Introducción a Kubernetes

Introducción a Kubernetes

En los últimos años se ha ido extendiendo el uso de contenedores como elementos esenciales para el uso de aplicaciones en entornos en producción, tanto más cuanto más variable sea la demanda, la frecuencia con la que se actualizan o la necesidad de que funcionen de forma ininterrumpida.

Gestionar una aplicación sobre contenedores, que pueda actualizarse rápidamente, que sea escalable o tolerante a fallos, es una tarea compleja que se realiza mediante un software específico que recibe el nombre de orquestador de contenedores.

Kubernetes es un software de orquestación de contenedores desarrollado inicialmente por Google, pero que hoy en día es un proyecto libre independiente utilizado en gran cantidad de entornos diferentes y que se ha convertido en muchos casos en la solución preferida para orquestar aplicaciones basadas en contenedores en entornos en producción.

En este curso conoceremos las principales características de Kubernetes y de las aplicaciones más adecuadas para poner en este entorno y comprobaremos de forma práctica la tolerancia a fallos, la escalabilidad de una aplicación o la gestión del versionado y los diferentes enfoques a la hora de hacerlo en entornos en producción, con o sin interrupciones.

Implantación de aplicaciones web en contenedores

El protocolo http, o su extensión https, ha ido convirtiéndose poco a poco en el "superprotocolo" de Internet y ha ido desplazando paulatinamente el uso de otros protocolos.

De igual forma, la mayor parte del software que se consume hoy en día se podría denominar de forma genérica como aplicación web, aunque hay diferencias importantes sobre la forma de presentarse, ya que no es lo mismo que una persona acceda a una aplicación a través de un navegador, a través de una aplicación móvil o que quien acceda a la aplicación sea una máquina.

En este curso no podemos entrar en detalle sobre las características de estas aplicaciones web, pero sí en las características que deben tener los sistemas que las ofrecen para que cumplan con los requisitos esperados.

Requisitos habituales de las aplicaciones web

Pensemos inicialmente en el caso de una aplicación interna de una empresa que está instalada localmente y que los únicos usuarios que tiene son la plantilla de empleados de la empresa. En ese caso, es fácil determinar los recursos necesarios para que la aplicación funcione de forma adecuada, porque ni el uso de la aplicación se dispara en unos instantes, ni el número de empleados de una empresa varía de forma abrupta.

Por otra parte, las actualizaciones se pueden hacer en momentos en los que el uso es mínimo y, si es necesario una interrupción del servicio, se puede programar para un momento determinado en que tenga muy poco impacto. Las aplicaciones de este tipo no se suelen modificar habitualmente, sino que lo hacen de forma bastante espaciada en el tiempo, por lo que los cambios entre una versión y otra son significativos. Esto, que podríamos llamar informática tradicional, también tiene un impacto importante en la forma de desarrollar las aplicaciones que funcionan bajo este esquema.

Por otra parte, una aplicación web que esté disponible en Internet, tiene miles de millones de potenciales usuarios, que la pueden usar las 24 horas del día y cualquier día del año. Esto tiene unas consecuencias muy importantes, ya que es muy difícil determinar los recursos necesarios para prestar servicios a una demanda muy variable e idealmente, el servicio no puede interrumpirse nunca.

Pero, ¿esto cómo se hace?. ¿Es posible que el mismo sistema se ajuste a una demanda que puede variar de un usuario a un millón?, ¿es posible tener un sistema siempre actualizado y que a la vez no se pare?, ¿cómo se aplican las actualizaciones de software?, ¿poco a poco o con grandes saltos?. Durante este curso, veremos que precisamente esto es lo que trata de proporcionar Kubernetes.

Componentes auxiliares de un servicio web

El componente esencial para servir una aplicación web es un servidor web, pero vamos a ver a continuación, que para poder proporcionar el servicio con los requisitos anteriores, debe apoyarse en un número importante de componentes auxiliares. En los siguientes apartados vamos a ir viendo paso a paso la forma de ir incluyendo diferentes componentes auxiliares y cómo esta inclusión va a ir cambiando la arquitectura de los sistemas que proporcionan el servicio.

Paso 1. Punto de partida

Supongamos que nuestra organización proporciona tres aplicaciones web diferentes que son accesibles a través de las URL:

https://example.com/app1

https://example.com/app2

https://example.com/app3

Estas aplicaciones pueden estar desarrolladas en el mismo lenguaje o en varios diferentes (Python, Java, PHP, etc.), pueden utilizar una base de datos, almacenamiento auxiliar y como se sirven a través de https, es necesario gestionar los certificados x509.

El esquema inicial que pensaríamos para proporcionar estas tres aplicaciones sería una máquina (física o virtual) en la que instalaríamos el servidor web, los servidores de aplicaciones (php, java, ...), el servidor de bases de datos, etc... tal y como aparece en la siguiente imagen:

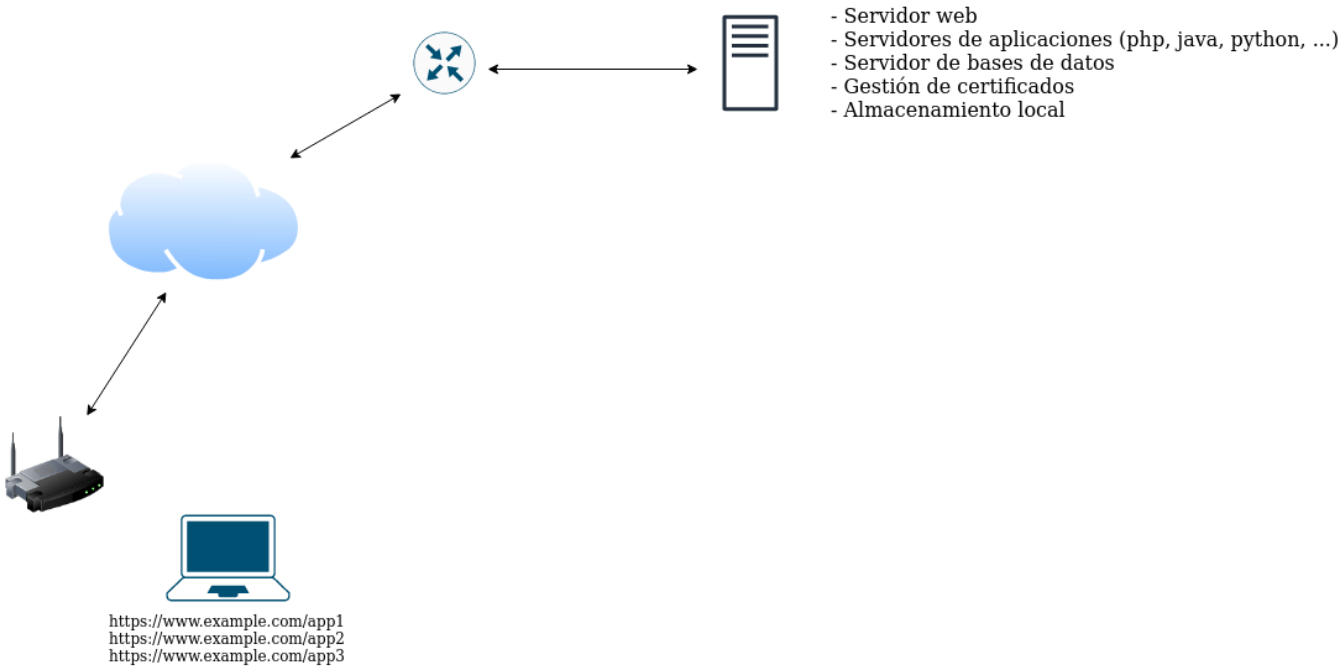


Imagen de elaboración propia (CC BY-NC-SA)

Paso 2. Servidor de bases de datos separado

Desde un punto de vista de seguridad, ubicar el servidor de bases de datos en el mismo equipo que el servidor web es totalmente inadecuado, ya que el servidor web, por su propia naturaleza debe permitir que cualquier usuario acceda desde Internet y una vulnerabilidad en este equipo podría exponer los datos que se ubican en las bases de datos a un potencial atacante. Además, desde el punto de vista del rendimiento y la disponibilidad, separar los servicios en diferentes equipos hace que no haya interacciones entre ellos y no compitan por los mismos recursos.

