

# **Administración de módulos en Apache**

Antonio Espín Herranz

# Contenidos

- Módulos más importantes de Apache.
- Introducción a `mod_rewrite`, `mod_ssl`, `mod_security`.
- Gestión de módulos dinámicos.
- Optimización de carga y rendimiento.
- Instalación y configuración de módulos

# Módulos

- Los **módulos de Apache** son componentes que amplían las funcionalidades del servidor web.
- Permiten añadir soporte a cosas como autenticación, compresión, reescritura de URLs, SSL, estadísticas, y mucho más.
- Se **activan y configuran** en Apache, especialmente en Windows.
- Sólo activar los módulos que necesitemos → activar módulos innecesarios penaliza en el rendimiento.

# Módulos más importantes de Apache

- Saber que módulos tenemos instalados:
- `httpd -M`
- Por funcionalidad tenemos:
  - Módulos de seguridad.
  - Módulos de rendimiento
  - Módulos de reescritura y redirección
  - Módulos de monitoreo y administración
  - Otros módulos útiles

# Módulos de Seguridad

- **mod\_security**: Firewall de aplicaciones web (WAF). Bloquea ataques como inyecciones SQL, XSS, etc.
- **mod\_evasive**: Protege contra ataques DoS y escaneos agresivos.
- **mod\_auth\_basic**: Implementa autenticación básica con usuario y contraseña.
- **mod\_ssl**: Habilita HTTPS y cifrado TLS.

# Módulos de rendimiento

- **mod\_deflate**: Comprime contenido con Gzip para reducir el tamaño de las respuestas.
- **mod\_pagespeed**: Optimiza automáticamente HTML, CSS, JS e imágenes.
- **mod\_cache**: Almacena contenido en caché para acelerar la entrega.
- **mod\_expires**: Controla la expiración de recursos estáticos (ideal para navegadores).

# Módulos de reescritura y redirección

- **mod\_rewrite:** Reescribe URLs dinámicamente. Muy usado para SEO y redirecciones.
- **mod\_alias:** Redirecciona rutas o mapea URLs a carpetas específicas.
- **mod\_speling:** Corrige errores ortográficos en URLs (útil en sistemas sensibles a mayúsculas/minúsculas).

# Módulos de monitoreo y administración

- **mod\_status:** Muestra el estado del servidor en tiempo real (procesos, carga, etc.).
- **mod\_info:** Expone la configuración activa del servidor (útil para auditorías).
- **mod\_log\_config:** Controla el formato y destino de los logs de acceso.



# Otros módulos útiles

- **mod\_userdir**: Permite acceder a carpetas personales vía `http://localhost/~usuario`.
- **mod\_autoindex**: Genera listados de directorios si no hay `index.html`.
- **mod\_cgi**: Ejecuta scripts CGI (como Perl o Python).

# Módulos: mod\_rewrite, mod\_ssl, mod\_security

- **mod\_rewrite**

- **Función:** Permite reescribir URLs de forma dinámica mediante reglas basadas en expresiones regulares.
- **Usos comunes:** Crear URLs amigables, redirecciones 301/302, forzar HTTPS, control de acceso por IP o cabeceras.
- **Ejemplo:** Convertir /producto/123 en producto.php?id=123.

- **mod\_ssl**

- **Función:** Habilita el soporte para conexiones HTTPS mediante el protocolo SSL/TLS.
- **Usos comunes:** Cifrado de datos entre el cliente y el servidor, configuración de certificados digitales, cumplimiento de normativas de seguridad.
- **Ejemplo:** Configurar un VirtualHost en el puerto 443 con un certificado SSL.

- **mod\_security**

- **Función:** Actúa como un firewall de aplicaciones web (WAF) para proteger contra ataques como inyecciones SQL, XSS, etc.
- **Usos comunes:** Filtrado de peticiones maliciosas, reglas de seguridad personalizadas, cumplimiento de OWASP.
- **Ejemplo:** Bloquear peticiones con patrones sospechosos en los parámetros de la URL.

# Expresiones Regulares

- Dentro de los módulos de **Apache** vamos a tener que utilizar expresiones regulares, por ejemplo: en el módulo de **mod\_rewrite**:
- En las **expresiones regulares** tendremos:
  - **Metacaracteres básicos**
  - **Cuantificadores**
  - **Caracteres especiales**

# Metacaracteres básicos

Símbolo	Significado	Ejemplo		
.	Cualquier carácter excepto salto de línea	a.c → coincide con abc, axc		
^	Inicio de línea o cadena	^hola → debe comenzar con "hola"		
\$	Fin de línea o cadena	fin\$ → debe terminar con "fin"		
*	0 o más repeticiones del carácter anterior	a* → "", a, aa, aaa		
+	1 o más repeticiones	a+ → a, aa, aaa		
?	0 o 1 repetición (opcional)	colou?r → color o colour		
		Alternativa (OR)	`rojo	azul` → coincide con uno u otro
()	Agrupar patrones y captura	(abc)+ → abc, abcabc		
[]	Clase de caracteres	[aeiou] → cualquier vocal		
[^]	Negación de clase	[^0-9] → cualquier no dígito		

# Cuantificadores

Símbolo	Significado	Ejemplo
<code>{n}</code>	Exactamente n repeticiones	<code>\d{3}</code> → exactamente 3 dígitos
<code>{n,}</code>	Al menos n repeticiones	<code>\d{2,}</code> → 2 o más dígitos
<code>{n,m}</code>	Entre n y m repeticiones	<code>\d{2,4}</code> → 2 a 4 dígitos

