



Menú



Los mejores cursos online para programadores



CATEGORÍAS

FRIKADAS: Objetos impresos en 3D que se conectan a WiFi sin electrónica alguna | Mezclando colores y creando efectos fotográficos en CSS mediante el uso de "blend modes"

¿Qué diferencia hay entre Docker (Contenedores) y Máquinas virtuales (VMWare, VirtualBox...)?

Por **José Manuel Alarcón**. Publicado el 14 de junio de 2018 a las 08:00

[Curso de Docker y Kubernetes: Desarrolla y despliega aplicaciones basadas en contenedores »](#)



Imagen por José Manuel Alarcón, CC-BY-NC

Mucha gente, cuando oye hablar de [Docker](#) y de lo que se puede hacer con él, lo primero que piensa es en máquinas virtuales. Al fin y al cabo, una máquina virtual es un software que permite aislarse del sistema operativo subyacente y compartirlo entre varias aplicaciones.

Sin embargo las diferencias entre las tecnologías de contenedores como Docker y las máquinas virtuales son enormes, tanto conceptualmente como en la práctica.

En este artículo vamos a repasar brevemente ambas tecnologías para ver cómo trabajan y entender bien sus diferencias. No volverás a tener dudas al respecto 😊

Entendiendo el funcionamiento de las máquinas virtuales

Como su propio nombre indica, una máquina virtual (o VM a partir de ahora, de sus siglas en inglés: *Virtual Machine*) 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.

Todo ese poder computacional no vale de mucho si no le añadimos el "cerebro" en forma de **sistema operativo**. En producción se utiliza algún S.O. especializado en ejecutarse en servidores, es decir, que será Windows o Linux (macOS raramente se usa en este tipo de entornos). Hasta aquí no tenemos nada que no conozcas ya, pues el computador que estés usando ahora mismo para leer esto, funciona de manera esencialmente igual (aunque sea un móvil o una tableta).

Para que las máquinas virtuales puedan ejecutarse es necesario instalar otro componente por encima del S.O.: **el hipervisor**. Un hipervisor es un software especializado en exponer los recursos *hardware* que están debajo del sistema operativo, de modo que puedan ser utilizados por otros sistemas operativos. Esto incluye las CPUs (o *cores*), la memoria y el espacio de almacenamiento además del resto del *hardware*. De este modo se pueden crear **máquinas virtuales** a las que se expone parte del *hardware* subyacente. Estas VMs "engañan" a un sistema operativo convencional para que crea que se está ejecutando sobre una máquina física. Los hipervisores vienen con productos como Hyper-V (incluido gratuitamente con Windows), VirtualBox o VMWare, entre otros.

Nota para el lector avanzado: soy consciente de que existen hipervisores de tipo 1 que se instalan directamente sobre el *hardware* y evitan la necesidad de instalar a mayores un sistema operativo convencional en el medio. De hecho, estos son los que se suelen utilizar en los Data Center para aumentar todavía más el rendimiento. No obstante, en este artículo quiero tratar el tema de la manera

