- **Páginas web estáticas:** Son archivos HTML simples que se entregan al navegador tal como están almacenados en el servidor. El contenido es fijo y no cambia a menos que un desarrollador edite manualmente el archivo. Son rápidas de cargar y fáciles de alojar, pero limitadas en interactividad y personalización.
- **Páginas web dinámicas:** Se generan en tiempo real en el servidor antes de ser enviadas al navegador. Utilizan lenguajes de programación del lado del servidor (como PHP, Python, Ruby, Node.js) y bases de datos para crear contenido personalizado y actualizado. Permiten interacción con el usuario, formularios, acceso a bases de datos y contenido adaptado.

¿Qué lenguajes se utilizan para la creación de web dinámicas?

Algunos de los lenguajes más comunes para la creación de webs dinámicas son:

- PHP: Muy popular, especialmente para sistemas de gestión de contenido como WordPress.
- Python: Con frameworks como Django y Flask, es muy versátil para diversas aplicaciones web.
- Ruby: Con el framework Ruby on Rails, conocido por su productividad.
- Node.js: Permite usar JavaScript tanto en el cliente como en el servidor.
- Java: Con tecnologías como JSP y Servlets, utilizado en entornos empresariales.
- .NET (C#): Desarrollado por Microsoft, ampliamente usado en entornos Windows.

¿Qué es una URL? ¿Qué elementos tiene?

Una URL (Uniform Resource Locator) es una dirección que identifica de forma única un recurso en la web (como una página, imagen, video, etc.) y especifica cómo acceder a él. Los elementos principales de una URL son:

- Esquema/Protocolo: Indica el protocolo que se debe utilizar para acceder al recurso (ej. http://, https://, ftp://).
- Host/Dominio: El nombre del servidor donde se encuentra el recurso (ej. www.ejemplo.com).
- Puerto (opcional): El número de puerto en el que el servidor está escuchando. Si no se especifica, se usan los puertos predeterminados para el protocolo (ej. 80 para HTTP, 443 para HTTPS).
- Ruta/Path: La ubicación específica del recurso dentro del servidor (ej. /blog/articulo.html).
- Parámetros/Query String (opcional): Información adicional que se envía al servidor, generalmente en pares clave-valor después de un ? (ej. ?id=123&categoria=web).
- Fragmento/Anchor (opcional): Una referencia a una sección específica dentro del recurso, indicada con un # (ej. #seccion2).

Ejemplo: https://www.ejemplo.com:8080/productos/camisetas.html?color=azul#descripcion

Si la web está en la capa de aplicación TPC/IP, ¿qué hay por debajo?

Por debajo de la capa de aplicación (donde se encuentra el protocolo HTTP/HTTPS de la web) en el modelo TCP/IP, encontramos las siguientes capas:

- **Capa de Transporte:** Aquí se encuentra TCP (Transmission Control Protocol), que proporciona una comunicación fiable y orientada a la conexión. Se encarga de la segmentación de datos, control de flujo y control de errores.
- **Capa de Internet:** Aquí se encuentra IP (Internet Protocol), que se encarga del direccionamiento lógico (direcciones IP) y el enrutamiento de los paquetes a través de la red.
- **Capa de Acceso a la Red/Enlace de Datos/Física:** Esta capa se encarga de la comunicación física de los datos a través del medio de transmisión (Ethernet, Wi-Fi, fibra óptica, etc.). Incluye protocolos como Ethernet, PPP, y maneja las direcciones MAC.

¿Qué es http?

HTTP (Hypertext Transfer Protocol) es el protocolo fundamental para la transferencia de información en la World Wide Web. Es un protocolo sin estado (stateless), lo que significa que cada solicitud del cliente es independiente de las anteriores. Se utiliza para la comunicación entre navegadores web (clientes) y servidores web. Por defecto, utiliza el puerto 80.

¿Qué es https?

HTTPS (Hypertext Transfer Protocol Secure) es una extensión segura del protocolo HTTP. Utiliza SSL/TLS (Secure Sockets Layer/Transport Layer Security) para cifrar la comunicación entre el navegador y el servidor, lo que proporciona autenticación, integridad de los datos y confidencialidad. Esto es crucial para transacciones seguras (banca online, compras, etc.). Por defecto, utiliza el puerto 443.

¿Qué es s-http?

S-HTTP (Secure HTTP) es un protocolo de seguridad que fue propuesto como una alternativa a SSL/TLS para proporcionar seguridad a HTTP. Sin embargo, S-HTTP no obtuvo la adopción generalizada y fue eclipsado por HTTPS/SSL/TLS. A diferencia de HTTPS que asegura la conexión de extremo a extremo, S-HTTP se enfoca en asegurar mensajes individuales dentro de la conexión HTTP. Hoy en día, HTTPS es el estándar de facto para la comunicación web segura.

¿Qué servidores web hay disponibles para Windows y para Linux?

Para Windows:

- IIS (Internet Information Services): El servidor web nativo de Microsoft, integrado en los sistemas operativos Windows Server.
- Apache HTTP Server: Disponible y ampliamente utilizado también en Windows.

- Nginx: También puede ejecutarse en Windows.
- XAMPP/WAMP: Paquetes que incluyen Apache, MySQL/MariaDB, PHP y Perl para desarrollo local.

Para Linux:

- Apache HTTP Server: El servidor web más popular y ampliamente utilizado en entornos Linux.
- Nginx: Muy popular por su alto rendimiento y eficiencia, especialmente para servir contenido estático y como proxy inverso.
- Lighttpd: Un servidor web ligero y eficiente, ideal para servidores con recursos limitados.
- Cherokee: Otro servidor web ligero y de alto rendimiento.
- Caddy: Un servidor web moderno que se enfoca en la simplicidad y seguridad, con HTTPS automático por defecto.
- XAMPP/LAMP: Paquetes que incluyen Apache, MySQL/MariaDB, PHP y Perl para desarrollo local.

2. Evolución del modo de funcionamiento de Apache

Contexto Histórico

En versiones **muy antiguas** de Apache (como Apache 1.3), el parámetro `ServerType` se utilizaba en el archivo de configuración principal (`httpd.conf`) y solo aceptaba dos valores posibles:

- `ServerType standalone`: Indicaba que Apache se ejecutaría como un demonio independiente (el modo más común y moderno).
- `ServerType inetd`: Indicaba que Apache se ejecutaría a través del super-servidor `inetd` (o `xinetd`). Este modo era raro y tenía implicaciones de rendimiento y seguridad, por lo que casi siempre se usaba el modo `standalone`.
  inetd (Internet Service Daemon) o xinetd (Extended Internet Service Daemon) es un "super-servidor" que escucha múltiples puertos para varios servicios (FTP, Telnet, POP3, etc.).
  - Cuando llega una petición a un puerto configurado (como el 80 para Apache), inetd/xinetd lanza una nueva instancia del programa del servidor (en este caso, Apache).
  - Esta instancia de Apache maneja la conexión y termina al finalizar la petición.
  - Desventaja: Este proceso de iniciar, cargar la configuración y terminar por cada solicitud es muy ineficiente para un servidor web de alto tráfico, ya que añade una sobrecarga considerable.
  - Uso: Este modo solo se recomendaba para servicios poco frecuentes o en escenarios donde la seguridad estricta (por ejemplo, mediante envoltorios TCP como tcp_wrappers) o el ahorro de memoria (si Apache no se usa casi nunca) son prioritarios sobre el rendimiento.