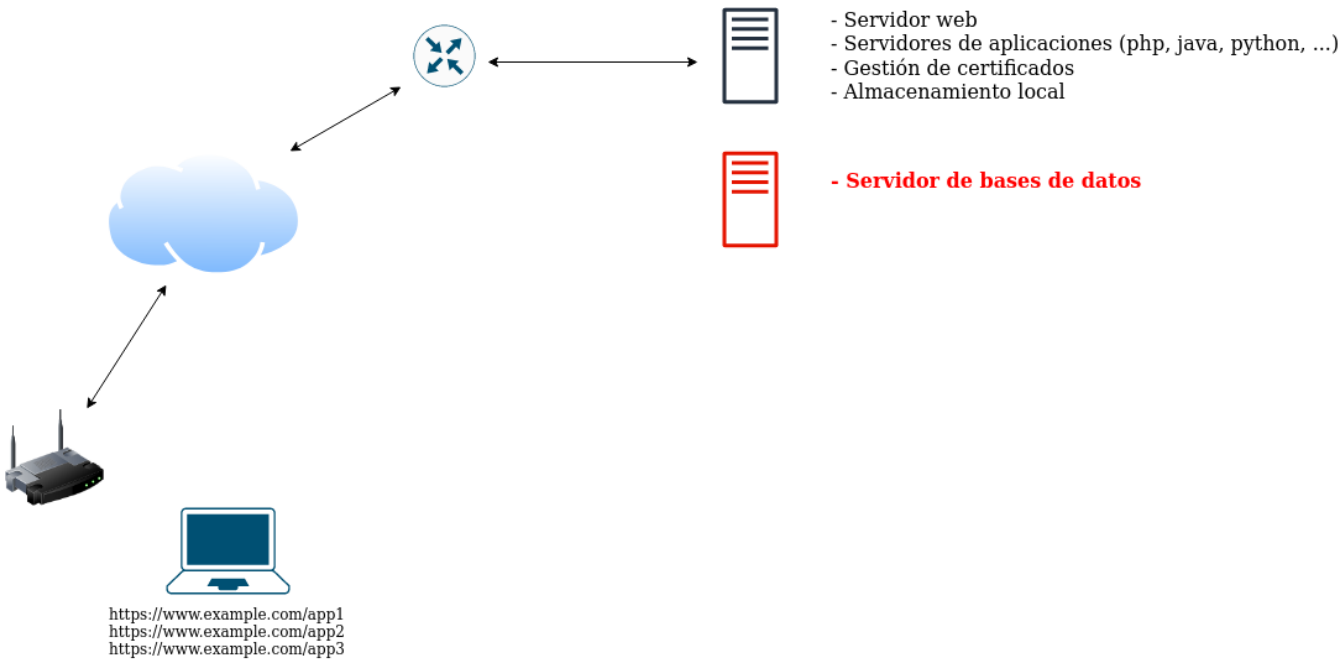


Imagen de elaboración propia (CC BY-NC-SA)

Paso 3. Servidores de aplicaciones en equipos separados

El coste computacional mayor en una aplicación web suele recaer en los servidores de aplicaciones, que son los que ejecutan código complejo, mientras que el servidor web se limita a servir el contenido generado por estos servidores de aplicaciones o los ficheros estáticos del sitio web. Al servir tres aplicaciones web diferentes desde el mismo equipo, podemos tener importantes interacciones entre ellas y que un aumento de uso de una aplicación, repercuta negativamente en las otras. Es por esto, por lo que se puede separar estos servidores de aplicación en equipos dedicados para cada una de ellas. La función del servidor web en este caso, se acerca más a la de un proxy inverso, que pasa la petición web a un equipo interno (el servidor de aplicaciones).

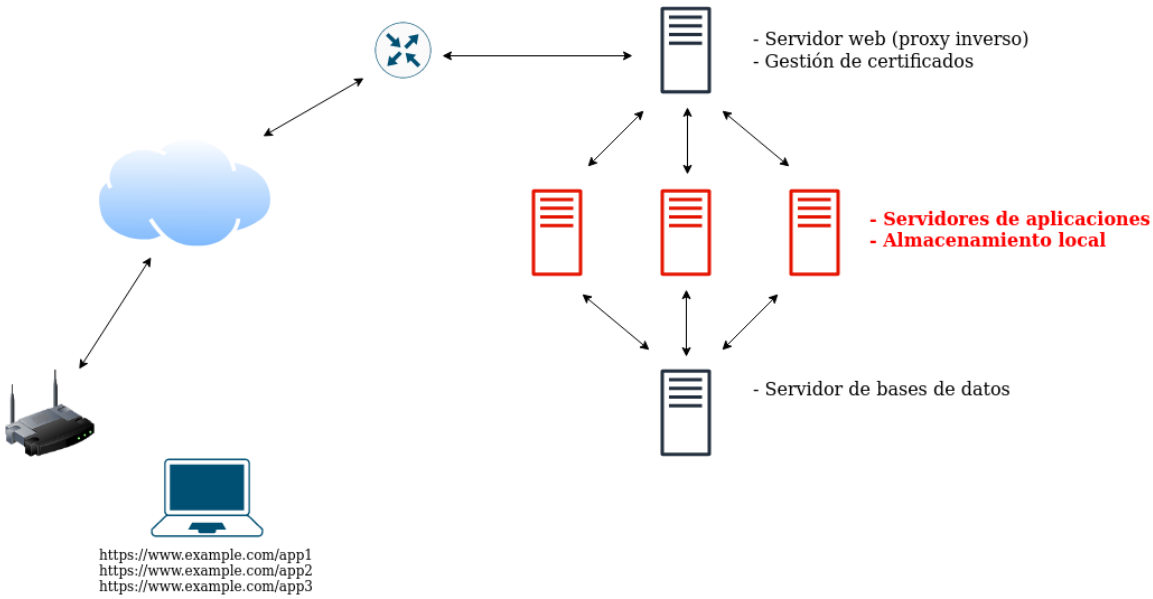


Imagen de elaboración propia (CC BY-NC-SA)

Paso 4. Caché SQL

Los servidores de aplicaciones consultan continuamente a los servidores de bases de datos y cada consulta conlleva un importante coste computacional y una ralentización de la respuesta. Si la misma consulta ya se ha realizado antes, se puede acelerar mucho la velocidad de respuesta con menor coste computacional utilizando un servicio de caché SQL, de manera que los servidores de aplicaciones se configuran para consultar al servidor caché, que servirá directamente la respuesta si ya lo ha hecho anteriormente, o consultará al servidor de bases

de datos en caso necesario. Memcached o redis son dos opciones muy utilizadas como caché SQL.

