boot configuration files.

/dev/ Device special files managed by devfs.

/etc/ System configuration files and script.

/etc/defaults/ Default system configuration files.

/etc/periodic/ Scripts that run daily, weekly, and monthly, via cron(8).

/lib/ Critical system libraries needed for binaries in /bin and /sbin

/libexec/ Critical system files

/media/ Contains subdirectories to be used as mount points for removable media

such as CDs, USB drives, and floppy disks

/mnt/ Empty directory commonly used by system administrators as a temporary

mount point.

/net/ Automounted NFS shares

/proc/ Process file system.

/root/ Home directory for the root account.

/sbin/ System programs and administration utilities fundamental to both single user

and multi-user environments.

/tmp/ Temporary files which are usually not preserved across a system reboot. A

memory-based file system is often mounted at /tmp.

/usr/ The majority of user utilities and applications.

/usr/bin/ Common utilities, programming tools, and applications.

/usr/include/ Standard C include files.

/usr/lib/ Archive libraries.

/usr/libdata/ Miscellaneous utility data files.

/usr/libexec/ System daemons and system utilities executed by other programs.

/usr/local/ Local executables and libraries. Also used as the default destination for the

FreeBSD ports framework. Within /usr/local, the general layout sketched out

by hier(7) for /usr should be used. Exceptions are the man directory, which is

directly under /usr/local rather than under /usr/local/share, and the ports

documentation is in share/doc/port.

/usr/ports/ The FreeBSD Ports Collection (optional).

/usr/sbin/ System daemons and system utilities executed by users.

/usr/share/ Architecture-independent files.

/usr/src/ BSD and/or local source files.

/var/ Multi-purpose log, temporary, transient, and spool files.

/var/log/ Miscellaneous system log files.

/var/tmp/ Temporary files which are usually preserved across a system reboot.

### Useful Tools to Install for the Project in FreeBSD

#### pkg Upgrade

After the installation of FreeBSD and getting familiar with its directory structure the next step is to upgrade the packages. For this purpose, we will first update the package repository to make sure it is up to date. To update the package repository following command will be executed:

**pkg update**

When this command is executed, FreeBSD’s package management system (pkg) contacts the package repositories specified in the configuration (usually under **/etc/pkg/FreeBSD.conf**). It fetches the latest information about the packages. After fetching, it updates the local package database. Now to upgrade to the latest version we will execute the following command:

**pkg upgrade**

This command then upgrades the installed packages to their latest versions available in the repositories.

#### Vim Installation

For the project, we need to edit and configure different files. For this purpose, there are different tools available. **Vi/Vim**, **Emacs**, **Nano** and **Ed** are some of the useful editors. We are mostly using Vi/Vim text editor in this project. Vim is a powerful text-editor tool which can be installed on FreeBSD using following command:

**pkg install vim**

Vim is a highly configurable and efficient tool for text editing in FreeBSD. It is an enhanced version of the traditional **vi** editor.

#### Process Management

FreeBSD can run different tasks at the same time. Each running program or task at any one time is called a process. Below are some useful commands for the interaction with processes.

**ps**: To view the information about the processes that are running.

**kill**: It is used to terminate a process using PID.

**killall**: It is used to kill the processes using there names.

**pgrep**: It searches for the processes based on their names and returns their PIDs.

#### Process Viewers

Process viewers are useful tools to monitor and manage the processes in FreeBSD. Some commonly used process viewers are **top**, **htop**, **ps**, **gtop** and **htop**. We are using **htop** and **pstree** in this project for this purpose.

**htop** is an interactive process viewer. It is user friendly, and the user can view the processes in real-time. It continuously keeps updating the process and system information. It retrieves the information by reading data from **kern.proc** in FreeBSD. It shows the important information such as Process ID (PID), User, Priority of the process (PRI), State of the process (S), TIME (TIME+) measured in clock ticks that the process has spent in user and system time, full command line of the process (Command) etc. To install **htop** following command needs to be executed:

**pkg install htop**

**pstree** is a command line tool that shows the process hierarchy in a tree-like structure. **pstree** makes it easy to understand the relationships between parent and child processes. To install **psrtree** we need to execute the following command:

**pkg install pstree**

#### Web Browser

As in this project we are working on the console FreeBSD, we need console-based web browsers in order to open the webpages when needed. There are different terminal web browsers available such as **lynx**, **links**, **w3m** and **Netsruf** etc. We will use **lynx** browser in this project. **Lynx** is simple, fast and supports basic html. It is one of the user-friendly browsers available in its category. To install **lynx**, we can execute the following command:

**pkg install lynx**

To open websites using lynx, the following command can be used as an example,

**lynx google.com** or **lynx 127.0.0.1**

## Apache2.4 Installation

After updating the installed packages, we can proceed to install Apache 2.4. To do this the following command needs to be executed in the terminal:

**pkg install apache24**

Once the installation is complete the following command can be executed to run Apache 2.4. Apache runs as the **www** user and group on FreeBSD.