Estado Actual

A partir de **Apache 2.0**, el parámetro `ServerType` **dejó de existir** como directiva de configuración. Apache siempre se ejecuta en modo **standalone** (como demonio) y el comportamiento del servidor es gestionado por los **Módulos de Multiplexación de Peticiones (MPMs)**, como `mpm_prefork`, `mpm_worker` o `mpm_event`.

- Apache funciona como un demonio independiente que se inicia al arrancar el sistema y permanece en ejecución, esperando peticiones. Es el modo predeterminado y más eficiente para la mayoría de los despliegues web, ya que el servidor está siempre listo para responder.
- Se puede gestionar con 'systemd' pues se inicia como una unidad de servicio usando 'systemctl'.
- Es el modo más común y eficiente para Apache (y el predeterminado en versiones modernas).
- Mantiene procesos o threads en memoria escuchando continuamente el puerto 80 (HTTP) o 443 (HTTPS) para gestionar peticiones de manera rápida.
- Al recibir una petición, utiliza sus propios procesos ya cargados para responder, lo cual es muy eficiente para un servicio de alto tráfico.

¿Puede haber diferentes estancias del servidor corriendo el alguno de los casos anteriores?

En el caso de ServerType standalone, sí, puede haber diferentes instancias del servidor corriendo si se configuran múltiples instancias de Apache para escuchar en diferentes puertos o direcciones IP (aunque esto es menos común que usar Virtual Hosts).
En el caso de ServerType inetd, generalmente no hay múltiples estancias del servidor corriendo al mismo tiempo en el mismo sentido que standalone. inetd lanzará un proceso Apache por cada conexión entrante, y ese proceso se cerrará una vez que la conexión se complete. No mantiene procesos hijos "en espera" como en el modo standalone.

Systemd como Reemplazo de inetd (Activación por Socket)

Una de las características clave de **systemd** es la **Activación por Socket** (*Socket Activation*), que funciona como un **reemplazo moderno y mejorado** de la funcionalidad que ofrecía `inetd` o `xinetd` (el "super-servidor").

Característica	inetd / xinetd	systemd (Activación por Socket)
Función	Espera en el puerto y lanza el servicio.	systemd espera en el puerto y lanza el servicio.
Implementación	Configurado a través de archivos `/etc/inetd.conf` o `/etc/xinetd.d/*`.	Configurado con dos tipos de unidades: `.socket` (define el puerto) y `.service` (define el programa a ejecutar).
Ventaja	Inicia servicios bajo demanda, ahorrando recursos.	Inicia servicios bajo demanda, pero puede pasar el **socket abierto** al servicio, permitiendo que el servicio se inicie solo cuando se necesita.

- Aunque esta activación por socket **podría usarse para Apache**, no es la práctica habitual ni la recomendada debido a los problemas de rendimiento mencionados anteriormente (iniciar el servidor completo para cada conexión).
- Sin embargo, para **servicios de bajo tráfico** (como servidores TFTP, o ciertas aplicaciones internas), la **Activación por Socket de systemd** es la manera moderna de lograr la funcionalidad "solo-si-es-necesario" que antes se gestionaba con `inetd`.

3. Primeros pasos

Concepto de servidor virtual

Un servidor web virtual (también conocido como Virtual Host en Apache o Server Block en Nginx) es una configuración que permite a un único servidor físico (o una única instancia de un servidor web) alojar múltiples sitios web distintos, cada uno con su propio nombre de dominio, contenido y configuraciones, compartiendo la misma dirección IP y puerto.

La forma en que esto funciona es que el servidor web utiliza el nombre de host solicitado por el cliente (en la cabecera HTTP con el parámetro SNI) para determinar qué sitio web virtual debe servir.

Existen dos tipos principales de servidores virtuales:

1. Virtual Hosts basados en nombres (Name-based Virtual Hosts): Son los más comunes. El servidor web utiliza el nombre de dominio en la solicitud HTTP para dirigir la petición al sitio web correcto. Múltiples dominios pueden apuntar a la misma dirección IP del servidor.
2. Virtual Hosts basados en IP (IP-based Virtual Hosts): Cada sitio web tiene su propia dirección IP dedicada en el servidor. Esto es menos común hoy en día debido a la escasez de direcciones IPv4 y la eficiencia de los hosts basados en nombres.

El concepto de servidor web virtual es fundamental para el alojamiento web, ya que permite a los proveedores de hosting alojar cientos o miles de sitios web en un número limitado de servidores físicos, optimizando recursos y costes.

Apache como servidor web. ¿De qué modos permite Apache añadir funcionalidad al servicio web?

Apache es altamente modular, lo que le permite añadir funcionalidad de diversas maneras:

- Módulos (Modules): Esta es la forma principal. Apache es un servidor extensible basado en módulos. Los módulos son librerías dinámicas (.so en Linux, .dll en Windows) que se cargan en el servidor y proporcionan funcionalidades específicas. Ejemplos incluyen mod_ssl (para HTTPS), mod_rewrite (para reescritura de URLs), mod_php (para procesar PHP), mod_proxy (para proxy inverso), etc.
- Directivas de configuración: Aunque no "añaden" funcionalidad en sí mismas, las directivas permiten configurar y habilitar las funcionalidades proporcionadas por los módulos.
- Archivos .htaccess: Permiten a los usuarios de directorios específicos (sin acceso a la configuración principal del servidor) anular o añadir directivas de configuración para ese directorio y sus subdirectorios. Esto es útil en entornos de alojamiento compartido.
- Scripts CGI/FastCGI/WSGI/AJP: Apache puede ejecutar programas externos (escritos en cualquier lenguaje) a través de interfaces como CGI (Common Gateway Interface), FastCGI, WSGI (para Python) o AJP (para Tomcat/Java). Esto permite que Apache sirva contenido dinámico generado por estos scripts.

¿De qué formas permite Apache el acceso a diferentes sitios web ubicados en el mismo servidor?

