DESARROLLO DE APLICACIONES WEB

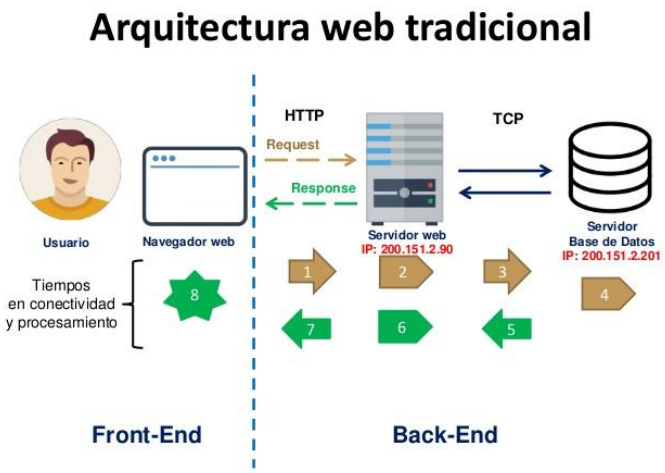
Alex Domínguez Domínguez

Listado de Ejercicios 3
Ejercicio 1. 3
Ejercicio 2. 3
Ejercicio 3. 4
Ejercicio 4. 5
Ejercicio 5. 7
Ejercicio 6. 8
Ejercicio 7. 9
Ejercicio 8. 10
Ejercicio 9. 10
Ejercicio 10. 10

Listado de Ejercicios

Ejercicio 1.

- Arquitectura básica de un servidor web
  - Un servidor Web es un programa que sirve datos en forma de páginas Web, hipertextos o páginas HTML (HyperText Markup Language): textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de sonidos.
  - La comunicación de estos datos entre cliente y servidor se hace por medio un protocolo*, concretamente del protocolo HTTP.
  - Con esto, un servidor Web se mantiene a la espera de peticiones HTTP, que son ejecutadas por un cliente HTTP; lo que solemos conocer como un navegador Web.



  - A modo de ejemplo: al teclear http://www.cnice.mec.es en un navegador, éste realizará una petición HTTP al servidor que tiene asociada dicha URL**. El servidor responde al cliente enviando el código HTML de la página; el navegador cuando recibe el código, lo interpreta y lo muestra en pantalla.
  - El cliente es el encargado de interpretar el código HTML, es decir, de mostrar las fuentes, los colores y la disposición de los textos y objetos de la página. El servidor se encarga de transferir el código de la página sin llevar a cabo ninguna interpretación de la misma.

Ejercicio 2.

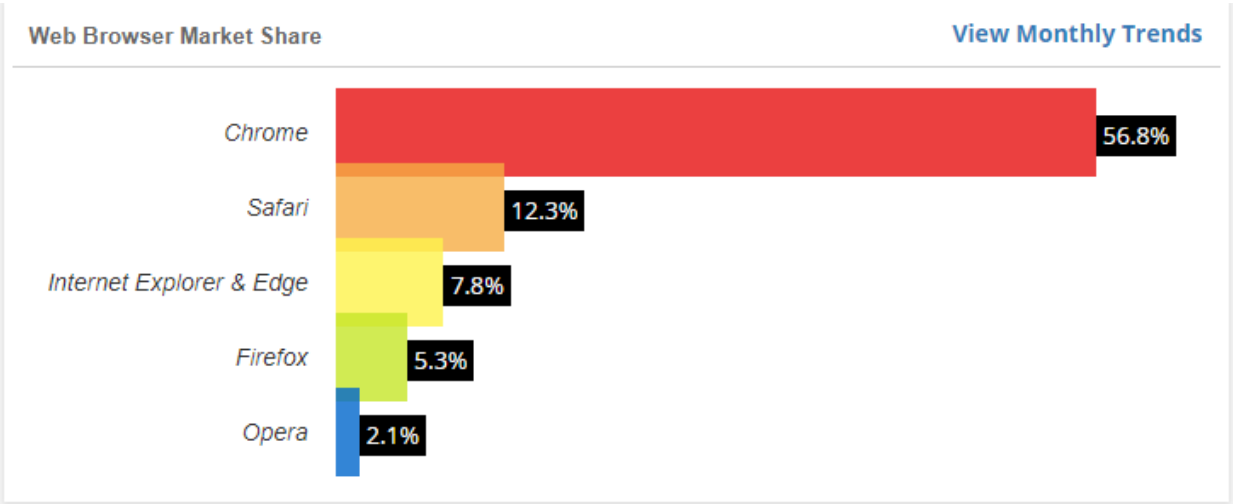
- URL: definición, sintaxis (estructura) y diferentes ejemplos.
  - Localizador de recursos único. Identifica un recurso en la red de forma única
  - Ejemplos:
    - http://www.ejemplo.com/directorio/index.php:8080

