

Configuración Avanzada de Apache

Antonio Espín Herranz

Contenidos

- Configuración de **hosts virtuales** y **múltiples sitios** en Apache.
- Ajustes de rendimiento con **KeepAlive**, **Timeout**, **MaxClients**.
- Administración de **logs** y monitoreo de **rendimiento**.

Virtual Hosts

Hosts Virtuales

- Host Virtuales son una funcionalidad que permite que un solo servidor web (una sola máquina con Apache instalado) **sirva múltiples sitios web diferentes.**
- Tendremos varios sitios independientes corriendo en el mismo servidor, cada uno con su propio dominio, configuración y contenido.

Host Virtuales

- Pueden ser basados en nombre o en dirección IP
- **Name-based Virtual Hosts:** Se basan en el nombre del dominio: www.ejemplo.es. Pueden estar en la misma dirección IP. Apache los distingue por el nombre del host que se envía la solicitud.
- **IP-based Virtual Hosts:** Se basan en direcciones IP. Cada sitio tiene su propia IP, apache responde según la dirección IP solicitada.

Configuración

- Primero tenemos que ir al fichero de configuración de Apache:
- C:\Apache24\conf**httpd.conf**
 - Localizar la línea:
 - **# Virtual hosts**
 - **#Include conf/extra/httpd-vhosts.conf**
- Descomentar y reiniciar.
- Ir al fichero (añadir un virtual host por cada proyecto)
- **C:\Apache24\conf\extra\httpd-vhosts.conf**

Múltiples sitios en Apache

```
<VirtualHost *:80>  
    ServerName proyecto1.local  
    DocumentRoot "C:/websites/proyecto1"  
    <Directory "C:/websites/proyecto1">  
        AllowOverride All  
        Require all granted  
    </Directory>  
</VirtualHost>
```

```
<VirtualHost *:80>  
    ServerName proyecto2.local  
    DocumentRoot "C:/websites/proyecto2"  
    <Directory "C:/websites/proyecto2">  
        AllowOverride All  
        Require all granted  
    </Directory>  
</VirtualHost>
```

Múltiples sitios en Apache

- Abrir como **administrador** el fichero (con el block de notas):
 - Seleccionar “todos los archivos”
 - **C:\Windows\System32\drivers\etc\hosts**
- Añadir:
 - ***127.0.0.1 proyecto1.local***
 - ***127.0.0.1 proyecto2.local***
- Tantas entradas como proyectos tengamos.

Múltiples sitios en Apache

- Reiniciar Apache
- Crear un fichero index.php en cada carpeta de los dos sitios:
 - C:/websites/proyecto1/index.php → <?php phpinfo(); ?>
 - C:/websites/proyecto2/index.php → <?php phpinfo(); ?>
- Desde un navegador en dos pestañas distintas:
 - <http://proyecto1.local>
 - <http://proyecto2.local>

Con páginas de error personalizadas

```
<VirtualHost *:80>
  ServerName www.sitio1.local
  DocumentRoot "C:/Apache24/htdocs/sitio1"

  <Directory "C:/Apache24/htdocs/sitio1">
    AllowOverride All
    Require all granted
  </Directory>

  ErrorDocument 404 /error/404_sitio1.html
  ErrorDocument 500 /error/500_sitio1.html
</VirtualHost>
```

```
<VirtualHost *:80>
  ServerName www.sitio2.local
  DocumentRoot "C:/Apache24/htdocs/sitio2"

  <Directory "C:/Apache24/htdocs/sitio2">
    AllowOverride All
    Require all granted
  </Directory>

  ErrorDocument 404 /error/404_sitio2.html
  ErrorDocument 500 /error/500_sitio2.html
</VirtualHost>
```

Con páginas de error personalizadas

```
htdocs/  
├── sitio1/  
│   ├── index.php  
│   └── errores/  
│       ├── 404.html  
│       └── 500.html  
├── sitio2/  
│   ├── index.php  
│   └── errores/  
│       ├── 404.html  
│       └── 500.html  
└──
```