Apache permite el acceso a diferentes sitios web ubicados en el mismo servidor principalmente a través de:

- Virtual Hosts (Virtual Hosts basados en nombres): Como se explicó anteriormente, esta es la forma más común. Apache examina el nombre de dominio en la cabecera Host de la solicitud HTTP para servir el sitio web correcto. Cada sitio web virtual tiene su propia raíz de alojamiento de documentos (DocumentRoot), logs, y configuraciones específicas. Se hace a través de un atributo denominado SNI.
- Virtual Hosts basados en IP: Menos común, pero posible. Cada sitio web se asocia a una dirección IP diferente en el servidor.
- Basado en puerto: Diferentes sitios web pueden estar accesibles a través de diferentes puertos en la misma dirección IP (ej. http://www.ejemplo.com:80 y http://www.ejemplo.com:8080). Aunque funcional, no es ideal para el usuario final.
- Subdirectorios/Aliases: Se pueden servir diferentes sitios web o secciones de sitios desde subdirectorios específicos de la raíz de documentos principal (DocumentRoot) o usando la directiva Alias para mapear una URL a una ruta de archivo diferente.

¿Cómo se configurar los diferentes parámetros del servidor Apache para que su comportamiento sea el que deseamos?

Por nuestro conocimiento sobre DNS y DHCP ya sabemos que las configuraciones de estos servicios se hacen a través de ámbitos que se pueden anidar para ir de lo más general a lo más particular. En el caso de Apache ocurre algo similar. ¿Cuáles son los ámbitos que Apache ofrece para poder hacer la configuración de más general a más particular?

1. Configuración del servidor global (httpd.conf): Estas son las directivas globales que afectan a todo el servidor Apache. Se definen en el archivo principal de configuración de Apache (típicamente httpd.conf o a través de archivos incluidos en conf-available y mods-available en Debian). Afectan a todos los Virtual Hosts a menos que sean anuladas.
2. Configuración de Virtual Host (<VirtualHost>): Estas directivas se aplican a un sitio web virtual específico. Anulan las directivas globales para ese Virtual Host. Se definen en bloques <VirtualHost> dentro de los archivos de configuración de los sitios (ej. 000-default.conf en Debian).
3. Configuración de directorio (<Directory>, <Location>, <Files>): Permiten aplicar directivas a directorios específicos, URLs o archivos dentro de un sitio web.
   - <Directory>: Aplica directivas a un directorio del sistema de archivos y sus subdirectorios.
   - <Location>: Aplica directivas a una URL específica (ruta virtual) sin importar dónde esté el recurso en el sistema de archivos.
   - <Files> / <FilesMatch>: Aplica directivas a archivos que coinciden con un nombre o patrón dentro de un directorio. Los nombres de los archivos se pueden especificar, en el primer caso con cadenas de caracteres literales con comodines básicos, en el segundo con expresiones regulares.
4. Archivos .htaccess (Distributed Configuration Files): Son archivos que se colocan en los directorios de contenido del sitio web. Permiten a los usuarios sin acceso al archivo de configuración principal de Apache aplicar directivas específicas a ese directorio y sus subdirectorios. Las directivas en .htaccess anulan las de los ámbitos superiores si AllowOverride lo permite. Son el nivel más particular y granular.

4. Práctica 0

Antes de comenzar con el trabajo:

- La configuración de red debe ser la habitual de las prácticas de DNS y ambos servidores deben estar funcionando
- El servidor DHCP tiene que estar funcionando y otorgar configuración de red, incluido servicio DNS a los dos clientes (LINUXXX y WINDOWXX)
- Dispones de toda la documentación sobre apache en: https://httpd.apache.org/docs/current/
- Puedes hacer las pruebas de conexión en esta práctica desde el propio servidor
- Observa que esta práctica presenta cierto contenido teórico que deberás estudiar en profundidad para poder responder a las preguntas que se planteen en las pruebas.

Instalación del servidor Apache en tu máquina BOOKWORMXXA:

- Utiliza apt-get para ello (apt-get install apache2)
- Asegúrate de que está corriendo utilizando el comando service/systemctl
- Comprueba su funcionamiento tecleando en tu navegador "localhost", que deberá mostrarte una página sobre Apache e información sobre la estructura y contenido de los archivos de configuración
- Explora la estructura de archivos de la configuración de Apache
  - Archivo apache2.conf:
    - Es el archivo de configuración principal desde el que se llama a otros archivos: Include/IncludeOptional
    - En él se configuran algunas directivas, que tras la instalación serán pocas.
    - Gran parte del archivo estará comentado, bien para hacer aclaraciones, bien para anular parte de las directivas apuntadas.
    - Algunas directivas:
      - La directiva que se utiliza para establecer el directorio base para los archivos de configuración y registro de Apache es ServerRoot que se puede establecer del siguiente modo: ServerRoot "/etc/apache2"
      - La directiva que se utiliza para establecer el lugar donde se almacena el número de proceso de Apache es PidFile que se puede establecer del siguiente modo: PidFile "/var/run/apache2/apache2.pid"
      - Directivas que tienen que ver con el comportamiento de Apache en las conexiones que se hacen sobre él para demandar sus servicios y que se verán posteriormente.
  - Archivo ports.conf:
    - Archivo en que se habilitan de forma general los puertos en que Apache escucha.
    - Algunos de los puertos pueden estar condicionados.
  - Dualidad de archivos "available" y "enabled" al respecto de los sitios (sites), módulos (mods) y configuración (conf):
    - Observa en qué se diferencia el archivo que se encuentra en 'sites-enabled' del que archivo con el mismo nombre que se encuentra en 'sites-available'.
    - Observa que el archivo 'default-ssl.conf' solo se encuentra en el directorio 'sites-available'.

- Abre el archivo '000-default.conf' utilizando el editor de texto y observa cómo está estructurado y qué directivas contiene
  - Con base en las directivas que aparecen deduce donde se encuentra el archivo 'index' del sitio.
  - Abre el archivo 'index' del sitio y extrae conclusiones.
- Observa los archivos que contiene el directorio 'conf-available' e intenta relacionar este directorio con el archivo de configuración 'apache2.conf'.
- Observa los archivos que contiene el directorio 'mods-available' y busca alguno cuyo nombre te pueda sonar.
- Intenta buscar registros log de Apache en los archivos que corresponda.

Realiza todas las capturas que consideres relevantes en el proceso.

5. Comportamiento de Apache ante solicitudes de conexión

¿Qué se entiende por conexión persistente?

Una conexión persistente (también conocida como Keep-Alive) en HTTP permite que un cliente (navegador) reutilice la misma conexión TCP con el servidor para múltiples solicitudes HTTP sucesivas, en lugar de establecer una nueva conexión TCP para cada solicitud individual.

Beneficios:

- Reduce la latencia: Elimina la sobrecarga de establecer y cerrar nuevas conexiones TCP para cada solicitud (negociación de tres vías, establecimiento de sesión SSL/TLS, etc.).
- Reduce el uso de CPU/memoria: Menos conexiones que gestionar para el servidor.
- Mejora la velocidad de carga de la página: Especialmente para páginas con muchos recursos (imágenes, CSS, JS) que se cargan desde el mismo dominio.

