1. **Que es el protocolo HTTP:**

<https://developer.mozilla.org/es/docs/Web/HTTP/Overview>

HTTP, de sus siglas en inglés: "Hypertext Transfer Protocol", es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos, como pueden ser documentos [HTML](https://developer.mozilla.org/es/docs/Glossary/HTML). Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor, esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (el cliente), normalmente un navegador Web. Así, una página web completa resulta de la unión de distintos sub-documentos recibidos, como, por ejemplo: un documento que especifique el estilo de maquetación de la página web ([CSS](https://developer.mozilla.org/es/docs/Glossary/CSS)), el texto, las imágenes, vídeos, scripts, etc...

Clientes y servidores se comunican intercambiando mensajes individuales (en contraposición a las comunicaciones que utilizan flujos continuos de datos). Los mensajes que envía el cliente, normalmente un navegador Web, se llaman peticiones, y los mensajes enviados por el servidor se llaman respuestas.

Diseñado a principios de la década de 1990, [HTTP](https://developer.mozilla.org/es/docs/Glossary/HTTP) es un protocolo ampliable, que ha ido evolucionando con el tiempo. Es lo que se conoce como un protocolo de la capa de aplicación, y se transmite sobre el protocolo [TCP](https://developer.mozilla.org/es/docs/Glossary/TCP), o el protocolo encriptado [TLS (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/TLS), aunque teóricamente podría usarse cualquier otro protocolo fiable. Gracias a que es un protocolo capaz de ampliarse, se usa no solo para transmitir documentos de hipertexto ([HTML](https://developer.mozilla.org/es/docs/Glossary/HTML)), si no que además, se usa para transmitir imágenes o vídeos, o enviar datos o contenido a los servidores, como en el caso de los formularios de datos. [HTTP](https://developer.mozilla.org/es/docs/Glossary/HTTP) puede incluso ser utilizado para transmitir partes de documentos, y actualizar páginas Web en el acto.

## [Arquitectura de los sistemas basados en HTTP](https://developer.mozilla.org/es/docs/Web/HTTP/Overview#arquitectura_de_los_sistemas_basados_en_http)

[HTTP](https://developer.mozilla.org/es/docs/Glossary/HTTP) es un protocolo basado en el principio de cliente-servidor: las peticiones son enviadas por una entidad: el agente del usuario (o un proxy a petición de uno). La mayoría de las veces el agente del usuario (cliente) es un navegador Web, pero podría ser cualquier otro programa, como por ejemplo un programa-robot, que explore la Web, para adquirir datos de su estructura y contenido para uso de un buscador de Internet.

Cada petición individual se envía a un servidor, el cuál la gestiona y responde. Entre cada petición y respuesta, hay varios intermediarios, normalmente denominados [proxies (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/Proxy_server" \o "Currently only available in English (US)), los cuales realizan distintas funciones, como: gateways o [caches](https://developer.mozilla.org/es/docs/Glossary/Cache).

En realidad, hay más elementos intermedios, entre un navegador y el servidor que gestiona su petición: hay otros tipos de dispositivos: como routers, modems ... Es gracias a la arquitectura en capas de la Web, que estos intermediarios, son transparentes al navegador y al servidor, ya que [HTTP](https://developer.mozilla.org/es/docs/Glossary/HTTP) se apoya en los protocolos de red y transporte. [HTTP](https://developer.mozilla.org/es/docs/Glossary/HTTP) es un protocolo de aplicación, y por tanto se apoya sobre los anteriores. Aunque para diagnosticar problemas en redes de comunicación, las capas inferiores son irrelevantes para la definición del protocolo [HTTP](https://developer.mozilla.org/es/docs/Glossary/HTTP) .

### [Cliente: el agente del usuario](https://developer.mozilla.org/es/docs/Web/HTTP/Overview#cliente_el_agente_del_usuario)

El agente del usuario, es cualquier herramienta que actué en representación del usuario. Esta función es realizada en la mayor parte de los casos por un navegador Web. Hay excepciones, como el caso de programas específicamente usados por desarrolladores para desarrollar y depurar sus aplicaciones.

El navegador es **siempre** el que inicia una comunicación (petición), y el servidor nunca la comienza (hay algunos mecanismos que permiten esto, pero no son muy habituales).

Para poder mostrar una página Web, el navegador envía una petición de documento [HTML](https://developer.mozilla.org/es/docs/Glossary/HTML) al servidor. Entonces procesa este documento, y envía más peticiones para solicitar scripts, hojas de estilo ([CSS](https://developer.mozilla.org/es/docs/Glossary/CSS)), y otros datos que necesite (normalmente vídeos y/o imágenes). El navegador, une todos estos documentos y datos, y compone el resultado final: la página Web. Los scripts, los ejecuta también el navegador, y también pueden generar más peticiones de datos en el tiempo, y el navegador, gestionará y actualizará la página Web en consecuencia.

Una página Web, es un documento de hipertexto ([HTTP](https://developer.mozilla.org/es/docs/Glossary/HTTP)), luego habrá partes del texto en la página que puedan ser enlaces (links) que pueden ser activados (normalmente al hacer click sobre ellos) para hacer una petición de una nueva página Web, permitiendo así dirigir su agente de usuario y navegar por la Web. El navegador, traduce esas direcciones en peticiones de HTTP, e interpretara y procesará las respuestas HTTP, para presentar al usuario la página Web que desea.

### [El servidor Web](https://developer.mozilla.org/es/docs/Web/HTTP/Overview#el_servidor_web)

Al otro lado del canal de comunicación, está el servidor, el cual "sirve" los datos que ha pedido el cliente. Un servidor conceptualmente es una unica entidad, aunque puede estar formado por varios elementos, que se reparten la carga de peticiones, (load balancing), u otros programas, que gestionan otros computadores (como cache, bases de datos, servidores de correo electrónico, ...), y que generan parte o todo el documento que ha sido pedido.

Un servidor no tiene que ser necesariamente un único equipo físico, aunque si que varios servidores pueden estar funcionando en un único computador. En el estándar HTTP/1.1 y [Host](https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Host) , pueden incluso compartir la misma dirección de IP.

### [Proxies](https://developer.mozilla.org/es/docs/Web/HTTP/Overview#proxies)

Entre el cliente y el servidor, además existen distintos dispositivos que gestionan los mensajes HTTP. Dada la arquitectura en capas de la Web, la mayoria de estos dispositivos solamente gestionan estos mensajes en los niveles de protocolo inferiores: capa de transporte, capa de red o capa física, siendo así transparentes para la capa de comunicaciones de aplicación del HTTP, además esto aumenta el rendimiento de la comunicación. Aquellos dispositivos, que sí operan procesando la capa de aplicación son conocidos como proxies. Estos pueden ser transparentes, o no (modificando las peticiones que pasan por ellos), y realizan varias funciones:

* caching (la caché puede ser pública o privada, como la caché de un navegador)
* filtrado (como un anti-virus, control parental, ...)
* balanceo de carga de peticiones (para permitir a varios servidores responder a la carga total de peticiones que reciben)
* autentificación (para el control al acceso de recursos y datos)
* registro de eventos (para tener un histórico de los eventos que se producen)

**2.Metodos que implementa:**

<https://developer.mozilla.org/es/docs/Web/HTTP/Methods>

HTTP define un conjunto de **métodos de petición** para indicar la acción que se desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados HTTP verbs. Cada uno de ellos implementan una semántica diferente, pero algunas características similares son compartidas por un grupo de ellos: ej. un request method puede ser [safe](https://developer.mozilla.org/es/docs/Glossary/Safe), [idempotent (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/Idempotent" \o "Currently only available in English (US)), o [cacheable](https://developer.mozilla.org/es/docs/Glossary/Cacheable).