Múltiples sitios en Apache

- ***OJO que el orden en que se colocan los virtualhosts puede ser determinante.***
- Si tenemos APIs de servicios (por ejemplo: con Slim), podemos tener problemas a la hora de que nuestro sitio responda.
- En este caso, podemos colocarlo el primero.
- ***Prueba: localhost o phpmyadmin (algo que esté en htdocs), después de configurar los virtual hosts, fuera de htdocs.***

Situaciones distintas en Virtual Hosts

- **1. Directorio raíz distintos:**
 - Cada sitio puede tener su propio **DocumentRoot**, apuntando a diferentes carpetas en el sistema de archivos.
- **2. Dominio o subdominio diferente**
 - Podemos servir: www.ejemplo.com y blog.ejemplo.com con configuraciones distintas
- **3. Certificados SSL distintos**
 - Cada host puede tener su propio certificado TLS/SSL para HTTPS
- **4. Logs separados**
 - Puedes definir ErrorLog y CustomLog por host para facilitar el análisis y la depuración.
 - ErrorLog "logs/app1-error.log"
 - CustomLog "logs/app1-access.log" common
 - Los logs van a la carpeta de logs de apache, si no, colocar rutas absolutas
- **5. Configuraciones PHP o CGI distintas**
 - Con .htaccess o directivas específicas para calbiar el comportamiento de PHP.

Situaciones distintas en Virtual Hosts II

- **6. Políticas de seguridad distintas**
 - Control de acceso (Allow, Deny, Require)
- **7. Reescrituras y redirecciones personalizadas:**
 - Con el módulo `mod_rewrite`
- **8. Configuración de alias o rutas especiales**
 - Se pueden definir Alias para servir contenido estático desde otras ubicaciones.
- **9. Configuración de permisos de usuarios**
 - Controlar que usuarios pueden acceder a que recursos
- **10. Configuración de módulos específicos**
 - Activar o desactivar módulos como `mod_proxy`, `mod_headers`, etc.

AllowOverride

- **AllowOverride**: pertenece al núcleo de Apache, controla si se permite usar archivos **.htaccess** en ese directorio:

Valor	Significado
None	No se permite <code>.htaccess</code> (más seguro y rápido)
All	Se permite cualquier directiva en <code>.htaccess</code>
FileInfo	Permite directivas como <code>mod_rewrite</code> , <code>mod_alias</code> , etc.
AuthConfig	Permite directivas de autenticación
Limit	Permite directivas de control de acceso (<code>Allow</code> , <code>Deny</code>)

Directory (directivas)

Directiva	¿Qué hace?
<code>Options</code>	Define qué características están habilitadas en ese directorio
<code>AllowOverride</code>	Permite o bloquea el uso de archivos <code>.htaccess</code>
<code>Require</code>	Controla quién puede acceder (por IP, usuario, grupo, etc.)
<code>Order</code>	(Obsoleta en Apache 2.4+) Define el orden de evaluación de <code>Allow</code> y <code>Deny</code>
<code>Allow</code> / <code>Deny</code>	(Obsoletas en Apache 2.4+) Permiten o bloquean acceso según IP
<code>AuthType</code>	Define el tipo de autenticación (ej. <code>Basic</code>)
<code>AuthName</code>	Texto que aparece en el cuadro de autenticación
<code>AuthUserFile</code>	Ruta al archivo de usuarios autorizados
<code>Require valid-user</code>	Permite acceso solo a usuarios autenticados

Require ip 192.168.1.0/24
Require not ip 10.0.0.1

Ajustes de Rendimiento

Ajustes de rendimiento

- **Fundamental para el rendimiento:**

- keepAlive
- Timeout
- ThreadsPerChild / MaxRequestWorkers

Críticas para la gestión de las Conexiones.

- **Otros parámetros:**

- Compresión GZIP
- Caché del Navegador
- Desactivar módulos innecesarios
- Optimizar el tamaño de los logs
- Usar HTTP/2
- Monitorizar con herramientas externas
- Tamaño de los buffers.

Mejoran la velocidad de respuesta
Uso de recursos y experiencia de Usuario.

KeepAlive

- **keepAlive** es una funcionalidad que permite **mantener abierta la conexión TCP** entre el cliente (navegador) y el servidor para múltiples solicitudes HTTP.
- **¿Qué hace keepAlive?**
 - No abre una conexión por cada recurso: HTML, CSS, img, etc.
 - Permite reutilizar la misma conexión para varias solicitudes.
 - Reduce la latencia (t. de respuesta), mejora el rendimiento y disminuye el uso de recursos del servidor.