Cómo habilitarlas en Apache

- Las conexiones persistentes están habilitadas por defecto en Apache.
- Se configuran con la directiva KeepAlive.
- Ejemplo para habilitar/deshabilitar:
  - `KeepAlive On` # Habilita las conexiones persistentes
  - `KeepAlive Off` # Deshabilita las conexiones persistentes

¿Cómo se configura el número máximo de peticiones permitidas durante una conexión persistente?

- Se configura con la directiva MaxKeepAliveRequests.
- Ejemplo: `MaxKeepAliveRequests 100`
- Este valor indica el número máximo de solicitudes que se pueden servir a través de una única conexión persistente antes de que el servidor la cierre.
- Un valor de 0 significa un número ilimitado de solicitudes.

¿Cuánto tiempo ha de estar esperando Apache antes de cerrar una conexión lanzada desde un cliente?

- Se controla mediante la directiva KeepAliveTimeout.
- Ejemplo: `KeepAliveTimeout 5`
- Este valor (en segundos) define cuánto tiempo esperará Apache una nueva solicitud en una conexión persistente inactiva antes de cerrarla. Si no llega ninguna solicitud dentro de este tiempo, la conexión se cierra.
- Si se establece KeepAlive Off, esta directiva no tiene efecto.

¿Por qué es importante limitar el número de conexiones que en cada momento Apache puede soportar?

- Se evita la sobrecarga del servidor: Cada conexión consume recursos (memoria, CPU, descriptores de archivo). Un número ilimitado de conexiones concurrentes podría agotar los recursos del servidor y llevarlo a un estado de inoperatividad o rendimiento extremadamente bajo (denegación de servicio por agotamiento de recursos).
- Estabilidad: Asegura que el servidor pueda manejar la carga esperada sin colapsar.
- Optimización de recursos: Permite asignar los recursos disponibles de manera efectiva.

¿Cómo hacerlo en Apache?

La forma de limitar el número de conexiones y controlar el comportamiento de los procesos hijos de Apache depende del Multi-Processing Module (MPM) que esté utilizando Apache (prefork, worker o event).

- Para MPM Prefork (el más común para PHP con mod_php):
  - MaxClients (Apache 2.2 y anteriores) / MaxRequestWorkers (Apache 2.4 y posteriores): Establece el número máximo de procesos hijos simultáneos que pueden atender peticiones. Este es el límite principal de conexiones concurrentes.
  - StartServers: Número de procesos hijos que se lanzan al inicio.

  - MinSpareServers: Número mínimo de procesos hijos inactivos.
  - MaxSpareServers: Número máximo de procesos hijos inactivos.
- Para MPM Worker/Event (basados en hilos, más eficientes):
  - ThreadsPerChild: Número de hilos por proceso hijo.
  - MaxRequestWorkers: Número máximo total de hilos que pueden atender peticiones concurrentemente (este es el límite de conexiones). Se calcula como ServerLimit * ThreadsPerChild.
  - StartServers: Número de procesos hijos que se lanzan al inicio.
  - MinSpareThreads: Número mínimo de hilos inactivos.
  - MaxSpareThreads: Número máximo de hilos inactivos.
  - ServerLimit: Límite superior para la configuración de MaxRequestWorkers y ThreadsPerChild.
- Ejemplo (MPM Prefork en apache2.conf o mpm_prefork.conf):

```
<IfModule mpm_prefork_module>
    StartServers          5
    MinSpareServers       5
    MaxSpareServers      10
    MaxRequestWorkers   150  # Límite máximo de conexiones concurrentes
    MaxConnectionsPerChild 0
</IfModule>
```

NOTA: La directiva **MaxConnectionsPerChild** (o **MaxRequestsPerChild** en versiones anteriores de Apache HTTP Server) sirve para limitar el número de peticiones (conexiones) que un proceso hijo individual puede manejar antes de que sea terminado y reemplazado por uno nuevo.

6. Práctica 1

Antes de comenzar con el trabajo:

- La configuración de red debe ser la habitual de las prácticas de DNS y ambos servidores deben estar funcionando
- El servidor DHCP tiene que estar funcionando y otorgar configuración de red, incluido servicio DNS a los dos clientes (LINUXXX y WINDOWXX)
- Dispones de toda la documentación sobre apache en: https://httpd.apache.org/docs/current/
- Alterna las pruebas entre tu servidor y alguno de los clientes

Creación de un sitio virtual en el servidor Apache de tu máquina BOOKWORMXXA:

1. Crea un registro alias en el DNS (BOOKWORMXXB) para que 'virtual1.asirXX.asir' se traduzca por 'bookwormXX**A**.asirXX.asir'
2. Crea y configura el sitio virtual. Para ello puedes utilizar el archivo de configuración del sitio por defecto, '000-default.conf' con el nombre 'virtual1.conf', y dejar lo esencial del mismo, del que previamente tienes que haber hecho copia
3. Deberás dejar en el archivo de configuración del sitio (virtual1.conf) en el directorio sites-available:
   1. <virtualhost></virtualhost>
   2. Ip local de tu máquina (10.0....)
   3. Puerto 80
   4. Directiva ServerName con el nombre del sitio "virtual1.asirXX.asir"
   5. Directiva DocumentRoot apuntando a **/var/www/virtual1**
4. Crea el directorio /var/www/virtual1 (que será el DocumentRoot) con los permisos adecuados (fíjate en los que tenga el sitio http) y añade en él un archivo denominado index.html con el texto "Sitio virtual1 de BookwormXXA"
5. Habilita el sitio con la orden "a2ensite virtual1.conf" lo que generará un enlace virtual en el directorio sites-enabled hacia el archivo que debe estar en el directorio sites-available
6. Comprueba el correcto funcionamiento. No olvides hacer los "reload" necesarios según modifiques la configuración para que Apache responda. El "status" debe salir sin error y al teclear en el navegador tanto del servidor como del cliente "http://virtual1.asirXX.asir/" deberá mostrarse "Sitio virtual1 de BookwormXX**A**"

Incorporación de PHP:

1. Modifica el archivo index.html para que se denomine index.php, e inserta en él algún código PHP, por ejemplo un "echo"
2. Prueba si al conectarte al sitio virtual se muestra además del contenido HTML que ya había el nuevo contenido PHP ("echo"). Si no se muestra deberás:
   1. Instalar PHP en tu máquina (apt-get install php o similar) con lo que se instalará el módulo para que Apache pueda servir código interpretado
   2. Si haces la instalación como se indica en el punto 1, el módulo quedará automáticamente habilitado (mods-enabled). Si no es así deberás activarlo con el comando "a2enmod"
3. Realiza las pruebas de conexión desde el cliente

Creación del sitio virtual utilizando SSL:

El contenido del sitio virtual va a ser el mismo que el anterior, es decir, el sitio se va a servir tanto por el puerto 80 como por el puerto 443, por tanto el 'DocumentRoot' será el mismo.