# Caracteres especiales

Secuencia	Significado	Equivalente
<code>\d</code>	Dígito	<code>[0-9]</code>
<code>\D</code>	No dígito	<code>[^0-9]</code>
<code>\w</code>	Carácter alfanumérico o guion bajo	<code>[A-Za-z0-9_]</code>
<code>\W</code>	No alfanumérico	<code>[^A-Za-z0-9_]</code>
<code>\s</code>	Espacio en blanco	espacio, tab, salto de línea
<code>\S</code>	No espacio en blanco	cualquier carácter visible

**mod\_rewrite**

# mod\_rewrite

- ***mod\_rewrite es un módulo de Apache que permite reescribir URLs de forma dinámica usando reglas basadas en expresiones regulares.***
- Es extremadamente potente y se usa para:
  - Redireccionamientos SEO-friendly
  - Reestructuración de URLs
  - Control de acceso
  - Forzado de HTTPS
  - Proxy inverso y balanceo de carga



# mod\_rewrite

- Para utilizar el módulo: **mod\_rewrite** hay que **activarlo** en el archivo de configuración.
- En **httpd.conf**
  - **LoadModule** rewrite\_module modules/mod\_rewrite.so
- En el archivo **.htaccess** o en la configuración de Apache:
  - **RewriteEngine On**

# mod\_rewrite

- Dispone de las directivas:
- **RewriteEngine On:**
  - Activa el motor de reescritura
- **RewriteCond:**
  - Condición que debe cumplirse para aplicar la regla
- **RewriteRule:**
  - Regla que define cómo se transforma la URL
- Estas tres forman el núcleo de mod\_rewrite.

# mod\_rewrite

- **Sintaxis general:**

RewriteEngine On

**# (Opcional) Una o más condiciones**

RewriteCond <condición 1>

RewriteCond <condición 2>

...

**# Regla que se aplica si TODAS las condiciones anteriores se cumplen**

RewriteRule <patrón> <sustitución> [flags]

# mod\_rewrite

- No hay limite de RewriteCond antes de RewriteRule
- Se evalúan en orden y tienen que cumplirse todas para aplicar la regla.
- Se puede aplicar un flag [OR] y se aplica la condición con OR, la condición que contiene el flag y la siguiente.
- **Ejemplo:**
  - **Bloquea el acceso a /admin si la IP es 123.45.67.89 OR 111.22.33.44**
  - RewriteCond %{REMOTE\_ADDR} ^123\.45\.67\.89\$ [**OR**]
  - RewriteCond %{REMOTE\_ADDR} ^111\.22\.33\.44\$
  - RewriteRule ^admin - [F,L]

# mod\_rewrite

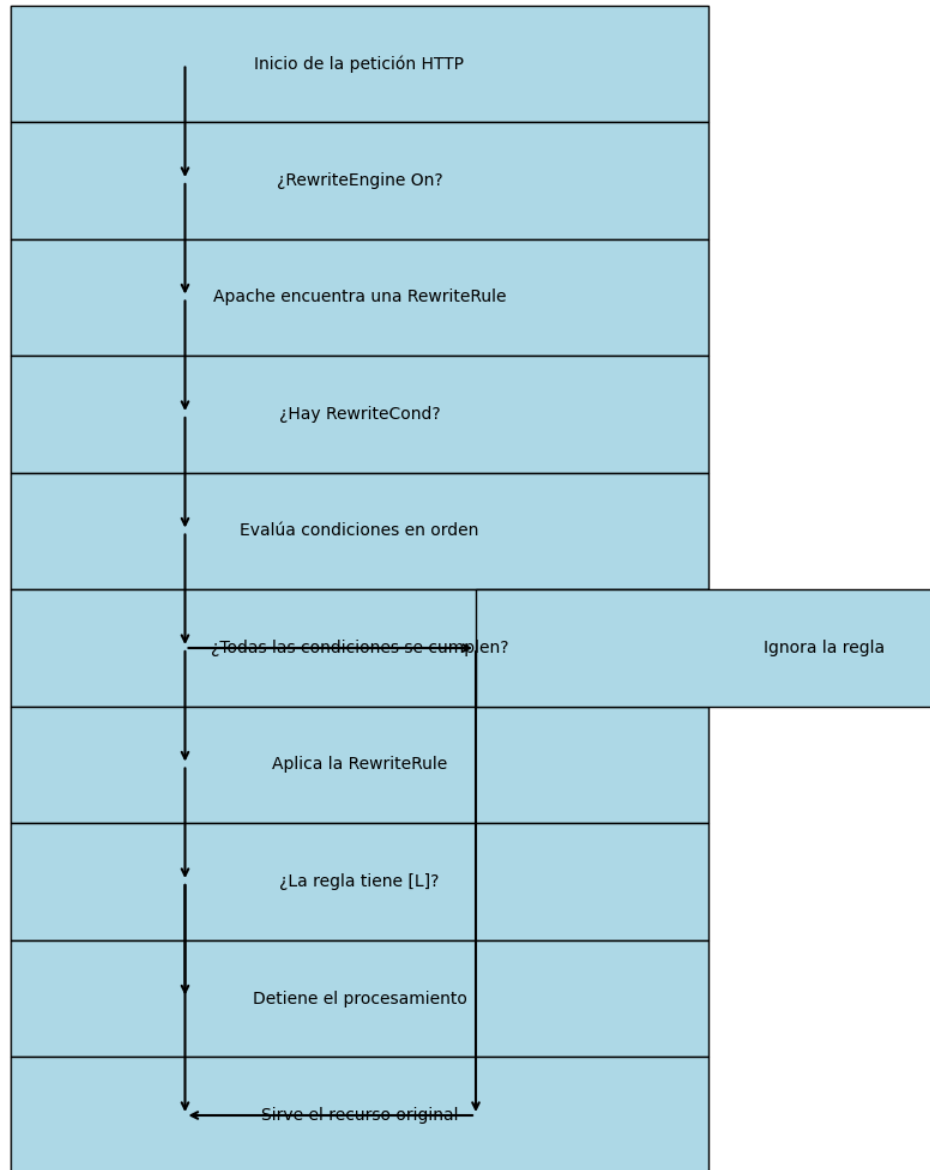
- **¿Qué pasa si no se cumplen las condiciones?**
  - Si no se cumplen las condiciones, la RewriteRule **no se ejecuta**.
  - En ese caso, Apache **sigue buscando otras reglas** (a menos que haya un **[L]** en una regla anterior que ya se aplicó).
  - Si **ninguna regla se aplica**, Apache simplemente **sirve el recurso original** como si no hubiera reglas.