KeepAlive

- Pueden estar definidas en el archivo httpd.conf.
- Si no están apache utiliza estos valores por defecto:
 - **KeepAlive On**
 - **MaxKeepAliveRequests 100**
 - **KeepAliveTimeout 5**
- Si queremos buscar el archivo de configuración de Apache
Podemos lanzar el comando: **httpd -V**
 - Y localizar la propiedad: **-D SERVER_CONFIG_FILE="conf/httpd.conf"**

KeepAlive

- **KeepAlive:** On / Off
 - Para reutilizar conexiones. En vez de cerrar / abrir conexiones, las mantiene abiertas.
- **MaxKeepAliveRequests:** número de conexiones
 - Número máximo de solicitudes HTTP que Apache permite por una sola conexión persistente (KeepAlive)
- **KeepAliveTime:** segundos
 - Es el número de segundos que espera el servidor a cerrar una conexión, antes de que venga otra petición

KeepAlive

- **Cuando un cliente (como un navegador) establece una conexión con Apache y KeepAlive está activado:**
 - Apache mantiene esa conexión abierta.
 - El cliente puede enviar múltiples solicitudes (por ejemplo, para HTML, CSS, JS, imágenes) **sin abrir nuevas conexiones**.
 - MaxKeepAliveRequests define cuántas de esas solicitudes puede procesar Apache **antes de cerrar la conexión**.
- Si tienes MaxKeepAliveRequests 100, Apache permitirá hasta 100 solicitudes HTTP por conexión.
- Después de la solicitud número 100, Apache cerrará la conexión, aunque el cliente quiera seguir usándola.

KeepAlive

- **Si no se activa KeepAlive:**
 - **El servidor cierra la conexión después de cada solicitud.**
 - Cada vez que el navegador necesita un recurso (HTML, CSS, JS, imagen, fuente...), debe:
 - Abrir una nueva conexión TCP.
 - Realizar la solicitud.
 - Recibir la respuesta.
 - Cerrar la conexión.
 - Esto genera más carga en el servidor y más latencia para el usuario, especialmente en páginas con muchos elementos.

KeepAlive

- Las directivas se colocan en el archivo de configuración, cerca de la propiedad **ServerRoot** o **Listen**.
- Se pueden hacer comprobaciones si están o no activadas, utilizar el comando **curl** en la consola.
 - Comprobar si tenemos el comando: **curl --version**
 - Si no lo tenemos se puede descargar del sitio oficial: <https://curl.se/windows/>
 - En Ubuntu:
 - `sudo apt update`
 - `sudo apt install curl`
 - curl nos sirve para hacer peticiones a servicios REST para pruebas:
 - **curl -X POST -d "nombre=Juan" https://api.ejemplo.com**

keepAlive

- Se pueden hacer comprobaciones con el comando: **curl**
- En una consola: `curl -I http://localhost --header "Connection: keep-alive"`
- **`curl -I http://localhost --header "Connection: keep-alive"`**
 - HTTP/1.1 200 OK
 - Date: Sat, 25 Oct 2025 14:50:48 GMT
 - Server: Apache/2.4.57 (Win64) PHP/8.2.29
 - Last-Modified: Mon, 11 Jun 2007 18:53:14 GMT
 - ETag: "2e-432a5e4a73a80"
 - Accept-Ranges: bytes
 - Content-Length: 46
 - **Keep-Alive: timeout=5, max=100**
 - Connection: Keep-Alive
 - Content-Type: text/html

Configuración keepAlive

Directiva	Valor recomendado	Descripción
<code>KeepAlive</code>	<code>On</code>	Activa la reutilización de conexiones TCP.
<code>MaxKeepAliveRequests</code>	<code>100</code> a <code>500</code>	Número máximo de solicitudes por conexión. Valores más altos benefician sitios con muchos recursos por página.
<code>KeepAliveTimeout</code>	<code>2</code> a <code>5</code> segundos	Tiempo que Apache espera antes de cerrar una conexión inactiva. Valores bajos reducen el uso de recursos.

