MetricSpot
SEO & Web Performance

Nº1

MetricSpot Executive Guides

WEB PERFORMANCE OPTIMIZATION

GUÍA PRÁCTICA PARA OPTIMIZAR EL RENDIMIENTO DE TU WEB

Acerca de este manual

Este manual forma parte de la serie **MetricSpot Executive Guides**.

En él encontrarás directrices que te ayudarán a **incrementar la velocidad de tu Web**, organizadas según el ámbito de aplicación:

- Servidor y Conectividad
- Hojas de estilos CSS
- Imágenes
- Tablets y SmartPhones
- Código HTML
- JavaScript
- Cookies

En ocasiones se incluyen **fragmentos de código**, a modo de ejemplo o para que puedas copiar y pegar en el código de tu Web. Aparecen dentro de cuadrados grises.

El último capítulo incluye enlaces a **herramientas gratuitas** de análisis de rendimiento Web, así como a **otros libros y guías** acerca del Rendimiento Web.

Acerca del autor

Ángel Díaz Ibarra es consultor SEO/WPO y responsable técnico de MetricSpot. Puedes conectar con él vía Twitter (@angeldiazibarra).

Acerca de MetricSpot

En MetricSpot **creamos herramientas y manuales** que te ayudarán a optimizar tu Web para que posicione mejor en los buscadores, atraiga tráfico relevante y genere más conversiones.

También ofrecemos **servicios de consultoría especializada** a empresas de desarrollo Web y Marketing Online en los ámbitos del SEO, rendimiento Web y usabilidad.

MetricSpot apoya la libre circulación de información. Por lo tanto, concedemos la licencia de uso Creative Commons Reconocimiento - Compartir Igual [1] de uso personal y comercial del contenido de este documento.

Enlaces:

1. bit.ly/cc-by-sa-es

Introducción al Web Performance Optimization (WPO)

La Optimización del Rendimiento Web o WPO (siglas inglesas para "Web Performance Optimization") consiste en una serie de técnicas para **acelerar el funcionamiento de una Web** con el fin de mejorar la usabilidad y la experiencia del usuario.

El pionero en este campo fue **Steve Souders**, ex ingeniero de Yahoo, actual responsable de rendimiento de Google y autor de varios libros sobre el tema.

Se han hecho numerosos estudios acerca de cómo la rapidez de una Web está directamente relacionada con la cantidad de tráfico que recibe y las probabilidades que hay de generar conversiones.

En este manual podrás encontrar **consejos y trucos** que, aplicados a tu Web, harán que ésta tenga un rendimiento mucho mayor.

1. Servidor y Conectividad

Lo más importante: un buen proveedor de Hosting

Sin un buen servidor el resto de las optimizaciones que hagamos en nuestra Web no servirán de mucho, ya que **seguirá cargando lenta**.

La mejor inversión que puedes hacer si tu Web carga lenta es contratar un buen servicio de Hosting. Además, **hay buenos servidores baratos** que pueden costar incluso menos que otros con peor rendimiento.

Reduce el número de peticiones HTTP

Cada vez que se carga un elemento, el navegador tiene que **hacer una llamada al servidor**, esperar una respuesta y recibir los datos antes de continuar.

Cada llamada externa puede demorar el renderizado algunos milisegundos. Si tenemos que cargar muchos elementos (archivos CSS, archivos JavaScript, imágenes, etc.) el tiempo de espera puede ser de **varios segundos**.

Una solución es **combinar varios ficheros** CSS o JavaScript en uno y combinar los iconos o **imágenes en Sprites**. Veremos cada punto detalladamente en su apartado correspondiente.

Utiliza HTTP/1.1 en vez de HTTP/1.0

De las dos versiones del protocolo HTTP que existen, la 1.1 ofrece la gran ventaja de que **permite la compresión de datos** con funciones como **gzip** o **deflate**.

Utiliza GZIP y DEFLATE

GZIP comprime los datos en el servidor antes de enviarlos al navegador del usuario. Es muy fácil de configurar, y todos los navegadores y servidores modernos lo soportan.

Hay **determinados archivos** que es mejor no comprimir (como las imágenes) pero gran parte del contenido de un sitio es simple texto (HTML, CSS, JavaScript, JSON, XML, etc.) y conviene comprimirlo.