# mod\_rewrite

- **Flujo de evaluación:**

- Apache encuentra una RewriteRule.
  - Si hay condiciones (RewriteCond), las evalúa:
  - Si todas se cumplen, aplica la RewriteRule.
  - Si alguna falla, ignora esa regla y pasa a la siguiente.
  - Si no hay condiciones, la RewriteRule se evalúa directamente.
  - Si una regla se aplica y tiene **[L]**, se detiene el procesamiento.
- 
- **[L]** → Significa: Last Rule, detiene el procesamiento de reglas si esta se cumple.

# Flujo de ejecución



# mod\_rewrite

- **Variables del Servidor**, se usan dentro de **RewriteCond**

Variable	Descripción
<code>%{REQUEST_URI}</code>	URI solicitada, incluyendo el path y la query string.
<code>%{REMOTE_ADDR}</code>	Dirección IP del cliente.
<code>%{HTTP_HOST}</code>	Host solicitado (dominio).
<code>%{QUERY_STRING}</code>	Cadena de consulta (lo que va después de <code>?</code> ).
<code>%{REQUEST_METHOD}</code>	Método HTTP ( <code>GET</code> , <code>POST</code> , etc.).
<code>%{SERVER_PORT}</code>	Puerto del servidor ( <code>80</code> , <code>443</code> , etc.).
<code>%{HTTPS}</code>	Si la conexión es segura ( <code>on</code> o vacío).
<code>%{TIME_YEAR}</code>	Año actual.
<code>%{ENV:VAR}</code>	Variable de entorno personalizada.
<code>%{HTTP:User-Agent}</code>	Cabecera del navegador del cliente.



# mod\_rewrite

- **Flags** que nos podemos encontrar en las reglas:

Flag	Significado
[L]	<b>Last rule:</b> detiene el procesamiento de reglas si esta se cumple.
[F]	<b>Forbidden:</b> devuelve un error 403 (acceso prohibido).
[R=301]	<b>Redirect permanente:</b> redirige con código HTTP 301 (útil para SEO).

# mod\_rewrite

- **Otros flags:**

Flag	Descripción
[R]	Redirección temporal (302 por defecto).
[G]	Gone: devuelve error 410 (contenido eliminado).
[NC]	No Case: ignora mayúsculas/minúsculas en la expresión regular.
[QSA]	Query String Append: conserva los parámetros de la URL original.
[NE]	No Escape: evita que caracteres especiales se escapen automáticamente.
[PT]	Pass Through: pasa la URL reescrita al siguiente handler (útil con mod_alias).
[NS]	No Subrequest: solo aplica en peticiones principales, no en subpeticiones.
[OR]	Or: permite combinar condiciones con lógica OR.

# mod\_rewrite

- **Grupo captura: \$1, \$2, etc.**
  - Hacen referencia al **grupo captura** dentro de la **expresión regular**
  - Es la parte de la expresión regular que va dentro de los **paréntesis**
  - Se toman de izquierda a derecha: **\$1, \$2**, etc.
  - A parte del \$ los podemos ver con %
- Los vamos a ver en **RewriteRule** y también en **RewriteCond**
- RewriteRule ^producto/([0-9]+)\$ producto.php?id=\$1 [L]

# mod\_rewrite

- **Redirección simple:**

- RewriteEngine On
- RewriteRule ^oldpage\.html\$ newpage.html [R=301,L]

- Expr. Reg:

- Si empieza por oldpage
- \. Escapar el ".", no utilizar como comodín.
- html\$ termina en extensión html
- 301 → código http: movido permanentemente.

# mod\_rewrite

- Forzar **HTTPS**
  - Redirigir todo el tráfico de http a https
  - RewriteCond %{HTTPS} off
  - RewriteRule ^(.\*)\$ https://%{HTTP\_HOST}%{REQUEST\_URI} [L,R=301]

# mod\_rewrite

- **URLs limpias:**

- Sobre todo se utiliza en servicios REST, en vez de utilizar la clásica queryString se utilizan URLs bonitas.
- Una petición entrante del estilo: producto/123 se traduce internamente a producto/?id=123
- RewriteRule ^producto/([0-9]+)\$ producto.php?id=\$1 [L]

# mod\_rewrite

- **Bloquear direcciones IP específicas:**
  - RewriteCond %{REMOTE\_ADDR} ^123.\456\789\000\$
  - RewriteRule ^.\*\$ - [F]
- [https://www.elarraydejota.com/ejemplos-de-redirecciones-utiles-y-comunes-con-mod\\_rewrite-para-apache/](https://www.elarraydejota.com/ejemplos-de-redirecciones-utiles-y-comunes-con-mod_rewrite-para-apache/)

# Ejemplo

- RewriteCond %{REQUEST\_URI} ^/admin
  - RewriteCond %{REMOTE\_ADDR} !^192\.168\.1\.
  - RewriteRule ^.\*\$ - [F,L]
- 
- Bloquea el acceso a /admin si la IP no es local.
  - Devuelve error 403.
  - Detiene el procesamiento de reglas.