Configuración KeepAlive

- Sitios con mucho **contenido estático** (imágenes, CSS, JS):
 - MaxKeepAliveRequests 200
 - KeepAliveTimeout 3
- Sitios dinámicos con **PHP o bases de datos**:
 - MaxKeepAliveRequests 100
 - KeepAliveTimeout 2
- Servidores con alta concurrencia:
 - Mantén KeepAliveTimeout bajo (1–2 segundos) para liberar conexiones rápidamente.

Ventajas / desventajas

- **Consecuencias de desactivar KeepAlive**

- **Más conexiones simultáneas** → mayor uso de CPU y memoria.
- **Mayor tiempo de carga** → cada recurso requiere su propia conexión.
- **Menor eficiencia** → especialmente en redes lentas o con alta latencia.

- **Con KeepAlive activo:**

- El rendimiento general del sitio.
- La experiencia del usuario.
- La eficiencia del servidor, al reducir el número de conexiones abiertas y cerradas constantemente.

Ejemplo

- Documento HTML con 4 imágenes, y una CSS.
- 6 recursos → 6 peticiones HTTP
- **Sin KeepAlive**
 - Cada petición requiere una nueva conexión TCP
 - 6 peticiones → 6 conexiones.
- **Con KeepAlive**
 - Se puede reutilizar la misma conexión para varias peticiones.
 - Puede ser una única conexión, dependiendo como se configuren las directivas de KeepAlive.

Timeout

- Por defecto, **timeout son 60 segundos**
- **Puede que no veamos la directiva en el archivo httpd.conf.**
 - Se puede colocar cerca de ServerRoot o Listen
- Si la ponemos: timeout 30
- Es el tiempo de espera máximo que el servidor espera por actividad es una conexión.
 - Si el timeout es 60, y en ese tiempo no hay actividad a través de una petición, la conexión se cierra.

¿Qué controla timeout?

- **Lectura de la solicitud:** cuánto tiempo Apache espera a que el cliente envíe la solicitud completa.
- **Lectura de datos POST/PUT:** cuánto tiempo espera a que el cliente envíe el cuerpo de la solicitud.
- **Escritura de la respuesta:** cuánto tiempo espera para enviar datos al cliente si este está lento o no responde.
- **Conexiones con backends:** si usas módulos como mod_proxy, también aplica al tiempo de espera para respuestas del servidor de destino.

Timeout vs KeepAliveTimeout

- Timeout no reemplaza a KeepAliveTimeout, pero puede actuar como un límite superior.
- Si una conexión KeepAlive está abierta pero inactiva, y el cliente no responde, Timeout puede cerrarla si se supera el tiempo definido.
- **Timeout 60**
- **KeepAliveTimeout 5**
 - *Apache espera hasta 5 segundos por una nueva solicitud en una conexión KeepAlive.*
 - *Pero si hay una solicitud en curso (por ejemplo, un POST grande), Apache puede esperar hasta 60 segundos antes de abortarla.*

Ajustes

- Para **servidores con alta concurrencia: Timeout 30 o incluso 15 puede ser más eficiente.**
- Para **aplicaciones lentas o con clientes móviles: Timeout 60–120 puede evitar cortes prematuros.**

Revisar access.log

- ¿Solicitan muchos archivos estáticos (CSS, JS, imágenes)?
- ¿Hay muchas peticiones desde la misma IP?
- ¿Qué páginas generan más tráfico?
- ¿Hay errores frecuentes (404, 500)?
- Este archivo contiene cada solicitud HTTP con detalles como:
 - IP del cliente
 - Fecha y hora
 - Recurso solicitado (HTML, CSS, imágenes...)
 - Código de estado (200, 404, etc.)
 - Tamaño de la respuesta

Herramientas

- Herramientas **GoAccess** (para Windows, pero con WSL)
 - <https://goaccess.io/download#windows>
- **AWStats**
 - <https://awstats.sourceforge.io/>

MaxClients → MaxRequestWorkers

- A partir de **Apache 2.4 se renombra** (la versión de apache, la podemos ver con **httpd -v**)
- Esta directiva tiene que ver con la **conurrencia**.
- Influye directamente en el **rendimiento** del Servidor.
- **Define el número máximo de solicitudes simultáneas que Apache puede atender al mismo tiempo.**
 - Si se supera el valor, las nuevas conexiones quedan en cola o son rechazadas.
- **Un valor bajo:**
 - Los **usuarios** pueden experimentar **lentitud** o **errores** al acceder al sitio.
- **Un valor alto:**
 - Se puede agotar la **RAM** y **sobrecargar** el **Servidor**.