En **servidores Apache** se activa editando el archivo **.htaccess** añadiendo estas líneas:

```
<IfModule mod_deflate.c>
# Filtramos los tipos de contenido
AddOutputFilterByType DEFLATE text/plain
AddOutputFilterByType DEFLATE text/html
AddOutputFilterByType DEFLATE text/xml
AddOutputFilterByType DEFLATE text/css
AddOutputFilterByType DEFLATE application/xml
AddOutputFilterByType DEFLATE application/xhtml+xml
AddOutputFilterByType DEFLATE application/rss+xml
AddOutputFilterByType DEFLATE application/javascript
AddOutputFilterByType DEFLATE application/x-javascript
AddOutputFilterByType DEFLATE application/x-httpd-php
AddOutputFilterByType DEFLATE application/x-httpd-fastphp
AddOutputFilterByType DEFLATE image/svg+xml

# Quitamos los navegadores que dan problemas con GZIP
BrowserMatch ^Mozilla/4 gzip-only-text/html
BrowserMatch ^Mozilla/4\.0[678] no-gzip
BrowserMatch \bMSI[E] !no-gzip !gzip-only-text/html

# Nos aseguramos de que los Proxy no muestren contenido equivocado
Header append Vary User-Agent env=!dont-vary
</IfModule>
```

Activa el Caché/Expiración

Hay algunos tipos de archivos, los llamados **contenidos estáticos** (CSS, JavaScript, imágenes, etc.) que no suelen cambiar en mucho tiempo.

Cada vez que se carga una página el navegador tiene que descargar todos los archivos. Podemos ahorrarle tiempo al usuario y **evitar solicitudes innecesarias** al servidor permitiendo al navegador cachear archivos.

Para ello, primero tenemos que **separar todos los contenidos estáticos** en ficheros aparte y después activar el cacheado editando el archivo **.htaccess** añadiendo estas líneas:

```
<IfModule mod_expires.c>
  ExpiresActive On
  ExpiresDefault "access plus 600 seconds"
  ExpiresByType image/x-icon "access plus 604800 seconds"
  ExpiresByType image/jpg "access plus 604800 seconds"
  ExpiresByType image/jpeg "access plus 604800 seconds"
  ExpiresByType image/png "access plus 604800 seconds"
  ExpiresByType image/gif "access plus 604800 seconds"
  ExpiresByType application/x-shockwave-flash "access plus 604800 seconds"
  ExpiresByType text/css "access plus 604800 seconds"
  ExpiresByType text/javascript "access plus 604800 seconds"
  ExpiresByType application/x-javascript "access plus 604800 seconds"
  ExpiresByType text/html "access plus 600 seconds"
  ExpiresByType application/xhtml+xml "access plus 600 seconds"
</IfModule>

<IfModule mod_headers.c>
  <FilesMatch "\\.(ico|jpeg|jpg|png|gif|swf|css|js)$">
    Header set Cache-Control "max-age=604800, public"
  </FilesMatch>
  <FilesMatch "\\.(x?html?|php)$">
    Header set Cache-Control "max-age=600, private, must-revalidate"
  </FilesMatch>
</IfModule>
```

Con las instrucciones de arriba le estamos diciendo al navegador que cachee los **contenidos estáticos durante una semana** (604800 segundos) y los **contenidos dinámicos durante cinco minutos** (600 segundos).

Si estás en versión de desarrollo o **aplicando cambios a la Web**, deberías desactivar el caché.

Activa las ETags

Las ETags son identificadores únicos que indican la **última vez que se modificó un archivo**.

Si la ETag indica que el documento no se ha modificado desde la última descarga, el servidor envía un **Código de Estado 304** indicando al navegador que no descargue el contenido, **ahorrando tiempo y ancho de banda**.

Puedes activar el uso de ETags añadiendo esta línea al archivo **.htaccess**:

```
FileEtag MTime Size
```

Utiliza la función FLUSH();

Casi todos los lenguajes de programación tienen una **función flush()** que permite limpiar el buffer de datos y enviarlos al navegador del usuario.

Un buen sitio donde utilizarla es **justo después del <head>** para que el **servidor disponga de más memoria** para procesar el resto de los scripts y el navegador del usuario **pueda ir descargando los CSS** de la cabecera.

Por ejemplo, si tenemos un WordPress (funciona **con PHP**) el código que se insertaría en el archivo **header.php** del Theme sería el siguiente:

```
<head>
  // Aquí va todo el código de la cabecera: TITLE, METAS, CSS, etc.
</head>

<?php flush(); ?>

<body>

<header>
  // Aquí comienza el HEADER visible, el logo, menú de navegación, etc.
```

Evita los enlaces a contenidos 404

Hacer peticiones a **archivos que no existen** (imágenes, CSS, JavaScript, etc.) aumenta mucho el tiempo de carga. Esto se debe a que si el navegador recibe una respuesta **404 not found** parará el resto de descargas hasta procesar el error.