1. Para que el servidor escuche el el puerto 443 (ver ports.conf) es necesario tener activado el módulo SSL. Si no lo has hecho ya lo tendrás que hacer usando el comando "a2enmod"
2. Genera un certificado utilizando openssl para el sitio "virtual1.asirXX.asir. Para ello utiliza el siguiente código y no olvides poner en el nombre común el nombre exacto del sitio (openssl req -x509 -sha256 -newkey rsa:2048 -keyout certificate.key -out certificate.crt -days 1024 -nodes). Debes poner el certificado y su clave en los lugares correspondientes con los permisos adecuados. Fíjate para ello en las claves pública y privada del "snakeoil", que es el sitio SSL por defecto. El CN del certificado ha de ser 'virtual1.asirXX.asir'.
3. Haz una copia del sitio 'default-ssl.conf' y denomínalo 'virtual1SSL.conf'. Modifica:
    1. DocumentRoot
    2. ServerName
    3. Clave pública
    4. Clave privada
    5. IP desde la que se puede acceder que podrá ser cualquiera que tenga ahora o en el futuro tu máquina
4. Realiza las pruebas de conexión desde el cliente LINUXXX y WINDOWXX. Quizá tengas que aceptar alguna excepción en configuración de seguridad por ser el certificado autofirmado.

Realiza todas las capturas que consideres relevantes en el proceso.

7. Práctica 2

Antes de comenzar con el trabajo:

- Es conveniente haber realizado previamente el resto de prácticas
- Dispones de toda la documentación sobre apache en: http://httpd.apache.org/docs/current/
- Esta práctica tiene una base teórica que se explica en la propia práctica y que será evaluada en los correspondientes controles teóricos

Recapitulación

- Como hemos visto, un servidor Web puede albergar diferentes sitios web utilizando diferentes nombres de hosts (alias) o diferentes IP's.
- Parámetros de configuración vistos hasta ahora:
    - <VirtualHost> </VirtualHost>: se utilizan para agrupar un conjunto de directivas que se aplicarán solo a un host virtual. Dentro de este par de directivas se pueden incluir cualquier otra que haga referencia a este host particular
    - Port: Define el puerto en el cual escucha el servidor (0 - 65535). Hay que tener en cuenta la relación que tiene esta directiva con el fichero /etc/services y que algunos puertos, especialmente los situados por debajo del 1024, están reservados para protocolos específicos. El puerto estándar para el protocolo HTTP es el 80
    - ServerName: Especifica el nombre de host que aparecerá en la cabecera de la petición:
        - Beneficio: Definir un número de servidores ilimitado, de fácil configuración y que no requiere hardware adicional
        - Desventaja: El software cliente debe soportar esta parte del protocolo (salvo excepción, todos lo hacen)
        - (*) Si no se define el nombre de servidor, intentará deducirlo a través de su dirección IP
    - ServerAdmin: Define la dirección de correo que el servidor incluirá en cualquier mensaje de error que devuelva al cliente
    - DocumentRoot: Define el directorio desde el cual el servidor servirá los documentos. (A menos que la URL solicitada coincida con una directiva Alias, el servidor añadirá el PATH a la URL)
    - IPs de escucha del servidor, en caso de que se tenga más de una.
    - En el caso de la configuración SSL:
        - Puerto específico
        - Motor SSL activado
        - Ruta de certificado
        - Ruta de clave privada

Utilización dela directiva 'Directory':

- La directiva (<Directory></Directory>) se utiliza para encerrar un grupo de directivas que se aplicarán al directorio en cuestión y sus sub-directorios (salvo que en ellos se indique lo contrario)
- ¿Qué ocurrirá si en nuestro sitio virtual actual modificamos el DocumentRoot y lo sustituimos por /var/virtual1?
    1. Creamos la carpeta /var/virtual1
    2. Creamos un fichero index.html en dicha carpeta en que ponemos el texto: "Este es el fichero /var/virtual1c/index.html" (para poderlo reconocer)
    3. Modificamos el DocumentRoot de nuestro sitio virtual y ponemos /var/virtual1
- Si lo hemos hecho de la forma indicada nos dará un forbidden (prohibido) ¿Por qué nos da dicho 'forbidden'?

1. En el archivo de configuración las directivas Directory establecen opciones concretas para directorios concretos, entre ellas que en / está el acceso denegado, mientras que para /var/www sí tenemos permisos ¿qué ocurre con /var?
2. ¿Qué ocurría hasta ahora?
3. Podríamos fijar en la configuración del sitio virtual los permisos básicos que aparecen, para el sitio concreto e ir añadiendo permisos y otras opciones con diferentes bloques “Directory”, teniendo en cuenta que se aplica lo más atómico frente a lo más general.

- Deshacemos el cambio que hemos hecho en el DocumentRoot de nuestro sitio y lo devolvemos a su valor (/var/www/virtual1)
- Realizamos lo siguiente:
  1. Creamos una carpeta con “/var/www/virtual1/restringido”
  2. Creamos en dicha carpeta un conjunto de 3 archivos con nombres restringido1.html, restringido2.html y restringido3.html (vacíos)
  3. Creamos en nuestro index.html original un enlace a dicha carpeta
  4. Observamos el resultado que obtenemos al pulsar el enlace creado y que nos debería llevar a la carpeta (1).
  5. Modificamos el archivo de configuración del sitio virtual para añadir un bloque “Directory” para la carpeta creada y ponemos la directiva “Require all denied”.
  6. Observamos el resultado que obtenemos al pulsar el mismo enlace anterior(2).
- Recuerda comprobar la sintaxis cada vez que hagas una modificación así como recargar Apache cada vez que sea necesario. Para ello se puede hacer uso de 'apachectl':
  - 'apachectl' es una interfaz para el servidor de protocolo de transferencia de hipertexto (HTTP) Apache. Está diseñado para ayudar al administrador a controlar el funcionamiento del demonio httpd de Apache.
  - Entre otras funciones permite hacer un test de configuración usando opción 'configtest'.

Recuerda hacer todas las capturas que consideres necesarias

8. Práctica 3

Antes de comenzar con el trabajo:

- Es conveniente haber realizado previamente el resto de prácticas
- Dispones de toda la documentación sobre apache en: http://httpd.apache.org/docs/current/
- Esta práctica tiene una base teórica que se explica en la propia práctica y que será evaluada en los correspondientes controles teóricos

Opción 'Indexes'

- Apache usa por defecto un valor de archivo de acceso a los sitios denominada 'index' (index.html, index.php, etc.)
- Si no existe ningún archivo con nombre 'index', pero se tiene habilitado la opción `Option Indexes` Apache devolverá un listado de ficheros y directorios correspondientes a la ruta.
- Se puede añadir o quitar a la lista de opciones por defecto, nuevas opciones usando los signos ‘+’ y ‘-’, en el caso que nos ocupa sería:
  - `Options +Indexes`
  - `Options -Indexes`
- Podemos observar ejemplos de uso en los bloques Directory de los sitios por defecto de Apache.
- Podemos especificar en cada bloque `Directory` que queramos declarar las opciones específicas que nos interesen.