# mod\_rewrite

- Con este módulo también se incluyen:
  - RewriteMap: para redirecciones basadas en archivos externos
  - Redirecciones dependientes del tiempo
  - Balanceo de carga por URL
  - Regeneración de contenido al vuelo
  - Reescritura condicional por encabezados HTTP o variables de entorno

# .htaccess vs httpd.conf

- Existen **diferencias** entre colocar la configuración de **mod\_rewrite** en el archivo **.htaccess** (por cada sitio) y **httpd.conf** (global).
- **Ventajas en httpd.conf:**
  - **Mejor rendimiento:** Apache no tiene que buscar y leer **.htaccess** en cada carpeta.
  - **Mayor seguridad:** Evita que usuarios sin privilegios modifiquen reglas críticas.
  - **Configuración más limpia y centralizada.**
- **Tener en cuenta:**
  - Requiere **acceso al sistema (root o sudo)**.
  - Cambios requieren **reiniciar Apache** (sudo systemctl restart apache2).
  - **No es práctico en entornos de hosting compartido.**

# Ejemplo

- Dentro de **httpd.conf**

```
<Directory "/var/www/html/misitio">
```

```
    AllowOverride None
```

```
    RewriteEngine On
```

```
    RewriteRule ^producto/([0-9]+)$ producto.php?id=$1 [L]
```

```
</Directory>
```

- Para Linux: /var/www/html/misitio
- Para Windows: C:/Apache24/htdocs/misitio

# Ubicación

- Dentro del fichero: **httpd.conf**
- Las reglas se encontrarán dentro de Directory o VirtualHost, pero si lo ponemos fuera (el contexto global), puede afectar a todo el servidor.

# Comparativa

Característica	<code>.htaccess</code>	<code>httpd.conf</code> (o <code>apache2.conf</code> )
 Ubicación	Dentro del directorio del sitio web	Archivo global de configuración del servidor
 Acceso requerido	Usuario con acceso al sitio (no root)	Requiere permisos de administrador (root/sudo)
 Rendimiento	Menor: Apache revisa <code>.htaccess</code> en cada petición	Mayor: se carga una vez al iniciar Apache
 Flexibilidad	Alta: permite cambios sin reiniciar Apache	Baja: requiere reinicio para aplicar cambios
 Activación de módulos	No se puede activar módulos	Se activan con <code>LoadModule</code>
 Uso de <code>mod_rewrite</code>	Requiere <code>AllowOverride FileInfo</code> o <code>All</code>	Se configura directamente sin restricciones
 Seguridad	Menor: usuarios pueden modificar reglas sensibles	Mayor: configuración centralizada y controlada
 Estructura	Solo admite directivas permitidas por <code>AllowOverride</code>	Permite bloques <code>&lt;Directory&gt;</code> , <code>&lt;VirtualHost&gt;</code> , etc.
 Aplicación por carpeta	Sí, afecta solo al directorio donde está ubicado	Sí, pero requiere definir rutas absolutas
 Uso recomendado	Hosting compartido, sitios con acceso limitado	Servidores dedicados, producción, entornos controlados

# Slim (framework de PHP para REST)

- **.htaccess**
- RewriteEngine On
- RewriteCond %{REQUEST\_FILENAME} !-f
- RewriteCond %{REQUEST\_FILENAME} !-d
- RewriteRule ^ index.php [QSA,L]
- Para redirigir todo el trafico a index.php excepto cuando el archivo o directorio solicitado existe en el sistema.
- -f fichero
- -d directorio
- ! negación
- ***Prueba: puesta en marcha con Slim***

# Slim (framework de PHP para REST)

- Con el proyecto\_empresa en la carpeta CODIGO\_OCT\_2025 del repositorio, una vez que se despliega en Apache la URL es:
  - [http://localhost/php8\\_avanzado/proyecto\\_empresa/web/backend/public/productos](http://localhost/php8_avanzado/proyecto_empresa/web/backend/public/productos)
- Después:
  - Definir un **VirtualHost** para: <http://proyecto.local/productos>
  - DocumentRoot (ajustar directorios)
    - D:/apache/Apache24/htdocs/php8\_avanzado/proyecto\_empresa/web/backend/public
  - En el index.php
    - Comentar la línea:
      - `$app->setBasePath('/php8_avanzado/proyecto_empresa/web/backend/public');`

**mod\_ssl**



# mod\_ssl

- **mod\_ssl** es el módulo de Apache que habilita el soporte para **SSL/TLS**, permitiendo conexiones seguras (**HTTPS**).
- Se basa en **OpenSSL** y permite cifrar el tráfico entre el cliente y el servidor.
- Activar el módulo en httpd.conf:
  - **LoadModule ssl\_module modules/mod\_ssl.so**
- Y el archivo de configuración SSL:
  - **Include conf/extra/httpd-ssl.conf**

# mod\_ssl

- Crear los certificados:
  - Podemos usar certificados autofirmados para pruebas o certificados reales para producción.
  - **Certificado público:** C:/Apache24/conf/ssl/ejemplo.crt
  - **Clave privada:** C:/Apache24/conf/ssl/ejemplo.key
- Utilizar el programa OpenSSL, disponible en la carpeta bin de Apache.  
`openssl req -x509 -nodes -days 365 -newkey rsa:2048 ^  
-keyout C:\Apache24\conf\ssl\ejemplo.key ^  
-out C:\Apache24\conf\ssl\ejemplo.crt`
- ^ se utiliza para indicar que el comando continua (en Windows).