[GET](https://developer.mozilla.org/es/docs/Web/HTTP/Methods/GET)

El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

[HEAD (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD)

El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

[POST](https://developer.mozilla.org/es/docs/Web/HTTP/Methods/POST)

El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

[PUT](https://developer.mozilla.org/es/docs/Web/HTTP/Methods/PUT)

El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

[DELETE](https://developer.mozilla.org/es/docs/Web/HTTP/Methods/DELETE)

El método DELETE borra un recurso en específico.

[CONNECT](https://developer.mozilla.org/es/docs/Web/HTTP/Methods/CONNECT)

El método CONNECT establece un túnel hacia el servidor identificado por el recurso.

[OPTIONS (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS)

El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.

[TRACE](https://developer.mozilla.org/es/docs/Web/HTTP/Methods/TRACE)

El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.

[PATCH](https://developer.mozilla.org/es/docs/Web/HTTP/Methods/PATCH)

El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.

1. **Respuestas codigos y significado**

<https://developer.mozilla.org/es/docs/Web/HTTP/Status>

Los códigos de estado de respuesta HTTP indican si se ha completado satisfactoriamente una solicitud HTTP específica. Las respuestas se agrupan en cinco clases:

1. Respuestas informativas (100–199),
2. Respuestas satisfactorias (200–299),
3. Redirecciones (300–399),
4. Errores de los clientes (400–499),
5. y errores de los servidores (500–599).

Los códigos de estado se definen en la sección 10 de[RFC 2616](https://tools.ietf.org/html/rfc2616#section-10). Puedes obtener las especificaciones actualizadas en [RFC 7231.](https://tools.ietf.org/html/rfc7231#section-6.5.1)

[**Respuestas informativas**](https://developer.mozilla.org/es/docs/Web/HTTP/Status#respuestas_informativas)

[100 Continue](https://developer.mozilla.org/es/docs/Web/HTTP/Status/100)

Esta respuesta provisional indica que todo hasta ahora está bien y que el cliente debe continuar con la solicitud o ignorarla si ya está terminada.

[101 Switching Protocol](https://developer.mozilla.org/es/docs/Web/HTTP/Status/101)

Este código se envía en respuesta a un encabezado de solicitud [Upgrade (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Upgrade" \o "Currently only available in English (US)) por el cliente e indica que el servidor acepta el cambio de protocolo propuesto por el agente de usuario.

[102 Processing (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/102) ([WebDAV (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV" \o "Currently only available in English (US)))

Este código indica que el servidor ha recibido la solicitud y aún se encuentra procesandola, por lo que no hay respuesta disponible.

[103 Early Hints (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/103)

Este código de estado está pensado principalmente para ser usado con el encabezado [Link](https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Link), permitiendo que el agente de usuario empiece a [pre-cargar (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/rel/preload" \o "Currently only available in English (US)) recursos mientras el servidor prepara una respuesta.

[**Respuestas satisfactorias**](https://developer.mozilla.org/es/docs/Web/HTTP/Status#respuestas_satisfactorias)

* GET: El recurso se ha obtenido y se transmite en el cuerpo del mensaje.
* HEAD: Los encabezados de entidad están en el cuerpo del mensaje.
* PUT o POST: El recurso que describe el resultado de la acción se transmite en el cuerpo del mensaje.
* TRACE: El cuerpo del mensaje contiene el mensaje de solicitud recibido por el servidor.

[200 OK](https://developer.mozilla.org/es/docs/Web/HTTP/Status/200)

La solicitud ha tenido éxito. El significado de un éxito varía dependiendo del método HTTP:

[201 Created](https://developer.mozilla.org/es/docs/Web/HTTP/Status/201)

La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado de ello. Ésta es típicamente la respuesta enviada después de una petición PUT.

[202 Accepted](https://developer.mozilla.org/es/docs/Web/HTTP/Status/202)

La solicitud se ha recibido, pero aún no se ha actuado. Es una petición "sin compromiso", lo que significa que no hay manera en HTTP que permite enviar una respuesta asíncrona que indique el resultado del procesamiento de la solicitud. Está pensado para los casos en que otro proceso o servidor maneja la solicitud, o para el procesamiento por lotes.

[203 Non-Authoritative Information](https://developer.mozilla.org/es/docs/Web/HTTP/Status/203)

La petición se ha completado con éxito, pero su contenido no se ha obtenido de la fuente originalmente solicitada, sino que se recoge de una copia local o de un tercero. Excepto esta condición, se debe preferir una respuesta de 200 OK en lugar de esta respuesta.

[204 No Content (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/204)

La petición se ha completado con éxito pero su respuesta no tiene ningún contenido, aunque los encabezados pueden ser útiles. El agente de usuario puede actualizar sus encabezados en caché para este recurso con los nuevos valores.

[205 Reset Content (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/205)

La petición se ha completado con éxito, pero su respuesta no tiene contenidos y además, el agente de usuario tiene que inicializar la página desde la que se realizó la petición, este código es útil por ejemplo para páginas con formularios cuyo contenido debe borrarse después de que el usuario lo envíe.

[206 Partial Content](https://developer.mozilla.org/es/docs/Web/HTTP/Status/206)

La petición servirá parcialmente el contenido solicitado. Esta característica es utilizada por herramientas de descarga como wget para continuar la transferencia de descargas anteriormente interrumpidas, o para dividir una descarga y procesar las partes simultáneamente.

[207 Multi-Status (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/207) ([WebDAV (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV" \o "Currently only available in English (US)))

Una respuesta Multi-Estado transmite información sobre varios recursos en situaciones en las que varios códigos de estado podrían ser apropiados. El cuerpo de la petición es un mensaje XML.

[208 Multi-Status (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/208) ([WebDAV (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV" \o "Currently only available in English (US)))

El listado de elementos DAV ya se notificó previamente, por lo que no se van a volver a listar.

[226 IM Used (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/226) ([HTTP Delta encoding](https://tools.ietf.org/html/rfc3229))

El servidor ha cumplido una petición GET para el recurso y la respuesta es una representación del resultado de una o más manipulaciones de instancia aplicadas a la instancia actual.

[**Redirecciones**](https://developer.mozilla.org/es/docs/Web/HTTP/Status#redirecciones)

[300 Multiple Choice (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/300)

Esta solicitud tiene más de una posible respuesta. User-Agent o el usuario debe escoger uno de ellos. No hay forma estandarizada de seleccionar una de las respuestas.

[301 Moved Permanently (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/301)

Este código de respuesta significa que la URI del recurso solicitado ha sido cambiado. Probablemente una nueva URI sea devuelta en la respuesta.

[302 Found](https://developer.mozilla.org/es/docs/Web/HTTP/Status/302)

Este código de respuesta significa que el recurso de la URI solicitada ha sido cambiado temporalmente. Nuevos cambios en la URI serán agregados en el futuro. Por lo tanto, la misma URI debe ser usada por el cliente en futuras solicitudes.

