UT2 - ADMINISTRACIÓN DE SERVIDORES WEB

Despliegue de Aplicaciones Web

2° DAW

Introducción

HTTP (Hyper Text Transfer Protocol)

- Es un protocolo de la capa de **aplicación** que facilita a los usuarios de forma sencilla e intuitiva el acceso a la información remota conectándose a una red TCP/IP.

WWW (World Wide Web)

- El modelo cliente/servidor y el protocolo http son la base de la WWW o simplemente web.
- WWW es un servicio de distribución de información que permite acceder a millones de recursos electrónicos y aplicaciones distribuidos en servidores por todo internet y localizados por direcciones (URIs o URLs). Los recursos se conectan a través de hiperenlaces/hipervínculos/links lo que permite navegar de uno a otro fácilmente.
- La www fue desarrollada por el CERN en 1989 y actualmente su desarrollo está controlado por W3C (World Wide Web Consortium) una comunidad que desarrolla estándares web como XHTML, CSS y XML.

Funcionamiento del servicio HTTP

- Componentes: el servicio que ofrece la Web se basa en el modelo cliente/servidor y está formado por los componentes:
  - Recursos
  - Nombres y direcciones
  - Clientes web (clientes HTTP o navegadores)
  - Servidores web (o servidores HTTP)
  - Proxies web (o proxies HTTP)
  - Protocolo HTTP
  - Tecnologías web (XHTML, CSS, XML, Ajax, XPath, etc.)

Funcionamiento del servicio HTTP

Recursos

- Documentos, vídeos imágenes, audio aplicaciones, buzones de correo, etc., accesibles a través de servidores web y conectados por hiperenlaces.
- Distinguimos 2 tipos:
  - Páginas web:
    - **Documento hipermedia** o conjunto de información relacionada (texto, audio, imágenes, …) que contiene links a otras web o recursos.
  - Sitio web:
    - **Conjunto de páginas web** relacionadas y accesibles a partir de un mismo nombre de dominio DNS. Todos los sitios web de internet constituyen la WWW.
    - Normalmente ese conjunto está almacenado en un directorio específico del servidor web.
    - En ese directorio se suele establecer una jerarquía de subdirectorios para organizar las distintas páginas web y el resto de elementos que lo componen.

Funcionamiento del servicio HTTP

Nombres y direcciones

- Sistema de nombres basado en cadenas de caracteres que identifican y localizan inequívocamente a los recursos en la Web.
- URL: Universal Resource Locator.
  - Cadena de texto que se utiliza para identificar un recurso y además nos da información sobre como acceder a él, como localizarlo.
- Formato URL:

Parte de la URL	Descripción	Ejemplo
Servicio:	Indica el servicio o protocolo a usar: http, https, ftp, telnet, …	http:
//	Separador	//
Servidor	Indica la IP o el nombre del servidor	www.ubuntu.com
Ruta	Indica el directorio o subdirectorios donde reside en recurso.	/desktop
Recurso	Recurso al que se quiere acceder.	/index.html

Funcionamiento del servicio HTTP

Clientes Web (navegadores)

- Programas con los que interactúa el usuario y que permiten, entre otras, introducir URLs para acceder a recursos de la red.
- Pueden actuar como clientes de diferentes protocolos pero su función principal es la de ejercer de clientes http.
- Mantienen una memoria caché para almacenar: historial, contraseñas, etc.
- Permiten opciones múltiples de configuración y personalización.
- Ejemplos: Mozilla Firefox, Google Chrome, Internet Explorer, Microsoft Edge, Safari, Opera, …

Funcionamiento del servicio HTTP

Servidores web o servidores http

- Son programas que atienden peticiones HTTP, procesan e interpretan código escrito en diferentes lenguajes y envían a los clientes los recursos solicitados.
- Estos recursos pueden estar en el propio equipo o en otros.
- Pueden enviar contenido estático (archivos en diferentes formatos) como dinámico (el resultado de ejecutar programas).
- Por defecto, escuchan las peticiones HTTP en el puerto **TCP 80**
- Algunos servidores web:
  - **Libres**: Apache HTTP server, nginx (engine X)
  - **Propietarios**: IIS (Internet Information Server) de MS, LiteSppeed

NGINX

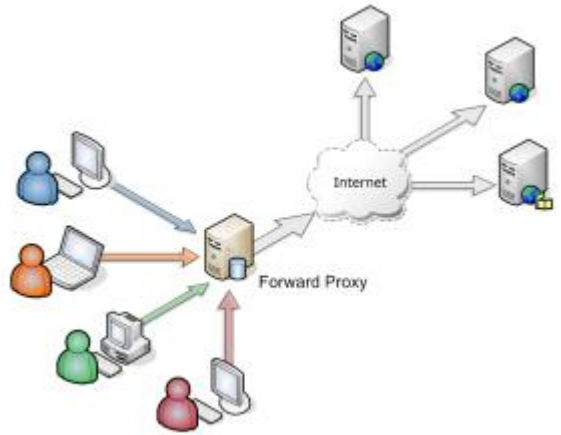
Microsoft IIS

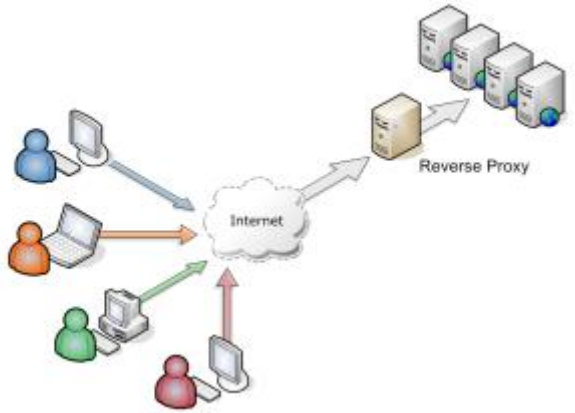
LITESPEED

Funcionamiento del servicio HTTP

Proxies web

- En redes: Proxy = Intermediario.
- Proxy web = Intermediario entre servidor y cliente HTTP
- Proxy directo (forward proxy):
  - Recibe la petición de un cliente web y la traslada al servidor.
  - La petición del cliente es **hacia el servidor**, no al proxy.
  - Usado para optimizar y controla accesos a Internet de los clientes de una empresa. Habitualmente el propio usuario no conoce de su existencia.
- Proxy inverso (reverse proxy):
  - Reciben la petición de un cliente y la reenvían al servidor.
  - La petición del cliente ahora es **hacia el proxy** (para los clientes es un servidor web).
  - Los clientes usan la URL del proxy.
  - Se usan para proporcionar acceso a servidores web que están detrás de cortafuegos y no son accesibles directamente.
  - Objetivo: balancear carga, aumentar la seguridad en los accesos, etc.