Verifica que no estés enlazando a ficheros que no existen, es fácil equivocarse tecleando el nombre del archivo... o puede que la Web externa de donde lo descargas ya no exista.

Evita las redirecciones

Las redirecciones permiten pasar de una página a otra automáticamente.

Hay muchas Webs cuya página de inicio redirige a una **subcarpeta para un idioma específico** (ej.: www.midominio.com/es/). Si bien puede resultar útil cuando nos visitan desde otros países, **desde el punto de vista del front-end** estamos duplicando una solicitud y esto puede demorar la carga de la página varios milisegundos.

Domain Sharding

Los navegadores vienen configurados con un **límite de conexiones por subdominio** (habitualmente 3 o 4). Esto hace que si tenemos muchos contenidos estáticos los últimos no se empezarán a bajar hasta que hayan terminado de bajarse los primeros, generando así una "cola de descarga".

Podemos hacer que se puedan descargar contenidos estáticos en paralelo **alojándolos en distintos subdominios**. Para empezar, una buena distribución sería colocar todos los Scripts y CSS en un subdominio y todas las imágenes en otro.

Posteriormente podemos analizar la **ruta crítica de descarga** en un diagrama de cascada para optimizar aún más el tiempo de bajada. Puedes **generar diagramas de cascada** con algunas de las herramientas indicadas en el capítulo 8 de este manual.

Red de Distribución de Contenidos (CDN)

Una Red de Distribución de Contenidos o CDN (siglas inglesas para "Content Delivery Network") es similar al Domain Sharding pero almacenando los contenidos en **centros de datos localizados por todo el mundo**.

De esta forma, el CDN entrega a todos los visitantes el mismo contenido pero desde servidores situados cerca de la localización de cada uno, evitando tener que enviar datos desde una punta del mundo a otra y ahorrando tiempo de descarga.

Los CDN son servicios externos y **suelen ser caros**, por lo que su uso está desaconsejado para sitios Web pequeños o cuyo tráfico venga mayoritariamente de una zona geográfica. Los CDN son utilizados habitualmente por Webs con **mucho tráfico** y que tengan cierta **proyección internacional**.

HERRAMIENTAS:
Akamai [1]
Amazon Cloud Front [2]
Windows Azure [3]
Coral CDN [4]
Google App Engine [5] (Gratis, hasta cierta cantidad de tráfico)
Cloud Flare [6] (Gratis, aunque no es muy fiable ni para gestión de DNS)

Enlaces:

1. www.akamai.com
2. aws.amazon.com/es/cloudfront
3. bit.ly/win-cdn
4. www.coralcdn.org
5. developers.google.com/appengine/
6. www.cloudflare.com/features-cdn

2. Código HTML

Evita el código en línea/embebido

El **código en línea** ocurre cuando agregamos un estilo o un evento a determinado DOM. El estilo se agrega con un **atributo "style"** y un evento con cualquier evento HTML [1].

Ejemplo de estilo en línea:

```
<span style="font-family: Arial, sans-serif; color: #555" >ejemplo de
estilo</span>
```

Ejemplo de evento en línea:

```
<div onclick="window.open('http://www.metricspot.com', 'popupwindow',
'scrollbars=yes,width=400,height=300');return true" >ejemplo de popup</div>
```

El **código embebido** se suele colocar en la cabecera de la página. Los estilos CSS se definen dentro de una **etiqueta <style>** y el JavaScript dentro de una **etiqueta <script>**.

Ejemplo de estilos CSS embebidos:

```
<style>
  .class1{
    font-family: Arial, sans-serif;
    color: #555;
  }
</style>
```

Ejemplo de JavaScript embebido (el botón “me gusta” de Facebook):

```
<script>
  (function(d, s, id) {
    var js, fjs = d.getElementsByTagName(s)[0];
    if (d.getElementById(id)) {return;}
    js = d.createElement(s); js.id = id;
    js.src = "//connect.facebook.net/es_ES/all.js#xfbml=1";
    fjs.parentNode.insertBefore(js, fjs);
  }(document, 'script', 'facebook-jssdk'));
</script>
```

Estas dos opciones reducen el número de peticiones HTTP y son válidas cuando el número de páginas de nuestra Web es reducido (como puede ser un Microsite).