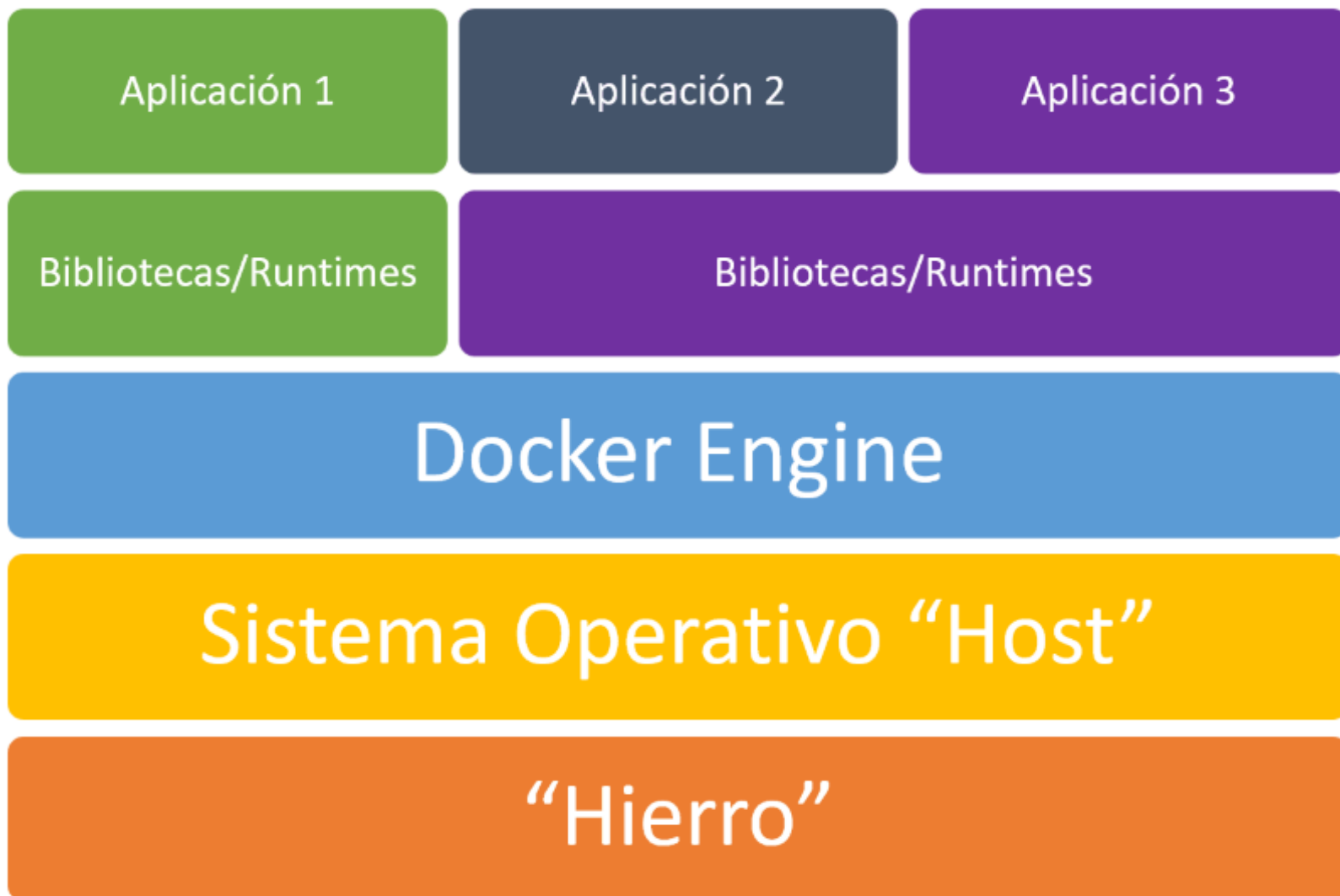
más general posible para que lectores con menos conocimientos puedan tener una buena idea general del funcionamiento tanto de las VMs como de los contenedores.

Gracias a todo esto podemos tener diferentes sistemas operativos ejecutándose en paralelo sobre la misma máquina física, cada uno con su memoria y espacio en disco reservados (los "cores" se pueden compartir), y completamente aislados unos de otros. Gracias al avance de los hipervisores en los últimos años, y a las tecnologías orientadas a la virtualización que ofrecen los procesadores modernos, la pérdida de rendimiento es mínima y es una manera muy eficiente de compartir el *hardware* para sacarle más partido.

Entendiendo el funcionamiento de los 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:



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**, como veremos enseguida.

Nota: he puesto Docker Engine porque me estoy centrando en Docker. Existen otras soluciones de contenedores ([Virtuozzo](#), [LXC/LXD](#), [OpenVZ](#), [DC/OS](#)...), pero Docker es sin lugar a dudas la más popular y utilizada, por lo que me ceñiré a ella. En el caso de otras soluciones, el componente que se encarga de realizar la "contenedorización" se llamará de otra forma, pero el concepto es el mismo.

Docker Engine se encarga de lanzar y gestionar los contenedores con nuestras aplicaciones, pero en lugar de exponer los diferentes recursos de hardware de la máquina de manera discreta (es decir, 1 procesador y "x" GB de RAM... para cada aplicación), lo que hace es compartirlos entre todos los contenedores **optimizando su uso** y eliminando la necesidad de tener sistemas operativos separados para conseguir el aislamiento.