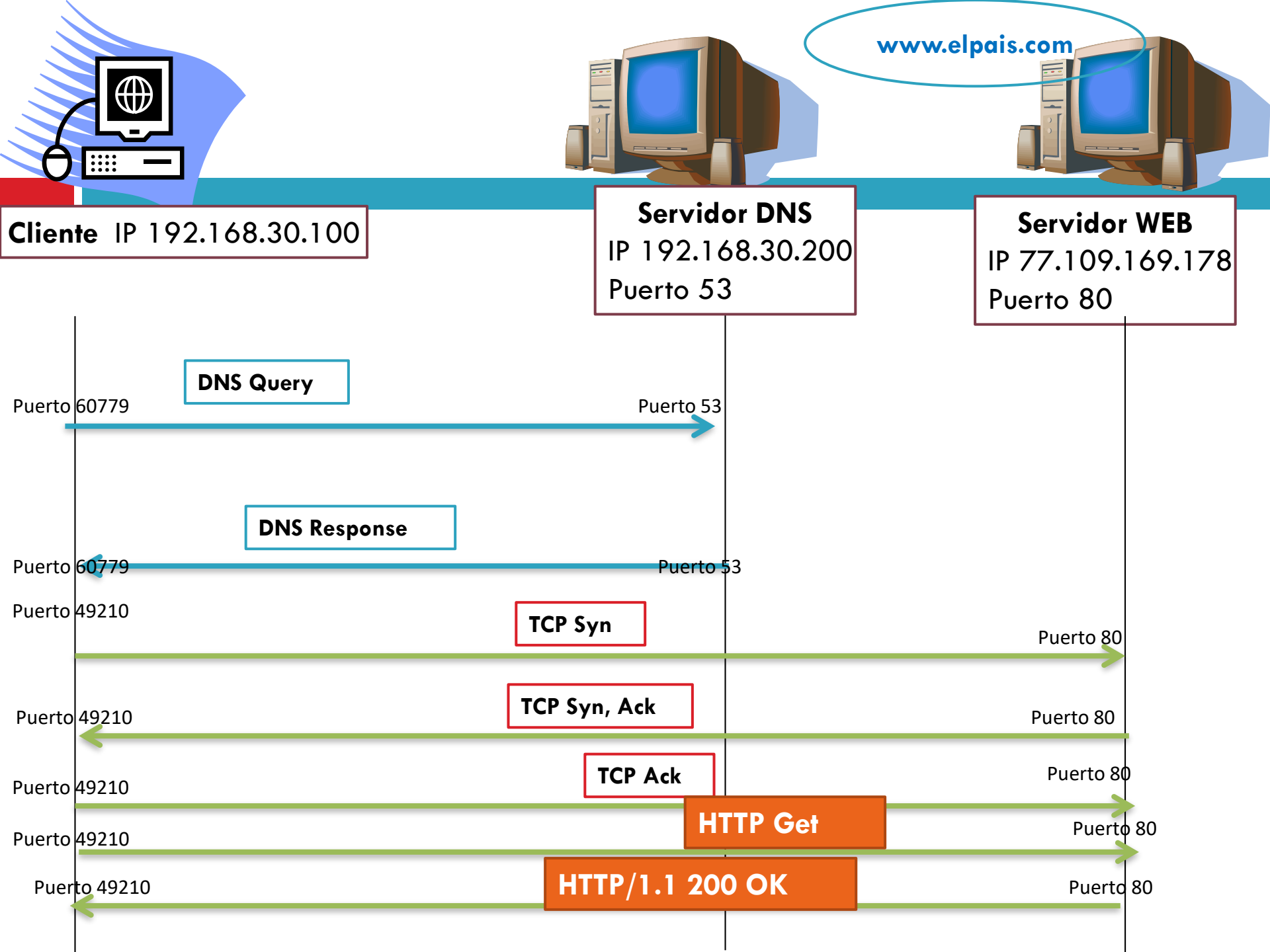
Funcionamiento del servicio HTTP

Protocolo HTTP

- Define las **reglas** que utilizan los componentes software (clientes, servidores y proxies) para comunicarse.
- Determina **los tipos de peticiones** que los clientes pueden enviar, así como el formato y la estructura de las **respuestas.**
- También define una estructura de metadatos, en forma de **cabeceras** que se envían tanto en las peticiones como en las respuestas.
- Versiones:
  - http/0.9 (obsoleta)
  - http/1.0
  - http/1.1 (versión actual)
  - http/1.2 (experimental)

Funcionamiento del servicio HTTP

1) El usuario introduce una URL en la barra de direcciones del navegador (o clic sobre link).
2) El navegador descompone la URL en:
    1) Protocolo de acceso.
    2) El nombre DNS o dirección IP del servidor.
    3) El puerto, si es que viene especificado.
    4) El objeto requerido.
3) Si se ha especificado un nombre DNS, se buscará la IP asociada.
4) Establece una conexión TCP con el servidor Web (puerto 80 por defecto).
5) El navegador envía el mensaje **HTTP de petición**.
6) El servidor envía el mensaje **HTTP de respuesta** en función del estado del servidor y de la petición enviada.
7) Se cierra la conexión TCP.


www.elpais.com
Cliente IP 192.168.30.100
Servidor DNS
IP 192.168.30.200
Puerto 53
Servidor WEB
IP 77.109.169.178
Puerto 80
DNS Query
Puerto 60779
Puerto 53
DNS Response
Puerto 60779
Puerto 53
Puerto 49210
TCP Syn
Puerto 80
Puerto 49210
TCP Syn, Ack
Puerto 80
Puerto 49210
TCP Ack
Puerto 80
HTTP Get
Puerto 49210
Puerto 80
Puerto 49210
HTTP/1.1 200 OK
Puerto 80

Funcionamiento del servicio HTTP

Mensajes HTTP: Peticiones

- Los **métodos de petición** especifican la operación que quiere realizar el cliente en el servidor. La versión http 1.1 contempla siete métodos de petición que son:
  - Método GET.
  - Método POST.
  - Método OPTIONS.
  - Método HEAD.
  - Método PUT.
  - Método DELETE.
  - Método TRACE.
- **Ejercicio**: Para los métodos GET y POST averigua los siguientes datos:
  - Para qué se emplean.
  - Cuándo se invocan.
  - Ejemplo en el que se utilizaría este mensaje.

