

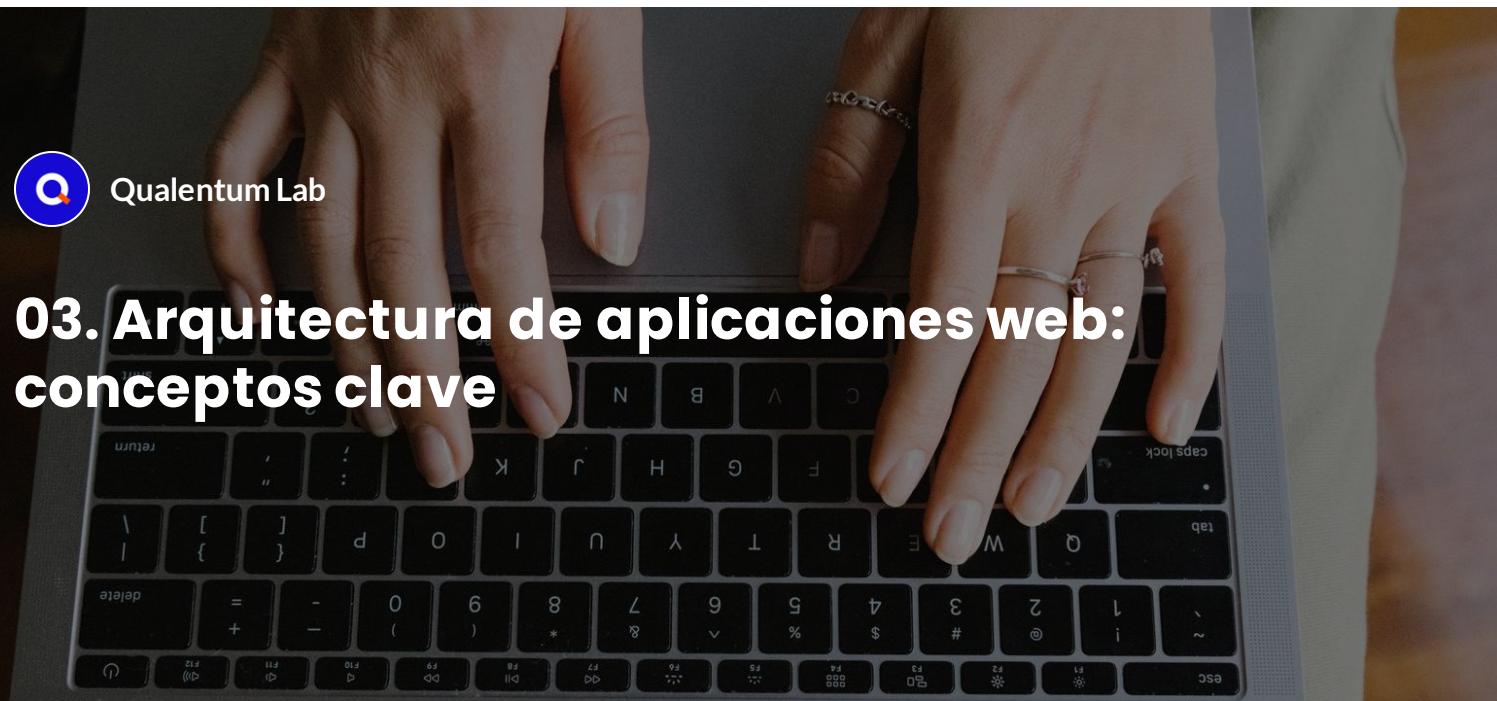
FASTBOOK 03

Arquitectura de aplicaciones web: conceptos clave

Core Desarrollo



03. Arquitectura de aplicaciones web: conceptos clave



A través de estas páginas descubriremos los principales conceptos y estructuras vinculados al mundo de arquitectura, por ejemplo, qué es un servidor, dónde se despliegan los microservicios o en qué consiste Kubernetes.

Por último, para cerrar el tema, dedicaremos todo un apartado al estudio de los patrones de desarrollo donde averiguaremos cuáles son los principales componentes de un software, sus responsabilidades y las relaciones entre ellos.