Sin embargo, en la mayoría de las Webs es **preferible no introducir CSS o JavaScript en medio del código HTML**. La alternativa es colocar el CSS y el JavaScript en **archivos externos** y enlazarlos de la siguiente manera:

```
<link rel="stylesheet" href="style.css" media="all" />

<script type="text/javascript" src="script.js" ></script>
```

De esta manera reducimos la cantidad de código en cada página, queda mejor organizado y podemos **permitir al navegador que lo cachee** para evitar solicitudes innecesarias durante la navegación por la Web.

Coloca los CSS arriba y los Scripts abajo

Debemos colocar los **CSS dentro del <head>** porque si los colocamos abajo la página se empezará a renderizar sin estilos hasta que se descargue el archivo.

Por el contrario, los archivos **JavaScript deberían ser cargados al final de la página** porque si no bloquean el renderizado mientras se descargan y se ejecutan. El lugar donde deben colocarse es justo **antes de la etiqueta de cierre </body>**.

Un ejemplo de una página con los CSS y JS bien emplazados:

```
<!DOCTYPE html>
<html lang="es">

  <head>
    <meta charset="UTF-8">

    <title>Mi Página Web</title>
    <meta name="description" content="La META DESCRIPTION de mi página." />

    <!-- Enlace a la hoja de estilos CSS -->
    <link rel="stylesheet" href="style.css" media="all" >

  </head>

  <body>

    <h1>El Título de mi Página</h1>
    El contenido de mi página

    <!-- Enlace al JavaScript -->
    <script src="script.js"></script>

  </body>

</html>
```

Reduce el número de <iframe>

Los **<iframe> permiten embeber contenido** de otras páginas en tu Web.

Esta tecnología es utilizada por YouTube, SlideShare, Vimeo y otros sitios de **contenido Multimedia**. También es utilizada por Facebook, Twitter y Google+ cuando insertas sus **botones sociales** de "me gusta" en tu Web.

Cada **<iframe>** es una nueva página Web anidada, y requiere sus correspondientes solicitudes HTTP, cargas de CSS, JS, elementos DOM, etc. Esto **retrasa bastante el renderizado** completo de tu Web, por lo que su uso debería de estar limitado.

Utiliza tablas de ancho fijo

Aunque **cada vez se utilizan menos las <table>** en HTML, si tenemos que presentar datos de esta manera es preferible **definir los estilos de las tablas** mediante CSS en vez de permitir al navegador que ajuste su tamaño al contenido de las celdas.

De esta forma, el navegador puede ir renderizando la tabla **a medida que se carga** en vez de tener que esperar hasta recibir todo el contenido para calcular el ancho de las celdas que tiene que mostrar.

En cualquier caso, **desaconsejamos el uso de <table>** frente a otros métodos de presentar datos, como los ** **.

Cierra los elementos HTML

Algunos elementos de HTML como **<p>**, **<li>** e **<img>** no tienen por qué llevar las **etiquetas de cierre** correspondientes.

Pero **cuando las omitimos** es el navegador el que tiene que calcular cuándo termina el bloque, lo que hace que el **renderizado sea más lento**.

Máximo 1000 elementos DOM por página

Los elementos DOM (p.ej: **<b>**, **<li>**, **<h2>**, ****, etc.) son la base del código HTML y permiten estructurar el código.

Sin embargo, cuando hay un número elevado de ellos en una única página el navegador puede **tardar mucho en procesarlos**, haciendo que se ralentice. Por ello es conveniente que su número sea **inferior a 1000 elementos por página**.

Una manera de **saber cuántos elementos DOM tiene** determinada página es ejecutar la siguiente consulta en la **consola de FireBug**:

```
document.getElementsByTagName('*').length
```

Comprime el código HTML

Es buena práctica usar **indentación**, **tabulaciones** y **comentarios** en el HTML para mantenerlo legible durante la fase de desarrollo.

```
<p>Un párrafo</p>

<!-- A continuación: una lista de enlaces -->

<ul>
  <li><a href="#1">Link 1</a></li>
  <li><a href="#2">Link 2</a></li>
</ul>
```

El navegador puede **prescindir perfectamente de ellos**, y para ellos el código de arriba es igual que el de abajo, con la diferencia de que **el segundo ocupa un 35% menos** de bytes.

```
<p>Un párrafo</p><ul><li><a href="#1">Link 1</a></li><li><a href="#2">Link
2</a></li></ul>
```

Sin embargo, como el desarrollo de una Web es algo que se prolonga en el tiempo (en muchos casos durante todo el periodo de existencia de una Web) es recomendable que **te centres en otros aspectos de la optimización** y dejes el HTML tal y como está para poder editarlo cómodamente en el futuro.