**service apache24 start**

To add Apache 2.4 to the startup, **rc.conf** file needs to be edited. Following are the steps that need to be performed:

**Step 1:** Open **/etc/rc.conf** file using **vim/vi** editor.

**Step 2:** Add **apache24\_enable=”YES”** in **rc.conf** file.

Some of the import directories related to Apache 2.4 in FreeBSD are as follows:

1. **Main Configuration: /usr/local/etc/apache24** contains the main configuration files for Apache.
2. **httpd.conf: /usr/local/etc/apache24/httpd.conf** contains the primary configuration file for Apache. This file sets most of the much-needed configurations. We will configure **httpd.conf** completely in this project according to our needs. This will be discussed later in the documentation.
3. **Modules: /usr/local/libexec/apache24** contains the Apache modules that will be used in the **httpd.conf** for the configuration.
4. **Web Root Directory:** **/usr/local/www/apache24/data/** is the default web root directory for Apache. Here we can place our website files.
5. **Log files:** Apache log files are stored by default in **/var/log/httpd/**. Similarly, Runtime files and PID files are stored in **/var/run/httpd/**.
6. **Executable Files:** Executable and binary files are found in **/usr/local/sbin/httpd**.

## Runit

Runit is a cross-platform Unix init scheme with service supervision. It is used to manage and monitor services in a system. It is efficient and straightforward, which makes it a good choice to manage services in Operating Systems such as FreeBSD. Some key features of Runit are as follows:

**Service Directory**

In Runit, we specify services with a separate directory for each service. We have located service directories under **/var/service**.

**Scripts**

We will create the subdirectories in the **/var/service** directorywhich will contain **run script.** These subdirectories will be supervised by the Runit tool.

**State of each Service**

Runit continuously maintains the state of each service using a **supervise** subdirectory, which contains the status of the service and the **process id (pid)**.

**Supervision**

Runit monitors the running services continuously and makes sure that they keep running. If a service stops, Runit automatically restarts it, which makes it crash resilient.

#### Utilities in Runit Tool

**runsv**

runsv starts and monitors a service (must be a directory). runsv switches to the directory and starts the run script. It maintains status information in a binary format in **/service/supervise/status** and in human readable format in **/service/supervise/stat**.

**runsvdir**

runsvdir is used to start and monitor a collection of **runsv** processes. Its synopsis looks like this:

**runsvdir [-P] dir**

Here **dir** indicates a directory. It will start a **runsv** process for each of the subdirectories.

**-P** uses setsid to run each **runsv** process in a new session and separate process group.

**chpst**

It is used to run a program with changed process state. It will be used in the run script to execute the service’s main process with custom configurations.

**sv**

It is a command-line tool that is used to control and manage services monitored by **runsv**.

The **sv** program can report the current status and controls the state of services monitored by runsv supervisor. The command looks like as follows:

**sv command service**

For example: To get the status of the service Apache 2.4 we use,

**sv status /var/service/apache24**

To start the service apache24 we use,

**sv start /var/service/apache24**

To restart the service apache24 we use,

**sv restart /var/service/apache24**

To stop the service apache24 we use,

**sv stop /var/service/apache24**

#### Installation of Runit in FreeBSD

Create a directory /package and execute the following commands,

**mkdir -p /package** (Only if /package doesn’t exist before)

**chmod 1755 /package**

Go to the website **smarden.org/runit/install.html**. Download **runit-2.1.2.tar.giz** into the directory **/package**, and then execute the following commands:

**cd /package**

**gunzip runit-2.1.2.tar** (Decompresses the **runit-2.1.2.tar.gz** file, creating **runit-2.1.2.tar**)

**tar -xpf runit-2.1.2.tar** (Extracts the contents of the **runit-2.1.2.tar** archive)

**rm runit-2.1.2.tar** (Removes the compressed **runit-2.1.2.tar** file)

**cd admin/runit-2.1.2** (Changes the current directory to **admin/runit-2.1.2**)

Once the above steps performed, navigate to **/package/admin/runit-2.1.2/src** andusing **lynx** browser install the patch files from <https://www.fehcom.net/patch-utmpx.diff>

Since we do not have **gcc** compiler installed in our FreeBSD environment, we will make the following changes in the **/package/admin/runit-2.1.2/src** folder. We will open **conf-cc** and **conf-ld** and replace **gcc** with **cc** using vim editor.

Now we will install the patches by using the following command on console:

**Patch < patch-utmpx.diff**

Now to install Runit we will move to the **/package/admin/runit-2.1.2** directory and install by using following commands on console:

**cd /package/admin/runit-2.1.2**

**package/install**

## rc.local and run script

To monitor **Runit** and supervise the Apache server at the startup, **rc.local** file needs to be modified. In this file we will specify the service directory that needs to be monitored. In the subdirectory of this service named **apache24**, we will create a run script that will be executed to start the Apache server at the startup under the supervision of Runit.

#### Run Script

A **run** scriptwill be used to start the Apache Server with the specific configuration file in the foreground while redirecting standard error to standard output. For the system management purpose, **chpst** will be used to run the server in a new process group. To create this script the following steps are required:

Navigate to **/var/service** and create directory **apache24**. In the directory **apache24**, **run** script needs to be created using **vim** editor.