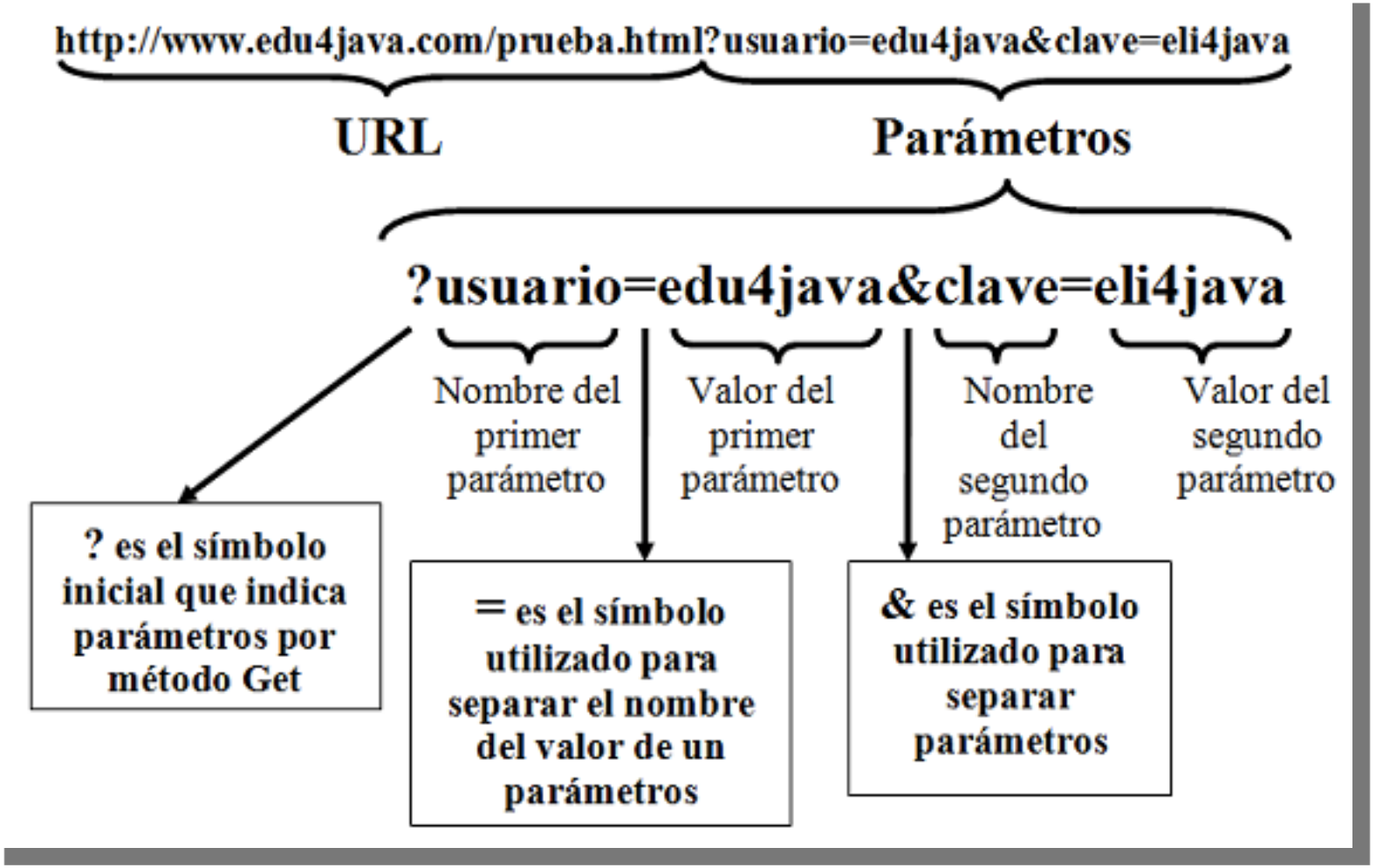
Funcionamiento del servicio HTTP

GET

- El método más utilizado y se usa para obtener información del servidor.
- Se invoca:
  - Al escribir una URL en el navegador.
  - Al pinchar un link.
  - Cuando se envía un formulario con GET.
- Permitirá enviar parámetros al servidor en la URL.
- Se agrega al final de la URL una "?" y seguido pares de "nombre" y "valor" separados por &.
- Las peticiones GET no envían cuerpo del mensaje.
- El tamaño de la información enviada estará limitada.
- No se puede usar para subir archivos o realizar otras operaciones que requieran enviar una gran cantidad de datos al servidor.

Funcionamiento del servicio HTTP

GET



Funcionamiento del servicio HTTP

POST

- Se emplea para solicitar al servidor que acepte información que se envía adjunta en una petición.
- Las peticiones ahora están en el **cuerpo del mensaje,** y en él se incluyen los parámetros y los datos que se deseen.
- Ahora los parámetros ya **no son visibles en la URL**.
- Se invoca normalmente para enviar un **formulario** HTML **POST**.
- No hay límite en la cantidad de datos a enviar, se usa para enviar archivos al servidor.
- Las respuestas a un POST no se cachean por defecto.

Tipos MIME

- Multipurpose Internet Mail Extensions
- HTTP soporta la negociación de contenidos entre cliente y servidor
- Estándar que especifica como deben transferirse en internet todo tipo de recursos (archivos).
- En principio se desarrolló para el protocolo SMTP (correo).
- Hay una organización que mantiene las listas de formatos. Se puede ver la última versión oficial en: https://www.iana.org/assignments/media-types/media-types.xhtml
- El tipo MIME de un documento lo tiene que enviar el servidor. Una vez enviado, el navegador lo recibe y lo utiliza para interpretar el documento (muchas veces sin tener en cuenta la extensión del archivo).
- Por lo tanto deberemos configurar nuestros servidores para que envíen adecuadamente los tipos MIME.
- Su sintaxis es muy sencilla. Dos cadenas de caracteres: “tipo/subtipo”. Algunos ejemplos son: text/html, text/csv, image/jpeg, image/bmp, video/mp4, etc.