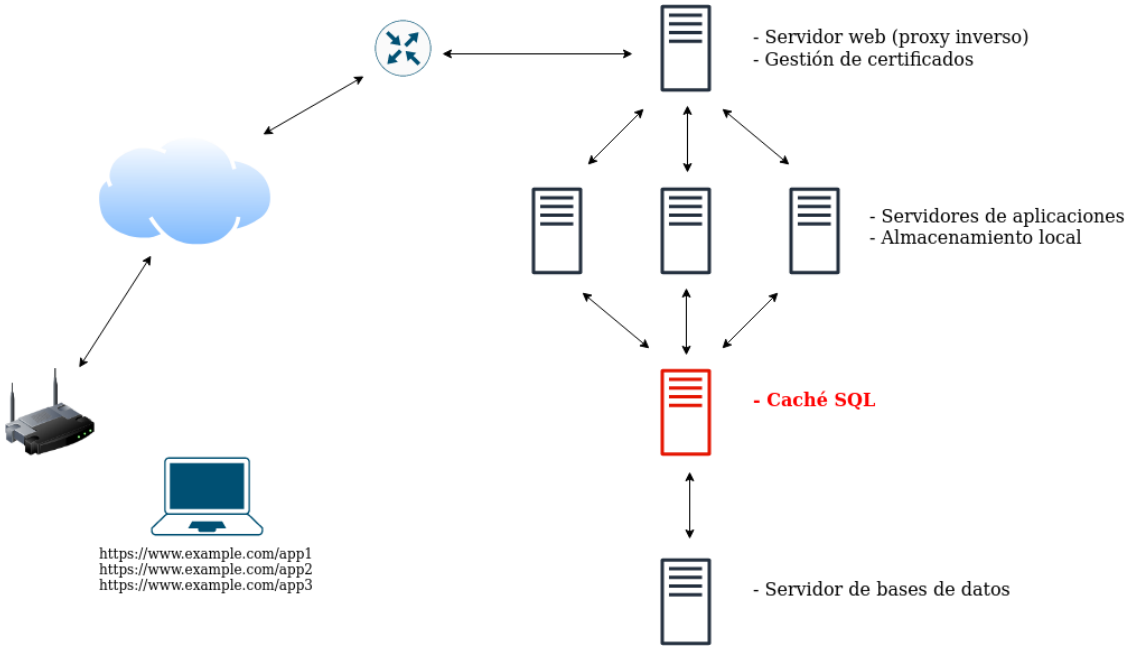


Imagen de elaboración propia (CC BY-NC-SA)

Paso 5. Caché HTTP

Al igual que se puede cachear la respuesta del servidor de bases de datos, se puede hacer lo mismo con la del servidor de aplicaciones o el servidor web. Dependiendo del servidor de aplicaciones, se puede ubicar este componente delante del servidor web o entre éste y el servidor de aplicaciones. Dicho de otro modo, podemos cachear http o algún otro protocolo como CGI, WSGI, etc. Un software muy conocido de caché http es varnish.

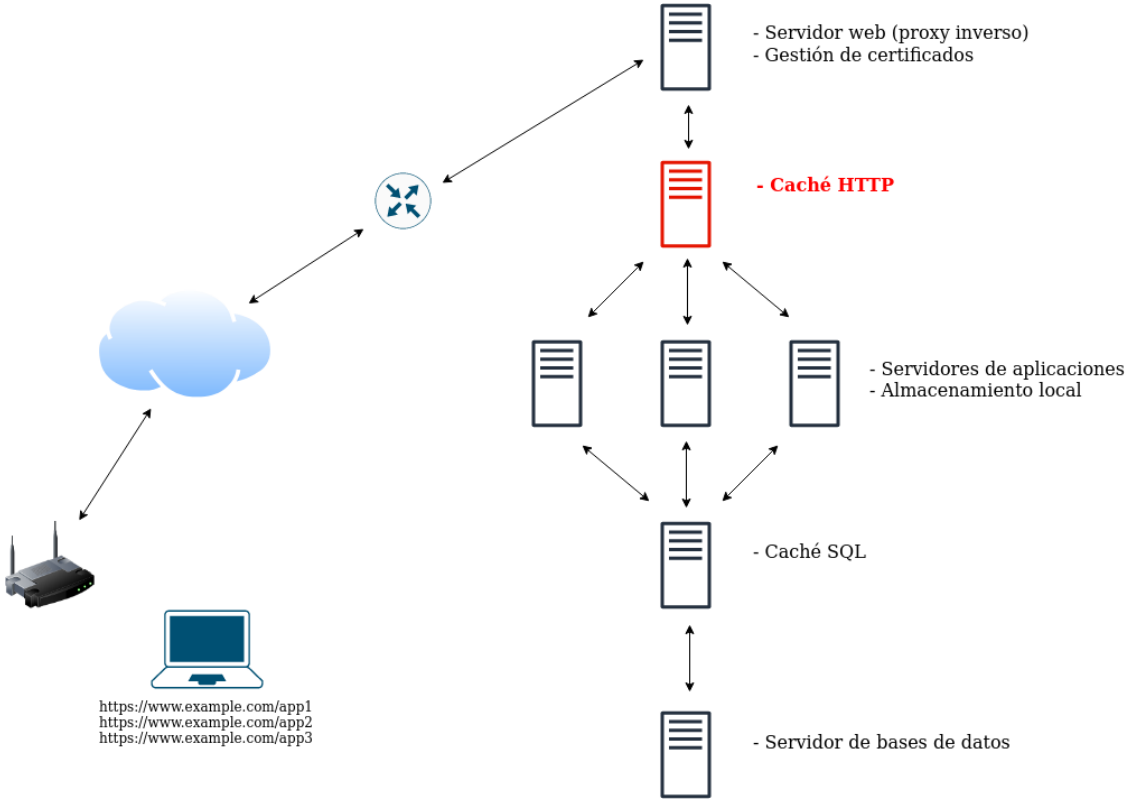


Imagen de elaboración propia (CC BY-NC-SA)

Paso 6. Varios servidores de aplicaciones

Si la demanda de alguna de las aplicaciones varía de forma importante, se puede utilizar escalado horizontal, aumentando el número de nodos de estos servidores de aplicaciones a la demanda de cada momento. Esto conlleva dos importantes modificaciones, el almacenamiento entre los servidores de aplicación de la misma aplicación tiene que estar distribuido de forma que garantice el uso concurrente y se deben repartir las peticiones a los diferentes servidores de aplicación a través de un balanceador de carga.