**cd /var/service**

**mkdir apache24**

**vim run**

Now the following script needs to be written in the run file:

**#!/bin/sh**

**exec 2>&1**

**exec env -i /command/chpst -P \**

**/usr/local/sbin/httpd -f /usr/local/etc/apache24/httpd.conf -DFOREGROUND 2>&1**

**/bin/sh** is the path to the default Bourne shell in FreeBSD. The **exec** command is used to replace the current shell process with a specific command. The **env -i** clears the environment to ensure a clean environment before running the command. The **/command/chpst** allows running a program with specific resource limits and in a process group. Next **-P** will make sure that the process is placed in its own process group. The **/usr/local/sbin/httpd** is an executable binary file. It contains machine code that FreeBSD can directly execute. It is the core of Apache Server. The **-f** specifies the path to the **httpd.conf** which is provided in the command as **/usr/local/etc/apache24/httpd.conf.** httpd.conf file contains the configurations that Apache server will use. The **-DEFOREGROUND** tells the Apache to run in the foreground. At the end **2>&1** redirects standard error to standard output. It basically ensures that standard error and standard output are combined and sent to the same output stream.

To make this **run** script executable, the following command needs to be executed:

**#chmod +x run (**Changes the file permission of run to be executable)

Now the run script is ready to be executed.

#### rc.local file

This file is not part of the standard FreeBSD but can be created in **/etc** directory and is used to execute certain commands or scripts during the system boot process. Commands written in this file are executed towards the end of the system’s boot process, where most of the system services and configurations are already initialized, which makes it suitable for using for the tasks depending on other services.

Navigate to the **/etc** directory and open **rc.local** using **vim** editor.

**cd /etc**

**vim rc.local**

The following lines need to be written in the file:

**PATH=/usr/local/bin:$PATH**

**csh -cf '/command/runsvdir -P /var/service &'**

The purpose of setting the **Path** in this file is to search the executables in **/usr/local/bin**.

The **csh -cf** will invoke the C shell to run the command **/command/runsvdir -P /var/service &**.

This will launch the **runsvdir** with **-P** and **/var/service** as the argument and **&** at the end will run this command in the background. As soon as the **rc.local** file runs these commands, the **runsvdir** will start supervision process for the service directory.

#### Pstree after Reboot

-+= 00001 root /sbin/init

|--= 00420 root dhclient: system.syslog (dhclient)

|--= 00423 root dhclient: em0 [priv] (dhclient)

|--= 00484 \_dhcp dhclient: em0 (dhclient)

|--= 00485 root /sbin/devd

|--= 00681 root /usr/sbin/syslogd -s

|--= 00782 root /usr/sbin/cron -s

|--= 00787 root sendmail: accepting connections (sendmail)

|--= 00790 smmsp sendmail: Queue runner@00:30:00 for /var/spool/clientmqueue (sendmail)

|--= 00810 root sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups (sshd)

|-+= 00739 root /usr/local/bin/runsvdir -P /var/service

| \-+= 00745 root runsv apache24

| \-+= 00757 root /usr/local/sbin/httpd -f /usr/local/etc/apache24/httpd.conf -DFOREGROUND

| |--- 00806 www /usr/local/sbin/httpd -f /usr/local/etc/apache24/httpd.conf -DFOREGROUND

| |--- 00807 www /usr/local/sbin/httpd -f /usr/local/etc/apache24/httpd.conf -DFOREGROUND

| \--- 00808 www /usr/local/sbin/httpd -f /usr/local/etc/apache24/httpd.conf -DFOREGROUND

|-+= 00823 root login [pam] (login)

| \-+= 00849 root -csh (csh)

| \-+= 07791 root pstree

| \--- 07792 root ps -axwwo user,pid,ppid,pgid,command

|--= 00824 root /usr/libexec/getty Pc ttyv1

|--= 00825 root /usr/libexec/getty Pc ttyv2

|--= 00826 root /usr/libexec/getty Pc ttyv3

|--= 00827 root /usr/libexec/getty Pc ttyv4

|--= 00828 root /usr/libexec/getty Pc ttyv5

|--= 00829 root /usr/libexec/getty Pc ttyv6

\--= 00830 root /usr/libexec/getty Pc ttyv7

Here is an overview of the running processes on the system. This visually represents the hierarchy of the processes. Each indentation level shows the parent-child relationship. The **runsvdir** process supervises the **apache24** service, which is running multiple instances of the Apache HTTP Server to handle incoming connections simultaneously.

Process **00739** shows the **runsvdir** which is part of the Runit service management setup.

Process **00745** is the child process of **00739** and is a supervisor of Apache http server **apache24** service. Similarly, processes such as **00806**, **00807**, **00808** are also the child processes and are instances of the Apache HTTP Server.

## httpd.conf

The httpd.conf is a crucial configuration file associated with the Apache server. It plays a vital role in defining how the Apache server will operate, allowing administrators to make customizations and tailor its behaviour according to the desired requirements.

The httpd.conf file contains a range of directives that govern various aspects of the server’s functionality. These directives include settings related to server ports, virtual hosts, access control, logging, module configurations, etc. Any changes made to the configuration file are only recognized by httpd when it is started or restarted. It is important to mention that directives in the configuration files are case insensitive but the arguments to the directives ae often case sensitive.