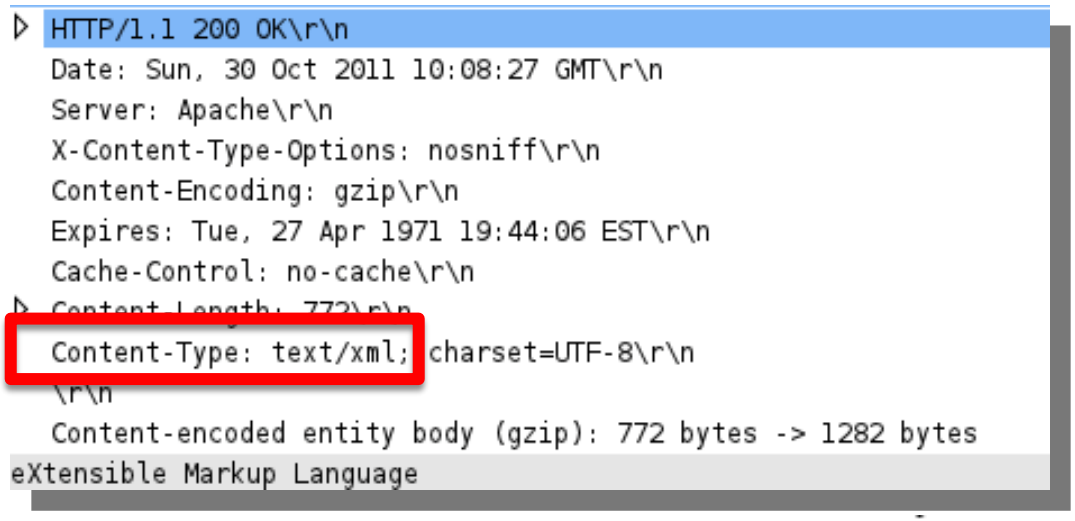
Tipos MIME

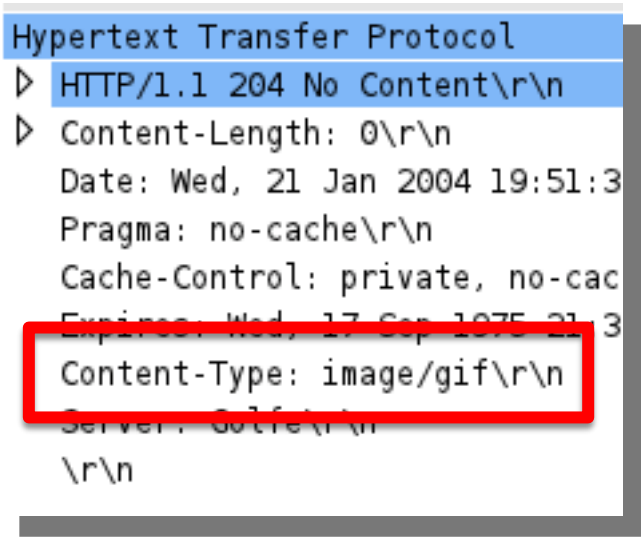
TIPO	DESCRIPCIÓN	SUBTIPOS
text	Los subtipos contendrán texto.	**text/plain, text/html, text/css,** text/javascript, text/csv
audio	Archivos que contienen audio en diferentes formatos.	audio/mpeg o audio/wav.
image	Incluye las imágenes y las imágenes animadas (como los gif). No incluye los vídeos.	image/gif, image/jpeg, image/png
video	Identifica a los distintos archivos de vídeo.	video/mp4, video/ogg
application	Para datos binarios.	application/pdf, application/xml

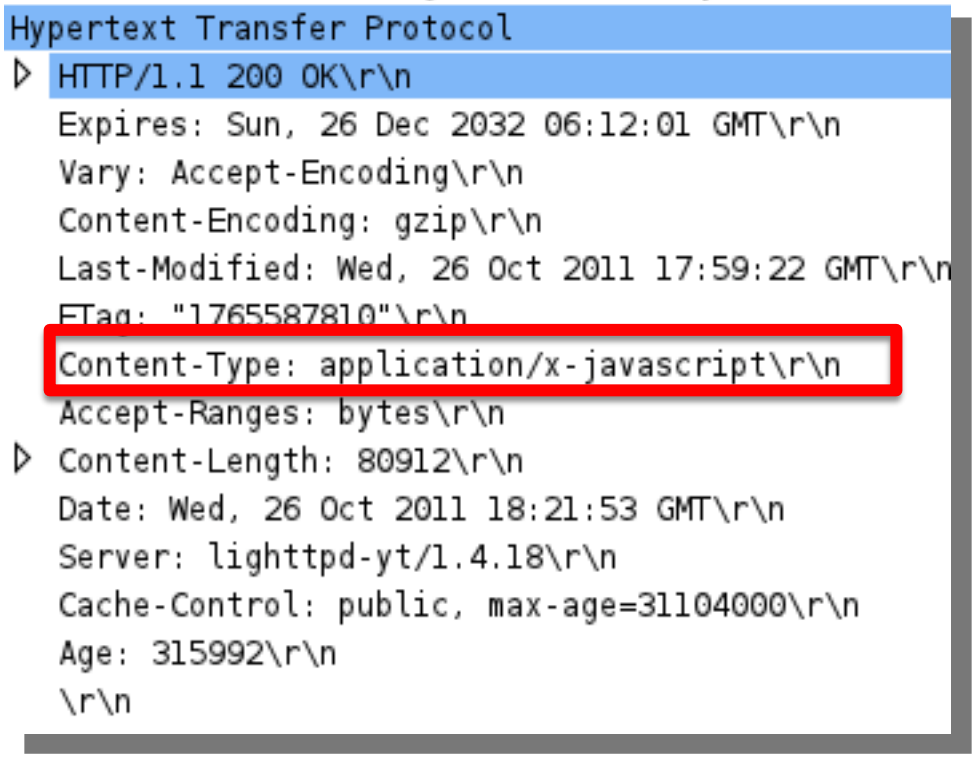
Tipos MIME

- En HTTP se utilizan las cabeceras y tipos MIME en:
  - Las **respuestas** del servidor para informar al cliente de los recursos que envía. El navegador en función del tipo MIME que recibe visualiza el recurso con una aplicación u otra.
  - Los **mensajes que envían los clientes** para informar al servidor de los tipos MIME que aceptan.
- Los desarrolladores de páginas web debemos especificar el tipo MIME para que los servidores lo envíen y los navegadores puedan mostrar correctamente la página o el recurso a mostrar. Por ejemplo:

```
<link rel="stylesheet" href="mi_hoja_de_estilo.css" type="text/css">
<script language="JavaScript" type="text/javascript"
   src="scripts/mijavascript.js">
<meta http-equiv="Content-Type" content="text/html">;
```