[303 See Other (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/303)

El servidor envía esta respuesta para dirigir al cliente a un nuevo recurso solicitado a otra dirección usando una petición GET.

[304 Not Modified](https://developer.mozilla.org/es/docs/Web/HTTP/Status/304)

Esta es usada para propósitos de "caché". Le indica al cliente que la respuesta no ha sido modificada. Entonces, el cliente puede continuar usando la misma versión almacenada en su caché.

[305 Use Proxy](https://developer.mozilla.org/es/docs/Web/HTTP/Status#305_use_proxy) Obsoleto

Fue definida en una versión previa de la especificación del protocolo HTTP para indicar que una respuesta solicitada debe ser accedida desde un proxy. Ha quedado obsoleta debido a preocupaciones de seguridad correspondientes a la configuración de un proxy.

[306 unused](https://developer.mozilla.org/es/docs/Web/HTTP/Status#306_unused)

Este código de respuesta ya no es usado más. Actualmente se encuentra reservado. Fue usado en previas versiones de la especificación HTTP1.1.

[307 Temporary Redirect (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/307)

El servidor envía esta respuesta para dirigir al cliente a obtener el recurso solicitado a otra URI con el mismo método que se usó la petición anterior. Tiene la misma semántica que el código de respuesta HTTP 302 Found, con la excepción de que el agente usuario *no debe* cambiar el método HTTP usado: si un POST fue usado en la primera petición, otro POST debe ser usado en la segunda petición.

[308 Permanent Redirect (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/308)

Significa que el recurso ahora se encuentra permanentemente en otra URI, especificada por la respuesta de encabezado HTTP Location:. Tiene la misma semántica que el código de respuesta HTTP 301 Moved Permanently, con la excepción de que el agente usuario *no debe* cambiar el método HTTP usado: si un POST fue usado en la primera petición, otro POST debe ser usado en la segunda petición.

[**Errores de cliente**](https://developer.mozilla.org/es/docs/Web/HTTP/Status#errores_de_cliente)

[400 Bad Request](https://developer.mozilla.org/es/docs/Web/HTTP/Status/400)

Esta respuesta significa que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.

[401 Unauthorized](https://developer.mozilla.org/es/docs/Web/HTTP/Status/401)

Es necesario autenticar para obtener la respuesta solicitada. Esta es similar a 403, pero en este caso, la autenticación es posible.

[402 Payment Required](https://developer.mozilla.org/es/docs/Web/HTTP/Status#402_payment_required)

Este código de respuesta está reservado para futuros usos. El objetivo inicial de crear este código fue para ser utilizado en sistemas digitales de pagos. Sin embargo, no está siendo usado actualmente.

[403 Forbidden](https://developer.mozilla.org/es/docs/Web/HTTP/Status/403)

El cliente no posee los permisos necesarios para cierto contenido, por lo que el servidor está rechazando otorgar una respuesta apropiada.

[404 Not Found](https://developer.mozilla.org/es/docs/Web/HTTP/Status/404)

El servidor no pudo encontrar el contenido solicitado. Este código de respuesta es uno de los más famosos dada su alta ocurrencia en la web.

[405 Method Not Allowed](https://developer.mozilla.org/es/docs/Web/HTTP/Status/405)

El método solicitado es conocido por el servidor pero ha sido deshabilitado y no puede ser utilizado. Los dos métodos obligatorios, GET y HEAD, nunca deben ser deshabilitados y no deberían retornar este código de error.

[406 Not Acceptable (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/406)

Esta respuesta es enviada cuando el servidor, después de aplicar una [negociación de contenido servidor-impulsado (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation#server-driven_negotiation), no encuentra ningún contenido seguido por la criteria dada por el usuario.

[407 Proxy Authentication Required (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/407)

Esto es similar al código 401, pero la autenticación debe estar hecha a partir de un proxy.

[408 Request Timeout](https://developer.mozilla.org/es/docs/Web/HTTP/Status/408)

Esta respuesta es enviada en una conexión inactiva en algunos servidores, incluso sin alguna petición previa por el cliente. Significa que el servidor quiere desconectar esta conexión sin usar. Esta respuesta es muy usada desde algunos navegadores, como Chrome, Firefox 27+, o IE9, usa mecanismos de pre-conexión HTTP para acelerar la navegación. También hay que tener en cuenta que algunos servidores simplemente desconecta la conexión sin enviar este mensaje.

[409 Conflict (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/409)

Esta respuesta puede ser enviada cuando una petición tiene conflicto con el estado actual del servidor.

[410 Gone (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/410)

Esta respuesta puede ser enviada cuando el contenido solicitado ha sido borrado del servidor.

[411 Length Required (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/411)

El servidor rechaza la petición porque el campo de encabezado Content-Length no esta definido y el servidor lo requiere.

[412 Precondition Failed (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/412)

El cliente ha indicado pre-condiciones en sus encabezados la cual el servidor no cumple.

[413 Payload Too Large](https://developer.mozilla.org/es/docs/Web/HTTP/Status/413)

La entidad de petición es más larga que los límites definidos por el servidor; el servidor puede cerrar la conexión o retornar un campo de encabezado Retry-After.

[414 URI Too Long (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/414)

La URI solicitada por el cliente es más larga de lo que el servidor está dispuesto a interpretar.

[415 Unsupported Media Type (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/415)

El formato multimedia de los datos solicitados no está soportado por el servidor, por lo cual el servidor rechaza la solicitud.

[416 Requested Range Not Satisfiable (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/416)

El rango especificado por el campo de encabezado Range en la solicitud no cumple; es posible que el rango está fuera del tamaño de los datos objetivo del URI.

[417 Expectation Failed (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/417)

Significa que la expectativa indicada por el campo de encabezado Expect solicitada no puede ser cumplida por el servidor.

[418 I'm a teapot](https://developer.mozilla.org/es/docs/Web/HTTP/Status/418)

El servidor se rehúsa a intentar hacer café con una tetera.

[421 Misdirected Request (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/421)

La petición fue dirigida a un servidor que no es capaz de producir una respuesta. Esto puede ser enviado por un servidor que no está configurado para producir respuestas por la combinación del esquema y la autoridad que están incluidos en la URI solicitada

[422 Unprocessable Entity (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/422) ([WebDAV (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV" \o "Currently only available in English (US)))

La petición estaba bien formada pero no se pudo seguir debido a errores de semántica.

[423 Locked (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/423) ([WebDAV (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV" \o "Currently only available in English (US)))

El recurso que está siendo accedido está bloqueado.

[424 Failed Dependency (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/424) ([WebDAV (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV" \o "Currently only available in English (US)))

La petición falló debido a una falla de una petición previa.

[426 Upgrade Required (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/426)

El servidor se rehúsa a aplicar la solicitud usando el protocolo actual pero puede estar dispuesto a hacerlo después que el cliente se actualice a un protocolo diferente. El servidor envía un encabezado Upgrade en una respuesta para indicar los protocolos requeridos.

[428 Precondition Required (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/428)

El servidor origen requiere que la solicitud sea condicional. Tiene la intención de prevenir problemas de 'actualización perdida', donde un cliente OBTIENE un estado del recurso, lo modifica, y lo PONE devuelta al servidor, cuando mientras un tercero ha modificado el estado del servidor, llevando a un conflicto.