In our project we modified the configuration file in such a way that it only contains directives which are essential for proper functioning of the Apache server. The location of the httpd.conf file in our project is **/usr/local/etc/apache24**. The httpd.conf file starts with contains following directives,

#### ServerRoot and ServerName

**ServerRoot “usr/local/**

**ServerName FreeBSD**

The **ServerRoot** directive specifies the root directory where the Apache will look for its configuration files, modules and for other server related resources. The **ServerName** directive is used by the server to identify itself by using the hostname and port. For example, if the machine hosting the web server is named as **my example.com** and the machine also has a DNS alias [**www.example.com**](http://www.example.com) and we want to identify the web server with its alias then we can do so with the help of following command,

**ServerName** [**www.example.com**](http://www.example.com)

The ServerName directive can be written anywhere in the configuration file, but each appearance will override the previous server’s name. If no port is specified in the ServerName, then the server will use the port from the incoming requests. In our case, FreeBSD is the ServerName.

#### Modules

Next, we loaded some modules in the configuration file by using the **LoadModule** directive. The **LoadModule** directive loads the named module from the **libexec** subdirectory of the **ServerRoot**. The following modules are loaded in the configuration file:

**LoadModule mpm\_event\_module libexec/apache24/mod\_mpm\_event.so**

**LoadModule log\_config\_module libexec/apache24/mod\_log\_config.so**

**LoadModule mime\_module libexec/apache24/mod\_mime.so**

**LoadModule authz\_core\_module libexec/apache24/mod\_authz\_core.so**

**LoadModule dir\_module libexec/apache24/mod\_dir.so**

**LoadModule unixd\_module libexec/apache24/mod\_unixd.so**

**LoadModule ssl\_module libexec/apache24/mod\_ssl.so**

**mpm\_event\_module**

The **mpm\_event\_module** in Apache is a clever way to handle many requests simultaneously, making websites faster and more responsive. It achieves this by efficiently using processes and threads.

Instead of having one big process, the event MPM has a main process that creates smaller processes. Each smaller process has multiple threads to handle incoming requests. Additionally, there's a special thread dedicated to listening for new connections.

The event MPM is particularly good at dealing with the **"keep-alive problem"** in HTTP. This means it can efficiently manage connections when clients want to keep the connection open after their first request. This efficient handling allows the server to serve more requests at the same time, enhancing overall performance and responsiveness.

**log\_config\_module**

The **log\_config\_module** is responsible for logging the requests made to the server. Logs are written in a customizable format and can be written directly to a file or to an external program. Administrators can control what data is logged such as the client IP address, requested URLs, response status, referrer and user-agent.

The **log\_config\_module** provides three directives namely **TransferLog** to create a log file, **LogFormat** to set a custom format and **CustomLog** to define a log file and format in one step. **TransferLog** and **CustomLog** directives can be used multiple times in a server to cause each request to be logged to multiple files.

**mime\_module**

The **mime\_module** handles the mapping of file extensions to MIME types. MIME types are used to indicate the type of data served by a web server, allowing clients (browsers) to interpret the content correctly. For example, when a client requests a file like "example.jpg," the mime\_module maps the ".jpg" extension to the corresponding MIME type for JPEG images (e.g., "image/jpeg"). This ensures that the client knows how to interpret and display the received content. The mime\_module also provides a directive **TypesConfig** that is used to specify a file which also maps extension onto media types.

**authz\_core\_module**

The **authz\_core\_module** provides core authorization capabilities in Apache. It defines access control rules based on various criteria such as IP address, user authentication or specific HTTP methods. It also provides some directives such as **<RequireAll>**, **<RequireAny>** and **<RequireNone>** which can be combined with each other to express complex authorization logic.

**dir\_module**

The **dir\_module** enables Apache to serve directory indexes when a client requests a URL corresponding to a directory path. If there is no specific index file (e.g., index.html) in the requested directory, Apache will generate a directory listing, allowing the client to see the files and subdirectories within that directory.

**unixd\_module**

The **unixd\_module** provides Unix-specific functionality to Apache server. It is responsible for handling basic Unix-specific tasks, such as initializing the Apache process and setting permissions and **User/Group** ownership on various resources. Some of the directives of **unixd\_module** are **Group** and **User** etc.

**ssl\_module**

The **ssl\_module** enables secure connections over **HTTPS** by providing support for the **SSL/TLS** protocols. It allows Apache to serve content securely using encryption, ensuring the data transmitted between the server and clients is encrypted and secure.

**TypesConfig**

After loading all modules next, we have the following directive in the configuration file:

**TypesConfig /usr/local/etc/apache24/mime.types**

As mentioned before, the **TypesConfig** directive will tell Apache where to find the mapping of file extensions to their corresponding **MIME types**.

#### PID File

Next, we have following directive written in the configuration file:

**PidFile /var/run/httpd.pid**

The **PidFile** directive specifies the location where the server records the process id of the daemon. This directive is often useful to be able to send different signals to the server, like closing and restarting its **ErrorLog**, **TransferLog** and re-read its configuration file.

#### User and Group

Subsequently we have following directives in the configuration file:

**User www**

**Group www**

The above two directives specify the user and group under which the Apache server will run. In this case, Apache will run as the user **www** and the group **www**. Running the web server under a dedicated user and group helps in enhancing security by limiting the access only to essential operations.