HERRAMIENTAS:
Pretty Diff [2]
HTML Compressor [3]
HTML Minifier [4]
Text Fixer [5]

Enlaces:

1. bit.ly/html-ev
2. prettydiff.com
3. bit.ly/html-comp
4. bit.ly/html-min
5. bit.ly/html-fix

3. Hojas de Estilos CSS

Externaliza todos los estilos

Poner todos los **estilos CSS en un archivo externo** y enlazarlo desde el **<head>** es una buena práctica porque permite el cacheado y ahorra solicitudes y ancho de banda.

```
<link rel="stylesheet" href="style.css" media="all" />
```

Carga los CSS primero

Como ya comentamos en el apartado de **Código HTML**, los CSS deben colocarse dentro del **<head>** para evitar que la página se empiece a renderizar antes de haber cargado los estilos.

Utiliza <link> en vez de @import

Existen **dos formas de cargar** una hoja de estilos CSS, mediante la etiqueta **<link>**:

```
<link rel="stylesheet" href="style.css" media="all" />
```

O bien mediante la directiva **@import**, que se coloca en una **hoja de estilos externa** o dentro de la etiqueta **<style>**:

```
@import url('style.css');
```

Utilizar la etiqueta **<link> es la mejor** opción porque mediante la directiva **@import** el navegador no puede **cargar otros archivos en paralelo**.

Comprime los CSS

Al igual que ocurre con el código HTML, es buena práctica usar **indentación**, **tabulaciones** y **comentarios** en los CSS para mantener el código legible:

```
/* Inicio de la cabecera */

.cabecera {
  width: 960px;
  margin: 0 auto;
}

/* Inicio del cuerpo */

.cuerpo {
  width: 960px;
  margin: 0 auto;
  padding: 10px;
  color: #555555;
}
```

El navegador puede **prescindir de ellos**, por lo que comprimir las hojas de estilos **utilizando herramientas automatizadas** hará que la descarga y análisis del CSS sea mucho más rápido.

Además, algunas herramientas **combinan elementos con propiedades iguales**, comprimen los códigos de color, etc. reduciendo aún más el tamaño de los archivos:

```
.cabecera,.cuerpo{width:960px;margin:0 auto;} .cuerpo{padding:10px;color:#555;}
```

HERRAMIENTAS:
CSS Compressor & Minifier (1)
CSS Compressor (2)
CSS Minifier (3)
YUI Compressor (4)
Clean CSS (5)
Code Beautifier (6)
CSS Beautify (7) - (Proceso inverso)
CSS Lint (8) - (Validador de CSS)

Combinar todos los CSS en uno

Separar los CSS en componentes modulares nos puede ayudar a organizar los estilos:

```
<link rel="stylesheet" href="header.css" media="all" />
<link rel="stylesheet" href="body.css" media ="all" />
<link rel="stylesheet" href="slider.css" media ="all" />
<link rel="stylesheet" href="sidebar.css" media ="all" />
<link rel="stylesheet" href="footer.css" media ="all" />
```

Los navegadores sólo pueden descargar un número limitado de recursos en paralelo, por lo que **es recomendable combinarlos** en un solo archivo para **reducir el número de solicitudes HTTP**.

```
<link rel="stylesheet" href="style.css" media="all" />
```

Enlaces:

1. www.minifycss.com/css-compressor
2. www.csscompressor.com
3. cssminifier.com
4. refresh-sf.com/yui
5. www.cleancss.com
6. www.codebeautifier.com
7. cssbeautify.com
8. csslint.net

4. JavaScript

Externaliza todos los Scripts

Poner todos los **estilos CSS en un archivo externo** y enlazarlo desde el **<head>** es una buena práctica porque permite el cacheado y ahorra solicitudes y ancho de banda durante la navegación por nuestra Web.

Además el código quedará **más limpio y ocupará menos**, haciendo que las descarga del HTML sea más rápida.

Aplaza la ejecución de Scripts

Los archivos JavaScript deben colocarse **abajo del todo**, justo antes de la etiqueta de cierre **</body>** para evitar que bloquee la descarga de otros elementos.