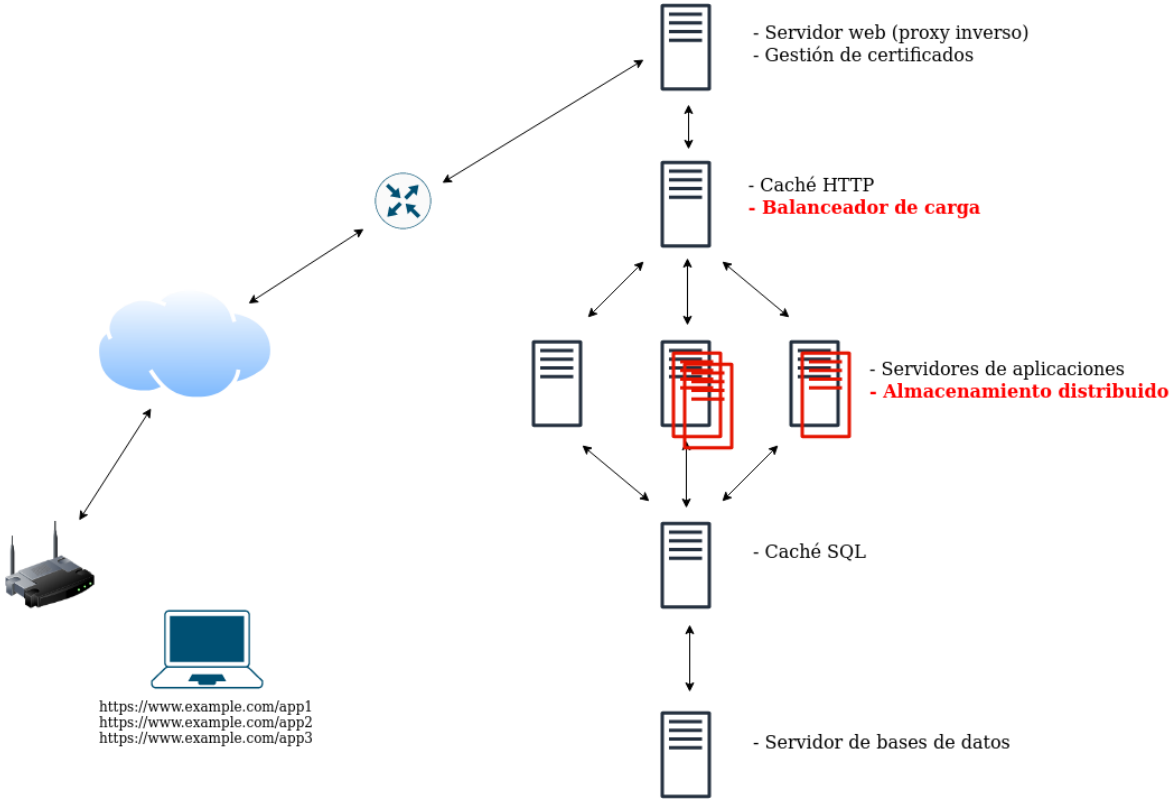


Imagen de elaboración propia (CC BY-NC-SA)

Paso 7. Alta disponibilidad en el resto de componentes

No solo se pueden escalar horizontalmente los servidores de aplicaciones, sino que si queremos ofrecer realmente alta disponibilidad en todos los niveles, debemos crear una arquitectura en la que la disponibilidad nunca dependa de uno solo nodo y el sistema pueda responder siempre ante incidencias puntuales en cualquier nivel.

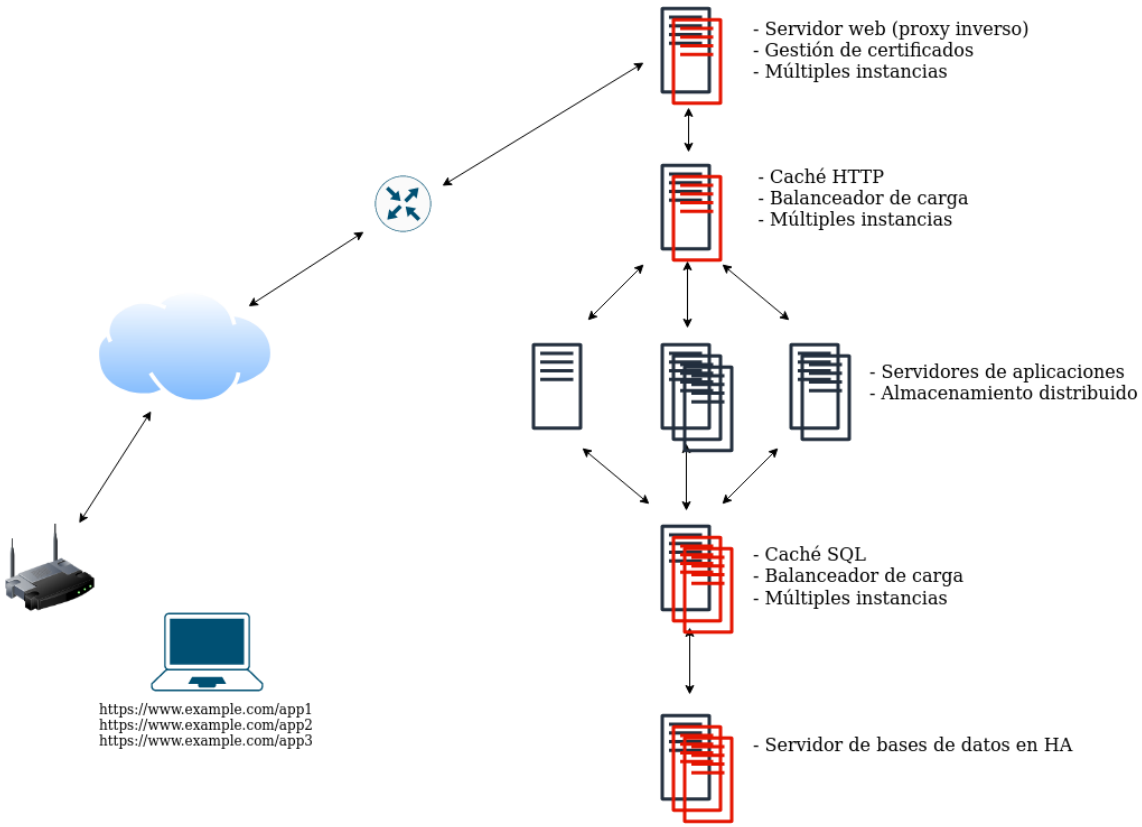


Imagen de elaboración propia (CC BY-NC-SA)

Paso 8. Microservicios y aplicaciones "tradicionales"

Una de las opciones que se considera más adecuada hoy en día para el desarrollo y puesta en producción de aplicaciones web es la utilización de microservicios. Con este enfoque los propios componentes de la aplicación se separan en múltiples componentes que se ejecutan en nodos independientes (típicamente contenedores) y se comunican unos con otros a través de servicios en red que ofrecen al resto.

Estos microservicios no solo incluirían de forma independiente los componentes que hemos explicado hasta ahora, sino que principalmente se refiere a la separación de los componentes internos de la aplicación en diferentes microservicios.

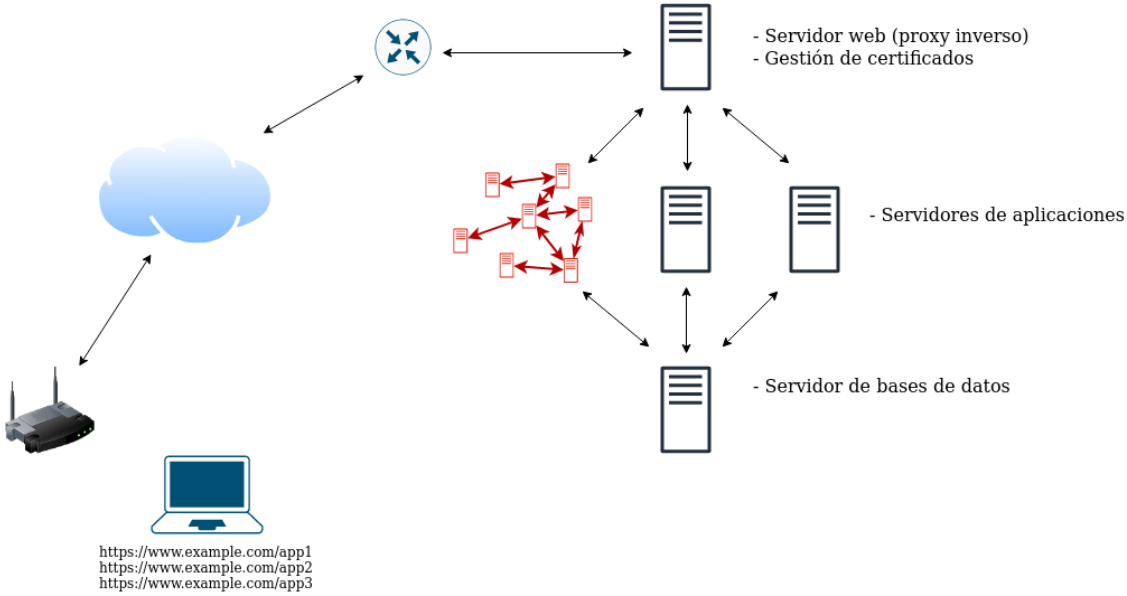


Imagen de elaboración propia (CC BY-NC-SA)

Paso 9. Escalabilidad en los microservicios

Al ofrecer microservicios no podemos tener dependencia de un solo nodo, por lo que al igual que en los pasos anteriores, se debe ofrecer la posibilidad de escalar cualquier componente a la demanda y que el sistema globalmente pueda responder ante cualquier error puntual.