Cálculo del valor

- Aproximadamente cada proceso de Apache consume 50 Mb
- Un servidor con 4 Gb de RAM → 4096 Mb
- $N = 4096 / 50 \rightarrow 81,92 \approx 80$
- **MaxRequestWorkers 80 (valor de partida)**
- **Revisar:**
 - Revisa los logs: si ves errores como **server reached MaxRequestWorkers setting**, necesitas aumentarlo.
 - Usa herramientas como `mod_status` o **Apache Bench** para simular carga y ver si el servidor responde bien.

Configuración

- En **Windows** se utiliza el módulo **npm_winnt_module**, es normal que no aparezca en una línea **LoadModule** en **httpd.conf**.
- Como comprobarlo: en una consola → **httpd -V**

```
Server version: Apache/2.4.57 (Win64)
Apache Lounge VS16 Server built:  Apr 14 2023 09:42:54
Server's Module Magic Number: 20120211:127
Server loaded:  APR 1.7.4, APR-UTIL 1.6.3, PCRE 10.42 2022-12-11
Compiled using: APR 1.7.4, APR-UTIL 1.6.3, PCRE 10.42 2022-12-11
Architecture:  64-bit
Server MPM:      WinNT
    threaded:    yes (fixed thread count)
    forked:      no
```

Configuración en Windows

- Se puede definir, en **httpd.conf**:

<IfModule mpm_winnt_module>

ThreadsPerChild 150

MaxConnectionsPerChild 0

</IfModule>

Apache puede manejar hasta **150** solicitudes simultáneas.

Necesitaríamos unas 8 Gb de RAM

150 x 50 Mb = 7500 Mb

El proceso hijo no se reinicia automáticamente.

- **Significado:**

- **ThreadsPerChild:** número de hilos por proceso (maneja **conurrencia**).
- **MaxConnectionsPerChild:** número máximo de conexiones antes de reiniciar el proceso (0 = ilimitado).

Configuración en Linux

- En **Linux** si tenemos **MaxRequestWorkers**

```
<IfModule mpm_prefork_module>  
    StartServers      5  
    MinSpareServers   5  
    MaxSpareServers   10  
    MaxRequestWorkers 150  
    MaxConnectionsPerChild 1000  
</IfModule>
```


Resumen

Sistema operativo	Directiva principal de concurrencia
Linux (prefork/worker/event)	<code>MaxRequestWorkers</code>
Windows (mpm_winnt)	<code>ThreadsPerChild</code>

Otros ajustes de rendimiento

Compresión GZIP

- Reduce el tamaño de los archivos enviados al navegador, acelerando la carga.
- `<IfModule mod_deflate.c>`
 - `AddOutputFilterByType DEFLATE text/html text/plain text/xml text/css application/javascript`
- `</IfModule>`

Caché del navegador

- Evitar que los clientes descarguen los mismos archivos repetidamente.
- Ideal para sitios con muchos recursos estáticos.
- `<IfModule mod_expires.c>`
 - ExpiresActive On
 - ExpiresByType image/jpg "access plus 1 month"
 - ExpiresByType text/css "access plus 1 week"
- `</IfModule>`

Desactivar módulos innecesarios

- Cada módulo activo consume recursos, puedes comentar los que no uses:
- `#LoadModule status_module modules/mod_status.so`
- `#LoadModule autoindex_module modules/mod_autoindex.so`
- Con esto liberamos memoria y mejoramos la velocidad.