Autor: Jaime Morales

≡ Servidor

≡ Servidor de web

≡ Kubernetes

≡ Patrones MVC

≡ Conclusiones

Servidor

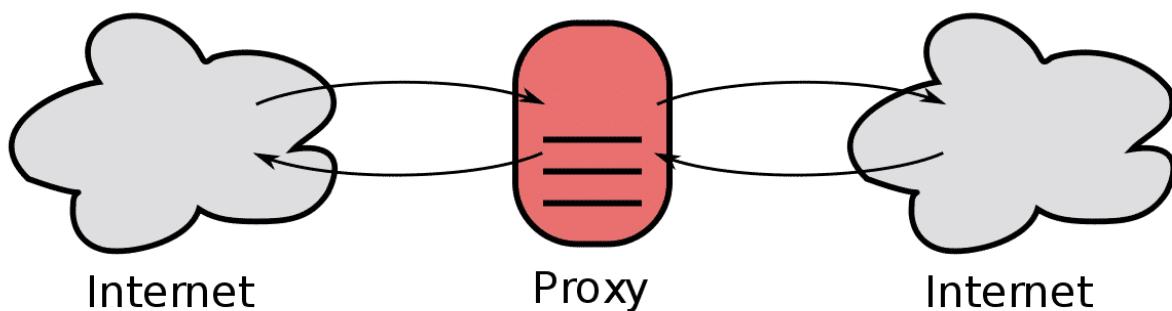


Qualentum Lab

Como ya sabemos, un servidor es un **sistema que proporciona recursos, datos, servicios o programas a otros ordenadores**, conocidos por el término 'clientes', a través de una red. Bajo este epígrafe exploraremos los tipos más relevantes.

Servidor proxy

El término proxy se refiere a un **intermediario que se instala entre el emisor y receptor** para poder recibir las peticiones, en el mundo informático, no es más que un punto entre medias de un usuario e internet, lo que permite dar una capacidad de anonimato tanto de dentro hacia fuera como de fuera hacia dentro.



Fuente: example.com

Tal y como se muestra en la ilustración, si un usuario se conecta directamente desde su dispositivo conectado a internet, la máquina donde esté haciendo la petición sabrá desde qué IP se está conectando, en cambio, si se hace a través del proxy, la IP de la petición será la de proxy por lo que de alguna forma se mantiene el anonimato del emisor, es decir, de quien realmente está realizando la petición.

Los proxys son utilizados, normalmente, para acceder a páginas desde una geolocalización diferente a la real por si tienen capados o rechazan conexiones desde ese país origen.

No se debe confundir un proxy que al final anonimiza tu IP con una VPN (*virtual private network*) que en teoría es más segura.



Puedes consultar esta fuente: [Xataka](#).

Nos podemos encontrar proxys gratuitos que podemos configurarlos para ocultar nuestra IP, por ejemplo: ProxySite, CroxyProxy, Proxium, Hide.me y Steganos.

Cuando se utiliza un proxy en una **red interna** para usarlo como conexión entre el **exterior** (internet) y el **interior** (cada ordenador interno) nos encontramos con muchas **ventajas**:

- Menos tiempo de configuración (solo hay que configurar el proxy).
 - Mayor seguridad.
 - Filtrados más eficientes.
 - Velocidad.
-

La gran ventaja es el anonimato.

En cuanto a las posibles **desventajas**, nos encontramos con tres que podemos agrupar en:

La sobrecarga

El proxy puede verse sometido a demasiada carga si muchos ordenadores realizan peticiones de forma simultánea.

El caché de datos entre 2 ordenadores

Algunos proxys pueden guardar copias de las transferencias, lo que supone cierta intromisión e inseguridad.

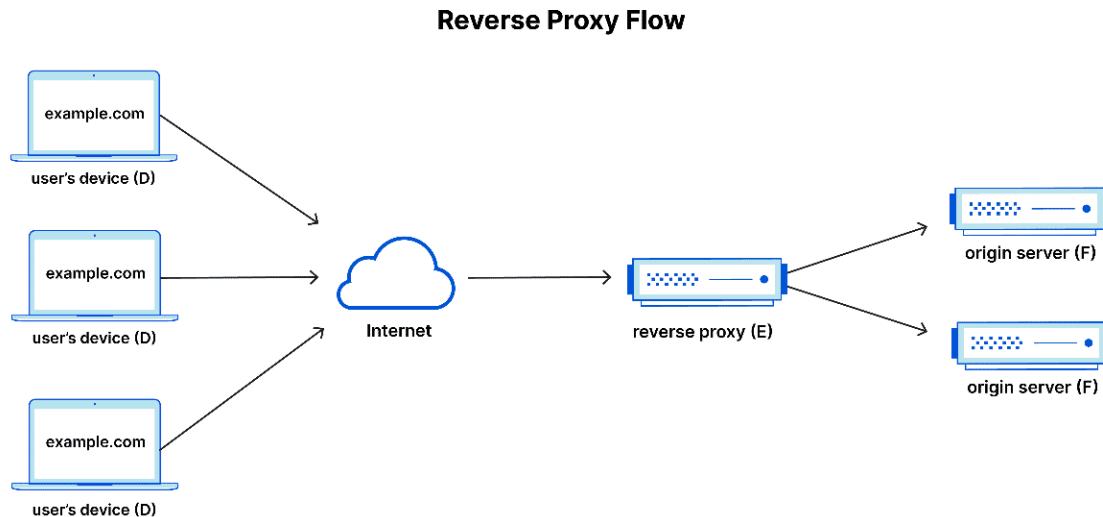
La desactualización

En algunos proxies, la información más actual puede verse afectada.