# mod\_ssl: open\_ssl

Fragmento	Significado
<code>openssl</code>	Llama al programa OpenSSL
<code>req</code>	Inicia el proceso de creación de un certificado X.509
<code>-x509</code>	Indica que se generará un certificado autofirmado (no se crea CSR)
<code>-nodes</code>	No cifra la clave privada con contraseña (útil para Apache)
<code>-days 365</code>	El certificado será válido por 365 días
<code>-newkey rsa:2048</code>	Crea una nueva clave RSA de 2048 bits junto con el certificado
<code>^</code>	En Windows, indica que el comando continúa en la siguiente línea
<code>-keyout</code> <code>C:\Apache24\conf\ssl\ejemplo.key</code>	Ruta donde se guardará la clave privada
<code>-out C:\Apache24\conf\ssl\ejemplo.crt</code>	Ruta donde se guardará el certificado público

# mod\_ssl

- **Configuración en httpd-ssl.conf en Windows**

```
<VirtualHost _default_:443>
```

```
    ServerName localhost
```

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

```
    SSLEngine on
```

```
    SSLCertificateFile "C:/Apache24/conf/ssl/ejemplo.crt"
```

```
    SSLCertificateKeyFile "C:/Apache24/conf/ssl/ejemplo.key"
```

```
    # SSLCertificateChainFile "C:/Apache24/conf/ssl/chain.crt" # Descomenta si usas certificados intermedios
```

```
    <Directory "C:/Apache24/htdocs">
```

```
        Options Indexes FollowSymLinks
```

```
        AllowOverride All
```

```
        Require all granted
```

```
    </Directory>
```

```
</VirtualHost>
```

- Reiniciar Apache: **httpd -k restart**

## **Options** Indexes FollowSymLinks

**Indexes:** permite mostrar listado de archivos si no hay index.html

**FollowSymLinks:** permite seguir enlaces simbólicos

## **AllowOverride All**

Permite que **.htaccess sobrescriba** cualquier configuración

## **Require all granted**

Permite acceso a **todos** los **usuarios** (sin restricciones)

# mod\_ssl

- Puedes duplicar o adaptar este archivo si quieres configurar **múltiples sitios HTTPS** con diferentes dominios o certificados.
- Solo asegúrate de usar **VirtualHost \*:443** y cambiar el **ServerName** y **DocumentRoot** según cada caso.

# mod\_ssl

- **Directivas**

Directiva	Función
<code>SSLEngine</code>	Activa o desactiva SSL ( <code>on</code> o <code>off</code> )
<code>SSLCertificateFile</code>	Ruta al certificado público ( <code>.cert</code> )
<code>SSLCertificateKeyFile</code>	Ruta a la clave privada ( <code>.key</code> )
<code>SSLCertificateChainFile</code>	Certificados intermedios (si aplica)
<code>SSLProtocol</code>	Protocolos permitidos ( <code>all -SSLv3 -TLSv1</code> )
<code>SSLCipherSuite</code>	Cifrado permitido (puede restringirse por seguridad)
<code>SSLHonorCipherOrder</code>	Prioriza el orden del servidor en el handshake

# mod\_ssl

- Más directivas

<code>SSLHonorCipherOrder</code>	Prioriza el orden de cifrado del servidor en el handshake	<code>SSLHonorCipherOrder on</code>
<code>SSLVerifyClient</code>	Solicita verificación del cliente (para certificados cliente)	<code>SSLVerifyClient optional</code>
<code>SSLVerifyDepth</code>	Profundidad máxima de verificación en la cadena de certificados	<code>SSLVerifyDepth 2</code>
<code>SSLOptions</code>	Opciones adicionales como exportación de claves o compatibilidad	<code>SSLOptions +StdEnvVars</code>
<code>SSLPassPhraseDialog</code>	Método para ingresar la frase de contraseña de la clave privada	<code>SSLPassPhraseDialog builtin</code>
<code>SSLLog</code> / <code>SSLLogLevel</code>	(Obsoleto) Usado en versiones antiguas para registrar eventos SSL	—

# mod\_ssl

- **Configuración en httpd-ssl.conf en linux**

```
<VirtualHost *:443>
```

```
    ServerName www.ejemplo.com
```

```
    DocumentRoot "/var/www/html"
```

```
    SSLEngine on
```

```
    SSLCertificateFile /etc/ssl/certs/ejemplo.crt
```

```
    SSLCertificateKeyFile /etc/ssl/private/ejemplo.key
```

```
    SSLCertificateChainFile /etc/ssl/certs/chain.crt
```

```
    SSLProtocol all -SSLv3 -TLSv1
```

```
    SSLCipherSuite HIGH:!aNULL:!MD5
```

```
    SSLHonorCipherOrder on
```

```
</VirtualHost>
```

***Prueba con dos dominios diferentes  
Uno por http y otro https***



# mod\_ssl

- Documentación: [https://httpd.apache.org/docs/trunk/es/mod/mod\\_ssl.html](https://httpd.apache.org/docs/trunk/es/mod/mod_ssl.html)
- El módulo expone variables de entorno que se pueden utilizar en los logs.
  - **CustomLog logs/ssl\_request\_log "%t %h %{SSL\_PROTOCOL}x %{SSL\_CIPHER}x \"%r\" %b"**
    - Fecha y hora
    - IP del cliente
    - Protocolo SSL (ej. TLSv1.2)
    - Cifrado usado (ej. ECDHE-RSA-AES256-GCM-SHA384)
    - Petición HTTP
    - Tamaño de respuesta
  - Tiene que estar dentro de un bloque HTTPS si no, las variables no las va a reconocer.