Optimizar el tamaño de los logs

- Los archivos de log pueden crecer rápidamente.
- Utilizar rotación de logs o limita su tamaño para evitar que afecten el rendimiento.
- ***En el siguiente apartado.***

Usar HTTP/2

- HTTP/2 permite multiplexar múltiples solicitudes en una sola conexión, reduciendo la latencia. Requiere:
 - Apache 2.4.17 o superior
 - mod_http2
 - Certificado SSL

Monitorear con herramientas externas

- Puedes usar:
 - **Apache Bench** (ab.exe) para simular carga.
 - **Wireshark** para analizar tráfico.
 - **AWStats** o **GoAccess** para visualizar estadísticas.
-
- *En el siguiente apartado.*

Ajustar el tamaño de los buffers

- En algunos casos, ajustar `LimitRequestBody`, `LimitRequestFields`, o `BufferSize` puede ayudar a controlar el uso de memoria y evitar abusos.

Logs y monitorización

Logs

- Tipos de logs en Apache
 - **access.log**
 - Registra cada solicitud HTTP que llega al servidor.
 - C:\Apache24\logs\access.log
 - **error.log**
 - Registra errores del servidor, problemas de configuración, fallos de módulos, etc.
 - C:\Apache24\logs\error.log

Access.log

- Contenido:
 - IP del cliente
 - Fecha y hora
 - Método HTTP (GET, POST...)
 - Recurso solicitado
 - Código de estado (200, 404, 500...)
 - Tamaño de la respuesta
- 127.0.0.1 - - [25/Oct/2025:18:56:12 +0200] "**GET** /index.html HTTP/1.1"
200 2326
 - 200 → status ok
 - 2326 → bytes (tamaño del recurso solicitado)

Error.log

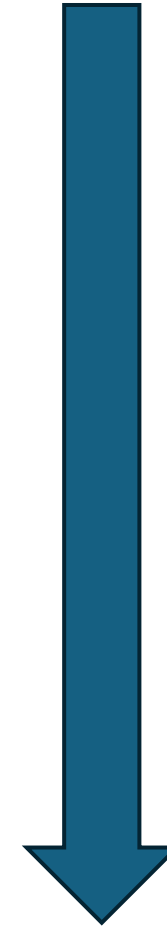
- Contenido:
 - Nivel del error (notice, warn, error, crit)
 - Fecha y hora
 - Mensaje descriptivo
 - Archivo y línea implicada (si aplica)
- [Sat Oct 25 18:56:12.123456 2025] [core:error] [pid 1234:tid 456] [client 127.0.0.1:54321] File does not exist: C:/Apache24/htdocs/favicon.ico

En httpd.conf

- **LogLevel:** controla qué tan detallado es el log de errores (debug, info, notice, **warn**, error, crit, alert, emerg).
 - **Por defecto, suele estar a nivel de warning**
- **ErrorLog "logs/error.log"**
- **CustomLog:** define el formato del log de acceso.
 - Puedes usar:
 - **common:** formato estándar
 - **combined:** incluye referer y user-agent

LogLevel

Nivel	Uso típico
<code>debug</code>	Diagnóstico detallado (útil para desarrollo)
<code>info</code>	Eventos informativos
<code>notice</code>	Eventos importantes pero no problemáticos
<code>warn</code>	Advertencias que podrían causar problemas
<code>error</code>	Fallos que afectan la funcionalidad
<code>crit</code>	Errores graves
<code>alert</code>	Requiere atención inmediata
<code>emerg</code>	El sistema está inutilizable



**Menos
importante**

**Más
importante**

LogLevel

- El nivel que establecemos es el mínimo de mensajes que se mostrarán.
- Se muestran del nivel establecido hacia el más importante (o grave) → **emerg**
- Desde **PHP** tenemos la función **error_log(“mensaje al log”)**
 - No podemos indicar el nivel, pero lo podemos simular.
 - `error_log("[ERROR] Fallo al conectar con la base de datos");`
 - `error_log("[WARN] Tiempo de respuesta alto");`
 - `error_log("[INFO] Usuario inició sesión");`
 - En PHP.ini tenemos que tener:
 - **log_errors = On** → Obligatoria para registrar los errores.
 - Para personalizar el fichero:
 - **error_log = “C:\Apache24\logs\php_errors.log”** → Si no se indica, va a error.log de Apache