Otra solución es utilizar el **atributo defer** que indica al navegador que **posponga la descarga de un <script>**. Sólo debe utilizarse para Scripts que no tengan **document.write**.

```
<script defer type="text/javascript" src="script.js" ></script>
```

Carga los Scripts asincrónicamente

El **atributo async** permite que el Script pueda ser **descargado en paralelo** de forma asincrónica, sin que bloquee el análisis del resto de la página.

```
<script async type="text/javascript" src="script.js" ></script>
```

¡Cuidado! El código se ejecutará en cuanto se termine de descargar.

Esto hace que si estamos descargando un archivo de **funciones que utilizan jQuery** y todavía no se ha descargado la librería de jQuery en sí, **el Script dará error** porque no se habrán declarado todavía las funciones que utiliza.

Es recomendable que **si vas a utilizar jQuery** lo enlaces desde el **<head>** y no le pongas el atributo async. La página **cargará más lenta**, pero al menos no dará error.

Combinar todos los Scripts en uno

Separar los Scripts en componentes modulares nos puede ayudar a tener el código organizado:

```
<script type="text/javascript" src="slider.js" ></script>
<script type="text/javascript" src="rotations.js" ></script>
<script type="text/javascript" src="plugins.js" ></script>
```

Se recomienda **combinar todos los Scripts** que puedas en un solo archivo para **reducir el número de solicitudes HTTP**:

```
<script type="text/javascript" src="script.js" ></script>
```

Utiliza la última versión de jQuery

Las **versiones más recientes** de jQuery tienen el **código mejor optimizado** y nuevas funcionalidades que las anteriores.

Pero **no utilices la librería jquery-latest.js** de su Web, ya que carga automáticamente las nuevas versiones y deberías **hacer pruebas con ellas primero** para comprobar que todo sigue funcionando correctamente.

Busca siempre cuál es la última versión en Google Developers [1] y enlaza a ella. Actualmente la última versión es la 2.0.0:

```
<script          src="//ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"
></script>
```

Reduce el uso de jQuery

La ventaja que ofrece jQuery es que **facilita muchísimo el trabajo** de escribir código. Pero hay veces que olvidamos que hay algunas funciones de **JavaScript** que hacen exactamente lo mismo que sus homólogas en jQuery y se ejecutan **mucho más rápido**.

Comprime los Scripts

Es buena práctica usar **indentación**, **tabulaciones** y **comentarios** en los Scripts para mantener el código legible.

Sin embargo, los navegadores no los necesitan para ejecutarlos, por lo que **comprimir los Scripts con herramientas automatizadas** hace que los archivos sean más pequeños, **ahorrando tiempo y ancho de banda**.

HERRAMIENTAS:
YUI Compressor [2]
JS Compress [3]
JS Beautifier [4] - (Proceso inverso)
JS Lint [5] - (Validador de JavaScript)

Enlaces:

1. bit.ly/ghl-jq
2. refresh-sf.com/yui
3. jscompress.com
4. jsbeautifier.org
5. www.jslint.com

5. Imágenes

Comprime las imágenes

Las imágenes en JPG y PNG contienen **mucha información adicional** (como metadatos acerca de la cámara, fecha, ubicación...) que puede ser eliminada sin que ello suponga **ninguna pérdida de calidad** de la imagen.

También hay otro tipo de reducción de tamaño de las imágenes, la llamada **compresión con pérdida**, que consiste en reducir la resolución o bien reducir la paleta de colores. Se pierde algo de calidad pero el **rendimiento aumenta**.

HERRAMIENTAS:
Compress PNG [1]
Compress JPG [2]
Kraken Image Optimizer [3]
SmushIt [4]
Tiny PNG [5]
JPEG Mini [6]

Evita el escalado de imágenes

Los atributos de ancho y alto de las **imágenes <img>** deben ser definidos obligatoriamente, bien sea en el propio HTML o mediante CSS. Si no lo hacemos, el navegador tendrá que esperar a descargar la imagen entera y analizarla antes de poder colocarla en la página.

Aun así, si el ancho y alto que hemos definido **no coincide con el original**, el navegador tiene que procesar la imagen y **hacer un "repaint"** con otro tamaño, lo cual lleva mucho tiempo.

Evidentemente hay que **guardar siempre un original** con buena calidad de la imagen, pero conviene disponer de una **copia con el tamaño final** para mostrar en la Web.

Utiliza Sprites

Una Web con **muchas imágenes** puede tardar mucho tiempo en cargar y genera **multitud de solicitudes HTTP**.

Los Sprites son **conjuntos de imágenes agrupadas** en un solo archivo con el fin de reducir el número de peticiones al servidor y **disminuir tiempo de carga**.