Tipos de proxys

Ahora veremos algunos **tipos de proxys** y sus características o funciones:

- Proxy web:** aquí el usuario accede desde el navegador web a este servicio de forma manual, a través una aplicación web. Ese servidor HTTP, el intermediario, mediante una URL recibe la petición, accede al servidor de la web solicitada y devuelve el contenido dentro una página propia.
- Proxy inverso:** llamamos así a un servidor que se sitúa delante de los servidores web y reenvía las solicitudes del cliente (por ejemplo, el navegador web) a esos servidores web.



Fuente: Cloudflare.com



Consulta [aquí](#) más datos del proxy inverso.

- Proxy NAT:** técnica mediante la cual las direcciones fuente o destino de los paquetes IP son reescritas, sustituidas por otras (de ahí el enmascaramiento).
- Proxy interdominio:** se usa por tecnologías web asíncronas (por ejemplo, Flash, AJAX, comet...) que tienen restricciones para establecer una comunicación entre elementos localizados en distintos dominios.
- Proxy abierto:** este tipo de proxy ejecutará cualquier petición de cualquier ordenador que pueda conectarse a él, realizándose como si fuera una petición del proxy.

El proxy más conocido es Nginx.

Existen infinidad de productos comerciales conocidos para evitar tener que desarrollar las funciones de un proxy, de cualquier tipo que sea. En este punto, se van a explicar dos de los proxys más conocidos y utilizados tanto a nivel de usuario como a nivel de compañía.

Como podemos imaginar, **el mapa de red de una compañía es un dato superconfidencial** que no debe compartirse fuera de la misma, ya que un atacante podría tener muchos detalles por ello:

- Conocer las IPs de las bases de datos.
- Los servidores web.
- Incluso localizar dónde están los HSMs (*hardware secure module*).

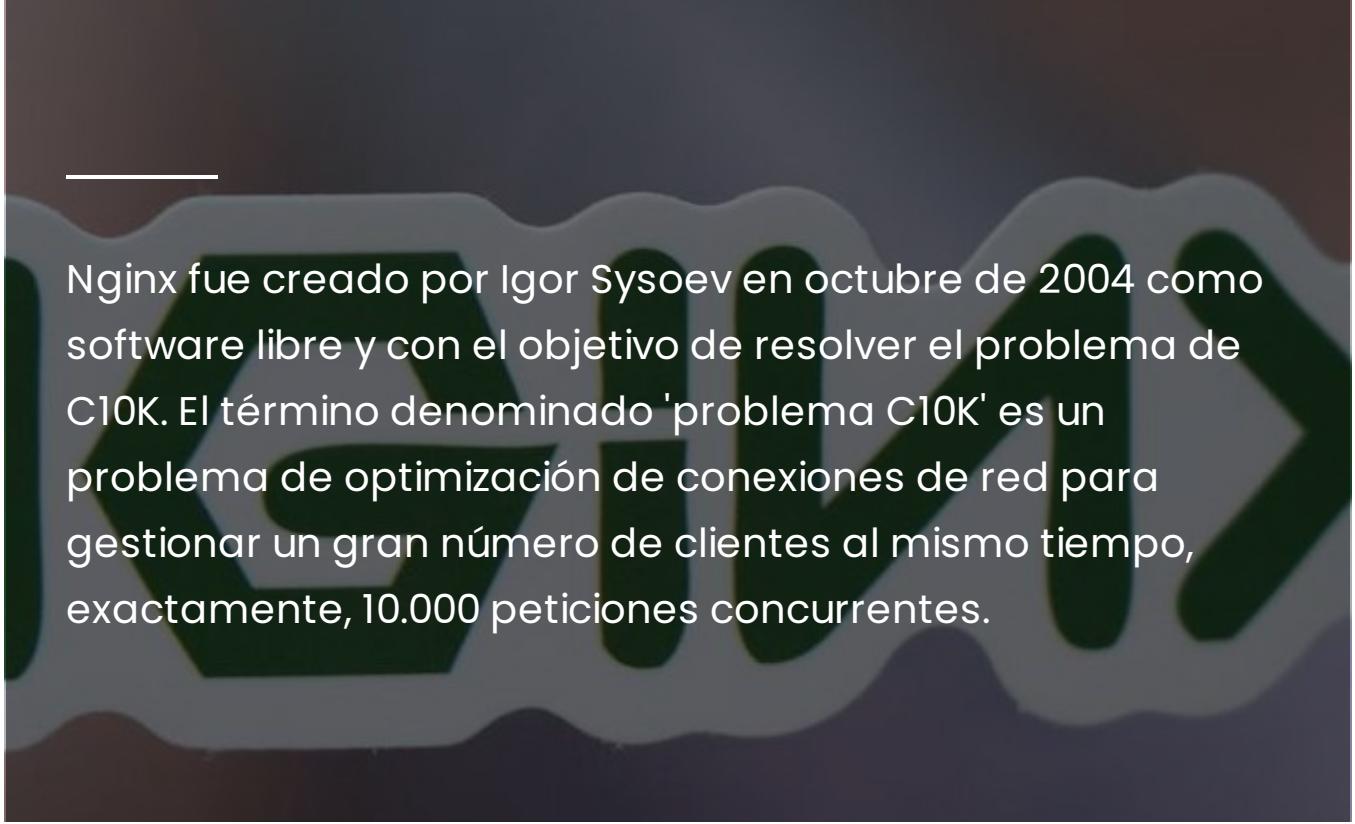
Por este motivo, se utilizan los proxys por delante, para que nadie conozca las IPs que hay detrás y sea **el propio proxy quien redirija internamente** sin que nadie conozca dicho mapa de red. Pues bien, el proxy más conocido para realizar este tipo de acciones es Nginx.