Puesta en práctica:

1. Creamos una carpeta '/var/www/virtual1/permitido'.
2. Creamos en dicha carpeta un conjunto de 3 archivos con nombres 'permitido1.html', 'prmitido2.html' y 'permitido3.html' (vacíos).
3. Creamos en nuestro index.html original un enlace a dicha carpeta.
4. Observamos el resultado que obtenemos al pulsar el enlace creado y que nos debería llevar a la carpeta (1)
5. Modificamos el archivo de configuración del sitio virtual para añadir un bloque 'Directory' para la carpeta creada y ponemos una línea con '`Options -Indexes`' (es decir, estamos quitando indexes que venía heredado de una configuración de directorio más general anterior).
6. Observamos el resultado que obtenemos al pulsar el mismo enlace anterior(2).
7. (*) Recuerda comprobar la sintaxis cada vez que hagas una modificación así como recargar Apache cada vez que sea necesario.

Directiva 'DirectoryIndex'

- Busquemos en el catálogo de directivas de la documentación de Apache2
  - ¿Qué hace la directiva?
  - ¿Dónde la podemos aplicar?
  - ¿Cómo se usa?
  - ¿Cómo podremos usarla para que cuando accedamos al directorio 'permitido' de nuestro sitio nos aparezca el fichero 'permitido.html'?

- En resumen:
  - Permite especificar una ruta a un archivo concreto diferente del usado por defecto por apache (index.html).
  - Si ponemos 'DirectoryIndex index.html index.php' Apache buscará primero un archivo index.html y en caso de no estar, buscará index.php. Si no se encuentra ninguno de ellos se mostrará un error salvo que tengamos activa la opción 'Indexes' en cuyo caso se mostrará el listado de archivos del directorio.

Puesta en práctica:

1. Creamos el archivo /var/www/virtual1/permitido/permitido.html
2. Como contenido del archivo ponemos el texto: 'Este es el archivo /var/www/virtual1/permitido/permitido.html'
3. En el sitio virtual, en la configuración del directorio correspondiente, quitamos la opción '`options -Indexes`' y añadimos la directiva '`DirectoryIndex permitido.html`'
4. ¿Qué ocurre cuando accedemos al directorio permitido ahora?

Recuerda hacer todas las capturas que consideres necesarias

9. Práctica 4

Antes de comenzar con el trabajo:

- Es conveniente haber realizado previamente el resto de prácticas
- Dispones de toda la documentación sobre apache en: http://httpd.apache.org/docs/current/
- Esta práctica tiene una base teórica que se explica en la propia práctica y que será evaluada en los correspondientes controles teóricos

Uso de la directiva 'Alias':

- Permite publicar ficheros de una carpeta en lugar distinto de `DocumetRoot` y acceder a los mismos ficheros a través de dos rutas distintas:
  - Con la directiva "`Alias /lugar2 /var/www/virtual1`" estaríamos accediendo a los archivos de virtual1 al poner en el navegador www.virtual1.asir/lugar2
  - Para poder acceder a ficheros fuera del `DocumentRoot` (imaginemos que queremos acceder a /var/fueradeDocumentRoot) tendremos que añadir a la configuración del sitio virtual la directiva "`Alias /fueradeDocumentRoot /var/fueradeDocumentRoot`", así como una declaración específica de Directory para /var/fueradeDocumentRoot
- También se podría utilizar para publicar el mismo sitio con nombres distintos
  - La directiva: "`Alias otrovirtual1.asirX.asir`" añadida a nuestro sitio virtual 'virtual1.conf' permite que una conexión al servidor sobre "otrovirtual1.asirX.asir" sirva el mismo sitio que virtual1.
  - Para que todo funcione correctamente será necesario añadir un registro CNAME a nuestra zona DNS que traduzca "otrovirtual1" por "virtual1" para que el cliente obtenga, en primer lugar, la traducción correcta del nombre a la IP a la que necesita conectarse.

Aplicación de la directiva 'Alias':

1. En la configuración del sitio "virtual1" añade la directiva de configuración para que /pruebaalias sea un alias de /permitido.
2. Añade un enlace en el index del sitio hacia "http://virtual1.asirX.asir/pruebaalias" como los otros que hemos creado.
3. Prueba que al pulsar sobre el enlace te lleva a la zona "permitida" del sitio.

Uso de la directiva 'FollowSymLinks':

- Permite activar el seguir los enlaces simbólicos (*).
- Hay que tener cuidado con los enlaces simbólicos por aspectos relacionados con la seguridad.
- Valorar la opción de usar alias en lugar de activar los enlaces simbólicos.

(*) Por ejemplo, enlaces simbólicos son los que se crean en sites-enabled. Los enlaces no simbólicos son enlaces "duros" y a todos los efectos para Apache serían como archivos.

Uso de la directiva 'MultiViews':

- Permite mostrar diferentes páginas en función del navegador que se use, por ejemplo permite mostrar una página en un idioma u otro en función de lo que se recibe en la cabecera de la solicitud.
- Podríamos tener una página "index" diferente para cada idioma diferente
- En el caso de español e inglés tendríamos que tener dos archivos: "index.html.es" e "index.html.en" (observar las dos letras que indican el país)
- Si el navegador está configurado con un idioma para el que no hay una página específica se mostraría lo que estuviera configurado en el módulo de negociación "negotation.conf" donde se especifica la prioridad de idiomas.

Uso de la directiva 'Redirect' para hacer redirecciones:

- Permite que Apache devuelva información al navegador indicándole que la conexión para obtener la información se obtiene en una URL distinta de la original.
- Su sintaxis es: "`Redirect url_origen url_destino`"
- Supone realizar una nueva conexión y petición al servidor de la URL de destino.
- Por ejemplo nos puede servir para:
  - Hacer que la url "…/correo" se redirija a https://mail.google.com con "Redirect /correo https://mail.google.com".
  - Puede servir para simplemente redirigir del sitio no seguro al sitio seguro por ejemplo: "Redirect / https://virtual1.asirX.asir".
  - Podemos redirigir a nuestro propio sitio pero a otro lugar. Sin embargo en este caso es preferible usar "alias" para evitar una nueva conexión.

Aplicación de la directiva 'Redirect'

1. Añade la directiva "`Redirect / https://virtual1.asirXX.asir`" al sitio "vitual1" (al sitio no seguro)
2. ¿Qué ocurre al intentar acceder al sitio https://virtual1.asirXX.asir"?
3. Una vez hecha la comprobación comenta la directiva para recuperar el funcionamiento del punto anterior sin redirección.

¿Cómo evitar los típicos mensajes de errores que aparecen en Apache como el 404 o el 403?