MetricSpot utiliza varios Sprites:

Las imágenes se posicionan **mediante CSS**:

```
.toplist {
  width: 20px;
  height: 20px;
  background: url("http://www.metricspot.com/img/topsprite.png") 0 0;
}

.toplist:hover {
  background: url("http://www.metricspot.com/img/topsprite.png") -20px 0;
}
```

HERRAMIENTAS:
Sprite Pad [7]
Sprite Me [8]
Sprite Cow [9]
Texture Packer [10]
Stitches [11]

Utiliza un FAVICON.ICO

Los FAVICON son unas pequeñas **imágenes en formato .ICO** que aparecen al lado de la dirección URL en el navegador.

Podemos indicar su localización en la **cabecera del HTML**:

```
<link rel="icon" href="http://www.metricspot.com/favicon.ico" type="image/x-icon" />
```

Aunque si no lo hacemos el navegador buscará automáticamente el fichero **FAVICON.ICO en la carpeta raíz** de la Web.

Conviene que el FAVICON exista y que esté localizado en la carpeta raíz para evitar que cuando se visite una Web genere un **Error 404 Not Found**.

Se recomienda que su tamaño **no sea superior a 1kb** y que esté **cacheado por mucho tiempo**.

Experimenta con SVG

Los **gráficos vectoriales escalables** o SVG (del inglés: Scalable Vector Graphics [12]) son imágenes creadas a partir de sentencias en XML.