#### Ports

**Listen 80**

**Listen 443**

The above two directives **Listen 80** and **Listen 443** specify the port on which Apache will listen for incoming client connections. Port **80** is the default port for **HTTP** and allows unencrypted web traffic. Port **443** is a default port for **HTTPS** which allows encrypted web traffic.

#### DocumentRoot

Next, we have following directives written in the configuration file:

**DocumentRoot "/usr/local/www/apache24/data"**

**DirectoryIndex index.html**

**<Directrory "/usr/local/www/apache24/data">**

**Require all granted**

**</Directory>**

The **DocumentRoot** directive specifies the base directory form which Apache serves content for the main server. In this case, the root directory is set to **/usr/local/www/apache24/data**. The **DirectoryIndex** directive specifies the default file that Apache will serve when the directory specified in **DocumentRoot** directive will be requested. The next block defines specific directives that only apply to directory specified, in this case **/usr/local/www/apache24/data**. In this block, we have used the directive **Require all granted** which allows access to the specified directory without any restrictions. Meaning that a client requesting for a resource form the directory **/usr/local/www/apache24/data** will be granted access without requiring any authentication.

#### VirtualHost

Thereafter we used **<VirtualHost>** directive to host multiple websites on Apache server. Each **Virtual Host** section defines the configuration settings that will apply only to a particular virtual host, allowing multiple websites to coexist on the same server. When the server receives a request for a document on a particular virtual host, it uses the configuration directives enclosed in the **<VirtualHost>** section. Each **Virtual Host** must correspond to a different IP address, different port number or a different host name of the server. In our configuration file we have used **<VitualHost>** twice for two different ports namely port **80** and port **443**.

**<VirtualHost \*:80>**

**DocumentRoot "/usr/local/www/nike"**

**<Directory "/usr/local/www/nike">**

**Options FollowSymlinks**

**AllowOverride AuthConfig**

**</Directory>**

**DirectoryIndex index.html**

**</VirtualHost>**

**<VirtualHost \*:443>**

**DocumentRoot "/usr/local/www/nike"**

**<Directory "/usr/local/www/nike">**

**Options FollowSymlinks**

**AllowOverride AuthConfig**

**</Directory>**

**DirectoryIndex index.html**

**SSLCertificateFile "/root/FreeBSD\_ECCert.pem"**

**SSLCertificateKeyFile "/root/FreeBSD\_ECCkey.pem"**

**</VirtualHost>**

The **<VirtualHost>** sections for **port 80** and **443** are quite similar to each other. As mentioned before, port 80 is a default port for **HTTP** and is used for unencrypted web traffic and port **443** is a default port for **HTTPS** which allows only encrypted web traffic. Port **80** and port **443** are using the same directory **/usr/local/www/nike** to which the client will be provided access but different index.html files will be served to the client because we have two different **index.html** files set for port **80** and **443**. Moreover, we are allowing **symlinks** on the directory in both port **80** and **443** which means that if there are any symbolic links inside the **/usr/local/www/nike** directory, the web server will follow them and serve the files or directories they point to. Both **VirtualHost** port **80** and port **443** also allow the **.htaccess** files to override the authentication settings **AuthConfig**.Furthermore, the **VirtualHost** port **443** also contains the **SSL certificates** by using the directives **SSLCertificateFile** and **SSLCertificateKeyFile** which are used to secure connections between the server and client. A more detailed explanation about how these **SSL certificates** work is explained in the later part of the document.

Apache HTTP server is also configured for logging. Logging part of the httpd.conf file is discussed later in this documentation.

Now here is a look at the httpd.conf file.

#### httpd.conf File Content

**# ServerRoot is the starting point for loading the modules**

ServerRoot “usr/local/

ServerName FreeBSD

LoadModule mpm\_event\_module libexec/apache24/mod\_mpm\_event.so

LoadModule log\_config\_module libexec/apache24/mod\_log\_config.so

LoadModule mime\_module libexec/apache24/mod\_mime.so

LoadModule authz\_core\_module libexec/apache24/mod\_authz\_core.so

LoadModule dir\_module libexec/apache24/mod\_dir.so

LoadModule unixd\_module libexec/apache24/mod\_unixd.so

LoadModule ssl\_module libexec/apache24/mod\_ssl.so

**# Setuid user and group**

User www

Group www

**# Listener**

Listen 80

Listen 443

DocumentRoot "/usr/local/www/apache24/data"

**# Global DocumentRoot**

DirectoryIndex index.html

<Directrory "/usr/local/www/apache24/data">

Require all granted

</Directory>

<VirtualHost \*:80>

DocumentRoot "/usr/local/www/nike"

<Directory "/usr/local/www/nike">

Options FollowSymlinks

AllowOverride AuthConfig

</Directory>

DirectoryIndex index.html

</VirtualHost>

<VirtualHost \*:443>

DocumentRoot "/usr/local/www/nike"

<Directory "/usr/local/www/nike">

Options FollowSymlinks

AllowOverride AuthConfig

</Directory>

DirectoryIndex index.html

SSLCertificateFile "/root/FreeBSD\_ECCert.pem"

SSLCertificateKeyFile "/root/FreeBSD\_ECCkey.pem"

</VirtualHost>

ErrorLog “/var/log/apache24/error.log“

LogLevel debug

<IfModule log\_config\_module>

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined

LogFormat "%h %l %u %t \"%r\" %>s %b" common

CustomLog “/var/log/apache24/access.log” combined

</IfModule>

## SSL Certificates

The **importance of an SSL** (Secure Sockets Layer) certificate lies in its crucial role in ensuring the security, privacy, and trustworthiness of online communication between a client (e.g., web browser) and a server (e.g., web server). SSL certificates enable the use of HTTPS, which encrypts data transmitted between the client and server, protecting it from interception, tampering, and eavesdropping by malicious actors.