LogLevel

- **error_log** genera los mensajes a nivel notice:
 - [Sat Oct 25 20:42:59.481115 2025] [**php:notice**] [pid 23468:tid 1168] [client ::1:56781] [ERROR] Fallo al conectar con la base de datos, referer: <http://localhost/apache/>
 - [Sat Oct 25 20:42:59.481115 2025] [**php:notice**] [pid 23468:tid 1168] [client ::1:56781] [WARN] Tiempo de respuesta alto, referer: <http://localhost/apache/>
 - [Sat Oct 25 20:42:59.481115 2025] [**php:notice**] [pid 23468:tid 1168] [client ::1:56781] [INFO] Usuario inici\xc3\xbb sesi\xc3\xbbn, referer: <http://localhost/apache/>

LogLevel

- En **PHP** la función **trigger_error** es más potente porque permite establecer el nivel de log.
- La función `error_log` se puede simular, pero el nivel es `notice`.
- `trigger_error("mensaje", NIVEL);`
- `E_USER_WARNING`, `E_USER_DEBUG`, `E_USER_ERROR`, etc.
- Sat Oct 25 20:50:56.442589 2025] [**php:warn**] [pid 23468:tid 1160] [client ::1:56866] PHP Warning: Esto es un aviso in D:\\apache\\Apache24\\htdocs\\apache\\logs\\index.php on line 8, referer: <http://localhost/apache/>
- [Sat Oct 25 20:50:56.442589 2025] [**php:error**] [pid 23468:tid 1160] [client ::1:56866] PHP Fatal error: Esto es un error in D:\\apache\\Apache24\\htdocs\\apache\\logs\\index.php on line 9, referer: http://localhost/apache/

Logs y Virtual Hosts

- Dentro de un host virtual se pueden definir logs, si no, hacemos esto los mensajes quedarán registrados en el log del servidor, si no, irán al del host.

Configuración: Access.log

```
<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
```

```
</IfModule logio_module>
```

```
# You need to enable mod_logio.c to use %I and %O
```

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" combinedio
```

```
</IfModule>
```

```
CustomLog "logs/access.log" common
```

```
#CustomLog "logs/access.log" combined
```

Podemos elegir el formato de los logs