Docker funciona a partir de **imágenes que se pueden reutilizar** entre varias aplicaciones (fíjate en el esquema anterior, en el que las aplicaciones 2 y 3 comparten el *runtime*). Cada una de esas imágenes se puede asimilar a una "capa" que se puede superponer a otras para formar un sistema de archivos que es la combinación de todas ellas. Por ejemplo, una capa puede llevar las bibliotecas o *runtimes* que necesitemos utilizar (como Node.js o PHP), otra con unas bibliotecas determinadas de las que hace uso nuestra aplicación, y otra capa final con el código de nuestra aplicación. La combinación resultante (una nueva imagen, única para nuestra app), es lo que forma la base de nuestro contenedor.

Cuando se lanzan uno o varios contenedores a partir de una imagen, a efectos de nuestra aplicación es como si estuviese ejecutándose en su propio sistema operativo, aislado de cualquier otra aplicación que

hubiese en la máquina en ese momento. Pero la realidad es que están compartiendo el sistema operativo "host" que hay por debajo. Un contenedor ve su propio sistema de archivos, el resultante de "superponer" las capas de las que hablaba antes, y los recursos hardware se van asignando dinámicamente en función de las necesidades, de lo cual se ocupa Docker Engine (o el equivalente si usamos otro tipo de contenedores). Es decir, **Docker aísla aplicaciones, no sistemas operativos completos**.

Nota: también es posible que acceda al sistema de archivos de la máquina local para persistir de forma permanente información propia de la aplicación, ya que si un contenedor se cierra, todos los cambios hechos en su sistema de archivos virtual se pierden. Aunque existen opciones para conseguirlo sin tener que hacer esto. Pero eso es lo máximo que se puede romper ese aislamiento entre el contenedor y el sistema operativo huésped.

Como vemos, **tecnologías que persiguen un fin similar, pero con enfoques totalmente diferentes**.

Ahora que ya entendemos lo básico de cómo funcionan ambas tecnologías, vamos a centrarnos en las diferencias y sus implicaciones prácticas, que es lo que nos interesa.

Docker vs. Máquinas Virtuales

En primer lugar debemos tener en cuenta que, en el caso de los contenedores, el hecho de que no necesiten un sistema operativo completo sino que reutilicen el subyacente **reduce mucho la carga** que debe soportar la máquina física, **el espacio** de almacenamiento utilizado **y el tiempo** necesario para lanzar las aplicaciones. Un sistema operativo puede ocupar desde poco menos de 1GB para algunas distribuciones


de Linux con lo mínimo necesario, hasta más de 10GB en el caso de un sistema Windows completo. Además, estos sistemas operativos, para funcionar requieren un mínimo de memoria RAM reservada, que puede ir desde 1 hasta varios GB, dependiendo de nuestras necesidades. Por lo tanto **los contenedores son mucho más ligeros que las máquinas virtuales**.

Cuando definimos **una máquina virtual debemos indicar de antemano cuántos recursos físicos le debemos dedicar**. Por ejemplo, podemos decir que nuestra VM va a necesitar 2 vCores (procesadores virtuales), 4GB de RAM y un espacio en disco de 100 GB. En el caso de los procesadores, es posible compartirlos entre varias máquinas virtuales (pero no conviene pasarse o irán fatal de rendimiento), y el espacio en disco se puede hacer que solo ocupe lo que de verdad se esté utilizando, de modo que crezca en función de las necesidades y no ocupe siempre tanto como habíamos reservado. Pero en el caso de la memoria y otros elementos (acceso a unidades externas o dispositivos USB) la reserva es total. Por eso, aunque nuestra aplicación no haga uso en realidad de los 4GB de RAM reservados da igual: no podrán ser utilizados por otras máquinas virtuales ni por nadie más. **En el caso de los contenedores** esto no es así. De hecho no indicamos qué recursos vamos a necesitar, sino que es Docker Engine, en función de las necesidades de cada momento, el encargado de **asignar lo que sea necesario para que los contenedores funcionen** adecuadamente.

Esto hace que los entornos de ejecución de Docker sean mucho más ligeros, y que se aproveche mucho mejor el *hardware*, además de permitir levantar muchos más contenedores que VMs en la misma máquina física. Mientras que una VM puede tardar un minuto o más en arrancar y tener disponible nuestra aplicación, **un contenedor Docker se levanta y responde en unos pocos segundos** (o menos, según la imagen). **El espacio ocupado en disco es muy inferior con Docker** al no necesitar que instalemos el sistema operativo completo.