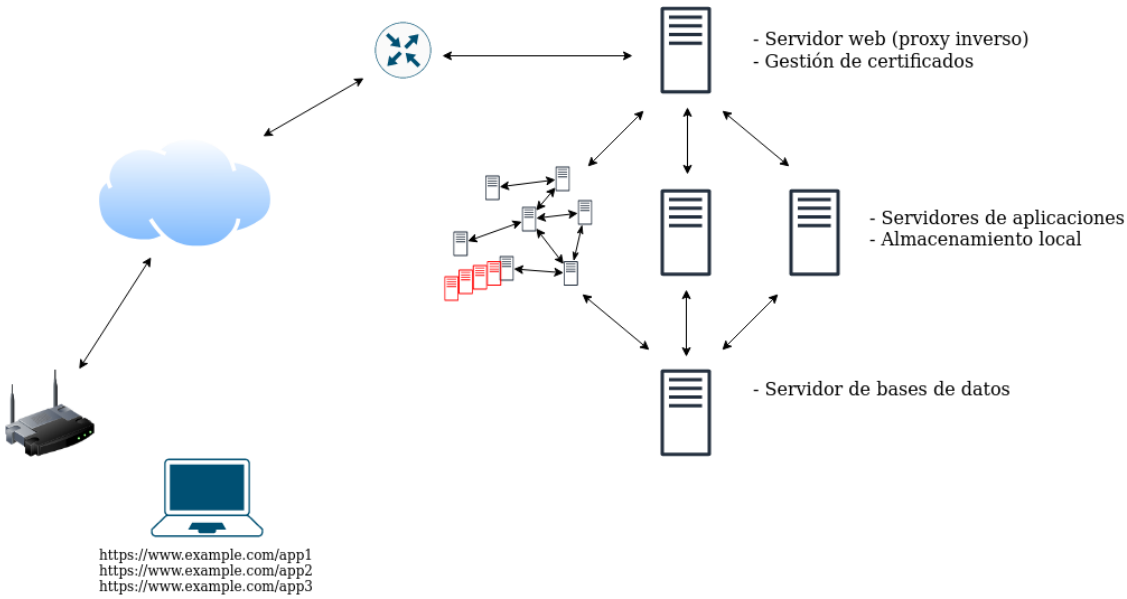


Imagen de elaboración propia (CC BY-NC-SA)

Paso 10. Microservicios en todas las aplicaciones

En lugar de utilizar microservicios en una aplicación, podríamos utilizarlos en todas, pero manteniendo los componentes auxiliares gestionados aparte.

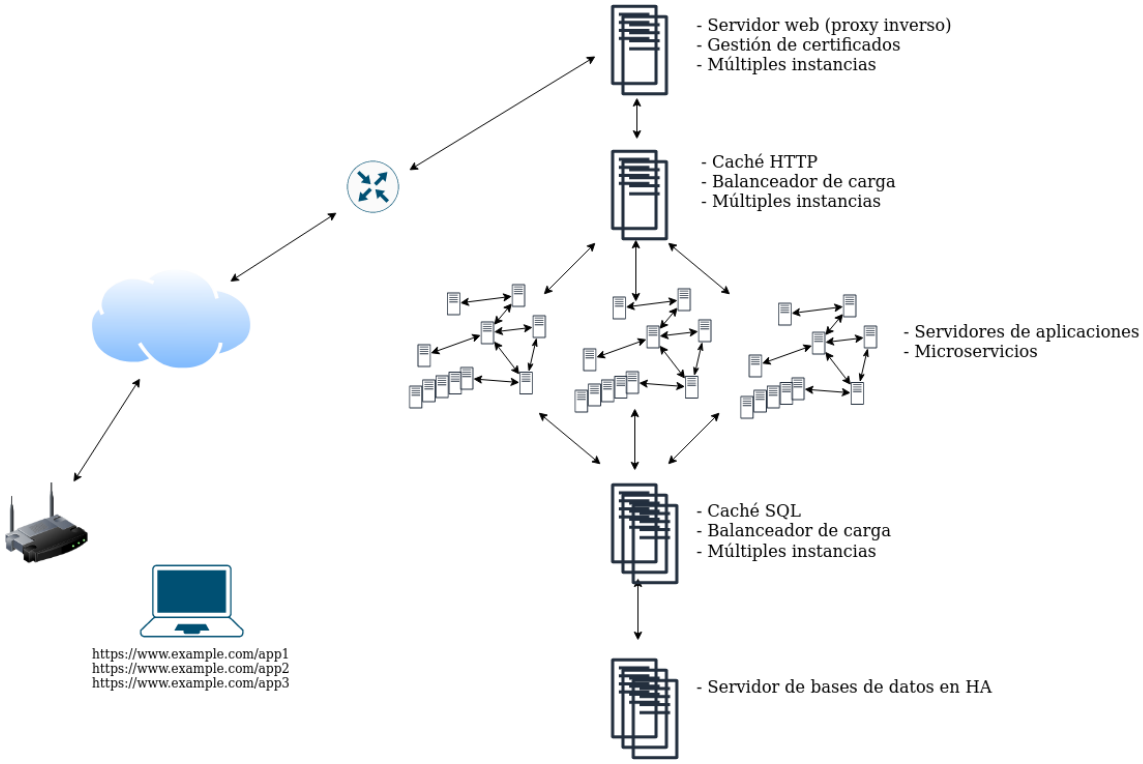


Imagen de elaboración propia (CC BY-NC-SA)

Paso 11. Todo en microservicios

O podríamos tener todo definido internamente en microservicios, tanto los componentes de cada aplicación, como los componentes auxiliares.

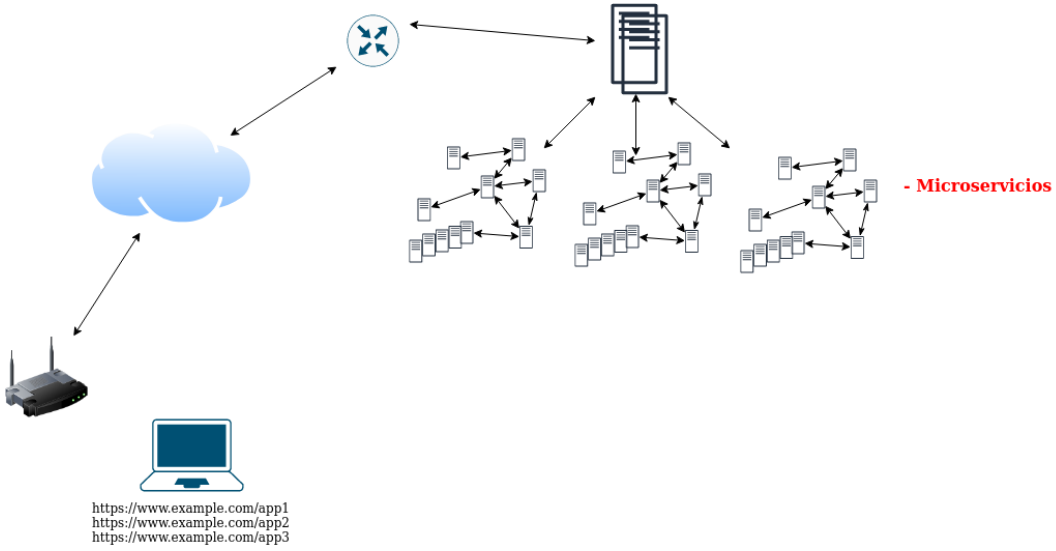


Imagen de elaboración propia (CC BY-NC-SA)