Ejercicio 3.

- HTTP/HTTPS
    - **HTTP**:
        - HTTP se basa en sencillas operaciones de solicitud/respuesta. Un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado. Todas las operaciones pueden adjuntar un objeto o recurso sobre el que actúan; cada objeto Web (documento HTML, fichero multimedia o aplicación CGI) es conocido por su URL. Usa el puerto 80 por defecto.
        - Cada vez que un cliente realiza una petición a un servidor, se ejecutan los siguientes pasos:
        - Un usuario accede a una URL, seleccionando un enlace de un documento HTML o introduciéndola directamente en el campo Location del cliente Web.
        - El cliente Web descodifica la URL, separando sus diferentes partes. Así identifica el protocolo de acceso, la dirección DNS o IP del servidor, el posible puerto opcional (el valor por defecto es 80) y el objeto requerido del servidor.
        - Se abre una conexión TCP/IP con el servidor, llamando al puerto TCP correspondiente.
        - Se realiza la petición. Para ello, se envía el comando necesario (GET, POST, HEAD,…), la dirección del objeto requerido (el contenido de la URL que sigue a la dirección del servidor), la versión del protocolo HTTP empleada (casi siempre HTTP/1.0) y un conjunto variable de información, que incluye datos sobre las capacidades del browser, datos opcionales para el servidor,…
        - El servidor devuelve la respuesta al cliente. Consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información.
        - Se cierra la conexión TCP.
    - **HTTPS**: Protocolo de transferencia de hipertexto seguro. Usa un cifrado SSL/TLS. Usa el puerto 443. Se utiliza para todos los sitios web en los que un usuario introduce datos. Un campo de aplicación importante es la banca online. En cualquier lugar donde se utilice una cuenta protegida por contraseña, sería sensato tener una conexión HTTPS. Esto incluye el acceso a redes sociales, o cuentas de correo electrónico y de compras, en las que de otro modo se podría causar un gran daño personal con la adquisición ilegal de datos personales. La información personal también puede ser enviada sin una cuenta. Si, por ejemplo, un vuelo o unas vacaciones enteras se reservan en línea, entonces los datos aplicables deben ser comunicados a los proveedores de una manera segura.

Ejercicio 4.

- Comparativa de diferentes navegadores.