HTTP/1.1 200 OK\r\n
Date: Sun, 30 Oct 2011 10:08:27 GMT\r\n
Server: Apache\r\n
X-Content-Type-Options: nosniff\r\n
Content-Encoding: gzip\r\n
Expires: Tue, 27 Apr 1971 19:44:06 EST\r\n
Cache-Control: no-cache\r\n
Content-Type: text/xml; charset=UTF-8\r\n
\r\n
Content-encoded entity body (gzip): 772 bytes -> 1282 bytes
eXtensible Markup Language


Hypertext Transfer Protocol
HTTP/1.1 204 No Content\r\n
Content-Length: 0\r\n
Date: Wed, 21 Jan 2004 19:51:3
Pragma: no-cache\r\n
Cache-Control: private, no-cac
Content-Type: image/gif\r\n
\r\n


Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
Expires: Sun, 26 Dec 2032 06:12:01 GMT\r\n
Vary: Accept-Encoding\r\n
Content-Encoding: gzip\r\n
Last-Modified: Wed, 26 Oct 2011 17:59:22 GMT\r\n
Content-Type: application/x-javascript\r\n
Accept-Ranges: bytes\r\n
Content-Length: 80912\r\n
Date: Wed, 26 Oct 2011 18:21:53 GMT\r\n
Server: lighttpd-yt/1.4.18\r\n
Cache-Control: public, max-age=31104000\r\n
Age: 315992\r\n
\r\n

Tipos MIME

- En un servidor web podemos especificar el tipo MIME por defecto para aquellos archivos que el servidor no pueda identificar automáticamente como pertenecientes a un tipo concreto, esto es, para aquellos los cuales no se resuelven según su extensión.
- Para el servidor web Apache se utilizan dos directivas: **AddType** y **ForceType**.
  - AddType asigna las extensiones de nombre de archivo proporcionadas al tipo de contenido especificado.
  - ForceType hace que todos los ficheros cuyos nombres tengan una equivalencia con lo que se especifique sean servidos como contenido del tipo MIME que se establezca.
- Ejemplos:
  - *AddType application/msword doc*
  - *AddType application/msexcel xls*
  - *ForceType image/gif*: desplegará todos los ficheros como un archivo de imagen gif.
  - *ForceType video/mp4*: desplegará todos los ficheros como un archivo de vídeo mp4.
- En el servidor web Apache existe el archivo **/etc/apache2/mods-available/mime.conf** donde encontrarás una referencia al archivo **/etc/mime.types,** el cual contiene la lista de tipos MIME reconocidos por el servidor.

Módulos en Apache

- Hemos visto la funcionalidad básica de Apache HTTP Server.
- Las herramientas básicas que vienen incluidas en el paquete de instalación es lo que se llama el "core" de Apache.
- Los módulos en Apache incrementan sus funcionalidades permitiendo que su utilidad sea mucho mayor que la que se permite en el paquete básico.
- Los nombres de estos módulos suelen comenzar con mod_ y una palabra clave que da información sobre la utilidad del módulo.
- Para ver la lista de módulos disponibles en Apache podemos entrar aquí:
  - https://httpd.apache.org/docs/2.4/en/mod
- Por otro lado, si queremos ver los módulos compilados en nuestro servidor de Apache se puede hacer con la orden "apache2ctl –l"
- Si queremos ver los cargados se puede hacer con "apache2ctl –M"

Módulos en Apache

- También podemos ver los módulos disponibles en **/etc/apache2/mods-available**
- Y los módulos habilitados en **/etc/apache2/mods-enabled**
- Si queremos habilitar un módulo (de los que están en disponibles) ejecutaremos **a2enmod modulo**
  - Después deberemos reiniciar Apache: sudo **/etc/init.d/apache2 restart**
- Para desactivar un módulo habilitado ejecutaremos **a2dismod modulo**
  - Después deberemos reiniciar Apache

- En caso de experimentar cualquier tipo de problema, el primer lugar dónde consultar qué está ocurriendo son los logs de error de Apache 2, en el archivo /var/log/apache2/error.log.
  - cat /var/log/apache2/error.log

Directivas de secciones

Directivas de secciones

- <Directory /ruta> …</Directory>
  - Engloba una o más directivas de configuración que sólo se aplican al directorio y subdirectorios especificados. Dentro, en el lugar de los … se usan las directivas siguientes:
    - **DirectoryIndex:** archivo índice del directorio. Lo habitual es que haya una página que se cargue automáticamente.
    - **Options**: controla que características estarán disponibles para un directorio particular
      - **Indexes**: muestra el contenido del directorio si no encuentra un archivo índice.
      - **FollowSymLinks**: permite seguir enlaces simbólicos dentro del directorio.
      - **Multiviews**: sirve para no tener que especificar en la URL la extensión de un archivo situado en este directorio. Por ejemplo: si un servidor tiene en el directorio /var/www/misitio/web.htm cuando un cliente escribe en la URL http://www.loquesea.con/misitio/web, si la opción multiviews está puesta para el directorio /var/www/misitio/ buscará algún archivo llamado web.* y mostrará el que encuentre. También serviría si pusiéramos DirectoryIndex indice y no especificáramos extensión.
      - Si ponemos **Options All** activará todas las opciones disponibles salvo Multiviews que hay que especificarla.

Host virtuales

- Configurando hosts virtuales podemos alojar varios sitios web (varios dominios) en el mismo servidor (en la misma máquina), siendo uno totalmente independiente del otro.
- Se pueden ver los dominios vecinos con alguna herramienta como https://dnslytics.com/reverse-ip
- El sitio por defecto que viene configurado (000-default) en apache es un host virtual más.
- Existen varios tipos de host virtuales:
  - Basados en **nombre**: permite alojar varios nombres de host o varios dominios en una misma máquina. Todos los hosts que compartan la misma IP deben declararse mediante la directiva NameVirtualHost.
  - Basados en **puerto**: se responde de diferente manera si se accede a un puerto u otro.
  - Basados en **IP**: una máquina tiene varias direcciones IP y a cada una de ellas servirá un sitio web diferente.

Host virtuales

- Para la definición de un host virtual se utilizará la sección <VirtualHost>.
- Las directivas que se aplican serán:
  - **DocumentRoot**: directorio donde apache2 va a buscar los documentos del servidor virtual.
  - **NameVirtualHost**: define la dirección IP y el puerto donde van a existir host virtuales basados en nombre. (**/etc/apache2/ports.conf**)
  - **ServerName**: define el nombre del servidor virtual.