Contenedores

En parte por lo que hemos explicado aquí, y en parte por las ventajas que proporciona en el desarrollo de software y en el rápido despliegue, muchos de los componentes que hemos presentado se ejecutan no sobre máquinas virtuales o físicas, sino que lo hacen sobre contenedores de aplicaciones tipo docker (hoy en día se plantean otras alternativas como podman o containerd, pero no vamos a entrar en esa explicación). Docker es capaz de gestionar esos contenedores de forma ágil y rápida, pero no tiene funcionalidad para ejecutar escenarios tan complejos como los anteriores, que además se ejecutarían lógicamente en diferentes nodos físicos o virtuales (que a su vez ejecutarían docker para los componentes de la aplicación).

Conclusión

Esto no son más que un conjunto de componentes y una explicación muy rápida de ellos, el orden y la ubicación de ellos es variable en función del caso de uso, pero en cualquier caso queríamos presentarlos aquí para tener una visión global de hacia dónde vamos. Algo que claramente podemos ver es que la gestión de este tipo de aplicaciones se convierte pronto en algo muy complejo, por lo que necesitamos apoyarnos en algún software que controle y gestione de forma adecuada estos sistemas tan complejos.

Vídeo: Implantación de aplicaciones web en contenedores

https://www.youtube.com/embed/vYB1hNXde0o

Vídeo: Implantación de aplicaciones web en contenedores

Docker

Docker es una empresa (Docker Inc.) que desarrolla un software con el mismo nombre, de forma más concreta el software denominado (docker engine), que ha supuesto una revolución en el desarrollo de software, muy ligado al uso de contenedores de aplicaciones, a las aplicaciones web y al desarrollo ágil.

Docker permite gestionar contenedores a alto nivel, proporcionando todas las capas y funcionalidad adicional y, lo más importante de todo, es que proporciona un nuevo paradigma en la forma de distribuir las aplicaciones, ya que se crean imágenes en contenedores que se distribuyen, de manera que el contenedor que se ha desarrollado es idéntico al que se utiliza en producción y deja de instalarse la aplicación de forma tradicional.

Componentes de docker

Docker engine tiene los componentes que a *grosso modo* se presentan a continuación:

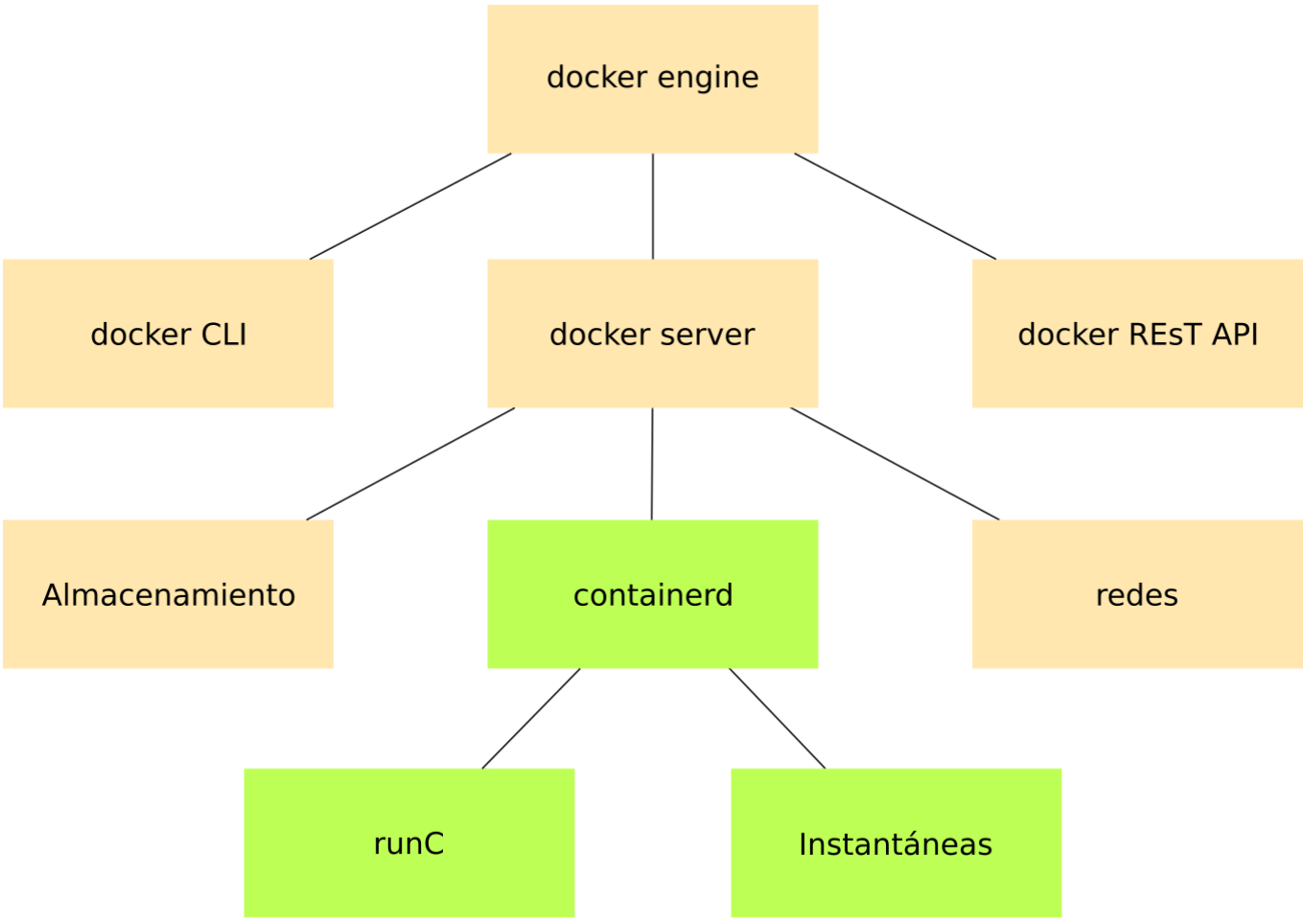


Imagen de elaboración propia (CC BY-NC-SA)

En la imagen se han destacado los componentes que son relevantes desde el punto de vista de este curso, ya que como veremos más adelante, docker podría ser un componente esencial de Kubernetes, pero realmente no lo es completo, solo containerd y los elementos que éste proporciona lo son, ya que k8s utiliza su propia API, su propia línea de comandos y gestiona el almacenamiento y las redes de forma independiente a docker.

Evolución del proyecto docker

Docker tuvo un enorme éxito y una gran repercusión, pero la empresa que lo desarrolla siempre se ha movido en el dilema de cómo sacar rendimiento económico a su software, que al ser desarrollado bajo licencia libre, no proporciona beneficio como tal. Este dilema se ha tratado de resolver con modificaciones en la licencia o con doble licenciamiento (docker CE y docker EE en estos momentos), pero esto a su vez ha propiciado que otras empresas desarrollasen alternativas a docker para no depender en el futuro de una empresa sin un modelo de negocio claro y ante posibles modificaciones de la licencia libre de docker.