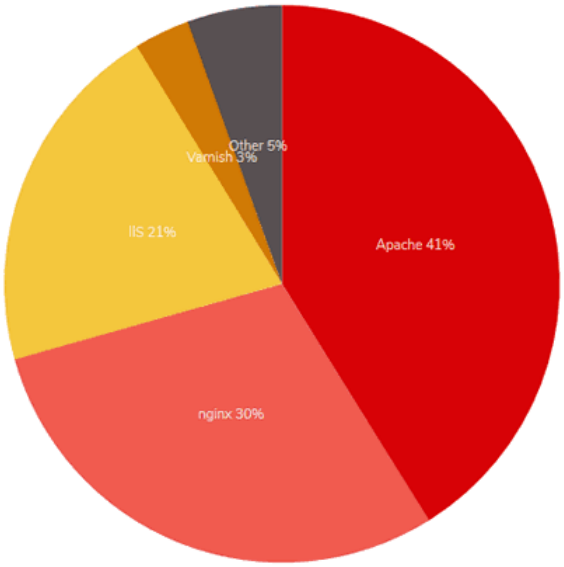


	Versión instalada	Version actual	Motor Renderizado	Interprete JavaScript	Empresa	Acceso a elementos	Acceso a configuración
LunaScape Orion	6.15	6.15	Trident, Gecko, WebKit		LunaScape	Ctrl+u	Herramientas->Configuración
Chromium	79	79	Blink	V8	The Chromium Projects	F12	chrome://settings/
Vivaldi	2.7.1628.33	2.7.1628.33	Blink	V8 7.6.303.30	Vivaldi Technologies	F12	Control + mayus + i
Chrome	76	76	Blink	V8	Google	F12	chrome://settings/
Firefox	69	69	Gecko	SpiderMonkey	Mozilla	F12	
Iexplorer	11	11.95	Trident	Chacra		F12	
Opera	63	63	Blink	V8	Opera	F12	
Safari	12	12	Webkit	JSCore	Apple		
Orca	126	2	Gecko	SpiderMonkey		F12	
SeaMonkey	2,9,03	2,9,03	Gecko	SpiderMonkey	SeaMonkey		
Epic	71.0.3578.98	71.0.3578.98	Blink	Chrome V8	Hidden Reflex	F12	
Iron	75.0.39	75.0.39	Webkit	V8	Srware	F12	
Flock	2.6.1	3.6.4	Webkit	Chromium	Flock Ink		
Avant	Build2	Build2	Gecko, Trinft,Webkit		Avant Force		
Kmeleon	75.1		Gecko		Mozilla	F12	
Falkon	3.1.0	3.1.0	QtWebEngine		KDE		

Maxthon	5,2,7,5000	5,2,7,5000	Blink	V8	Maxthon Internationa	F12	

Ejercicio 5.

- Comparatitva de servidores web más utilizados actualmente.
  - Características de los servidores
  - **Apache**: Ventajas de Apache:
    - Como es de código abierto, no hay tarifas de licencia.
    - Es flexible, lo que significa que puede elegir los módulos que desee.
    - Tiene un alto nivel de seguridad.
    - Fuerte comunidad de usuarios para proporcionar soporte de back-end.
    - Funciona igualmente bien en UNIX, Linux, MacOS, Windows.
    - Desventaja de Apache:
    - Es un servidor basado en procesos, por lo tanto tiene el alcance de tener una sobrecarga debido a las características del hilo.

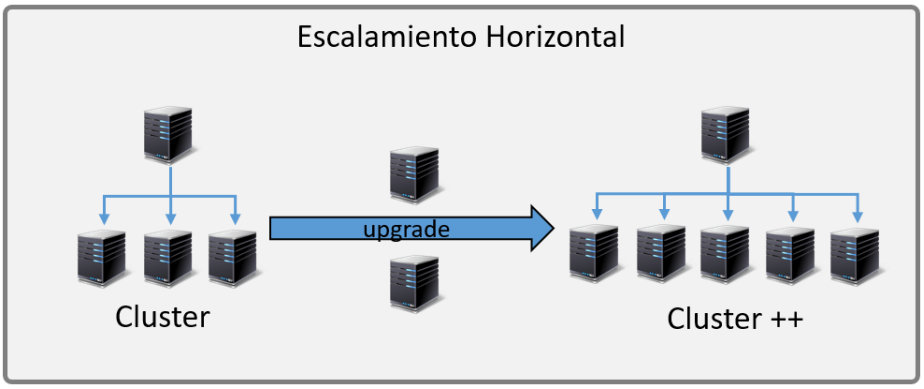


  - **Nginx**: Ventajas de Apache:
    - Como es de código abierto, no hay tarifas de licencia.
    - Es flexible, lo que significa que puede elegir los módulos que desee.
    - Tiene un alto nivel de seguridad.
    - Fuerte comunidad de usuarios para proporcionar soporte de back-end.
    - Funciona igualmente bien en UNIX, Linux, MacOS, Windows.
    - Desventaja de Apache:
    - Es un servidor basado en procesos, por lo tanto tiene el alcance de tener una sobrecarga debido a las características del hilo.
  - **IIS**: Ventajas de IIS:
    - Cuenta con el soporte de Microsoft.
    - Puede tener acceso al marco .NET junto con los scripts ASPX.
    - Se puede integrar fácilmente con otros servicios de Microsoft como ASP, MS SQL, etc.
    - Desventaja de IIS:

- No es tan personalizable como los servidores web de código abierto, por lo que podría enfrentar un problema allí.