[429 Too Many Requests (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/429)

El usuario ha enviado demasiadas solicitudes en un periodo de tiempo dado.

[431 Request Header Fields Too Large (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/431)

El servidor no está dispuesto a procesar la solicitud porque los campos de encabezado son demasiado largos. La solicitud PUEDE volver a subirse después de reducir el tamaño de los campos de encabezado solicitados.

[451 Unavailable For Legal Reasons (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/451)

El usuario solicita un recurso ilegal, como alguna página web censurada por algún gobierno.

[**Errores de servidor**](https://developer.mozilla.org/es/docs/Web/HTTP/Status#errores_de_servidor)

[500 Internal Server Error](https://developer.mozilla.org/es/docs/Web/HTTP/Status/500)

El servidor ha encontrado una situación que no sabe cómo manejarla.

[501 Not Implemented (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/501)

El método solicitado no está soportado por el servidor y no puede ser manejado. Los únicos métodos que los servidores requieren soporte (y por lo tanto no deben retornar este código) son GET y HEAD.

[502 Bad Gateway](https://developer.mozilla.org/es/docs/Web/HTTP/Status/502)

Esta respuesta de error significa que el servidor, mientras trabaja como una puerta de enlace para obtener una respuesta necesaria para manejar la petición, obtuvo una respuesta inválida.

[503 Service Unavailable](https://developer.mozilla.org/es/docs/Web/HTTP/Status/503)

El servidor no está listo para manejar la petición. Causas comunes puede ser que el servidor está caído por mantenimiento o está sobrecargado. Hay que tomar en cuenta que junto con esta respuesta, una página usuario-amigable explicando el problema debe ser enviada. Estas respuestas deben ser usadas para condiciones temporales y el encabezado HTTP Retry-After: debería, si es posible, contener el tiempo estimado antes de la recuperación del servicio. El webmaster debe también cuidar los encabezados relacionados al caché que son enviados junto a esta respuesta, ya que estas respuestas de condición temporal deben usualmente no estar en el caché.

[504 Gateway Timeout](https://developer.mozilla.org/es/docs/Web/HTTP/Status/504)

Esta respuesta de error es dada cuando el servidor está actuando como una puerta de enlace y no puede obtener una respuesta a tiempo.

[505 HTTP Version Not Supported](https://developer.mozilla.org/es/docs/Web/HTTP/Status/505)

La versión de HTTP usada en la petición no está soportada por el servidor.

[506 Variant Also Negotiates (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/506)

El servidor tiene un error de configuración interna: negociación de contenido transparente para la petición resulta en una referencia circular.

[507 Insufficient Storage (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/507)

El servidor tiene un error de configuración interna: la variable de recurso escogida está configurada para acoplar la negociación de contenido transparente misma, y no es por lo tanto un punto final adecuado para el proceso de negociación.

[508 Loop Detected (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/508) ([WebDAV (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV" \o "Currently only available in English (US)))

El servidor detectó un ciclo infinito mientras procesaba la solicitud.

[510 Not Extended (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/510)

Extensiones adicionales para la solicitud son requeridas para que el servidor las cumpla.

[511 Network Authentication Required (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/511)

El código de estado 511 indica que el cliente necesita autenticar para obtener acceso a la red.

1. **Que es un formato JSON**

<https://www.ibm.com/docs/es/baw/20.x?topic=formats-javascript-object-notation-json-format>

Se describe el formato JSON (JavaScript Object Notation).

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos. JSON es de fácil lectura y escritura para los usuarios. JSON es fácil de analizar y generar por parte de las máquinas. JSON se basa en un subconjunto del lenguaje de programación JavaScript, Estándar ECMA-262 3a Edición - Diciembre de 1999. JSON es un formato de texto completamente independiente del lenguaje, pero que utiliza convenios que resultan familiares a los programadores de lenguajes de la familia C, incluidos C, C++, C#, Java™, JavaScript, Perl, Python y mucho otros. Estas características hacen de JSON un lenguaje de intercambio de datos ideal. Encontrará más detalles acerca del formato JSON en la sección [Iniciación a JSON](https://www.ibm.com/links?url=http%3A%2F%2Fwww.json.org%2F).

Nota: el nombre de propiedad de los datos JSON debe coincidir exactamente con el nombre de propiedad del objeto de negocio.

1. **Que es una API**

Las API son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos. Por ejemplo, el sistema de software del instituto de meteorología contiene datos meteorológicos diarios. La aplicación meteorológica de su teléfono “habla” con este sistema a través de las API y le muestra las actualizaciones meteorológicas diarias en su teléfono.

## ¿Qué significa API?

API significa “interfaz de programación de aplicaciones”. En el contexto de las API, la palabra aplicación se refiere a cualquier software con una función distinta. La interfaz puede considerarse como un contrato de servicio entre dos aplicaciones. Este contrato define cómo se comunican entre sí mediante solicitudes y respuestas. La documentación de su API contiene información sobre cómo los desarrolladores deben estructurar esas solicitudes y respuestas.

## ¿Cómo funcionan las API?

La arquitectura de las API suele explicarse en términos de cliente y servidor. La aplicación que envía la solicitud se llama cliente, y la que envía la respuesta se llama servidor. En el ejemplo del tiempo, la base de datos meteorológicos del instituto es el servidor y la aplicación móvil es el cliente.

Las API pueden funcionar de cuatro maneras diferentes, según el momento y el motivo de su creación.

### API de SOAP

Estas API utilizan el protocolo simple de acceso a objetos. El cliente y el servidor intercambian mensajes mediante XML. Se trata de una API menos flexible que era más popular en el pasado.

### API de RPC

Estas API se denominan llamadas a procedimientos remotos. El cliente completa una función (o procedimiento) en el servidor, y el servidor devuelve el resultado al cliente.

### API de WebSocket

La [API de WebSocket](https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api-overview?pg=wianapi&cta=websocketapi) es otro desarrollo moderno de la API web que utiliza objetos JSON para transmitir datos. La API de WebSocket admite la comunicación bidireccional entre las aplicaciones cliente y el servidor. El servidor puede enviar mensajes de devolución de llamada a los clientes conectados, por lo que es más eficiente que la API de REST.

### API de REST

Estas son las API más populares y flexibles que se encuentran en la web actualmente. El cliente envía las solicitudes al servidor como datos. El servidor utiliza esta entrada del cliente para iniciar funciones internas y devuelve los datos de salida al cliente. Veamos las API de REST con más detalle a continuación.

1. **Cuantos tipos de Api existen**

Hay básicamente cuatro tipos de API en lo que se refiere a sus políticas de uso compartido, como veremos a continuación.

API según sus políticas de uso

1. APIs públicas o abiertas

Las APIs públicas también son conocidas como API abiertas y están disponibles para que otros usuarios o desarrolladores las empleen con mínimas restricciones o, en algunos casos incluso, están totalmente accesibles.

2. APIs privadas o internas

Las APIs privadas o internas están ocultas de los usuarios externos y se exponen únicamente para los sistemas internos de una organización. Se emplean para el desarrollo interno de la empresa, optimizando la productividad y la reutilización de servicios.

3. APIs de aliados comerciales

Las APIs de aliados comerciales son aquellas que se exponen entre los miembros de una alianza comercial. Como no están disponibles para todos, se necesita una autorización especial para usarlas.