**X509 Certificates in Https**

A digital certificate contains the public key of a server. It’s an electronic document that proves the ownership of a public key. It contains Serial number, Identity of owner, validity, issuer, public key etc. A X509 is standard that defines the format of digital certificate. This format is used in IP security and SSL/TLS. It is based on the use of public key Cryptography (ECC or RSA) and digital signatures.

**Working of a Self-Signed SSL Certificate and Private/Public Key generation**

A self-signed SSL certificate works by enabling secure communication between a server and clients without involving a trusted third-party Certificate Authority (CA). The process of a self-signed SSL certificate involves the server generating its key pair (public and private keys), creating a Certificate Signing Request (CSR) containing its public key, and using its private key to sign the CSR, producing a self-signed certificate. The certificate, along with the private key, is installed on the server. When clients connect, the server presents the self-signed certificate.

**Generating a Self-Signed Certificate in Project**

In order to create a self-signed Certificate for our webserver, certain Scripts and Configuration files were downloaded from <http://www.fehcom.net>

(make\_CA.sh) (make\_server.sh) (CA.cfg) (server.cfg)

**chmod+x make\_CA.sh**

**./make\_CA.sh**

First the make\_CA.sh was made executable and then executed in order to generate a Root Certificate using the OpenSSL command line tool so that we can become a Certificate Authority ourselves and creation of Public/Private Key pair is done to sign the certificates.

#### make\_CA.sh Script

The following is a brief description of the various commands used in this file:

**Openssl ecparam -genkey -name ${TYPE} -out ${KEY}**

This command generates an Elliptic Curve Cryptography (ECC) private key using the OpenSSL **ecparm** utility. The **-genkey** option indicates that a new private key should be generated. The **-${TYPE}** option specifies the elliptic curve type, and the **-out ${KEY}** option designates the output file where the generated private key will be saved.

**Openssl req -new -key ${KEY} -config ${CONFIG} -extensions v3\_ca -out ${CERT}.csr**

This command generates a Certificate Signing Request (CSR) using the ECC private key generated in the previous step. The CSR is a request sent to a Certificate Authority (CA) to obtain a certificate, which will be used to identify the entity associated with the ECC private key.

**Openssl req -x509 -days 365 -key ${KEY} -in ${CERT}.csr -out ${CERT}**

This command creates a self-signed X.509 certificate using the ECC private key, and the CSR generated in the previous step. **-x509** Specifies that the output should be a self-signed certificate rather than a CSR. **-days 365** Sets the validity period of the self-signed certificate to 365 days.

#### Server.cfg Configuration File

**Server.cfg** is a configuration file used by OpenSSL when generating an ECC (Elliptic Curve Cryptography) certificate for a server. The configuration file contains settings and parameters that specify how the certificate request should be processed and what information should be included in the resulting certificate. Firstly, we added all our relevant information such as name, place organization name, email, host name etc. Then this file was changed to our own hostname with the following command:

**mv server.cfg labpc107.cfg**

After a Certificate Signing Request (CSR) for the web server has been created using the Server.cfg file in the previous step (CSR contains public key and desired attributes for web server certificate), it is then used in conjunction with server.cfg to create a self-signed X.509 certificate, which is then signed using the RootCA’s Private key, to enable HTTPS connection.

First, we made the Make\_server.sh executable by executing the following command:

**chmod + x make\_server.sh.**

Then we execute the make\_server.sh by entering the hostname of our server:

**./make\_server.sh labpc-107**

#### make\_server.sh Script

Following commands are used in make\_server.sh, with their short description:

**Openssl ecparam -genkey -name ${TYPE} -out ${KEY}**

This command generates an Elliptic Curve Cryptography (ECC) private key using the OpenSSL ecparam utility. The **-genkey** option indicates that a new private key should be generated. The **-${TYPE}** option specifies the elliptic curve type, and the **-out ${KEY}** option designates the output file where the generated private key will be saved.

**openssl req -new -key ${KEY} -out ${HOST}.csr -config ${CONFIG}**

This command generates a Certificate Signing Request (CSR) using the ECC private key generated in the previous step. The CSR is a request sent to a Certificate Authority (CA) to obtain a certificate, which will be used to identify the entity associated with the ECC private key.