Ejercicio 6.

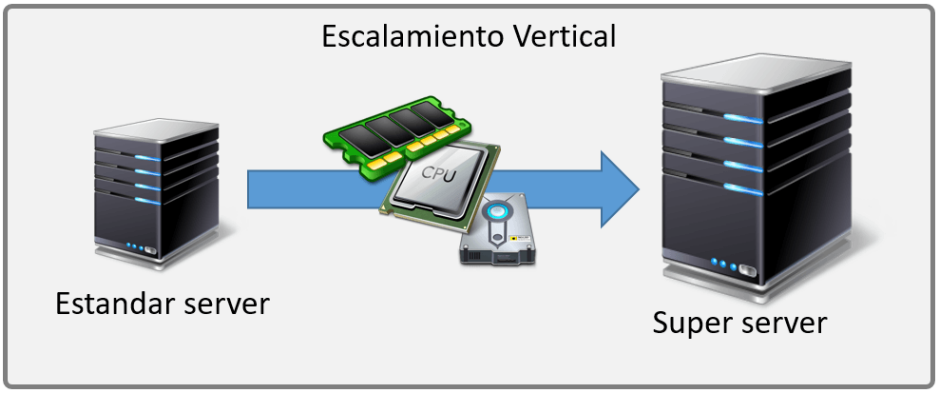
- Escalabiliad horizontal: Ventajas y inconvenientes



- El escalamiento **horizontal** es sin duda el más potente, pero también el más complicado. Este modelo implica tener varios servidores (conocidos como Nodos) trabajando como un todo. Se crea una red de servidores conocida como Cluster, con la finalidad de repartirse el trabajo entre todos nodos del cluster, cuando el performance del cluster se ve afectada con el incremento de usuarios, se añaden nuevos nodos al cluster, de esta forma a medida que es requeridos, más y más nodos son agregados al cluster.
- Ventajas:
  - El crecimiento es prácticamente infinito, podríamos agregar cuantos servidores sean necesarios
  - Es posible combinarse con el escalamiento vertical.
  - Soporta la alta disponibilidad
  - Si un nodo falla, los demás sigue trabajando.
  - Soporta el balanceo de cargas.
- Desventajas:
  - Requiere de mucho mantenimiento
  - Es difícil de configurar
  - Requiere de grandes cambios en las aplicaciones (si no fueron diseñadas para trabajar en cluster)
  - Requiere de una infraestructura más grande.

Ejercicio 7.

- Escalabilidad vertical: Ventajas y inconvenientes.
- La escalabilidad **vertical** o hacia arriba, este es el más simple, pues significa crecer el hardware de uno de los nodos, es decir aumentar el hardware por uno más potente, como disco duro, memoria, procesador, etc. pero también puede ser la migración completa del hardware por uno más potente. El esfuerzo de este crecimiento es mínimo, pues no tiene repercusiones en el software, ya que solo será respaldar y migrar los sistemas al nuevo hardware.



- Ventajas:
  - No implica un gran problema para las aplicaciones, pues todo el cambio es sobre el hardware
  - Es mucho más fácil de implementar que el escalamiento horizontal.
  - Puede ser una solución rápida y económica (compara con modificar el software)
  -
- Desventajas:
  - El crecimiento está limitado por el hardware.
  - Una falla en el servidor implica que la aplicación se detenga.
  - No soporta la Alta disponibilidad.
  - Hacer un upgrade del hardware al máximo pues llegar a ser muy caro, ya que las partes más nuevas suelen ser caras con respecto al rendimiento de un modelo anterior.

Ejercicio 8.