4. APIs compuestas

Las APIs compuestas utilizan distintos datos o diversas APIs de servicio y permiten que los desarrolladores puedan acceder a varios terminales.

Asimismo, podemos también dividir las APIs en cuatro según lo que ofrecen o casos de uso, como verás ahora.

El término API significa Application Programming Interface y se basa en un conjunto de definiciones y protocolos que tienen el propósito de integrar sistemas y facilitar la comunicación entre aplicaciones de software según una serie de reglas.

La transformación digital ha permitido que personas y organizaciones tengan acceso a miles de aplicaciones e interfaces con el propósito de simplificar sus rutinas y procesos diarios incluso de forma integrada.

Y las APIs están diseñadas justamente para integrar las diferentes aplicaciones.

¿Quieres saber de qué trata esta tecnología? ¡Sigue leyendo!

¿Qué es API?

API o Application Programming Interface, que en español quiere decir Interfaz de Programación de Aplicaciones, es un conjunto de funciones y procedimientos que permite integrar sistemas, permitiendo que sus funcionalidades puedan ser reutilizadas por otras aplicaciones o software.

Una API sirve para intercambiar datos entre diferentes tipos de software y así automatizar procedimientos y desarrollar nuevas funcionalidades.

¿Cómo funciona?

Una API es una especie de puente que conecta diversos tipos de software o aplicaciones y puede crearse en varios lenguajes de programación. Además de un buen desarrollo, una API debe tener una documentación clara y objetiva para poder facilitar su implementación.

Asimismo, suele utilizarse un formato predefinido de datos para compartir información entre los sistemas con el objetivo de lograr la integración entre ellos. Los más usados son XML (Extensible Markup Language), YAML (originalmente Yet Another Markup Language, pero oficialmente YAML Ain’t Markup Language) y JSON (JavaScript Object Notation) para las aplicaciones web.

También existe un patrón en las APIs web llamado REST (Representational State Transfer), que es un conjunto de reglas y definiciones que permite desarrollar proyectos con interfaces bien definidas.

¿Qué tipos de API hay?

Hay básicamente cuatro tipos de API en lo que se refiere a sus políticas de uso compartido, como veremos a continuación.

API según sus políticas de uso

1. APIs públicas o abiertas

Las APIs públicas también son conocidas como API abiertas y están disponibles para que otros usuarios o desarrolladores las empleen con mínimas restricciones o, en algunos casos incluso, están totalmente accesibles.

2. APIs privadas o internas

Las APIs privadas o internas están ocultas de los usuarios externos y se exponen únicamente para los sistemas internos de una organización. Se emplean para el desarrollo interno de la empresa, optimizando la productividad y la reutilización de servicios.

3. APIs de aliados comerciales

Las APIs de aliados comerciales son aquellas que se exponen entre los miembros de una alianza comercial. Como no están disponibles para todos, se necesita una autorización especial para usarlas.

4. APIs compuestas

Las APIs compuestas utilizan distintos datos o diversas APIs de servicio y permiten que los desarrolladores puedan acceder a varios terminales.

Asimismo, podemos también dividir las APIs en cuatro según lo que ofrecen o casos de uso, como verás ahora.

Tipos de API

APIs según sus casos de uso

1. API de datos

Las APIs de datos proporcionan a varios bancos de datos o proveedores SaaS (Software as a Service o Software como Servicio) acceso CRUD (Create, Read, Update, Delete) a conjuntos de datos subyacentes, permitiendo la comunicación entre una aplicación y un sistema de gestión de bases de datos.

2. API de sistemas operativos

Este grupo de APIs definen cómo las aplicaciones usan los recursos disponibles y servicios del sistema operativo. Por lo que cada OS (Operative System) posee un conjunto de APIs, por ejemplo, Windows API o Linux API tienen el kernel-user space API y kernel internal API.

3. APIs remotas

Este grupo define los estándares de interacción que las aplicaciones tienen en diferentes dispositivos, es decir, un software accede a ciertos recursos ubicados fuera del dispositivo que los solicita, como dice su nombre. Como dos aplicaciones se conectan de forma remota a través de una red, las APIs remotas usan protocolos para lograr la conexión.

4. APIs web

Esta clase de API es la más común, dado que las APIs web proporcionan datos que los dispositivos pueden leer y transferirlos entre sistemas basados en la web o arquitectura cliente-servidor.

1. **Que es una API REST**

Representational State Transfer (REST)

REST es un estilo de arquitectura de software con seis restricciones para crear aplicaciones que funcionen sobre HTTP, sobre todo servicios web.

Es considerado como una alternativa de SOAP, dado que múltiples desarrolladores encuentran dificultades en su uso al tener que escribir grandes cantidades de código para realizar una tarea. Y, por otro lado, REST sigue otra lógica ya que facilita la disponibilidad de datos como recursos.

1. **Que es un monolito**

Una arquitectura monolítica es un estilo de arquitectura de software en el que una aplicación se desarrolla como una sola unidad o componente indivisible, tienen como característica el uso de una base de código única para sus servicios o funcionalidades. En este enfoque, todos los componentes y funcionalidades de la aplicación están interconectados y desplegados juntos como una sola entidad en un entorno de ejecución.

En una arquitectura monolítica, la aplicación se compila y se despliega como un único archivo o un conjunto de archivos interdependientes. Todos los aspectos funcionales, como la lógica de negocio, la interfaz de usuario, el acceso a datos y otros servicios, están empaquetados y ejecutados dentro del mismo proceso.

Los componentes de una arquitectura monolítica se comunican directamente entre sí a través de llamadas a funciones o métodos internos. La aplicación monolítica generalmente se ejecuta en un solo servidor o instancia, lo que significa que no hay una separación física o lógica entre los diferentes componentes de la aplicación.

Este enfoque es común en aplicaciones tradicionales y heredadas, donde todas las funcionalidades están estrechamente acopladas y dependen de una única base de código. Aunque puede ser más fácil de desarrollar y desplegar inicialmente, una arquitectura monolítica puede volverse más compleja y difícil de mantener a medida que la aplicación crece en tamaño y complejidad.

1. **Que son los micro servicios**

“La arquitectura de microservicios tiene como objetivo aislar los distintos componentes de una aplicación, con el fin de que cada uno sea una aplicación por sí misma.”

Los microservicios son una evolución del Service Oriented Architecture o SOA, cuya función se basa en desarrollar servicios independientes para el negocio, estando cada uno de estos asociados o unidos a una misma aplicación.

En contraste con las arquitecturas monolíticas, donde todos los componentes están estrechamente acoplados en una sola unidad, los microservicios promueven la descomposición de la aplicación en servicios más pequeños y aislados, que se comunican entre sí a través de interfaces bien definidas, como APIs.

Cada microservicio es responsable de una funcionalidad específica y puede ser desarrollado utilizando diferentes tecnologías y lenguajes de programación, lo que facilita la adopción de nuevas tecnologías y la elección de herramientas más adecuadas para cada servicio.

1. **Que es Docker y para que se usa**

Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

Docker es un sistema operativo para contenedores. De manera similar a cómo una máquina virtual virtualiza (elimina la necesidad de administrar directamente) el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor.

Docker se instala en cada servidor y proporciona comandos sencillos que puede utilizar para crear, iniciar o detener contenedores.

La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

Es algo auto contenido en sí, que se puede llevar de un lado a otro de forma independiente, es portable.