# Ejemplo (*prueba, comprobar el log*)

```
<VirtualHost *:443>
```

```
    ServerName seguro.midominio.local
```

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

```
    SSLEngine on
```

```
    SSLCertificateFile "C:/Apache24/conf/ssl/server.crt"
```

```
    SSLCertificateKeyFile "C:/Apache24/conf/ssl/server.key"
```

```
    CustomLog "logs/ssl_request_log" "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
```

```
<Directory "C:/Apache24/htdocs/seguro">
```

```
    Require all granted
```

```
</Directory>
```

```
</VirtualHost>
```

# Variables SSL para CustomLog

- Para utilizarlas \${NOMBRE\_VAR}x
  - **La x tiene que estar en la cadena formato.**

Variable	Uso en CustomLog	Descripción
SSL_PROTOCOL	\${SSL_PROTOCOL}x	Protocolo negociado (ej. TLSv1.2)
SSL_CIPHER	\${SSL_CIPHER}x	Cifrado negociado (ej. AES256-GCM-SHA384)
SSL_CLIENT_S_DN	\${SSL_CLIENT_S_DN}x	DN completo del certificado del cliente
SSL_CLIENT_S_DN_CN	% \${SSL_CLIENT_S_DN_CN}x	Nombre común del cliente
SSL_CLIENT_VERIFY	\${SSL_CLIENT_VERIFY}x	Resultado de la verificación del certificado
SSL_SERVER_S_DN_CN	% \${SSL_SERVER_S_DN_CN}x	Nombre común del servidor

# Variables NO disponibles en CustomLog

- SSL\_SESSION\_ID
- SSL\_COMPRESS\_METHOD
- SSL\_CIPHER\_USEKEYSIZE

# Otro Ejemplo de CustomLog

- CustomLog logs/ssl\_access.log "%t %h %{SSL\_PROTOCOL}x %{SSL\_CIPHER}x %{SSL\_CLIENT\_S\_DN\_CN}x \"%r\" %b“
- El **CN**: Common Name (cuando estamos creando el certificado)
  - El **Common Name (CN)** es parte del **Distinguished Name (DN)** del certificado X.509. Es un campo que normalmente identifica al propietario del certificado.
  - Puede ser:
    - El **nombre completo** de una persona (ej. Juan Pérez)
    - El **nombre de una empresa** (ej. Acme Corp)
    - Un **nombre de usuario** o **ID** (ej. usuario123)
    - Un **dominio** (en certificados de servidor)

# Activar autenticación por certificado de cliente

- Crear una CA local y firmar certificados
- **1.Crear la CA (Autoridad Certificadora)**
  - A. Generar clave privada:
    - `openssl genrsa -out ca.key 4096`
  - B. Crear el certificado autofirmado:
    - `openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.crt`
- **2.Crear certificado de servidor firmado por la CA**
  - A. Generar clave privada del servidor
    - `openssl genrsa -out server.key 2048`
  - B. Crear CSR (solicitud de firma)
    - `openssl req -new -key server.key -out server.csr`
  - C. Firmar el certificado con la CA
    - `openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 365 -sha256`

# Activar autenticación por certificado de cliente

- **3. Crear certificado de cliente firmado por la CA**
  - **A. Generar clave privada del cliente**
    - `openssl genrsa -out client.key 2048`
  - **B. Crear CSR del cliente (CSR: Solicitud de firma de certificado)**
    - `openssl req -new -key client.key -out client.csr`
  - **C. Firmar el certificado del cliente con la CA**
    - `openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client.crt -days 365 -sha256`

# Resumen de ficheros

## CA: Autoridad Certificadora

Archivo	Extensión	Descripción
ca.crt	.crt	Certificado público de la CA. Se distribuye para que otros confíen en ella.
ca.key	.key	Clave privada de la CA. Se usa para firmar certificados de servidor y cliente. ¡Debe mantenerse segura!
ca.srl	.srl	Archivo de número de serie. OpenSSL lo usa para llevar el control de los certificados emitidos.

## Archivos del Servidor

Archivo	Extensión	Descripción
server.key	.key	Clave privada del servidor. Se usa para descifrar datos y demostrar identidad.
server.csr	.csr	Solicitud de firma de certificado. Contiene datos del servidor y se firma con la CA.
server.crt	.crt	Certificado público del servidor, firmado por la CA. Se presenta a los clientes.



# Resumen de ficheros

## Archivos del Cliente

Archivo	Extensión	Descripción
<code>client.key</code>	<code>.key</code>	Clave privada del cliente. Se usa para autenticarse ante el servidor.
<code>client.csr</code>	<code>.csr</code>	Solicitud de firma de certificado. Contiene datos del cliente y se firma con la CA.
<code>client.crt</code>	<code>.crt</code>	Certificado público del cliente, firmado por la CA. Se presenta al servidor.
<code>client.p12</code>	<code>.p12</code>	Archivo PKCS#12 que contiene el certificado y la clave privada del cliente. Se usa para importar fácilmente en navegadores como Firefox.