Nginx

Nginx es un servidor web de código abierto, es decir, es gratuito.

Desde su éxito inicial como servidor web, Nginx avanzó creando nuevas funcionalidades como proxy inverso, el balanceador de carga o el caché HTTP.

Grandes compañías utilizan Nginx, por ejemplo, Google, GitLab, VMWare, LinkedIn, Facebook, Twitter... Lo que **demuestra el potencial y el buen funcionamiento** de esta tecnología tan utilizada mundialmente.



Nginx fue creado por Igor Sysoev en octubre de 2004 como software libre y con el objetivo de resolver el problema de C10K. El término denominado 'problema C10K' es un problema de optimización de conexiones de red para gestionar un gran número de clientes al mismo tiempo, exactamente, 10.000 peticiones concurrentes.

Nginx ofrece un **uso de gasto de memoria bajo y alta concurrencia**: el objetivo es crear varias peticiones en un solo hilo de tal manera que no realiza solicitudes diferentes al servidor web y provoca que descargue de manera notable. Estas serían sus características definitorias:

- Proxy inverso con caché.
- IPv6.
- Balanceo de carga.
- Soporte FastCGI con almacenamiento en caché.
- Websockets.
- Manejo de archivos estáticos, archivos de índice y autoindexación.
- TLS / SSL con SIN.

Entre los servidores webs siempre hay mucha competencia en cuanto a cuál ofrece mejores características para las aplicaciones que se van a desplegar. Nosotros vamos a realizar una comparación con Apache que es, sin duda, uno de los servidores webs más extendidos.

Nginx vs. Apache

Compatibilidad del sistema operativo

Aquí ambas tecnologías casi empatarían, ya que ambas son compatibles con Linux, pero si se habla de Windows, Nginx no funciona tan bien como Apache.

Soporte al usuario

Existe una amplia documentación en internet de ambos servidores en este aspecto, incluso hay una comunidad específica en Stack Overflow, sin embargo, Apache no tiene soporte por parte de la compañía, lo que hace a Nginx que resulte más fuerte en este punto.

Rendimiento

Nginx puede ejecutar simultáneamente miles conexiones y es dos veces más rápido que Apache, además, utiliza menos memoria. Aunque si se habla de ejecución dinámica, ambos servidores tienen las mismas capacidades, pero podemos afirmar que Nginx funciona mejor para página más estáticas.