Host virtuales

Los pasos para crear un host virtual:

1) Crear una nueva carpeta (normalmente dentro de /var/www/), y copiar allí dentro las páginas web.
2) Crear un nuevo archivo de configuración en el directorio sites-available (crearemos uno por sitio). Lo más cómodo es copiar el que viene por defecto.

   cp /etc/apache2/sites-available/000-default /etc/apache2/sitesavailable/nombresitio
3) Configurar en el nuevo fichero de configuración (nombresitio) las opciones necesarias. Lo más importante que DocumentRoot apunte al directorio creado.
4) Habrá que añadir o modificar directivas adicionales en función de si estamos sirviendo sitios virtuales por IP, por nombre o por puerto (ver las próximas diapositivas para los detalles).
5) Activar este nuevo sitio con el comando:

   a2ensite nombresitio
6) Si luego lo queremos desactivar:

   a2dissite nombresitio

Host virtuales basados en nombre

- Se trata de servir desde la misma máquina, en la misma IP (y a través del mismo puerto) dos sitios, por ejemplo: web1.com y web2.com
- Para ello, tras seguir los pasos de la diapositiva anterior debemos indicar en el archivo de configuración correspondiente (por ejemplo /etc/apache2/sites-available/web1.conf) lo siguiente:
  - Que el sitio se va a proporcionar a través de todas las IPs y del puerto 80. Si no está ya añadido, lo añadiremos en /etc/apache2/ports.conf:
    - NameVirtualHost *:80
  - Una directiva <VirtualHost *:80> </VirtualHost> para el sitio que queremos publicar.
  - Indicar dentro de la directiva <VirtualHost *:80> el nombre con el que la web será accesible, y el directorio donde los archivos (html por ejemplo) están almacenados, así como cualquier otra opción que necesitemos.

```
<VirtualHost *:80>
    ServerName  www.web1.com
    DocumentRoot /var/www/web1
    <Directory /var/www/web1>
        Options FollowSymLinks AllowOverride None
    </Directory>
</VirtualHost>
```

- No debemos olvidar añadir las entradas correspondientes en el DNS o en /etc/hosts

Host virtuales basados en puerto

- Ahora serviremos dos sitios distintos en la misma IP pero será uno u otro dependiendo el puerto al que se conecte el usuario.
- Esto lo especificaremos en la directiva <VirtualHost *:puerto>.
- Si por ejemplo quiero publicar web1.com por el puerto 80 y web2.com por el puerto 8080:

```
<VirtualHost *:80>
        ServerName www.web1.com
        DocumentRoot /var/www/web1
</VirtualHost>
<VirtualHost *:8080>
        ServerName www.web2.com
        DocumentRoot /var/www/web2
</VirtualHost>
```

- Además hay que añadir el puerto 8080 en ports.conf: Listen 8080

Host virtuales basados en IP

- Requerirá que el servidor disponga de dos direcciones IP.
- Si por ejemplo, las IPs son 192.168.1.1 y 192.168.1.2, el archivo de configuración deberá contener las directivas VirtualHost como sigue:

```
<VirtualHost 192.168.1.1>
        ServerName www.web1.com
        DocumentRoot /var/www/web1
</VirtualHost>

<VirtualHost 192.168.1.2>
        ServerName www.web2.com
        DocumentRoot /var/www/web2
</VirtualHost>
```

Control de acceso

- Control de acceso" vs "Autenticación y autorización".
  - **Control de acceso** se refiere al proceso y los medios para gestionar el acceso a los diferentes recursos del servidor.
  - **Autenticación** es el proceso por el cual verificamos que un usuario es quien dice ser.
  - **Autorización** es cualquier proceso por el cual a una persona (una vez autenticada) se le permite acceder al recurso que requiere.
  - Delgada es la línea que separa "Control de acceso" de "Autorización". Son conceptos a menudo confundidos. Se podría decir que Autorización es un concepto centrado en el usuario y en los permisos de acceso que puede tener mientras el Control de acceso es un término más general. Para implementar una política de Control de acceso debemos tener en cuenta cosas como las IPs de los clientes, la hora del día en la que se pretende acceder, el navegador usado, etc.

Control de acceso por host

- Las reglas de control de acceso se van a basar en la información proveniente de la cabecera HTTP, o incluso de las cabeceras TCP/IP que la encapsulan.
- La directiva que más se utiliza a la hora de establecer las reglas de control de acceso es **Require**. Esta directiva establece un filtro con restricciones y/o normas que afectan al acceso.
  - Por ejemplo: "Require all granted": en este caso el filtro sería "todos los accesos" y la norma sería "permitido".
  - Otro ejemplo sería "Require valid-user". en este caso establece que solo accedería un usuario validado
- En versiones anteriores de Apache se permitían directivas como Allow, Deny u Order que han sido retiradas del uso en las versiones actuales. Siguen funcionando pero en versiones posteriores desaparecerán.

Control de acceso por host

- Hay dos construcciones típicas de directivas con Require:
  - **Require host address:** por ejemplo, "Require host elpais.com" solo permitirá el acceso a peticiones proveniente de dicho dominio.
  - **Require ip ip_address:** por ejemplo, "Require ip 10.234.12.233" establece que desde esa IP se puede acceder al recurso.
  - También se pueden usar ambas construcciones en forma negada: "Require not ip 10.234.12.233" bloquea el acceso desde esa IP.
- Algunos ejemplos de uso son:
  - Require all granted / Require all denied
  - Require not host example.com
  - Require not host gov (Se puede poner una parte del host)
  - Require not ip 192.168 (también se puede poner parte de una IP)

Control de acceso por host

- Otras directivas que afectan al acceso son <RequireAll>, <RequireAny> y <RequireNone>.
  - **<RequireAll>** agrupa una serie de directivas de las que no puede fallar ninguna para que el acceso se permita.
  - **<RequireAny>** agrupa una serie de directivas de las que al menos una debe cumplirse para que el acceso se permita.
  - **<RequireNone>** agrupa una serie de directivas de las cuales ninguna debe cumplirse para que se permita el acceso.
- Estas directivas permiten hacer combinaciones complejas de control de acceso.