```
</IfModule>
```

Access.log

- **combined**

- *LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" combined*
 - IP del cliente (%h)
 - Identidad remota (%l, casi siempre -)
 - Usuario autenticado (%u)
 - Fecha y hora (%t)
 - Solicitud completa (%r)
 - Código de estado (%>s)
 - Tamaño de la respuesta (%b)
 - Referer (%{Referer}i)
 - User-Agent (%{User-Agent}i)

Access.log

- **common**
 - ***LogFormat "%h %l %u %t \"%r\" %>s %b" common***
- Podemos elegir entre **common** (más **ligero**) y **combined** (más **detallado**)
- ***Si estás depurando o analizando tráfico, combined te da más contexto sobre el origen y tipo de cliente.***
- ***Si activas mod_logio, puedes medir el volumen de datos transferido por solicitud.***

Error.log

- Apache no dispone de una etiqueta **IfModule** para configurar **log_error**.
- Se hace sólo a través de las directivas: **ErrorLog** → path al fichero de log y **LogLevel** (nivel de log).

Resumen: Configuración de logs

Característica	Access Log (<code>access.log</code>)	Error Log (<code>error.log</code>)
Módulo asociado	<code>log_config_module</code>	Parte del núcleo
Bloque <code><IfModule></code>	Sí	No necesario
Formatos personalizables	Sí (<code>LogFormat</code>)	No (solo texto plano)
Nivel de detalle	Configurable con <code>LogLevel</code>	Configurable con <code>LogLevel</code>

Rotación de Logs

- Apache no rota los logs automáticamente.
- Apache para Windows incluye una utilidad llamada **rotatelog.exe** que permite **rotar los logs por tamaño o por tiempo**.
 - **Configuración por tiempo:**
 - CustomLog "|bin/rotatelog.exe logs/access-%Y-%m-%d.log 86400" combined
 - ErrorLog "|bin/rotatelog.exe logs/error-%Y-%m-%d.log 86400"
 - |bin/rotatelog.exe: ejecuta el programa de rotación.
 - logs/access-%Y-%m-%d.log: crea un nuevo archivo cada día con la fecha en el nombre.
 - 86400: número de segundos entre rotaciones (86400 = 1 día).

Rotación de Logs

- **Configuración por tamaño:**

- CustomLog "|bin/rotatelogs.exe -l logs/access-%Y-%m-%d_%H-%M.log 10M" combined

- Ubicación de la configuración:

- La línea **CustomLog** debe ir **dentro** del bloque **log_config_module**.
- La línea **ErrorLog** puede ir **fuera** del bloque, ya que no depende de log_config_module.

Ejemplo

- **En httpd.conf**

- `<IfModule log_config_module>`
 - `LogFormat "%h %l %u %t \"%r\" %>s %b" common`
 - `LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" combined`
 - **CustomLog** `"|bin/rotatelog.exe logs/access-%Y-%m-%d.log 86400" combined`
- `</IfModule>`
- **ErrorLog** `"|bin/rotatelog.exe logs/error-%Y-%m-%d.log 86400"`

Formatos personalizados

- Se puede definir formatos personalizados en los logs
- `LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %D" detallado`
- `CustomLog "logs/detallado.log" detallado`
- **Utilizando las directivas: LogFormat y CustomLog**
- **Reiniciar apache**

Formatos personalizados

- "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %D"
- Este formato incluye:
 - IP del cliente
 - Usuario autenticado
 - Fecha y hora
 - Solicitud HTTP
 - Código de estado
 - Tamaño de respuesta
 - Referer
 - User-Agent
 - Tiempo de respuesta en microsegundos

Tabla de formatos I

Especificador	Descripción
%a	Dirección IP del cliente (remota)
%A	Dirección IP del servidor (local)
%B	Tamaño de la respuesta en bytes (sin encabezados)
%b	Tamaño de la respuesta en bytes, o <code>-</code> si es cero
%D	Tiempo que tardó en servir la solicitud (en microsegundos)
%h	Nombre del host remoto o IP del cliente
%H	Protocolo utilizado (por ejemplo, HTTP/1.1)
%l	Identidad del cliente (usualmente <code>-</code>)
%m	Método HTTP (GET, POST, etc.)
%p	Puerto del servidor

Tabla de formatos II

<code>%q</code>	Cadena de consulta (parte de la URL después de <code>?</code>)
<code>%r</code>	Línea de solicitud completa: método, recurso y protocolo
<code>%>s</code>	Código de estado HTTP
<code>%t</code>	Fecha y hora de la solicitud
<code>%T</code>	Tiempo que tardó en servir la solicitud (en segundos)
<code>%u</code>	Nombre de usuario autenticado
<code>%U</code>	URL solicitada sin la cadena de consulta
<code>%v</code>	Nombre del servidor (según la configuración de Apache)
<code>%V</code>	Nombre del servidor usado en la solicitud (Host header)

En un Virtual Host

- Puedes definir un formato de log personalizado **dentro de un bloque** <VirtualHost> en Apache para que **solo afecte a ese sitio específico**.
- Esto es muy útil cuando tienes múltiples sitios alojados en el mismo servidor y quieres que cada uno tenga su propio estilo de logging.

Ejemplo

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

```
    ServerName ejemplo.com
```

```
    DocumentRoot "C:/Apache24/htdocs/ejemplo"
```

```
    # Formato de log personalizado solo para este sitio
```

```
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{User-Agent}i\"" sitioFormato
```

```
    CustomLog "logs/ejemplo_access.log" sitioFormato
```

```
</VirtualHost>
```

Monitorización

Monitorización de Logs

- Disponemos de varias herramientas para la monitorización de logs:

Herramienta	Tipo	Ideal para...
Log Parser Studio	GUI + SQL	Análisis avanzado
BareTail	Tiempo real	Monitoreo en vivo
SnakeTail	Tiempo real	Alertas y múltiples archivos
Apache Log Viewer	Apache específico	Estadísticas y filtrado
Glogg	Visualizador	Archivos grandes y búsqueda

Monitorización de logs

- **Apache Log Viewer**

- Diseñado específicamente para logs de Apache.
- Filtra por IP, fecha, código de estado, etc.
- Exporta a CSV y genera estadísticas básicas.
- <https://www.apacheviewer.com/>