**Openssl req -x509 -pkeyopt ec\_paramgen\_curve: ${TYPE} -nodes -days 365 -in ${HOST}.csr -out ${CERT} -config ${CONFIG} -extensions x509v3 -key rootCA\_ECCkey.pem**

The **-x509** specifies that the output should be a self-signed certificate rather than a CSR. The **-pkeyopt ec\_paramgen\_curve: ${TYPE}** specifies the elliptic curve type for the ECC private key used in the self-signed certificate. The **-nodes** indicate that the private key should not be encrypted with a passphrase. The **-days 365** sets the validity period of the self-signed certificate to 365 days (1 year). The **-in ${HOST}.csr** specifies the input CSR file (generated in the second step) that provides the subject information for the certificate. The **-out ${CERT}** specifies the output file name for the self-signed certificate. The **${CERT}** is a variable representing the name of the certificate, so **${CERT}** is the certificate file.

The **-config ${CONFIG}** specifies the configuration file to be used for certificate generation. The configuration file can contain additional certificate-related settings and extensions. The **-extensions x509v3** specifies the certificate extensions to be used. **x509v3** extensions allow custom extensions to be included in the certificate.

#### View Certificate

To view the certificate in a human readable form on console we can type the following command:

**Openssl x509 -in labpc107\_ECCert.pem -text -noout | more**

This command is useful for inspecting the details of an X.509 certificate, including the certificate's subject, issuer, validity dates, public key details, extensions, and other relevant information.

#### Trust Chain

Finally in order to establish a Trust Chain Integration, once the SSL certificate is generated, we need to add the following files in the Httpd.conf:

**SSLCertificateFile “/root/Server\_ECCert.pem”**

This directive specifies the path to the SSL certificate file (e.g., labpc107\_ECCert.pem) that you obtained or generated for your server. The certificate file contains the public key, the server's identity, and the signature created by the private key. Apache uses this certificate to identify itself to clients during the SSL/TLS handshake.

**SSLCertificateKeyFile “/root/Server\_ECCkey.pem”**

This directive specifies the path to the SSL certificate's private key file (e.g., labpc107\_ECCkey.pem). The private key is used to decrypt encrypted data received from clients and to sign data sent to clients.

## Logging

Logging is an important part of any web server. Similarly, Apache server also makes logs for the successful client requests in the form of access logs and all error events in the form of error log. One can use these logs in case of any troubleshooting or maintenance.

#### Access Log

The server access log records all requests processed by the server. The location and content of the access log are controlled by the **CustomLog** directive. The format of the access log is highly configurable. The format is specified using a format string that looks much like a C-style printf format string. There are different kinds of log format strings used for access logs. Access Log file needs to be added in the **httpd.conf** file in order to store the access logs of the Apache Server. For this purpose, we will add an **<IfModule>** block which is a conditional configuration directive that defines the settings specific to the **log\_config** module, which is responsible for logging requests and responses made to the web server.

**<IfModule log\_config\_module>**

**LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined**

**LogFormat "%h %l %u %t \"%r\" %>s %b" common**

**CustomLog ”/var/log/apache24/access.log” combined**

**</IfModule>**

In this **<IfModule>**, two formats of logging have been defined first and then used later for the logging.

**Common Log Format**

A typical configuration for the access log might look as follows:

**LogFormat "%h %l %u %t \"%r\" %>s %b" common**

%h:This is the IP address of the client (remote host) which made the request to the server.

%l:Represents the remote logname (RFC 1413 identity) of the client.

%u:This is the userid of the person requesting the document as determined by HTTP authentication.

%t:Tells the time at which the request was received.

\”%r\”:Provides information about the method used by client, The requested resource by client and the protocol used by the client.

%>s:This is the status code that the server sends back to the client.

%b:The last part indicates the size of the object returned to the client.

**Combined Log Format**

Another commonly used format string is called the **Combined Log Format**. It can be used as follows:

**LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined**

This format is the same as the Common Log Format, with the addition of two more fields. The additional fields are:

\"%{Referer}i\":This gives the information about the site from where the client reports are referred.

\"%{User-agent}i\":This is the information that the client browser reports to identify itself.

After defining the log formats in the **<IfModule>**, we can use either **common** or **combined** format to make entries to our access log file. Here is the syntax:

**CustomLog “/var/log/apache24/access.log” combined**

This line in the **<IfModule>** will set the location of **access.log** file to **/var/log/apache24/. Common** or **Combined** at the end will specify the format of the entries that will be made by Apache Server to the access file.

#### Error Log

Similar to access logs, we will add the following line in httpd.conf file to make the error logs of the Apache Server:

**ErrorLog “/var/log/apache24/error.log”**

This is the place where Apache httpd will send all the information about diagnostics and errors encountered during processing the request. The format of the error log is defined by the ErrorLogFormat directive, with which you can customize what values are logged.

**ErrorLogFormat “[%t] [%l] [pid %P] %F: %E: [client %a] %M"**

A default error log format is defined if you don't specify one. A typical log message follows:

**[Sun Jul 30 2023] [core:info][pid 882:tid 35248015360] [client 192.168.178.26] File does not exist: /usr/local/www/example/favicon.ico, referrer: http://192.168.178.79/**

The **LogLevel** directive allows you to specify a log severity level on a per-module basis. For example:

**LogLevel warn:** Provides warning messages that indicate potential issues that need attention.

**LogLevel debug:** Provides detailed debugging information.

**LogLevel alert:** Provides details about critical conditions that require immediate attention.

**LogLevel error:** Provides standard logs for error messages that are significant.

We have used the **LogLevel debug** in our **httpd.conf** file.