Control de acceso por variables arbitrarias

- También se puede controlar el acceso mediante <If> o mediante "Require expr".
- Estas directivas sirven para establecer reglas de control de acceso en base a los valores de variables de entorno o a los datos contenidos en la cabecera HTTP.
- <If> permite controlar el flujo del archivo de configuración en base a ciertas variables. Por ejemplo:

```
<If "%{HTTP_USER_AGENT} =~ /Android/">
        Require all denied
</If>
```

  - Esta parte de la configuración hará que se rechacen las conexiones provenientes de un dispositivo Android.
- Esto mismo se puede conseguir con:
  - Require expr "%{HTTP_USER_AGENT} =~ /Android/"
- Si queremos negar el acceso:
  - Require expr "%{HTTP_USER_AGENT} !~ /Android/"

Control de acceso por variables arbitrarias

- La documentación de Apache recoge múltiples expresiones con las que se puede controlar el flujo y que pueden ser utilizadas en una estructura “Require expr”. Todas ellas podrían utilizarse para controlar el acceso.
- Se pueden encontrar en:
  - https://httpd.apache.org/docs/trunk/es/expr.html#examples
- Algunos ejemplos útiles serían:
  - <If "%{HTTP_HOST} == 'ejemplo.com'">
  - <If "%{QUERY_STRING} =~ /forcetext/">
  - Require expr %{TIME_HOUR} -gt 9 && %{TIME_HOUR} -lt 17
  - <If "%{HTTP:X-example-header} in { 'foo', 'bar', 'baz' }">
  - <If "md5('foo') == 'acbd18db4cc2f85cedef654fccc4a4d8'">

Autenticación

- La autenticación en Apache se puede llevar a cabo a través de dos módulos diferentes, **Basic** y **Digest**. Estos módulos implementan respectivamente las normas de autenticación de HTTP de mismo nombre.
- La **autenticación Basic de HTTP** es un mecanismo de consulta-respuesta con el que un servidor puede pedir credenciales (usuario y contraseña) a un cliente. El cliente devolverá dichos datos en la "Authentication header". Estará codificada en base-64. Este método solo debe usarse si se considera que la comunicación entre servidor y cliente es segura, es decir, no hay riesgo de que el paquete sea interceptado. Dada la falta de seguridad de la codificación que usa, se podría romper dicha codificación y descubrir las credenciales de acceso.
- La **autenticación Digest de HTTP** es otra forma de intercambiar credenciales entre cliente y servidor. En este caso no se envía la clave directamente. El servidor envía un "nonce value" (valor aleatorio que se usa una sola vez) al cliente. El cliente responderá con un "checksum" (valor calculado con una determinada función) con datos de entrada como el nombre de usuario, la contraseña, el valor nonce, el método HTTP y la URI. Este checksum será por defecto MD5.
- Digest se considera un método más fiable que Basic y, en teoría estaba llamado a sustituirlo.

Autenticación

- AuthName: Frase que aparecerá en la pantalla de autenticación.
  - Ejemplo: AuthName "Este directorio es privado introduce tus datos. "
- AuthType: Basic o Digest
  - (*) Para usar la autenticación básica se podría requerir activar el módulo correspondiente. Para ello: sudo a2enmod auth_basic
- AuthUserFile: archivo que contiene user/password a usar para autenticación
  - Ejemplo: AuthUserFile /etc/apache2/passwd
- Require user: selecciona qué usuarios van a tener acceso al recurso.
  - Ejemplos: Require user juan
  - Require group daw
  - Require valid-user

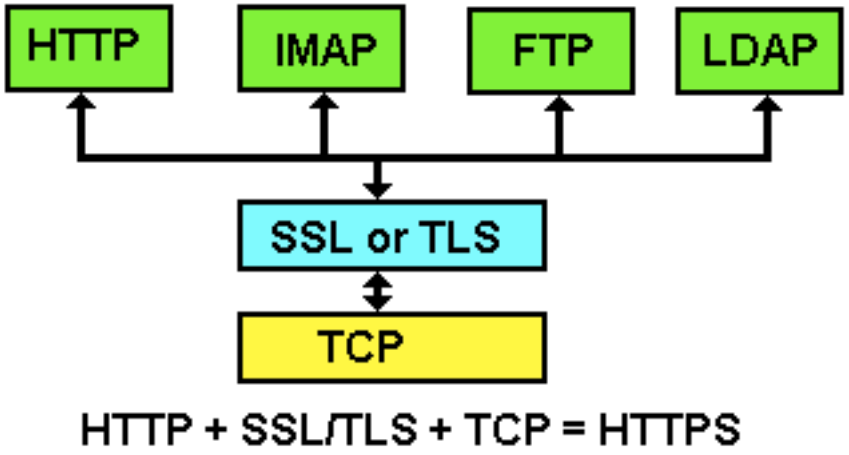
```
<Directory /var/www/profesor>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        AuthType Basic
        AuthName "Acceso restringido"
        AuthUserFile /etc/apache2/passwd
        Require user profesor1 profesor2
</Directory>
```

- Para poder usar la autenticación de usuarios hay que tener un fichero de usuarios y contraseñas. El nombre y la ruta ha de coincidir con la de AuthUserFile
  - sudo htpasswd –c /etc/apache2/passwd usuario1 (crea el archivo y añade usuario1)
  - sudo htpasswd /etc/apache2/passwd juan (añade el usuario juan)
  - sudo htpasswd –D /etc/apache2/passwd juan (borra el usuario juan)