Por otro lado, Docker no permite utilizar en un sistema operativo "host" contenedores/aplicaciones que no sean para ese mismo sistema operativo. Es decir, no podemos ejecutar un contenedor con una aplicación para Linux en Windows ni al revés. Lo cual puede suponer un impedimento en algunas ocasiones.

Nota: Docker para Windows, paradójicamente, utiliza una máquina virtual Linux en segundo plano para poder ejecutar aplicaciones Linux. Los contenedores Windows se ejecutan nativamente en Windows, claro.

Además para poder hacer despliegues avanzados de aplicaciones en contenedores hay que ir más allá de Docker y utilizar tecnologías como [Kubernetes](#) , que nos permiten orquestar y controlar los despliegues con mucha partes en movimiento. Estas tecnologías pueden llegar a ser complejas de aprender y dominar.

En resumen

Los contenedores permiten desplegar aplicaciones más rápido, arrancarlas y pararlas más rápido y aprovechar mejor los recursos de *hardware*. Las máquinas virtuales nos permiten crear sistemas completos totalmente aislados, con mayor control sobre el entorno y mezclando sistemas operativos *host* y huésped.

Cada tecnología tiene sus aplicaciones y sus ventajas según las necesidades y circunstancias de cada desarrollo. En la actualidad los contenedores en general y Docker en particular se están convirtiendo en una tecnología indispensable y cada vez se utilizan para más cosas, no solo para desplegar aplicaciones en producción, sino también para **crear entornos de desarrollo replicables** entre todos los miembros de un equipo, **asegurar que las aplicaciones se van a ejecutar igual en todos los entornos** (desarrollo,

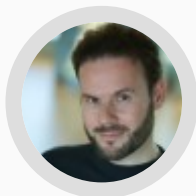
pruebas y producción), etc. Hay quien asegura que, **a medio plazo, la mayoría de los desarrolladores usaremos Docker para desarrollar y desplegar aplicaciones**. A ver qué ocurre, pero Docker sin duda ofrece ventajas muy importantes en todas las fases de un desarrollo de software.

Te dejo algunos enlaces por si quieres profundizar un poco más sobre el tema:

- ▶ [Las ventajas principales de usar Docker](#)
- ▶ [Cómo empezar a desarrollar utilizando Docker](#)
- ▶ [Docker vs Vagrant: diferencias y similitudes y cuándo usar cada uno](#)
- ▶ [Docker y Kubernetes: desarrollo y despliegue de aplicaciones basadas en contenedores](#)

¡Espero que te resulte útil!

[Curso de Docker y Kubernetes: Desarrolla y despliega aplicaciones basadas en contenedores »](#)



Fundador de [campusMVP](#), es ingeniero industrial y especialista en consultoría de empresa. Ha escrito diversos libros, habiendo publicado hasta la fecha cientos de artículos sobre informática e ingeniería en publicaciones especializadas. Puedes seguirlo en Twitter en [@jm_alarcon](#) o leer sus blog [técnico](#) o [personal](#).

[Ver todos los posts de José Manuel Alarcón](#)

¿Te ha gustado este artículo? ¡Compártelo!



No te pierdas nada, recibe lo mejor en tu email

Si te ha gustado este artículo, únete a miles de desarrolladores que ya reciben cada mes nuestro boletín por email. No te pierdas los mejores trucos, noticias y frikadas.

Enviamos poco, pero bueno. Palabra de desarrollador.

[Suscríbete aquí](#)

Síguenos también en:



Comentarios (6)



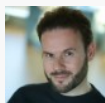
Javier
15/06/2018 18:02:01

Hola, José Manuel.

Gracias por este artículo base para empezar a entender mejor este tema de los contenedores, que cada vez leo y escucho más. Particularmente, me llamó la atención el caso de uso "crear entornos de desarrollo replicables". En mi trabajo, estamos desarrollando un producto para el cual cada developer debe instalar todo lo necesario para montarse su entorno de desarrollo. Esto ya se está volviendo cada vez más enrevesado porque ahora el mismo producto tiene una nueva versión con la mismas tecnologías pero en diferentes versiones, y la verdad terminamos con una ensalada de tecnologías que ha veces generan conflictos en tiempo de desarrollo.