Los cambios más significativos que han ocurrido en docker se enumeran a continuación:

- Moby Docker engine se desarrolla ahora como proyecto de software libre independiente de Docker Inc. denominándose Moby. De este proyecto se surten las distribuciones de linux para desarrollar los paquetes docker.io
- Docker Engine Versión desarrollada por Docker Inc.
- runC Componente que ejecuta los contenedores a bajo nivel. Actualmente desarrollado por OCI
- containerd Componente que ejecuta los contenedores e interactúa con las imágenes. Actualmente desarrollado por la CNCF.

Limitaciones de docker (docker engine)

Docker (docker engine) gestiona completamente la ejecución de un contenedor en un determinado nodo a partir de una imagen, pero no proporciona toda la funcionalidad que necesitamos para ejecutar aplicaciones en entornos en producción.

Existen diferentes preguntas que nos podemos hacer acerca de esto :

- ¿Qué hacemos con los cambios entre versiones?

- ¿Cómo hacemos los cambios en producción?
- ¿Cómo se balancea la carga entre múltiples contenedores iguales?
- ¿Cómo se conectan contenedores que se ejecuten en diferentes demonios de docker?
- ¿Se puede hacer una actualización de una aplicación sin interrupción?
- ¿Se puede variar a demanda el número de réplicas de un determinado contenedor?
- ¿Es posible mover la carga entre diferentes nodos?

Las respuestas a estas preguntas no pueden venir de docker engine, ya que no es un software desarrollado para eso, tiene que venir de algún software que pueda utilizar docker o parte de él y que sea capaz de comunicar múltiples nodos para proporcionar de forma coordinada estas funcionalidades. Ese software se conoce de forma genérica como **orquestador de contenedores**.

Vídeo: Docker

https://www.youtube.com/embed/UdPsknw30OE

Vídeo: Docker

Orquestadores de contenedores

Tan pronto como se fue extendiendo el uso de docker para el desarrollo de aplicaciones web, surgió la necesidad de desarrollar software de orquestadores de contenedores para gestionar de forma coordinada múltiples nodos en los que se estuvieran ejecutando contenedores y para proporcionar funcionalidad no ofrecida por docker engine y que es necesaria en la puesta en producción de la aplicación.

Docker swarm

Lógicamente la propia empresa Docker Inc. comenzó pronto el desarrollo de su orquestador que extendía la funcionalidad de docker y creó el proyecto Swarm (enjambre), aunque en las últimas versiones de docker-engine, ya se incluye swarm como un componente y no es necesario instalarlo de forma separada.

Actualmente se considera que docker swarm es una solución de orquestación de contenedores sencilla y que es adecuada para determinados entornos no muy exigentes, pero no puede competir con Kubernetes en grandes entornos expuestos a Internet. Su desarrollo continúa, pero ya no como un competidor de Kubernetes, de hecho la propia Docker Inc. publicita Kubernetes como un producto sobre el que ofrecen servicios.

Apache Mesos

Apache Mesos es un proyecto de software libre que proporciona un orquestador de contenedores. Este proyecto actualmente está bajo el paraguas de la Apache Software Foundation, pero originalmente fue desarrollado por la empresa Mesosphere, hoy renombrada a D2iQ. Mesos ha ido derivando de un competidor de Kubernetes hacia un software que proporcione soluciones más específicas, como DC/OS.

Hashicorp Nomad

Nomad es un proyecto de la prestigiosa empresa Hashicorp (Vagrant, Terraform, etc.) y su idea es ser un software de orquestación más simple, que se centre en la gestión del cluster de nodos y la ejecución de contenedores en ellos, pero sin proporcionar todo el resto de

funcionalidad adicional, por lo que Nomad se utiliza junto a otro software cuando se pone en producción.

Vídeo: Orquestadores de contenedores

https://www.youtube.com/embed/XYh8PSQY5gs

Vídeo: Orquestadores de contenedores

El proyecto Kubernetes

El proyecto Kubernetes lo inicia Google en 2014 como un software (libre) para orquestar contenedores. En aquel momento había varios proyectos de software que querían extender las posibilidades del uso de contenedores de aplicaciones tipo docker a entornos en producción, lo que de forma genérica se conoce como orquestadores de contenedores. A diferencia del resto, Kubernetes no es un proyecto que se desarrolla desde cero, sino que aprovecha todo el conocimiento que tenía Google con el uso de la herramienta interna Borg, de manera que cuando se hace pública la primera versión de Kubernetes, ya era un software con muchas funcionalidades.

Un proyecto se convierte en software libre cuando utiliza una licencia libre, pero otro aspecto importante es la gobernanza del proyecto, es decir, si el desarrollo es abierto o no, si las decisiones sobre las nuevas funcionalidades las toma una empresa o se consensúan, etc. Si un proyecto de software libre lo inicia una única empresa, siempre existe la desconfianza de que ese proyecto vaya a ir encaminado a beneficiar a esa empresa. En este caso, la empresa en cuestión era un gigante como Google, por lo que aunque el proyecto era muy interesante, existía cierto recelo de gran parte del sector inicialmente. Para conseguir que una parte importante del sector se sumase al proyecto, Google tomó la decisión de desvincularse del mismo y ceder el control a la Cloud Native Compute Foundation (CNCF), por lo que Kubernetes es un proyecto de software libre de fundación, en el que se admiten contribuciones de forma abierta y donde las reglas de la gobernanza recaen sobre los miembros de la fundación, normalmente un conjunto amplio de grandes empresas del sector. Es decir, aunque hoy en día hay quien habla de Kubernetes como el software de orquestación de contenedores de Google, esto es un error, es un proyecto que gestiona desde hace años la CNCF, a la que ni siquiera pertenece Google.

¿Qué es Kubernetes?

Kubernetes es un software pensado para gestionar completamente el despliegue de aplicaciones sobre contenedores, realizando este despliegue de forma completamente automática y poniendo un gran énfasis en la escalabilidad de la aplicación, así como el control total del ciclo de vida. Por destacar algunos de los puntos más importantes de Kubernetes, podríamos decir:

- Despliega aplicaciones rápidamente
- Escala las aplicaciones al vuelo
- Integra cambios sin interrupciones
- Permite limitar los recursos a utilizar

Kubernetes está centrado en la puesta en **producción** de contenedores y por su gestión es indicada para administradores de sistemas y personal de equipos de operaciones. Por otra parte, afecta también a los desarrolladores, ya que las aplicaciones deben adaptarse para poder desplegarse en Kubernetes.

Características principales

Kubernetes surge como un software para desplegar aplicaciones sobre contenedores que utilicen infraestructura en nube (pública, privada o híbrida). Aunque puede desplegarse también en entornos más tradicionales como servidores físicos o virtuales, no es su "entorno natural".

Kubernetes es extensible, por lo que cuenta con gran cantidad de módulos, plugins, etc.

El nombre del proyecto proviene de una palabra de griego antiguo que significa timonel y habitualmente se escribe de forma abreviada como k8s.

Características del software

Kubernetes está desarrollado en el lenguaje Go como diversas aplicaciones de este sector. La primera versión de Kubernetes se publicó el 7 de junio de 2014, aunque la más antigua disponible en el repositorio es la v0.2, de septiembre de 2014.

La licencia utilizada en Kubernetes es la Apache License v2.0, licencia de software libre permisiva, muy utilizada últimamente en proyectos de fundación en los que están involucrados empresas, ya que no se trata de una licencia copyleft, que no permitiría su inclusión en software que no sea libre, mientras que la licencia Apache sí lo permite en determinadas circunstancias.