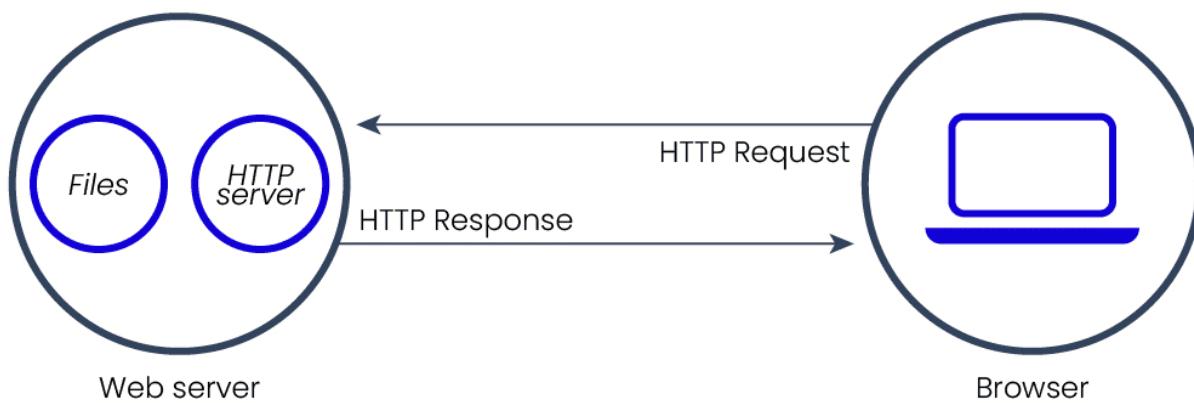
Esta es la bibliografía utilizada para construir este apartado, recomendamos consultarla para ampliar información: [la documentación de Nginx](#), [¿Qué es Nginx?](#) o [Tutorial: qué es Nginx](#).

Servidor de web



Un servidor web es el contenedor donde se van a desplegar las páginas webs que serán visibles desde el navegador.

Este tipo de servidores se encargan de desplegar el backend que recibirá las peticiones desde el navegador. Recordemos que esta funcionalidad se basa en el **modelo de arquitectura cliente-servidor**, tal y como se muestra en la imagen:



La comunicación entre el servidor y el cliente se basa en HTTP, abreviatura de *hypertext transfer protocol*. Este protocolo de comunicación permite operar transferencias de información o archivos en internet. Su variante cifrada, HTTPS, salvaguarda la confidencialidad de los datos de usuarios en esos intercambios.

La clave de todo el proceso radica en la propia petición. Existen muchos tipos de peticiones, aunque las más comunes son las de tipo GET y POST.

LAS PETICIONES GET

LAS PETICIONES POST

Solicitan al servidor algún tipo de información (página HTML, fichero XML, fotos, vídeos...).

LAS PETICIONES GET

LAS PETICIONES POST

Envían datos al servidor como, por ejemplo, cuando se rellena un formulario en un ecommerce.

A continuación, te presentamos los **servidores de aplicaciones** más conocidos.

1

Apache

Es un software que se ejecuta en un servidor.

Su objetivo es establecer una conexión entre un servidor y los navegadores de los visitantes del sitio web (Firefox, Google Chrome, Safari...), mientras envían archivos entre ellos (recuerda que esta estructura se denomina cliente-servidor). Apache es un **software multiplataforma**, esto significa que funciona tanto en servidores Unix como en Windows.

2

Tomcat

Es un servidor web **desarrollado también por la Apache Software Foundation**, cuyo nombre oficial es Apache Tomcat. También es un servidor HTTP, sin embargo, está hecho para aplicaciones Java en lugar de para sitios web estáticos.

Tomcat puede ejecutar varias especificaciones diferentes de Java, como Java Servlet, JavaServer Pages (JSP), Java EL y WebSocket.

Tomcat se creó **específicamente para aplicaciones Java**, mientras que Apache es un servidor HTTP de propósito general. Cabe resaltar que podemos utilizar Apache junto con diferentes lenguajes de programación (PHP, Python, Perl...) apoyándonos en el módulo de Apache apropiado (*mod_php*, *mod_python*, *mod_perl...*).

Aunque también puedes utilizar un servidor Tomcat para servir páginas web estáticas, es menos eficiente para ese propósito que Apache.

Por ejemplo, Tomcat precarga Java Virtual Machine y otras bibliotecas relacionadas con Java que no necesitarás en la mayoría de los sitios web. También es menos configurable en comparación con otros servidores web. Por ejemplo, para ejecutar WordPress, la mejor opción es un servidor HTTP de propósito general como Apache o Nginx.

3

Microsoft IIS

Internet information services o IIS¹ es un servidor web y un **conjunto de servicios** para el sistema operativo Microsoft Windows. Originalmente, era parte del Option Pack para Windows NT. Luego, fue integrado en otros sistemas operativos de Microsoft destinados a ofrecer servicios, por ejemplo: Windows 2000 o Windows Server 2003, 2016 y 2019. Windows XP Profesional incluye una versión limitada de IIS y los servicios que ofrece son FTP, SMTP, NNTP y HTTP/HTTPS.²

El servicio Microsoft IIS convierte a un PC en un servidor web para internet o una intranet, es decir, que en los ordenadores que tienen este servicio instalado se pueden publicar páginas web tanto local como remotamente.