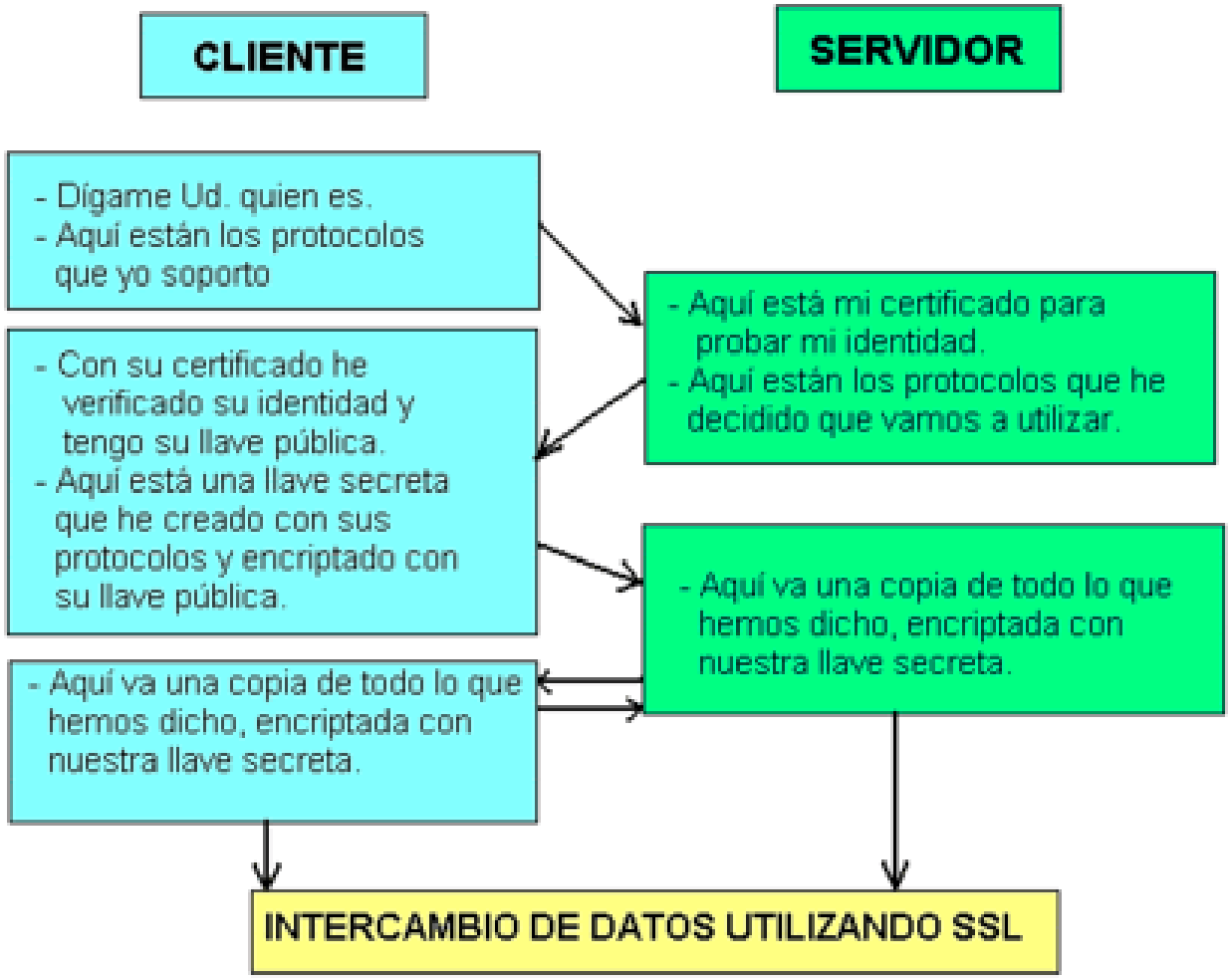
SSL/TLS

Protocolo SSL: Introducción

- SSL: Secure Sockets Layer
- Protocolo creado en 1992 por Netscape para el intercambio de información de manera segura entre cliente y servidor.
- La IETF (Grupo de Trabajo de Ingeniería de Internet) creó su propia versión a la que llamó TLS.
- Por eso vemos que se hace referencia a SSL/TLS.
- SSL es un protocolo que opera entre las capas de aplicación y de transporte.
- Puede recibir datos de varias aplicaciones: http, ftp, ldap, ...
- Opera sobre TCP en el nivel de transporte.



SSL


CLIENTE
SERVIDOR
- Dígame Ud. quien es.
- Aquí están los protocolos que yo soporto
- Aquí está mi certificado para probar mi identidad.
- Aquí están los protocolos que he decidido que vamos a utilizar.
- Con su certificado he verificado su identidad y tengo su llave pública.
- Aquí está una llave secreta que he creado con sus protocolos y encriptado con su llave pública.
- Aquí va una copia de todo lo que hemos dicho, encriptada con nuestra llave secreta.
- Aquí va una copia de todo lo que hemos dicho, encriptada con nuestra llave secreta.
INTERCAMBIO DE DATOS UTILIZANDO SSL

HTTPS

- Con http la información viaja por la red en **texto plano**.
- Pero en ocasiones cliente y servidor puede requerir intercambiar información confidencial.
- Para paliar este problema surge **https**.
- https se apoya en una conexión establecida en SSL/TLS.
- La información http viajará encapsulada en el protocolo seguro **SSL**.
- El puerto para este tipo de conexiones será el **443** en lugar del 80.
- El cliente accederá a este servicio usando **https** en la URL en vez de http.

Certificados autofirmados

- Cuando accedemos a un sitio web con https, necesitamos conocer previamente que el servidor es realmente quien dice. Para demostrarlo deberá poseer un certificado que debe estar firmado por una CA (autoridad de certificación) que sea de nuestra confianza.
- La obtención de un certificado acreditado por una CA para nuestra web suele conllevar gastos.

- **Alternativa**: utilizar certificados autofirmados, que se pueden generar con herramientas software como **OpenSSL**.

Configurar SSL en Apache (I)

1. Lo primero será **activar el módulo ssl** correspondiente:
   - sudo a2enmod ssl
   - service apache2 restart
2. Nos aseguramos de que en **/etc/apache2/ports.conf** se encuentra la directiva **Listen 443.**
3. A continuación crearemos un nuevo directorio donde se almacenará la clave pública del servidor y el certificado autofirmado que posteriormente generaremos.
   - mkdir /etc/apache2/ssl
4. Ahora **creamos la clave** (apache.key) **y el certificado** (apache2.crt) que en este caso tendrán una validez de 1 año (365 días).
   - sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache2.key -out /etc/apache2/ssl/apache2.crt
5. Rellenaremos los campos del formulario que se inicia.
6. Ahora debemos **configurar el archivo de configuración del sitio seguro.** (Podremos crear un archivo de configuración con seguridad y otro sin seguridad y activar uno u otro según las necesidades).

Configurar SSL en Apache (II)

7. Debemos hacer que el sitio se sirva a través del puerto https, es decir el 443: <VirtualHost *:443>
   - <VirtualHost *:443>
8. Añadiremos las líneas siguientes y nos aseguramos de que los directorios y los nombres de clave y certificado coinciden con los que hemos creado en los pasos anteriores:
   - SSLEngine on
   - SSLCertificateFile /etc/apache2/ssl/apache2.crt
   - SSLCertificateKeyFile /etc/apache2/ssl/apache2.key
9. Activaremos el sitio seguro que estemos creando si no está ya añadido:
   - a2ensite sitio-ssl
10. Y reiniciamos el servicio de apache:
    - service apache2 restart
11. Para probar que funciona nos conectaremos a https://www.dominio.com y comprobamos que se descarga el certificado que hemos creado.