El código de Kubernetes se gestiona a través de Github en cuyo repositorio se puede ver la gran cantidad de código desarrollado en estos años (más de 100000 "commits") y las miles de personas que han participado en mayor o menor medida. La última versión de Kubernetes en el momento de escribir esta documentación es la 1.23 y el proyecto actualmente está publicando dos o tres versiones nuevas cada año.

En cualquier caso la versión de Kubernetes no es algo esencial para los contenidos de este curso, porque se van a tratar los elementos básicos, que ya están muy establecidos y, salvo algún detalle menor, se puede realizar este curso al completo con una versión de Kubernetes diferente a la utilizada para la documentación.

El ecosistema

De entre todas las opciones de orquestadores de contenedores disponibles, hoy se considera que la opción preferida en la mayor parte de los casos es k8s y se ha desarrollado un enorme ecosistema de aplicaciones alrededor que proporcionan algunas funcionalidades que no tiene k8s o que de alguna forma utiliza o se pueden integrar de diferente forma con k8s. Este ecosistema de aplicaciones está actualmente en plena "ebullición" y es posible que en unos años algunos de esos proyectos se estabilicen y otros desaparezcan, ya que en muchos casos solapan unos con otros.

https://landscape.cncf.io/

Vídeo: El proyecto Kubernetes

https://www.youtube.com/embed/MtA74Hc4FAo

Vídeo: El proyecto Kubernetes

Arquitectura básica de Kubernetes

Nodos

k8s es un software que se instala en varios nodos que se gestionan de forma coordinada, es decir, un clúster de nodos. Aunque es posible en casos muy peculiares instalar algunos nodos sobre sistemas Windows, la situación normal es que se trate de un cluster de nodos linux. No es necesario que todos los nodos tengan la misma versión y ni siquiera que sean la misma distribución, aunque en muchos casos sí lo sea por simplicidad en el despliegue y mantenimiento.

Los nodos del clúster pueden ser máquinas físicas o virtuales, pero quizás lo más habitual es que se traten de instancias de nube de infraestructura, es decir, máquinas virtuales ejecutándose en algún proveedor de IaaS (AWS, GCP, OpenStack, etc.)

Se distingue entre dos tipos de nodos:

- Los nodos *master*: Son los que ejecutan los servicios principales de k8s y ordenan a los otros nodos los contenedores que deben ejecutar. Como el uso del término master es últimamente muy controvertido en los paises de habla inglesa, se está cambiando su denominación por *control plane* node.
- Los nodos *worker*: Son los que reciben las órdenes de los controladores y en los que se ejecutan los contenedores de las aplicaciones.

Componentes de un nodo master

- **kube-apiserver** Gestiona la API de k8s
- **etcd** Almacén clave-valor que guarda la configuración del clúster
- **kube-scheduler** Selecciona el nodo donde ejecutar los contenedores
- **kube-controller-manager** Ejecuta los controladores de k8s
- **docker/rkt/containerd/...** Ejecuta los contenedores que sean necesarios en el controlador
- **cloud-controller-manager** Ejecuta los controladores que interactúan con el proveedor de nube:
  - nodos
  - enrutamiento
  - balanceadores

    - volúmenes

Componentes de un nodo worker

- **kubelet** Controla los Pods asignados a su nodo
- **kube-proxy** Permite la conexión a través de la red
- **docker/rkt/containerd/...** Ejecuta los contenedores
- **supervisord** Monitoriza y controla kubelet y docker

Complementos (addons)

Los elementos anteriores forman la estructura básica de k8s, pero es muy habitual que se proporcione funcionalidad adicional a través de complementos de k8s, que en muchas ocasiones se ejecutan a su vez como contenedores y son gestionados por el propio Kubernetes. Algunos de estos complementos son:

- **Cluster DNS** Proporciona registros DNS para los servicios de k8s. Normalmente a través de CoreDNS
- **Web UI** Interfaz web para el manejo de k8s
- **Container Resource Monitoring** Recoge métricas de forma centralizada. Múltiples opciones: prometheus, sysdig
- **Cluster-level Logging** Almacena y gestiona los logs de los contenedores

Esquema de nodos y componentes

Se crea un cluster de k8s en los que algunos nodos actúan como master (normalmente se crea un conjunto impar de nodos master que proporcionen alta disponibilidad) y el resto actúa como worker en los que se ejecutan los contenedores de las aplicaciones. Los nodos se comunican entre sí a través de una red que proporciona la capa de infraestructura y se crea una red para la comunicación de los contenedores, que suele ser una red de tipo overlay.

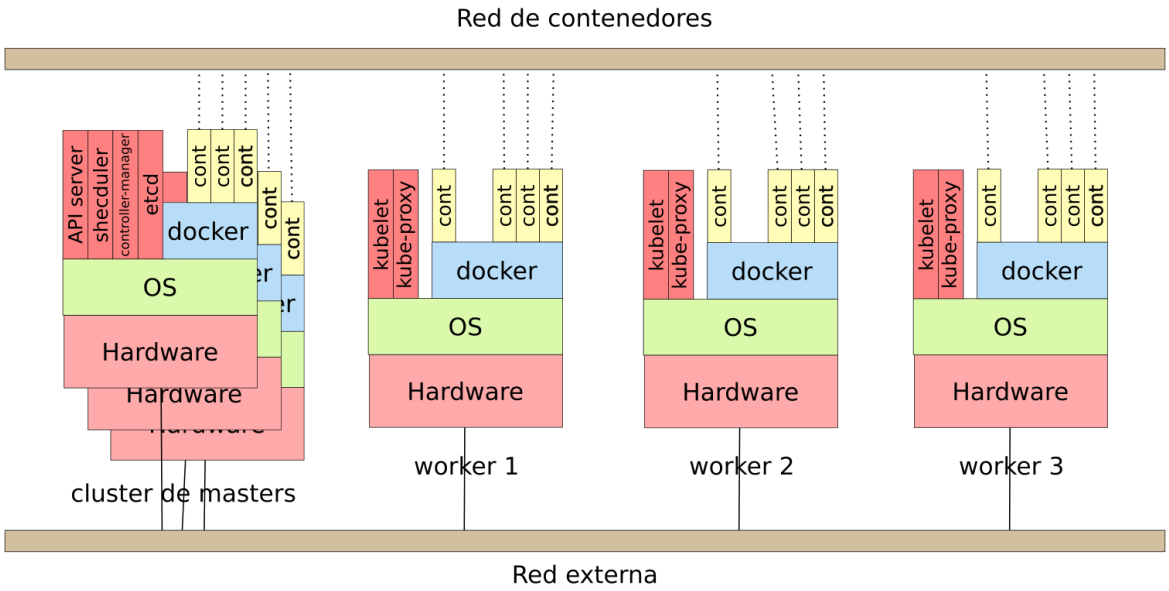


Imagen de elaboración propia (CC BY-NC-SA)

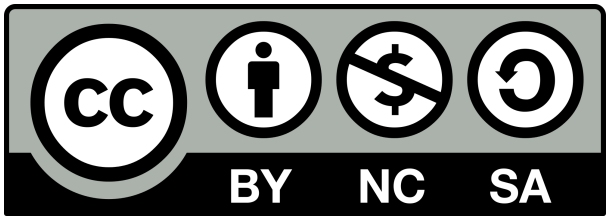
Vídeo: Arquitectura básica de Kubernetes

https://www.youtube.com/embed/mF-OGBbA57k

Vídeo: Arquitectura básica de Kubernetes

Créditos

Materiales desarrollados por:
Alberto Molina Coballes
José Domingo Muñoz Rodríguez
Propiedad de la Consejería de Educación y Deporte de la Junta de Andalucía



2021