Dicha aplicación software necesita estar ejecutándose en una máquina, en un ordenador. Pero además, dependiendo del tipo de aplicación, dicho ordenador también necesita tener instaladas una serie de cosas para que la aplicación se ejecute correctamente: cierta versión de Java instalado, un servidor de aplicaciones (p.e tomcat, que es el software que realmente estará ejecutando mi aplicación y haciendo que pueda interactuar con ella).

Docker, me permite meter en un contenedor (“una caja”, algo auto contenido, cerrado) todas aquellas cosas que mi aplicación necesita para ser ejecutada (java, Maven, tomcat…) y la propia aplicación. Así yo me puedo llevar ese contenedor a cualquier máquina que tenga instalado Docker y ejecutar la aplicación sin tener que hacer nada más, ni preocuparme de qué versiones de software tiene instalada esa máquina, de si tiene los elementos necesarios para que funcione mi aplicación , de si son compatibles…

Al ejecutar la aplicación software desde el contenedor de Docker, y dentro de él estarán todas las librerías y cosas que necesita dicha aplicación para funcionar correctamente.

¿Qué beneficios tiene esto?

Docker es una herramienta diseñada para beneficiar tanto a desarrolladores, testers, como administradores de sistemas, en relación a las máquinas, a los entornos en sí donde se ejecutan las aplicaciones software, los procesos de despliegue, etc.

En el caso de los desarrolladores, el uso de Docker hace que puedan centrarse en desarrollar su código sin preocuparse de si dicho código funcionará en la máquina en la que se ejecutará.

Por eso Docker también es muy bueno para el testing, para tener entornos de pruebas. Por un lado, es muy sencillo crear y borrar un contenedor, además de que son muy ligeros, por lo que podemos ejecutar varios contenedores en una misma máquina (donde dicho contenedor tendría el entorno de nuestra aplicación: base de datos, servidor, librerías…). Por otro, un mismo contenedor funcionará en cualquier máquina Linux: un portátil, el ordenador de tu casa, máquinas alojadas en Amazon, tu propio servidor…

Esto además beneficia a la parte de sistemas, ya como los contenedores son más ligeros que las máquinas virtuales, se reduce el número de máquinas necesarias para tener un entorno.

¿Y qué diferencia a Docker respecto a una máquina virtual?

Puede que cuando he hablado de que en un mismo ordenador podemos tener varios contenedores Docker te hayas preguntado: ¿y esto no es lo mismo que una máquina virtual?

Realmente el concepto es algo similar, pero un contenedor no es lo mismo que una máquina virtual. Un contenedor es más ligero, ya que mientras que a una máquina virtual necesitas instalarle un sistema operativo para funcionar, un contenedor de Docker funciona utilizando el sistema operativo que tiene la máquina en la que se ejecuta el contenedor.

Digamos que el contenedor de Docker toma los recursos más básicos, que no cambian de un ordenador a otro del sistema operativo de la máquina en la que se ejecuta. Y los aspectos más específicos del sistema que pueden dar más problemas a la hora de llevar el software de un lado a otro, se meten en el interior del contenedor.

Para que te hagas una idea, el concepto de portabilidad de un contenedor de Docker es algo similar a la máquina virtual de Java.

Que un contenedor Docker tome los aspectos básicos de funcionamiento del sistema operativo de la máquina en la que se ejecuta lo vuelve más ligero que una máquina virtual.

1. **Que son los kubernetes y como se usan**

Kubernetes automatiza las tareas operativas de la administración de contenedores e incluye comandos integrados para implementar aplicaciones, actualizarlas, escalarlas a fin de que se ajusten a tus necesidades, supervisarlas y mucho más. Todo esto facilita la administración de las aplicaciones.

Operaciones automatizadas

Cuenta con comandos integrados para manejar gran parte del trabajo pesado que forma parte de la administración de aplicaciones, lo que te permite automatizar las operaciones diarias. Puedes asegurarte de que las aplicaciones siempre se ejecuten de la manera que planeaste.

### Abstracción de la infraestructura

Kubernetes, se encarga del procesamiento, las herramientas de redes y el almacenamiento en nombre de tus cargas de trabajo. Esto les permite a los desarrolladores enfocarse en las aplicaciones sin preocuparse por el entorno subyacente.

### Supervisión del estado de los servicios

jecuta verificaciones de estado de manera continua en los servicios, reinicia los contenedores con errores o que se detuvieron y solo pone los servicios a disposición de los usuarios una vez confirma que se están ejecutando.

Kubernetes es una plataforma para gestionar aplicaciones en contenedores. Para ello, proporciona una API que permiten controlar y gestionar cómo se despliegan, amplían y organizan las aplicaciones en contenedores. Kubernetes se puede utilizar local o en la nube, y actualmente es la plataforma más conocida para gestionar aplicaciones en contenedores.

## Una de las valiosas ventajas de Kubernetes es su capacidad para escalar las aplicaciones. Kubernetes puede ampliar automáticamente las aplicaciones en función de la utilización de la CPU, el uso de memoria y otras métricas. Esto garantiza que Herramientas de Kubernetes

Kubernetes ofrece una gama de recursos, servicios y herramientas para la gestión de aplicaciones. A continuación se exploran algunas de las opciones más utilizadas.

### ¿Cómo funciona el equilibrio de carga de Kubernetes?

Kubernetes tiene un recurso denominado Ingress, que se utiliza para una variedad de funciones, incluido como equilibrador de carga. El equilibrio de carga mediante entrada permite distribuir el tráfico entre un juego de pods, exponiéndolos como un único servicio. Esto puede mejorar tanto la disponibilidad como el rendimiento de las aplicaciones.

El equilibrador de carga funciona inspeccionando las cabeceras de cada solicitud que recibe. Selecciona un pod según el destino de la solicitud y las reglas definidas. A continuación, reenvía la solicitud a ese pod.

El equilibrador de carga también soporta la comprobación del sistema, lo que permite especificar un juego de criterios que se deben cumplir para un pod antes de que el equilibrador de carga envíe solicitudes a él. Si un pod no cumple los requisitos, el equilibrador de carga dejará de enviar solicitudes.

También puedes utilizar el equilibrador de carga para enrutar el tráfico según la dirección IP de origen de la solicitud. Esto puede resultar útil si limitas el acceso a las aplicaciones desde orígenes específicos.

### ¿Cómo funciona la red de Kubernetes?

La red de Kubernetes funciona mediante la creación de pods y servicios. Un pod es un grupo de uno o más contenedores que se despliegan juntos y comparten un espacio de nombres de red y una dirección IP.

Los contenedores de un pod se pueden comunicar entre sí mediante un host local.

Los servicios se utilizan para exponer uno o más pods al mundo exterior. Los servicios se implementan como equilibradores de carga y pueden equilibrar la carga del tráfico en varios pods.

### ¿Cómo funciona el programador de Kubernetes?

El [programador de Kubernetes](https://www.oracle.com/ar/cloud/cloud-native/container-engine-kubernetes/) es una parte fundamental del sistema Kubernetes. Es responsable de asignar recursos a los pods y de garantizar que se puedan ejecutar correctamente.