- Podemos hacer uso de los errores personalizados.
- Podemos cambiar el texto por otro, añadiendo al archivo de configuración del sitio virtual las directivas:
  - `ErrorDocument 404 "Página no encontrada"`
  - `ErrorDocument 403 "No tienes permisos para esta página"`
- Podemos personalizar, en su lugar, la página: Para ello hay que crear páginas de error personalizadas, por ejemplo "not_found.html" y "forbidden.html" en un directorio específicamente creado para ello para el que habremos creado un bloque "Directory" en el sitio virtual y que podemos denominar /var/www/virtual1/errors:
  - `ErrorDocument 404 /errors/not_found.html`
  - `ErrorDocument 403 /errors/forbidden.html`

Aplicación del uso de los errores personalizados:

1. Crea un directorio específico para almacenar los archivos de errores personalizados del sitio.
2. Crea un archivo específico para el error 403 que ponga "No tienes permisos para visitar esta zona del sitio" (forbidden.html).
3. Prueba a acceder al sitio restringido.

Recuerda hacer todas las capturas que consideres necesarias

10. Práctica 5

Antes de comenzar con el trabajo:

- Es conveniente haber realizado previamente el resto de prácticas
- Dispones de toda la documentación sobre apache en: http://httpd.apache.org/docs/current/
- Esta práctica tiene una base teórica que se explica en la propia práctica y que será evaluada en los correspondientes controles teóricos

¿Cómo se valora quién puede conectarse al sitio o acceder a determinados recursos del mismo?

- Tradicionalmente se han usado la directivas "allow" y "deny" para permitir y denegar el acceso en función de múltiples circunstancias sobre directorios concretos del sitio web.
- Junto con la directiva 'Order' que indicaba el orden en que deberían aplicarse los permisos anteriores configuraban un escenario que permitía arbitrar diversas configuraciones.
- Estas directivas pertenecen al módulo 'mod_access_compat', pero están obsoletas y van a ser retiradas en futuras versiones por lo que es recomendable evitar su uso.

¿Cómo se usan Order, Allow y Deny? (Por si nos encontramos con algún sitio que lo mantenga configurado de esta manera)

- Order:
  - allow,deny: Primero aplicará los permisos de allow y luego los de deny. Es decir, las directivas 'allow' son evaluadas, al menos una debe coincidir o la petición será rechazada. Después se evalúan todas las directivas 'deny' y con que una se cumpla la petición será rechazada.
  - deny,allow: Primero aplicará los permisos de deny y luego los de allow. Es decir todas las directivas 'deny' son evaluadas y si alguna se cumple la petición es rechazada salvo que se cumplan las directivas 'allow'. Cualquier petición que no cumpla ninguna de 'allow' o 'deny' será permitida.
- Allow

  - from all: Admite cualquier acceso al directorio
  - from IP: Admite cualquier acceso al directorio proveniente de la dirección IP indicada
  - from <dominio>: Admite cualquier acceso al directorio desde el dominio especificado
- Deny
  - from all: Deniega cualquier acceso al directorio
  - from IP: Deniega cualquier acceso al directorio proveniente de la dirección IP indicada
  - from dominio: Deniega cualquier acceso al directorio desde el dominio especificado.

Si hemos entendido bien lo anterior deberíamos saber qué hacen las siguientes directivas:

- allow from 10.0.128+X.250 127.0.0.1
- O lo que ocurriría si configuramos las directivas:
  - Order deny,allow
  - Deny from 10.0.128+X.250
- Podríamos de esta forma configurar las directivas por ejemplo:
  - Para permitir todos los accesos desde tu propio servidor solamente
  - Para permitir los accesos solamente desde tu máquina cliente.

¿Cuál es la recomendación actual y la que vamos a aplicar nosotros?

La recomendación actual es utilizar al menos una autenticación básica (mod_auth_basic)

- La protección básica se basa en el uso de la directiva “AuthUserFile” que usamos para especificar un fichero de claves, que será de texto con los usuarios y sus contraseñas: AuthUserFile “/etc/apache2/usuarios.txt”
- Se añade además con la directiva “AuthName” un texto para invitar al acceso: AuthName “Identificación”, por ejemplo
- La directiva AuthType Basic permite establecer el tipo de autenticación y la directiva Require valid-user configurarla
- El fichero lo creamos con el comando htpasswd:
  - La primera vez podría ser: htpasswd –c usuarios.txt usuario1, y se nos pediría la contraseña para el “usuario1” dos veces, que se almacena con un hash
  - Si el fichero ya existe no es necesario añadir “-c”, por ejemplo htpasswd usuarios.txt usuario2
  - Podemos editar el fichero para verlo o para borrar usuarios
- Podemos usarla para todo el sitio o para un directorio concreto al que solamente queramos que se acceda con usuario y contraseña

De este modo también es posible la autenticación básica basada en grupos

- Será necesario tener un fichero de texto con los grupos en que aparecerá una línea por grupo con el nombre seguido de “:” y de los usuarios que pertenecen a cada grupo separados por espacios. Por ejemplo /etc/apache2/groups.txt
- Será necesario tener activado el módulo “authz_groupfile” (obligado restart)
- Modificamos el sitio virtual para añadir:
  - La directiva: AuthGroupFile “/etc/apache2/groups.txt”
  - Sustituimos la directiva de 'Require' de usuario por la de grupos, por ejemplo: Require group grupo1 grupo2, lo que permitirá que aquellos usuarios que pertezcan a los grupos 1 o 2 tendrán acceso pero no otros, aunque estén en el fichero de usuarios

Aplicación de la autorización básica de Apache:

Configura tu sitio 'virtual1.asirXX.asir' para que solamente se pueda acceder al directorio del sitio ./usuarios previa introducción de un usuario y una contraseña. Para ello:

1. Crea el directorio correspondiente
2. Añade un fichero html en dicho directorio en que ponga 'Este es la zona de usuarios de virtual1.asirXX.asir'
3. Añade un enlace a dicho directorio en la página principal del sitio
4. Crea un fichero denominado '/etc/apache2/usuariosbasic.txt' con el usuario 'alumno' y la contraseña '232425'.
5. Prueba el funcionamiento

Un segundo método de autenticación es 'Digest0 (mod_auth_digest):

- Es el segundo método nativo de autenticación de Apache
- Es necesario activar el módulo de apache “auth_digest”
- El fichero que se crea es único en vez de los dos del caso “basic”
- Se crea con el comando “htdigest –c usuarios_d.txt grupo1 usuario1”, que permite crear el “grupo1” y añadirle el “usuario1” (“-c” solo para la creación del fichero, después no hay que ponerlo). Podemos añadir diferentes usuarios y asignarlos a diferentes grupos con la repetición del comando con distintos usuarios y grupos
- Configuración de Apache
- Habría que añadir en la configuración del sitio virtual las directivas:
  - AuthType Digest (en lugar de basic)
  - AuthName “grupo1” para permitir acceso de los usuarios pertenecientes al grupo1
  - AuthUserFile “/etc/apache2/usuarios_d,txt” ...archivo de usuarios y claves Digest
  - Require valid-user