- Razona/relaciona entre la demanda de tu aplicación (número de usuarios que demanda tu aplicación) y el escalamiento horizontal y vertical.
  -

Ejercicio 9.

- ¿Qué es un cuello de botella en un sistema informático? Consecuencias que produce el cuello de botella o bottleneck
  - Un cuello de botella es cuando el PC está realizando una tarea muy exigente y parece que algunos aspectos del sistema deberían desempeñarse mejor. En los juegos, un signo revelador de un cuello de botella es que la tarjeta de vídeo obtiene puntuaciones y rendimiento de referencia mucho más bajas que la mayoría de las personas con la misma instalación. Como el rendimiento del procesador y la tarjeta gráfica son en gran parte responsables de lo bien que va el juego, la culpa se establece en el procesador por causar un cuello de botella en el rendimiento, de ahí que se lo denomine “cuello de botella del procesador”.
  - El disco duro del PC es un ejemplo típico. Casi siempre es el más lento entre los componentes. De hecho, nada sucede hasta que pueda entregar los bits de información que el procesador necesita para empezar a hacer algo. Probablemente has notado que algunas de tus grandes aplicaciones toman un largo tiempo antes de que suceda cualquier cosa. Lo más probable es que la culpa sea de un disco duro lento.

Ejercicio 10.

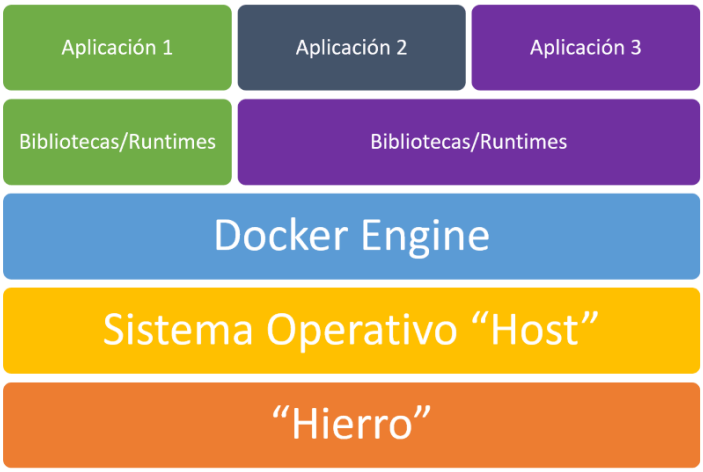
- Comparativa uso de virtualización frente a contenedores.
  - **Máquinas virtuales**: es un sistema operativo completo funcionando de manera aislada sobre otro sistema operativo completo.

La tecnología de VMs permite compartir el *hardware* de modo que lo puedan utilizar varios sistemas operativos al mismo tiempo.

Un esquema simplificado de su arquitectura es el siguiente (léelo de abajo a arriba):

Obviamente, por debajo siempre tiene que haber algún tipo de *hardware* que lo sustente todo. Lo que yo llamo "hierro" y de forma más marketiniana se suele denominar "infraestructura". Puede ser tu ordenador personal para desarrollo, pero si estamos hablando del despliegue de una aplicación real, lo más probable es que sean servidores en el Data Center del proveedor que hayas elegido: AWS, Azure, Digital Ocean, Arsys, OVH, etc. Se puede complicar lo que queramos, pero al final se trata de "hierro": las máquinas físicas sobre las que se ejecuta todo lo demás.

- **Contenedores:** La filosofía de los contenedores es totalmente diferente a la de las VMs. Si bien tratan también de aislar a las aplicaciones y de generar un entorno replicable y estable para que funcionen, en lugar de albergar un sistema operativo completo lo que hacen es compartir los recursos del propio sistema operativo "host" sobre el que se ejecutan.
- Si vemos el esquema equivalente al anterior para el caso de los contenedores, lo que tenemos es algo como esto:
- Arquitectura simplificada de trabajo de contenedores



- A simple vista puede parecer que no hemos ganado mucho. Al fin y al cabo solo desaparece la capa del sistema operativo huésped, y se sustituye el hipervisor por lo que he denominado "Docker Engine". Sin embargo las diferencias son enormes.