El programador funciona asignando a cada pod una prioridad y buscando nodos con recursos suficientes para alojar el pod. Si los recursos no están disponibles, el programador puede asignar un nodo con pods de mayor prioridad. Esto también influye en las diferentes prioridades de los pods para realizar la asignación. En general, al asignar pods a un nodo, los nodos de prioridad más baja se utilizan para minimizar la interrupción y la preferencia. Si no hay nodos con recursos suficientes, el programador esperará hasta que uno esté disponible.

El programador también es responsable de reiniciar los pods que han fallado. Si un pod falla, el programador lo continuará en un nodo diferente. Esto garantiza que el pod siempre tenga acceso a los recursos que necesita para ejecutarse correctamente.

### ¿Cómo funciona la ampliación automática de Kubernetes?

El escalado automático de Kubernetes es una gran función que permite ampliar o reducir automáticamente los pods en función de la utilización de la CPU u otras métricas. La ampliación automática puede ayudar a mantener un número óptimo de pods en el cluster, mejorando el rendimiento y la estabilidad de las aplicaciones.

Hay dos tipos de escala automática de Kubernetes: horizontal y vertical.

La ampliación automática horizontal amplía o reduce los pods agregando o eliminando nodos del cluster. En cambio, la ampliación automática vertical amplía o reduce los pods cambiando los límites de CPU o memoria de pods individuales.

La ampliación automática de Kubernetes se basa en dos conceptos: escalar disparadores y escalar políticas. Una motivación de ampliación es una condición que hace que Kubernetes amplíe o reduzca los pods. Una política de ampliación es una acción que Kubernetes realiza cuando se produce un disparador de escala.

### Cómo funciona DNS de Kubernetes

DNS representa el sistema de nombres de dominio y es un sistema utilizado para traducir los nombres de dominio legibles por el usuario en las direcciones IP numéricas utilizadas por los equipos. Kubernetes utiliza DNS para gestionar sus servicios. Cada servicio de Kubernetes tiene un nombre de DNS único. Al crear un servicio, Kubernetes crea un registro de DNS para ese servicio. El registro DNS contiene la dirección IP del servicio y el número de puerto. Kubernetes utiliza esta información para direccionar el tráfico al servicio.

El nombre DNS de un servicio consta de dos partes:

* Nombre de dominio
* El nombre del servicio

El nombre de dominio es la parte del nombre DNS familiar para todos los servicios del dominio. El nombre del servicio es la parte del nombre DNS exclusiva de cada servicio.

la aplicación esté siempre disponible y satisfaga las necesidades de los usuarios.

1. **Que es terraform**

Terraform es una herramienta de configuración de software diseñada para potenciar la automatización de múltiples procesos a través de conceptos como el de infrastructure as code. A través de su lenguaje permite crear definiciones (llamadas resources) de objetos o recursos de infraestructura.

Su principal objetivo es facilitar la creación de infraestructura de manera declarativa, por ejemplo un centro de datos en servicios como Azure, Amazon Web Services (AWS) o Google Cloud Platform (GCP). Es decir, esta configuración se almacena en la nube y permite automatizar una infraestructura.

Terraform consta de elementos como:

Núcleo Terraform. Consiste en recibir la información y tomar decisiones con ella.

Proveedores. Son las herramientas con las que se crea la infraestructura.

¿Qué se puede hacer con Terraform?

La propia web de Terraform propone varios escenarios en los que se podría usar esta tecnología:

Automatización de la infraestructura digital: servidores, bases de datos, firewall…

Administrar un entorno multinube (AWS, Azure, GCP…) a pesar de sus diferentes interfaces o flujos de trabajo. Esto es ideal cuando el entorno se ha construido antes de implantar Terraform y hay que administrar diferentes entornos cloud. En cualquier caso, también puede administrar una sola aplicación y de un solo proveedor.

Gestionar clústeres de Kubernetes e imágenes de máquinas virtuales. Sobre estas últimas, HCP Packer recopila imágenes de entornos multinube y las puede suministrar para una API.

Las configuraciones se pueden compartir y volver a usar cuando se crea necesario.

1. **Que es Openshift**

es un software que puede utilizarse para la construcción de desarrollo y aplicaciones enfocadas a la descentralización y escalabilidad. Tomando de base a OpenShift se pueden desarrollar e implementar diferentes soluciones, depende de lo que necesite el usuario:

Platform as a Service (PaaS).

Software as a Service (SaaS).

Containers as a Service (CaaS).

No solo es versátil en este sentido, sino que además, permite la gestión completa de los ciclos de vida de las aplicaciones que en él se desarrollan. Es decir, trabajar en el desarrollo, implantación, operaciones y mantenimientos de las aplicaciones.

Características que ofrece OpenShift como software

Al tratarse de una infraestructura que puede ser utilizada como herramienta por desarrolladores de aplicaciones web, agilizando el trabajo en distintos lenguajes de programación, la implantación de aplicaciones es muy sencilla gracias a las características que ofrece OpenShift de Cloud Computing.

Elasticidad: uno de los problemas que a veces encuentran los desarrolladores de apps es la demanda de su aplicación. OpenShift ofrece para estas situaciones la capacidad de escalabilidad sencilla, ofreciendo contenedores para servir la aplicación y llegar a más usuarios.

Pago por uso: como todo producto de una empresa, en este caso de Red Hat, es de pago, pero su metodología es de lo más interesante. Al pagar por OpenShift lo haces en función de los recursos que necesitas. ¿Qué quiere decir esto? Que en el momento en que requieras de mayor cantidad de recursos, el pago por el uso del software aumentará, pero si decides disminuir la cantidad de recursos, el costo final también se reduce. Esto es atractivo para desarrolladores ya que se ajusta a las necesidades.

Trabajo en clúster de servidores: ¿recuerdas que te dijimos que la aplicación estaría alojada en la nube? Aquí hay dos opciones, puedes alojarlo en un servidor externo a la nube o instalarlo en uno de los servidores del centro de datos.

Conocimiento base de OpenShift: a diferencia de otros software de esta industria, OpenShift no obliga al desarrollador a conocer todas las tecnologías que en él se encuentran. Si bien trabaja con Docker para la gestión de contenedores y Kubernetes para la ejecución de los mismos en los nodos del clúster, el desarrollador no debe conocer dichas tecnologías, lo que le permite centrarse en sus características de Cloud Computing para llevar a cabo su proyecto.

Adaptabilidad al entorno: una de las principales ventajas que ofrece OpenShift es que puedes trabajar en múltiples entornos. Puedes tener un clúster centrado en el desarrollo para probar tu aplicación, mientras que en el flujo de integración continua, puedes seguir trabajando con OpenShift para generar una aplicación implantada, facilitando el trabajo de desarrollo.

Diferentes metodologías de trabajo: el nivel práctico que ofrece a los desarrolladores es de otro nivel. Ya sea que quieran trabajar desde una aplicación web o un cliente de línea de comandos, facilitando la automatización de tareas. Sin embargo, lo más interesante es la API RESTful. Esta permite realizar diferentes aplicaciones junto a la API del software trabajando con el clúster.

1. **Que es CI**

La CI (integración continua) es la práctica de integrar cambios de código en un repositorio múltiples veces al día. La CD tiene dos significados: la entrega continua automatiza las integraciones de código, mientras que la implementación continua entrega automáticamente las versiones finales a los usuarios finales. Las pruebas frecuentes de CI/CD reducen los errores y defectos del código.