4

WebLogic

Oracle WebLogic es un servidor de aplicaciones Java EE (J2EE) y también un servidor web HTTP, desarrollado por BEA Systems y posteriormente, adquirida por Oracle Corporation. Se ejecuta en Unix, Linux, Microsoft Windows y otras plataformas.

WebLogic puede utilizar Oracle, DB2, Microsoft SQL Server y otras bases de datos que se ajusten al estándar JDBC. El servidor WebLogic es compatible con WS-Security y cumple con los estándares de J2EE 1.3 desde su versión 7, con la J2EE 1.4, desde su versión 9, y Java EE para las versiones 9.2 y 10.x.

Oracle WebLogic Server es parte de Oracle WebLogic Platform. Los componentes de esta plataforma son los siguientes:

- El portal, que incluye el servidor de comercio y el servidor de personalización (construido sobre un motor de reglas producido también por Bea, Rete).
- Weblogic Integration.
- Weblogic Workshop, una IDE para Java.
- JRockit, que es una máquina virtual Java (JVM) para las CPU de Intel.

WebSphere es una familia de productos de software privado de IBM, aunque popularmente se utiliza este término para hacer referencia a un producto específico, al WebSphere Application Server (WAS).

WebSphere ayudó a definir la categoría de software middleware y está diseñado para configurar, operar e integrar aplicaciones de e-business a través de varias plataformas de red, usando las tecnologías de la web. Incluye, por tanto, componentes de *run-time* (como el WAS) y las herramientas para **desarrollar aplicaciones** que se ejecutarán sobre el WAS.

La familia de productos WebSphere, además, agrupa las herramientas para **diseñar procesos de negocio** (WebSphere Business Modeler), para **integrarlos en las aplicaciones existentes** (WebSphere Designer) y para **ejecutar y monitorizar dichos procesos** (WebSphere Process Server o WebSphere Monitor).

Kubernetes



Qualentum Lab

Kubernetes, también conocido como k8s (8 por el número de letras que hay entre la k y la s), es una **plataforma portable y extensible de código abierto** para administrar cargas de trabajo y servicios desarrollados por los ingenieros de Google.

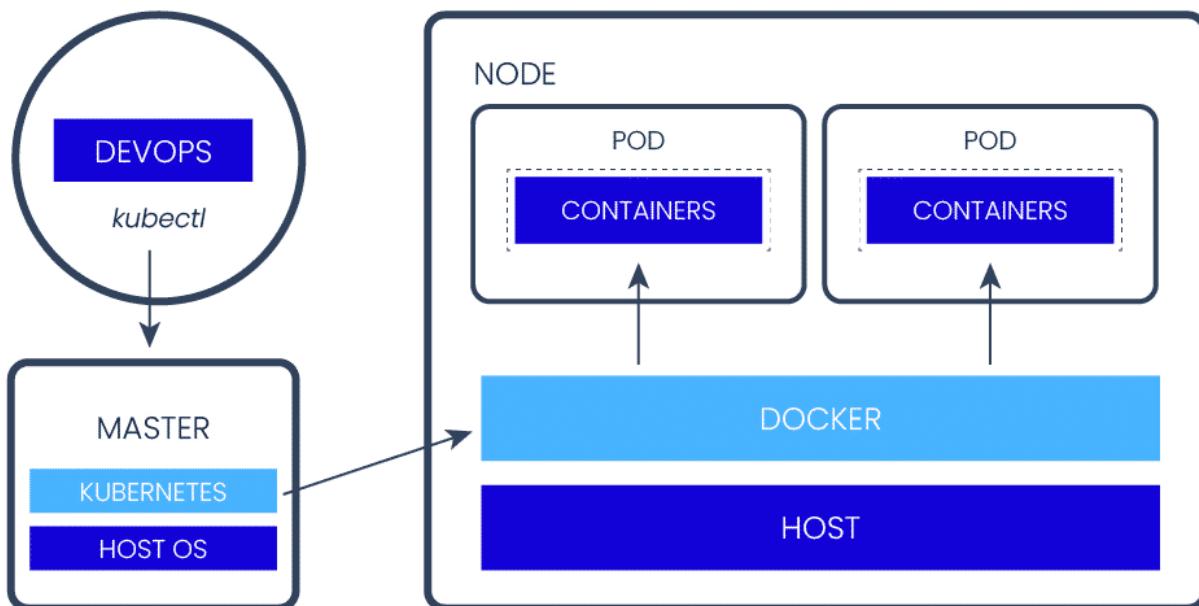
La esencia de Kubernetes es que facilita la automatización y la configuración declarativa.