Recuerda hacer todas las capturas que consideres necesarias

11. Ejemplos de sitios virtuales

En el siguiente enlace (https://httpd.apache.org/docs/2.4/vhosts/examples.html)podrás encontrar ejemplos sobre:

- Ejecutar varios sitios web basados en nombres en una sola dirección IP
- Hosts basados en nombres en más de una dirección IP
- Servir el mismo contenido en diferentes direcciones IP (como una dirección interna y externa)
- Ejecutar diferentes sitios en diferentes puertos
- Alojamiento virtual basado en IP
- Hosts virtuales mixtos basados en puertos y en IP
- Vhosts mixtos basados en nombres y en IP

12. Práctica 6

Antes de comenzar con el trabajo:

- Es conveniente haber realizado previamente el resto de prácticas
- Dispones de toda la documentación sobre apache en: http://httpd.apache.org/docs/current/
- La práctica es autocontenida, es decir, todo lo nuevo que es necesario está explicado en la propia práctica pero tendrás que usar aspectos que ya hemos visto en otras prácticas
- En el caso de php y su conexión con bases de datos se utlizará 'mysqli' cuya documentación podéis consultar en: https://www.php.net/manual/es/class.mysqli.php

La práctica tiene como objetivo familiarizarnos con la conexión de Apache con bases de datos de MySQL. En los siguientes puntos se detallan los pasos que será necesario dar:

1. Instalación del Servidor de Bases de datos y prueba
   - Se va a usar el servidor MySQL (MariaDB), por lo que necesitaremos instalarlo en nuestro servidor (apt-get install mariadb-server)
   - Comprobar que podemos acceder al servidor como usuario root del servidor de bases de datos (mariadb -u root -p)
   - Crea una base de datos en el servidor que contenga una tabla con atributos 'idusuario, nombreusuario, urlusuario', que se corresponderá con un número de usuario entero, un nombre de usuario, por ejemplo 'alumno' y una URL, por ejemplo 'https://virtual1.asirXX.asir/%usuario%'. Es necesario poblar la base de datos con algunos registros, entre ellos el usuario creado en la práctica anterior para la autenticación 'Basic'.
   - Para que Apache pueda acceder al servidor de bases de datos y a la base de datos vamos a crear un usuario específico para tal fin (por control y seguridad), el usuario se denominará 'apache' y la contraseña será '123456'. Además, a la vez que lo creamos le concederemos permisos explícitos de lectura (y solo lectura) sobre la base de datos de usuarios que acabamos de crear. Si la base de datos se denominase 'BD_USUARIOS': 'GRANT SELECT ON BD_USUARIOS.* TO apache@localhost IDENTIFIED BY '123456';
   - Para comprobar la corrección de lo realizado nos conectaremos con el usuario creado (mariadb -u apache -p) y verificaremos que podemos seleccionar la base de datos y que podemos hacer operaciones de lectura (SELECT) pero no de escritura (DELETE, UPDATE, INSERT...)
2. Instalación de la extensión PHP para MySQL
   - Es necesario instalar la extensión que permite que apache se pueda conectar a bases de datos de MySQL si no está ya instalada. En caso de no estarlo ejecutaremos 'apt-get install php-mysql'
3. Para comprobar que Apache se puede conectar a la base de datos y recuperar información para mostrarla al usuario tendremos que dar varios pasos. Lo recomendable es ir dando los pasos de uno en uno y chequeando el correcto funcionamiento. Para ello vamos a introducir en el archivo 'permitido.html' (que se convertirá en 'permitido.php') un conjunto de instrucciones que nos permitan comprobar que podemos conectarnos con el servidor de bases de datos, que nos podemos conectar con la base de datos en cuestión, que podemos extraer datos de dicha base de datos y por último, que podemos mostrar al usuario los registros recuperados.
   1. En primer lugar podemos definir en el archivo permitido.php las variables PHP que nos interesan (se usa de modo genérico y como ejemplo la base de datos 'BD_USUARIOS', la tabla 'TABLA1' y los atributos 'atributo1' y 'atributo2', de modo que solamente tendréis que cambiarlos por los vuestros)
      - $host= 'localhost';
      - $usuario= 'apache';
      - $password= '123456';
      - $bd= 'BD_USUARIOS';

- $consulta= 'SELECT atributo1, atributo2 FROM TABLA1';

2. Conexión con el servidor: Se lanza la conexión y se comprueba con un 'if' si la conexión ha sido o no establecida y en caso de no serlo que nos muestre el error concreto que se haya producido. Aunque en un entorno de producción no se haría de este modo porque hay otros modos de hacerlo, en una práctica como ésta, de aprendizaje, nos permite hacer las pruebas sin conocimientos adicionales:
   - $conexion= mysqli_connect($host,$usuario,$password);
   - if(!$conexion) {echo'
Conexión fallida con el servidor: '.mysqli_connect_error().'
</br>';}
   - else {echo '
Conexión establecida con el servidor.
</br>';}
3. Conexión con la base de datos: Se lanza la conexión y se comprueba con un 'if 'si la conexión ha sido o no establecida y en caso de no serlo que nos muestre el error concreto que se haya producido.:
   - $conexionBD= mysqli_select_db($conexion,$bd);
   - if(!$conexionBD) {echo'
Conexión fallida a la base de datos.
</br>';}
   - else {echo '
Conexión establecida con la base de datos.
</br>';}
4. Consulta sobre la base de datos: Se lanza la consulta y se comprueba con un 'if' si ha tenido éxito o no y en caso de no tenerlo que nos muestre el error concreto que se haya producido.:
   - $resultado= mysqli_query($conexion,$consulta);
   - if(!$resultado) {echo'
No se ha podido obtener la consulta a la base de datos: '.mysqli_sqlstate().'
</br>';}
   - else {echo '
Consulta ejecutada correctamente.
</br>';}
5. Para mostrar en el navegador los resultados de la consulta se puede hacer en una tabla. Para ello podemos hacer:
   - echo '
La consulta anterior devolvió el siguiente número de registros: '.mysqli_num_rows($resultado).'
</br>';
   - while ($fila=mysqli_fetch_array($resultado,MYSQLI_NUM)){printf("Atributo1: %d Atributo2: %s",$fila[0],$fila[1]);}
   - echo '</br>';
6. Liberamos el resultado de la consulta con: 'mysql_free_results()'; (por ejemplo).
7. Cerramos la conexión con: mysqli_close($conexion);

4. Lo ideal es ir anidando los if-else, de modo que si se produce un error de conexión ya no siga ejecutándose la consulta y demás, es decir, sería algo como:
   - if
   - else
     - if
     - else
       - if
       - else

5. Disponéis de un documento PDF en que se pueden ver algunos resultados que podríais obtener incluyendo algunos errores

Recuerda hacer todas las capturas que consideres necesarias