En este sentido, quería consultarte sobre algún recurso (artículos, libros, etc.) sobre las buenas practicas para gestionar mejor estos entornos de desarrollo replicables.

Saludos y gracias!
[Responder](#)



José Manuel Alarcón
15/06/2018 19:38:28

Hola Javier:

Me alegro de que te haya gustado el artículo.

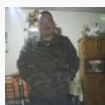
Los contenedores son muy útiles para crear entornos de desarrollo replicables entre todo el equipo, además sin tener que transferir archivos enormes entre unos y otros ya que solo son archivos de configuración basados en texto y luego las imágenes se las descarga automáticamente cada uno (puedes hacer control de versiones de la "infraestructura": ¡mola!)

Tenemos un artículo en el blog de hace unos pocos meses que habla precisamente de esto: www.campusmvp.es/.../...lar-utilizando-docker.aspx

En el curso de Eduard (www.campusmvp.es/.../...s-en-contenedores_237.aspx) hay un módulo dedicado a esto en el que se usan contenedores para compilar sin necesidad de tener nada instalado en el equipo (aparte de Docker, claro).

Saludos!

[Responder](#)



Leonardo Velazquez

17/07/2018 6:57:31

Hola... como que cada quien trabaja de forma independiente ? y luego para juntar el código ??? porque no se crean su propio servidor GIT o SubVersion... yo tengo mi trabajo y también trabajo independiente, y pues tengo una pequeña VPS donde instale Subversion y así desarrollo en mi trabajo en tiempos libres, llego a mi casa, tomo mi laptop y sincronizo.. sigo desarrollando, creo una versión para asegurar los cambios, por si luego tengo que volver,..subo los cambios al server.... llego a la oficina al siguiente y sincronizo y sigo desarrollando..

Esto independiente a docker, docker es para probar o poner en producción tu aplicación

[Responder](#)



José Manuel Alarcón

17/07/2018 8:19:18

Hola:

Nadie está diciendo que el equipo trabaje de forma independiente. Todo lo contrario. Por supuesto que hay que usar un sistema de gestión de código (aunque Subversion no sea hoy en día la mejor solución precisamente), que debería usarse incluso cuando trabaja uno solo. De hecho Docker ofrece otra gran ventaja a la hora de usar un gestor de código y es que tu entorno de desarrollo se vuelve versionable. Dado que Docker se basa en el uso de archivos de texto para configurar y lanzar, puedes llevar un control exhaustivo de los cambios que se producen en el tiempo en tu entorno de desarrollo, y asegurarte no solo de que todo el equipo tiene lo mismo, sino incluso de poder "volver atrás en el tiempo" y ver el entorno exacto que se utilizaba en un momento determinado, lo cual

puede ayudar mucho en ocasiones.

En un equipo y en un entorno complejo Docker puede brindarte una solución a la complejidad que nos comentaba Javier. Y Docker se usa en la actualidad para todo: desde definir y sincronizar el entorno de trabajo del equipo, hasta el despliegue en entornos complejos de producción usando Kubernetes, pasando por testing, pre-producción, etc...

Y por supuesto insisto, eso no tiene nada que ver con usar o no un sistema de control de versiones como Git (o incluso Subversion) que es algo que deberías usar siempre, utilices Docker o no.

Saludos

[Responder](#)



Milko

27/09/2018 0:54:58

Hola buenas tardes,

Tengo 2 ambientes de trabajo:

En uno necesito eclipse, java, tomcat y postgresql, además los fuentes del proyecto

En otro necesito Microsoft Visual Studio, SQL Server y IIS, además los fuentes del proyecto.

Ahora mismo uso 1 máquina virtual por cada cliente, la ventaja que tengo es que mi hosting está limpio de todas esas instalaciones y si la tengo que formatear no pierdo la información de las virtuales porque siempre las respaldo.

¿Con Docker puedo manejar ese escenario?

Otra cosa, leí que para Docker en Windows 10 debo habilitar el Hyper-V ¿Sabes si es así? y creo que el Hyper-V tiene conflicto con el VirtualBox y no quiero eliminar el uso de máquinas virtuales de forma definitiva.