Tiene un ecosistema grande, en rápido crecimiento, y el soporte, las herramientas y los servicios para Kubernetes están ampliamente disponibles.

¿Cuál es la clave de su éxito? Sin duda que es una plataforma que elimina muchos procesos manuales involucrados en la implementación y escalabilidad de las aplicaciones en contenedores.

En cuanto a su historia, tenemos que remitirnos a Google...

Google genera más de 2 mil millones de implementaciones en contenedores por semana, impulsadas por una plataforma interna: **Borg**. Borg fue la precursora de Kubernetes y las lecciones aprendidas durante el desarrollo de Borg fueron la principal influencia detrás de gran parte de la tecnología de Kubernetes.



Fuente: Openinnova.es

Para entender Kubernetes es fundamental que repasemos algunos **conceptos importantes** del orquestador:

- Master:** llamamos así a la máquina que controla los nodos Kubernetes. Aquí es donde se originan todas las asignaciones de tareas.
- Nodo:** estas máquinas realizan las tareas requeridas y asignadas. El master de Kubernetes las controla.

- Pod:** un grupo de uno o más contenedores implementados en un nodo único. Todos los contenedores de un pod comparten la dirección IP, la IPC, el nombre del host y otros recursos. Los pods abstraen la red y el almacenamiento del contenedor subyacente. Esto le permite mover los contenedores por el clúster con mayor facilidad.
- Controlador de replicación:** controla cuántas copias idénticas de un pod deben estar ejecutándose en algún lugar del clúster.
- Servicio:** separa las definiciones de tareas de los pods. Los proxys de servicios de Kubernetes envían automáticamente las solicitudes de servicio al pod correspondiente, sin importar a dónde se traslada en el clúster o incluso si está siendo reemplazado.
- Kubelet:** este servicio se ejecuta en los nodos y lee los manifiestos del contenedor. También garantiza que los contenedores definidos estén iniciados y ejecutándose.
- Kubectl:** la herramienta de configuración de la línea de comandos de Kubernetes.

¿Pero cuáles son las **grandes ventajas o beneficios** de usar contenedores?

De acuerdo con [la documentación oficial de Kubernetes](#) serían los siguientes:

- 1 **Ágil creación y despliegue de aplicaciones:** esto supone mayor facilidad y eficiencia al crear imágenes de contenedor en lugar de máquinas virtuales.
- 2 **Desarrollo, integración y despliegue continuo:** permite que la imagen de contenedor se construya y despliegue de forma frecuente y confiable, facilitando así los *rollbacks* ya que la imagen es inmutable.

3

Separación de tareas entre Dev y Ops: podemos crear imágenes de contenedor al momento de compilar y no al desplegar, desacoplando la aplicación de la infraestructura.

4

Observabilidad: no solamente se presentan la información y las métricas del sistema operativo, sino la salud de la aplicación y otros signos o señales.

5

Consistencia entre los **entornos de desarrollo**, pruebas y producción: la aplicación funciona igual en un laptop como en la nube.

6

Portabilidad entre nubes y distribuciones: funciona en Ubuntu, RHEL, CoreOS, tu datacenter físico, Google Kubernetes Engine.

7

Administración centrada en la aplicación: eleva el nivel de abstracción del sistema operativo y el hardware virtualizado a la aplicación que funciona en un sistema con recursos lógicos.

8

Microservicios distribuidos, elásticos, liberados y débilmente acoplados: las aplicaciones se separan en piezas pequeñas e independientes que pueden ser desplegadas y administradas de forma dinámica y no como una aplicación monolítica que opera en una sola máquina de gran capacidad.

9

Aislamiento de recursos: logra que el rendimiento de la aplicación sea más predecible.

10

Utilización de recursos: permite mayor eficiencia y densidad.

Patrones MVC



El patrón MVC es un patrón de arquitectura o guía para ver cómo exponer los principales componentes de un software, sus funciones y las relaciones entre ellos. MVC viene del **modelo vista-controlador**, un diseño de la capa de presentación que ya fue utilizado allá por 1979, en el campo de las interfaces gráficas de usuario, y que en la actualidad se sigue aplicando.

**Los tres componentes que definen a un patrón MVC son
el modelo, la vista y el controlador.**

Estudiemos cada uno de ellos.