**La CA firma los .csr del servidor y del cliente, generando los .crt.**

**El servidor usa su .crt y .key para ofrecer HTTPS.**

**El cliente usa su .p12 (que incluye .crt y .key) para autenticarse ante el servidor si se requiere certificado de cliente.**

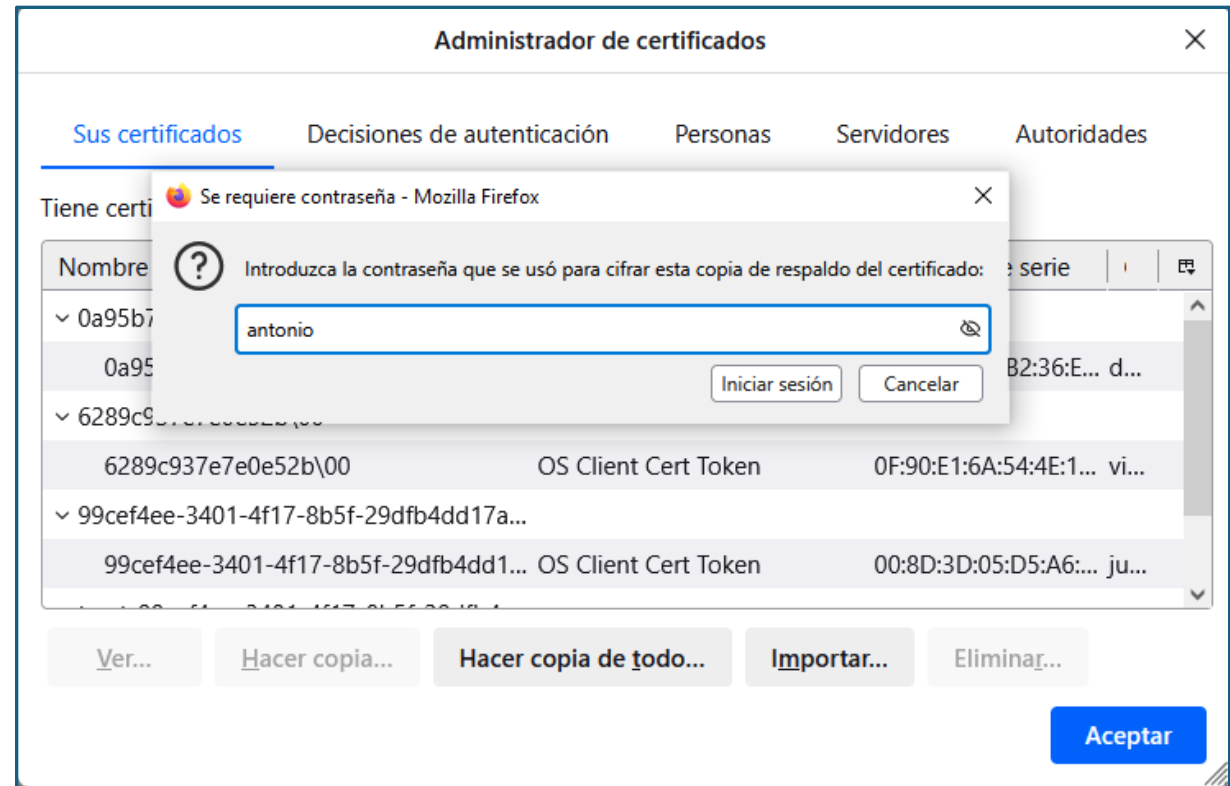
# Prueba

- **Instala client.crt en el navegador del cliente.**
- Necesitamos:
  - El archivo del certificado de cliente: **client.crt**
  - La clave privada: **client.key** (solo si el navegador lo solicita)
  - Lo ideal es tener ambos combinados en un archivo .p12 o .pfx (PKCS#12), que incluye certificado + clave
- ***openssl pkcs12 -export -out client.p12 -inkey client.key -in client.crt -name "Juan Pérez"***
- **Instalar en el Navegador...**
- Accede al sitio **HTTPS**.
- Apache pedirá el certificado, lo validará contra ca.crt, y si es válido, permitirá el acceso.

# Instalar en el Navegador

## En Firefox

- **Firefox**
  - Ve a **Configuración** → **Privacidad y seguridad** → **Certificados** → **Ver certificados**
  - Pestaña **Sus certificados** → **Importar**
  - Selecciona client.p12 y escribe la contraseña
  - El certificado estará disponible para autenticación SSL
- **Chrome / Edge (Windows)**
  - Chrome y Edge usan el almacén de certificados de Windows:
  - Abre el archivo client.p12 (doble clic)
  - Se abrirá el asistente de importación de certificados
  - Elige **Almacén personal** o **Usuarios actuales**
  - Introduce la contraseña
  - Finaliza la importación



# Activar autenticación por certificado de cliente (**en el Servidor**)

- Bloque <VirtualHost \*:443>
- SSLEngine on
- SSLCertificateFile "C:/Apache24/conf/ssl/**server.crt**"
- SSLCertificateKeyFile "C:/Apache24/conf/ssl/**server.key**"
- SSLCACertificateFile "C:/Apache24/conf/ssl/**ca.crt**"

# Activar autenticación por certificado de cliente (**en el Cliente**)

- **Para autenticar clientes:**
- SSLVerifyClient require
- SSLCACertificateFile "C:/Apache24/conf/ssl/**ca.crt**"

# Bloque virtualhost 443 (final)

```
<VirtualHost *:443>
```

```
ServerName seguro.midominio.local
```

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

## # Activar SSL

```
SSLEngine on
```

## # Certificado del servidor firmado por tu CA local

```
SSLCertificateFile "C:/Apache24/conf/ssl/server.crt"
```

```
SSLCertificateKeyFile  
"C:/Apache24/conf/ssl/server.key"
```

## # Certificado de la CA que firmó los certificados de cliente

```
SSLCACertificateFile "C:/Apache24/conf/ssl/ca.crt"
```