## Log Rotation

Log rotation in FreeBSD is the process in which log files are managed in such a way that it prevents them from becoming too large. To make the logs manageable and organized on the system where logging activity is very high, log rotation is a vital feature. Under FreeBSD there is a service available that is used for the log rotation called **newsyslog**.

#### Newsyslog

It is a utility that is used to maintain log files to manageable sizes. It needs to be scheduled to run periodically. It is used to manage log files by rotating and compressing them based on size and time. It prevents the log files to become too large so that they occupy bigger space on hard disk.

The **newsyslog** service can be configured to **watch** these log files and perform specific actions based on predefined criteria specified in the **configuration file (/etc/newsyslog.conf).** When a service directly writes to a file, it needs to be stopped first, and the log file must be closed. Then, a new empty file with the same name is created, and the service can be restarted to continue writing log data. We have configured the **/etc/newsyslog.conf** as follows:

**Logfilename [owner:group] mode count size when flags**

**[/pid\_file] [sig\_num]**

/var/log/apache24/access.log 644 7 \* @T00 JC var/run/httpd.pid 30

/var/log/apache24/error.log 644 7 \* @T00 JC /var/run/httpd.pid 30

Here is the information that has been provided in the **newsyslog** configuration file:

**Logfilename:** Name of the system log file to be archived.

**Mode:** The permission/mode for the log file.

**count:** The number of backup files to keep.

**size:** Maximum size of the log file before rotation in kilobytes, \* means no size-based rotation.

**when:** Rotation time or interval from midnight. @T00 means daily rotation at midnight.

**flags:** JC means to copy and truncate log files in all cases.

**[/pid\_file]:** The path to the process ID (pid) file.

**[sig\_num]:** Thesignal number to send to the process when rotating.

So basically, in the **/etc/newsyslog.conf** we are specifying the access and error log files to keep **7** backup files with no size specified to rotate at midnight. **JC** will copy and truncate log files in all cases and will make sure that log data is preserved in a backup copy, even if log file is not used by any other process at the rotation time. Pid file preserves the process id of the Apache Server. Signal number **30** indicates **SIGUSR1**, which is a user defined signal. It is sent to inform the process to close and reopen its log file.

## Links

[What Is Apache? An In-Depth Overview of Apache Web Server (hostinger.com)](https://www.hostinger.com/tutorials/what-is-apache)

[What is Apache? In-Depth Overview of Apache Web Server | Sumo Logic](https://www.sumologic.com/blog/apache-web-server-introduction/)

<https://www.twaino.com/en/blog/website-creation/apache-server-2/>

https://man.freebsd.org/cgi/man.cgi

<https://docs.freebsd.org/en/books/handbook/>

<https://freebsdfoundation.org/freebsd-project/what-is-freebsd/#:~:text=FreeBSD%20is%20a%20free%20Unix,key%20features%20and%20applications%20here>.

[About FreeBSD | The FreeBhttps://de.wikipedia.org/wiki/Syslog](https://www.freebsd.org/about/)

[https://cr.yp.to/daemontools/multilog.html](https://www.freebsd.org/about/)

[https://man.freebsd.org/cgi/man.cgi?newsyslog.conf(5)](https://www.freebsd.org/about/)

[https://httpd.apache.org/docs/2.4/logs.html](https://www.freebsd.org/about/)

[https://en.wikipedia.org/wiki/Log\_rotationSD Project](https://www.freebsd.org/about/)

https://de.wikipedia.org/wiki/Syslog

https://cr.yp.to/daemontools/multilog.html

https://man.freebsd.org/cgi/man.cgi?newsyslog.conf(5)

https://httpd.apache.org/docs/2.4/logs.html

https://en.wikipedia.org/wiki/Log\_rotation<https://httpd.apache.org/docs/2.4/configuring.html>

<https://www.hostgator.com/help/article/httpd-configuration-httpd-conf>

<https://httpd.apache.org/docs/2.4/mod/core.html#serverroot>

<https://httpd.apache.org/docs/2.4/mod/core.html#servername>

https://httpd.apache.org/docs/2.4/mod/event.html

<https://httpd.apache.org/docs/2.4/mod/mod_log_config.html>

<https://httpd.apache.org/docs/current/mod/mod_mime.html>

<https://httpd.apache.org/docs/2.4/mod/mod_authz_core.html>

<https://httpd.apache.org/docs/2.4/mod/mod_dir.html>

<https://httpd.apache.org/docs/2.4/mod/mod_unixd.html>

https://httpd.apache.org/docs/current/mod/mpm\_common.html#pidfile

<https://httpd.apache.org/docs/2.4/mod/core.html#documentroot>

<https://httpd.apache.org/docs/2.4/mod/core.html#virtualhost>

<https://de.wikipedia.org/wiki/Syslog>

<https://cr.yp.to/daemontools/multilog.html>

<https://man.freebsd.org/cgi/man.cgi?newsyslog.conf(5)>

<https://httpd.apache.org/docs/2.4/logs.html>

<https://en.wikipedia.org/wiki/Log_rotation>