1

El modelo

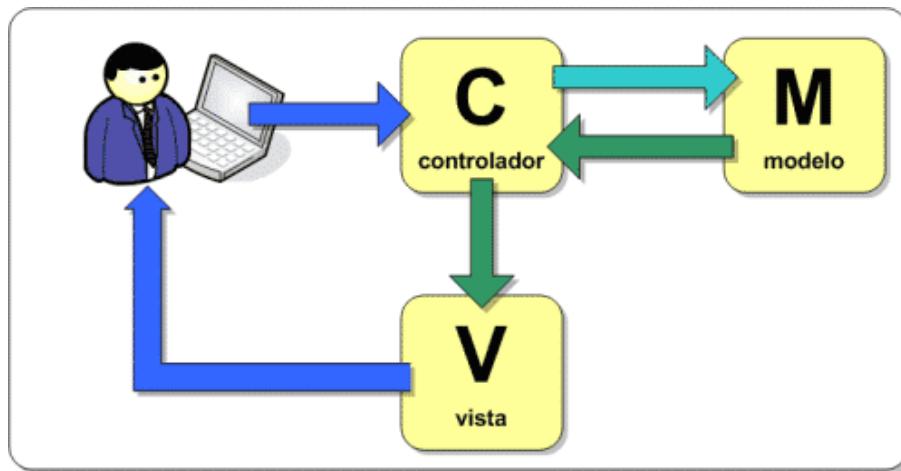
Aquí se encuentra la representación de la base del dominio en el que se desarrollarán las clases que tengan que ver con el objetivo de negocio, por ejemplo: si vamos a crear una calculadora digital de conversión, las clases serán las operaciones para convertir de euros a dólares, de dólares a euros...

Se suele conocer también como **la capa de DTO** (*data transfer object*) para intercambio de información. El modelo será también el encargado de gestionar el almacenamiento y recuperación de datos y de las entidades del dominio.

2

La vista

Dentro de ella se incluyen los componentes de la generación de la interfaz de nuestra aplicación, es decir, la representación del modelo. Cuando las vistas componen la interfaz de usuario de una aplicación, deben contener los elementos de interacción que permitan al usuario **enviar información y ordenar acciones en el sistema**, como botones, cuadros de edición o cualquier otro tipo de elemento, convenientemente adaptados a la tecnología del cliente.



3

El controlador

Actúa como intermediario entre el usuario y el sistema. El controlador captura las acciones del usuario sobre la vista (por ejemplo, clicar un botón, seleccionar un menú o realizar un filtrado). Pero además podrá **interpretarlas y actuar en función de esas acciones**. Por ejemplo, puede llevar al usuario a una nueva vista que represente el estado actual del sistema o invocar a acciones definidas en el modelo para consultar o actualizar información.

Por así decirlo, el controlador funciona como el director de una orquesta, el coordinador general del sistema, el que regula la navegación y el flujo de información con el usuario, ejerciendo también como intermediario entre la capa de vista y el modelo.

Ventajas y desventajas

Como ya podemos deducir del apartado anterior, **el patrón MVC** ofrece interesantes **ventajas**, por ejemplo:

- La separación impuesta por el patrón, por lo que los sistemas serán más limpios y mantenibles.
- Una mayor velocidad del equipo de desarrollo ya que los desarrolladores pueden ocuparse de cada uno de los componentes por separado de tal manera que se puede trabajar en paralelo.
- Obtener múltiples vistas a partir del mismo modelo.
- Gran facilidad para la realización de las pruebas unitarias.

Como en todo, también nos encontramos algunas **desventajas**:

- Hay que ceñirse a las convenciones y al patrón marcado por los componentes.
- La curva de aprendizaje no sencilla.

Conclusiones



En este fastbook, hemos tratado diversos temas vinculados a la arquitectura en los proyectos. Hemos repasado los servidores de aplicaciones web, de proxy... Nos hemos detenido en dos muy populares en la actualidad, Tomcat y Apache, al menos, en cuanto a aplicaciones web. Aunque si son **aplicaciones más grandes**, ya hemos visto que podemos utilizar otros servidores como WebLogic o WebSphere (de pago).

En cuanto a los orquestadores, Kubernetes ha sido el gran protagonista ya que es **el orquestador más utilizado en la actualidad**. De hecho, la mayoría de las grandes compañías se están migrando para poder tener desplegadas las aplicaciones mediante contenedores en Kubernetes; y potentes empresas como AWS, Google Cloud o Azure ofrecen ya Kubernetes como servicio para no tener que montarlo.

Por último, hemos dedicado todo un apartado a los patrones de desarrollo, en concreto, al MVC (modelo, vista, controlador), muy utilizado a la hora de poder definir nuestras aplicaciones y lograr **patrones más lógicos y homogéneos** para el desarrollo de estas.

¡Enhорabuena! Fastbook superado



Qualentum.com