Con el siguiente código podemos generar un círculo naranja en SVG:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" >
  <circle cx="40" cy="40" r="35" stroke="#ff5500" stroke-width="2" fill="#ff6600" />
</svg>
```

Algunas de las características que tienen los SVG son que **ocupan muy poco**, pueden ser **animados** y también pueden ser **escalados sin pérdida** de resolución.

La mayoría de los navegadores modernos soportan SVG y **Google lo utiliza** para muchos de sus servicios, incluyendo **Google Analytics**.

HERRAMIENTAS:
Inkscape [13]
Adobe Illustrator [14]
Google Chart Tools [15]
SVG Edit [16]

Enlaces:

1. compresspng.com
2. compressjpg.com
3. kraken.io
4. www.smushit.com/ysmush.it/
5. tinypng.org
6. jpegmini.com
7. wearekiss.com/spritepad
8. www.spriteme.org
9. www.spritecow.com
10. www.codeandweb.com/texturepacker
11. draeton.github.io/stitches
12. bit.ly/svg-ex
13. inkscape.org
14. bit.ly/ill-buy
15. developers.google.com/chart
16. code.google.com/p/svg-edit

6. Cookies

Elimina Cookies innecesarias

Las Cookies son pequeños ficheros de datos que se guardan en el **ordenador del usuario**. Contienen información necesaria para facilitar el proceso de navegación a través de la Web.

Cada vez que se abre **una nueva página** durante la navegación se tiene que enviar la Cookie, procesarla, posiblemente modificarla y volverla a guardar en el ordenador del usuario. Este proceso lleva tiempo.

En ocasiones las Cookies que utilizamos tiene **poca o ninguna utilidad** para mejorar la navegación y la experiencia del usuario. Deberíamos eliminar este tipo de Cookies.

Reduce el tamaño de las Cookies

El envío de muchos datos en las Cookies puede **ralentizar la navegación**.

Una alternativa es utilizar la Cookie como un **identificador de sesión** de tamaño mínimo y almacenar la información temporalmente en la **base de datos** de nuestro servidor.

Aplica las Cookies al Dominio/Subdominio necesario

Si utilizas técnicas como el **Domain Sharding para almacenar recursos estáticos** en subdominios, es importante que las Cookies queden **limitadas al dominio principal** de la Web.

De esta forma, no se enviarán Cookies cada vez que se haga una petición a alguno de los subdominios de almacenaje de recursos estáticos, ahorrando tiempo y ancho de banda.

Pon una fecha de caducidad adecuada a las Cookies

Conviene limitar el tiempo de vida de una Cookie al estrictamente necesario porque su uso puede **ralentizar la navegación**.

Algunas Cookies deberían tener una vida muy corta, incluso de minutos, como es el caso de las Webs de Banca Online. Por seguridad, no conviene que se guarde la sesión durante mucho tiempo, incluso si el usuario ha dejado el navegador abierto.

Otras Webs, como Google, establecían un tiempo de expiración de décadas para sus Cookies. Esto suscitó algunas quejas por temas de privacidad.

Lo habitual suele ser de establecer una duración **entre una semana y un mes**.

7. Tablets y SmartPhones

Reduce el uso de JavaScript

Los **nuevos SmartPhones** tienen navegadores capaces de procesar JavaScript sin problemas.

Sin embargo, **algunos terminales antiguos** no soportan JavaScript o lo procesan con mucha lentitud, por lo que su uso debería **reducirse al estrictamente necesario**.

Evita redirecciones a subdominios dedicados

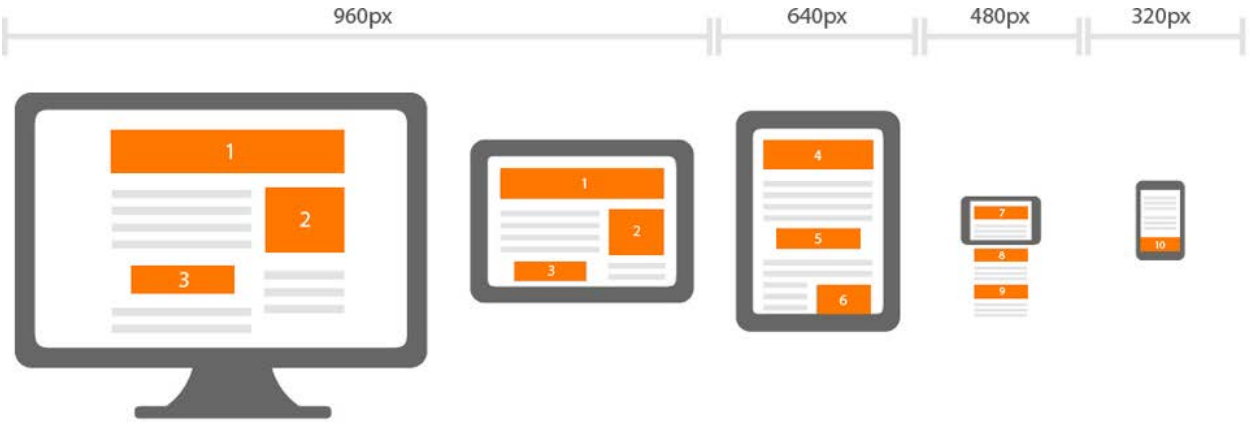
Hasta hace poco, la mejor solución para presentar contenidos adaptados para móviles era **crear un subdominio y redirigir el tráfico** de móviles al mismo.

Sin embargo, con el **Responsive Design** podemos servir el mismo contenido adaptado a móviles y **evitarnos una redirección**, con su ahorro de tiempo y ancho de banda.

Utiliza Responsive Design

El **Responsive Design** ha permitido servir el mismo contenido desde las mismas URLs adaptando los estilos a cada dispositivo. Además de ser **bueno para el SEO** (se evitan problemas de contenido duplicado) es la opción recomendada por Google [1] frente a utilizar subdominios dedicados para móviles.

Se entregan siempre los **mismos códigos HTML desde las mismas URLs** y se utilizan Media Queries en CSS3 [2] para adaptar los estilos a cada dispositivo.



Utilizar Responsive Design puede evitarnos tener que desarrollar Apps para los distintos dispositivos móviles, ahorrando horas de programación. Además, alojar las **Apps en el propio servidor** es mucho más seguro.

Enlaces:

1. bit.ly/goo-res
2. bit.ly/med-que

8. Más Información

Libros y Guías

Libros escritos por Steve Souders:
High Performance Web Sites: Essential Knowledge for Front-End Engineers [1]
Even Faster Web Sites: Performance Best Practices for Web Developers [2]

Directrices de Google:
Google PageSpeed Insights Rules [3]

Directrices de Yahoo:
Yahoo Web Performance Best Practices [4]

Herramientas de Análisis Online

PageSpeed Insights [5]
GT Metrix [6]
Web Pagetest [7]
Pingdom Website Speed Test [8]
Loads In [9]

Extensiones para Navegadores

YSlow [10]
PageSpeed Insights [11]

Enlaces:

1. bit.ly/ss-hpws (Comprar en Amazon)
2. bit.ly/ss-efws (Comprar en Amazon)
3. bit.ly/psi-rul
4. bit.ly/per-rul
5. bit.ly/psi-too
6. gtmetrix.com
7. www.webpagetest.org
8. tools.pingdom.com/fpt
9. loads.in
10. developer.yahoo.com/yslow
11. bit.ly/psi-ext