## # Autenticación de cliente

```
SSLVerifyClient require
```

```
SSLVerifyDepth 1
```

## # Log personalizado con CN del cliente

```
CustomLog "logs/ssl_access.log" "%t %h  
{SSL_CLIENT_S_DN_CN}x \"%r\" %>s %b"
```

## # Opcional: restringir acceso por CN específico

```
<Location />
```

```
SSLRequire %{SSL_CLIENT_S_DN_CN} eq "Juan  
Pérez"
```

```
</Location>
```

## # Permisos de directorio

```
<Directory "C:/Apache24/htdocs/seguro">
```

```
Options Indexes FollowSymLinks
```

```
AllowOverride All
```

```
Require all granted
```

```
</Directory>
```

```
</VirtualHost>
```

***OJO el certificado tiene que ser generado para Juan Perez en el CN  
El nombre que hayamos indicado al crear el certificado.***

# Bloque virtualhost 443

- SSL activado: **SSLEngine on**
- Certificados del servidor: SSLCertificateFile y SSLCertificateKeyFile
- Certificado de la CA: SSLCACertificateFile.
- Autenticación de cliente: SSLVerifyClient require obliga al navegador a presentar un certificado.
- Profundidad de verificación: **SSLVerifyDepth 1.**
  - Una CA local (sin intermediarios) Certificados de servidor y cliente firmados directamente por esa CA, hay un nivel
  - Verificarlo con el comando (en la carpeta de los certificados):
  - ***openssl verify -CAfile ca.crt client.crt***
  - ***Tiene que devolver OK***

# Bloque virtualhost 443

- Log personalizado %{SSL\_CLIENT\_S\_DN\_CN}x para registrar el **nombre común del cliente**.
- Restricción por CN: SSLRequire filtra el acceso solo si el CN del certificado cliente es "Juan Pérez" o el nombre indicado.
- Permisos de directorio para permitir acceso.



# Hosts y ficheros

- En el fichero hosts de Windows:
  - 127.0.0.1 **seguro.midominio.local**
  - C:\Windows\System32\drivers\etc\hosts
- La aplicación estará en **C:\apache24\htdocs\seguro**
- Copiar los archivos de los certificados a: **conf/ssl**
  - **ca.crt | server.crt | server.key**
- URL:
  - <https://seguro.midominio.local>

# **mod\_security**

# mod\_security

- **mod\_security** es un módulo que actúa como un **WAF (Web Application Firewall)**. Analiza las peticiones HTTP y bloquea ataques comunes como:
  - Inyección SQL
  - Cross-site scripting (XSS)
  - Inclusión de archivos (LFI/RFI)
  - Escaneos automatizados

# mod\_security

- **Directivas:**

Directiva	Función
<code>SecRuleEngine</code>	Activa el motor de reglas ( <code>On</code> , <code>DetectionOnly</code> , <code>Off</code> )
<code>SecRequestBodyAccess</code>	Permite inspeccionar el cuerpo de la petición
<code>SecRule</code>	Define una regla personalizada
<code>Include</code>	Carga reglas externas (como OWASP CRS)
<code>SecAuditLog</code>	Ruta del log de auditoría
<code>SecDefaultAction</code>	Acción por defecto para las reglas ( <code>deny</code> , <code>pass</code> , <code>log</code> , etc.)

# Gestión de módulos dinámicos

- **Módulos dinámicos vs estáticos**

- **Estáticos:** compilados directamente en Apache (no necesitan **LoadModule**).
- **Dinámicos:** se cargan con **LoadModule** desde la carpeta modules/.
- En Windows, la mayoría son dinámicos y están en C:\Apache24\modules\

- Los módulos se activan / desactivan descomentando las líneas, borrando la #

- LoadModule rewrite\_module modules/mod\_rewrite.so
- #LoadModule rewrite\_module modules/mod\_rewrite.so

- Una vez activado, podemos usar sus directivas en **httpd.conf**, **.htaccess** o dentro de bloques **<Directory>** o **<VirtualHost>**.

# Carga y rendimiento

- **1. Cada módulo consume recursos**

- Al activarse, un módulo se carga en memoria y puede ejecutar procesos en cada solicitud.
- Módulos como `mod_security`, `mod_rewrite` o `mod_deflate` analizan o transforman el tráfico, lo que puede aumentar la carga si no están optimizados.

- **2. Módulos críticos vs. opcionales**

- **Críticos:** necesarios para funciones básicas (SSL, autenticación, redirección).
- **Opcionales:** como `mod_status`, `mod_info`, `mod_autoindex`, pueden desactivarse si no se usan.

# Carga y rendimiento

- **3. Desactivar lo innecesario mejora el rendimiento**
  - Apache permite desactivar módulos no usados para liberar RAM y CPU.
  - En Linux: `a2dismod nombre_modulo`
  - En Windows: comenta la línea `LoadModule` en `httpd.conf`

# Carga y Rendimiento

- Mejoran el rendimiento

Módulo	Beneficio principal
<code>mod_deflate</code>	Compresión Gzip para reducir tamaño de respuesta
<code>mod_cache</code>	Almacenamiento en caché de contenido estático
<code>mod_expires</code>	Control de caché en navegador del cliente
<code>mod_pagespeed</code>	Optimización automática de contenido web



# Carga y Rendimiento

- Módulos que pueden ralentizar si no se configuran bien

Módulo	Riesgo de carga excesiva
<code>mod_security</code>	Inspección profunda de cada solicitud
<code>mod_rewrite</code>	Reescritura compleja puede ralentizar
<code>mod_proxy</code>	Redirección de tráfico externo