[Responder](#)



José Manuel Alarcón

27/09/2018 8:58:19

Hola Miko:

Como comento en el artículo, Docker no es lo mismo que una máquina virtual, aunque pueda parecerse. Por lo tanto no pienses en Docker como una forma de instalar una especie de máquina virtual en la que vas a ejecutar tus aplicaciones gráficas. En el escenario que planteas lo suyo sería que tuvieses ciertos contenedores con el JDK, Tomcat y PostgreSQL (quizá uno con todo ello) y otros con además con .NET o .NET Core, SQL Server, e IIS para dar soporte a la compilación, pruebas y backend, pero que tengas en la máquina local instaladas los IDEs (editores o entornos gráficos). En este artículo se da una idea de como empezar un entorno de desarrollo con Docker: www.campusmvp.es/.../...lar-utilizando-docker.aspx

En cuanto a lo de Hyper-V, es cierto. Por defecto en Windows Docker utiliza Hyper-V para virtualizar el Linux que necesita para trabajar por debajo con contenedores Linux (si usas contenedores Windows no es necesario). Antes disponían de Docker Toolbox que instalaba Docker para trabajar con VirtualBox en lugar de Hyper-V, pero lo desaconsejan por ser algo no soportado ya por ellos. También puedes tener un arranque dual en Windows que arranque con VirtualBox o Hyper-V según te interese. Te lo cuento aquí: www.jasoft.org/.../...o-equipo-con-windows-10.aspx

Saludos.
[Responder](#)

Agregar comentario

Nombre*:

Correo Electrónico*:

Sitio Web:

Comentario*:

☐ Notificarme cuando se publiquen nuevos comentarios:

Los datos anteriores se utilizarán exclusivamente para permitirte hacer el comentario y, si lo seleccionas, notificarte de nuevos comentarios en este artículo, pero no se procesarán ni se utilizarán para ningún otro propósito. Lee nuestra [política de privacidad](#).

Huevo, perro, abeja o viernes, ¿cuál es un día de la semana?:

Guardar comentario



Aprende online
la base que necesitas

Ver curso

Artículos recientes

[Los principales errores en los 100 sitios web más importantes del mundo y cómo evitarlos](#)

[TRUCO: Cómo resaltar la inicial de un texto con CSS](#)

[Creación de aplicaciones WPF con Xamarin.Forms](#)

[FRIKADAS: Orecchio, comunicación no verbal con las orejas para gente con movilidad reducida](#)

[Creando aplicaciones Linux con Xamarin y Xamarin.Forms](#)

Nuestros Cursos y Libros

- Cursos de Programación Web
- Programación para Móviles y Tablets
- Cursos de Acceso a Datos
- Cursos de Lenguajes y Plataformas
- Preparación de Certificaciones de Desarrollo
- Otros cursos y libros

Últimos vídeos





RecentComments

Entity Framework: Code First, Database First y Model First ¿En qué consiste cada uno? (6)

fabiola escribió: Hola , que buen post , me aclararon mis dudas acer... [\[Más\]](#)

Cómo concatenar subconsultas en un solo campo con SQL Server (13)

John escribió: Hola amigo, muchas gracias Hermano por la explicac... [\[Más\]](#)

Creación de aplicaciones WPF con Xamarin.Forms (1)

DN escribió: Hola, Me surge una duda... entonces con todo es... [\[Más\]](#)

¿Tiene sentido Yarn ahora que tenemos npm 5? (2)

julio escribió: excelente!, justo lo que estaba buscando, muy bien... [\[Más\]](#)

Desarrollador web: Front-end, back-end y full stack. ¿Quién es quién? (14)

Miguel Bolett escribió: Muy buena aclaratoria de los términos, deberíamos ... [\[Más\]](#)

[Comment RSS](#) 



campus
MVP



» Contacta con nosotros



- » Por qué campusMVP
- » Nuestros tutores y autores
- » Cómo aprender a programar
- » Servicios para empresas
- » Cursos gratuitos para empresas
- » Descuentos
- » Preguntas Frecuentes
- » Boletín de Novedades



Programación web



Acceso a datos



Lenguajes y plataformas



Prep. de exámenes oficiales



Desarrollo móvil



Libros de programación

[Ver catálogo completo de cursos y libros](#)

© CampusMVP.com - Aviso Legal y Política de Privacidad - Política de cookies