La CI es una práctica recomendada de DevOps y es la etapa del ciclo de vida de DevOps en que los desarrolladores registran el código en su repositorio de código compartido, a menudo, múltiples veces al día. Lo ideal es que, cada vez que esto sucede, una herramienta de compilación automatizada verifique el registro o la rama para garantizar que no haya errores y que esté listo para ingresar a producción. El principal beneficio de esto es que los inconvenientes generalmente se detectan en una etapa temprana, antes de que se conviertan en problemas mayores.

La práctica de CI implica integrar pequeños subconjuntos de cambios en un período más corto, en lugar de integrar actualizaciones importantes que toman más tiempo con menos frecuencia. Automatizar los flujos de trabajo para probar, fusionar y verificar los cambios en un repositorio compartido significa que los equipos pueden entregar un código más limpio a un ritmo más rápido. Un código más limpio significa una validación más rápida, versiones más limpias y un canal de desarrollo más eficaz que es más fácil de modificar.

1. **Que es TDD**

El test driven development (TDD) o en español desarrollo guiado por pruebas, es un enfoque de programación que se utiliza durante el desarrollo de software en el que se realizan pruebas unitarias antes de escribir el código. Básicamente, el objetivo consiste en hacer un código limpio, robusto y simple, para ello; es necesario crear en primer lugar, casos de prueba para cada funcionalidad que se quiera desarrollar, se testea, y en caso de detectar algún fallo, se rescribe un código libre de errores.

De esta manera, se pretende que el desarrollo llegue a ser más rápido (independientemente de lo que se pueda llegar a pensar inicialmente por el tiempo destinado a este propósito), ¿Por qué? Básicamente porque el enfoque TDD pretende eliminar la duplicación de código y dejar que los desarrolladores sólo escriban un código nuevo, en caso de que las pruebas fallen, subsanando los incontables errores que puedan producirse en miles de líneas de código, además de permitir generar una documentación mucho más transparente y que consecuentemente se cree una red de seguridad ante cualquier modificación.

¿En qué se diferencia un TDD de otros test de programación?

Existen muchos tipos de pruebas de software, porque cada una trata de cubrir una serie de requisitos y necesidades en la que podemos encontrarnos: test de aceptación, de integración, usabilidad, etc.. Sin embargo, la característica principal que define al TDD VS otros es que se basa en test de pruebas que guían el proceso y no se realizan en un momento posterior.

¿Cuáles son las fases de un TDD?

Realizar un proceso de Test Driven Development consta de 3 fases:

Desarrollo y Escritura de la prueba: En este paso, el desarrollador debe definir de forma clara los requerimientos que se llevarán a cabo para completar la escritura del código y que se usarán para codificar y validar las diferentes pruebas que se realicen. Por normal general, este proceso de escritura contendrá errores ya que aún no se ha realizado la implementación necesaria para su validación. También se le conoce como fase red porque los errores registrados se mostrarán con este color.

Validación de las pruebas: Una vez validadas las pruebas establecidas podemos decir que se cumplen los requisitos establecidos. Durante esta fase no se contemplan buenas prácticas, sino que simplemente se verifica que los test propuestos se cumplen, por ello se le conoce como fase green ya que este color es el más representativo para este proceso.

Refactorización: En esta parte, se revisará el código escrito con el objetivo de verificar que cumpla con las buenas prácticas y se verifique que el código sea limpio.

Ventajas de trabajar un Test Driven Development:

Reducción de Errores

Detección de requisitos no especificados

Eliminación de código duplicado

Menor coste de mantenimiento

Menor redundancia

Mayor productividad

Mayor documentación

Potenciamiento de metodología ágil

Escalabilidad de producción

Calidad del código

1. **Que es DEVOPS**

DevOps es un marco de trabajo y una filosofía en constante evolución que promueve un mejor desarrollo de aplicaciones en menos tiempo y la rápida publicación de nuevas o revisadas funciones de software o productos para los clientes.

Con DevOps se promueve una comunicación continua más fluida, la colaboración, la integración, la visibilidad y la transparencia entre equipos de desarrollo de aplicaciones (Dev) y sus homólogos en operaciones tecnológicas (Ops).

Esta relación estrecha entre «Dev» y «Ops» se extiende a cada una de las fases del ciclo de vida de DevOps: desde la planificación inicial del software a las fases de codificación, compilación, pruebas y publicación, y en la puesta en marcha, las operaciones y la supervisión continua. Esta relación impulsa un bucle de retroalimentación continua con los clientes sobre las mejoras, el desarrollo, las pruebas y la puesta en marcha. Uno de los resultados de todos estos esfuerzos puede ser la publicación continua y más rápida de las adiciones y los cambios que se necesitan en las funciones.

Algunas personas agrupan los objetivos de DevOps en cuatro categorías: cultura, automatización, medición y uso compartido (CAMS, por sus siglas en inglés), y las herramientas de DevOps pueden ayudar en estas tareas. Con estas herramientas, los flujos de trabajo de desarrollo y operaciones se convierten en tareas más optimizadas y colaborativas al automatizar tareas que antes eran manuales, estáticas o que llevaban mucho tiempo, y que son necesarias para la integración, el desarrollo, las pruebas, la puesta en marcha o la supervisión.

Además de los esfuerzos por romper las barreras de comunicación y fomentar la colaboración entre los equipos de desarrollo y operaciones tecnológicas, uno de los principales valores de DevOps es lograr la satisfacción del cliente y prestar sus servicios en menos tiempo. DevOps también se ha creado para impulsar la innovación empresarial y ser el motor de continuas mejoras en los procesos.

La práctica de DevOps propicia que cada empresa se ponga como objetivo ofrecer un mejor servicio cada vez, en menos tiempo, de mejor calidad y con mayor seguridad a sus clientes finales; por ejemplo, con actualizaciones, funciones o versiones de producto más frecuentes. Puede reflejarse en la rapidez con la que llega al cliente una nueva versión del producto o una nueva función manteniendo los mismos niveles de calidad y seguridad, o en el poco tiempo que se necesita para identificar un problema o un error y, a continuación, solucionarlo y volver a publicar una versión corregida.

Sin duda, todo este trabajo de DevOps se sustenta en una infraestructura subyacente con un rendimiento, una disponibilidad y una fiabilidad fluida y sin interrupciones del software, el cual se desarrolla y se prueba en primer lugar y, luego, se lanza a la fase de producción.

MÉTODOS DE DEVOPS

Existen varios métodos de DevOps comunes que las organizaciones usan para acelerar y mejorar el desarrollo y las publicaciones de productos. Normalmente se presentan como prácticas y metodologías de desarrollo de software. Entre los más populares están Scrum, Kanban y Agile:

Scrum. Scrum define la forma en la que los miembros de un equipo deben colaborar para conseguir entre todos acelerar los proyectos de desarrollo y control de calidad. Las prácticas de Scrum incluyen flujos de trabajo principales y terminología específica (sprints, bloques de tiempo, scrum diario [reunión]), y roles designados (Scrum Master, propietario del producto [product owner]).

Kanban. Kanban se originó a partir de las eficiencias que se alcanzaron en la fábrica de Toyota. Kanban prescribe que el estado «en curso» (WIP, del inglés «work in progress») de un proyecto de software debe controlarse en un tablero Kanban.

Agile. Los anteriores métodos de desarrollo de software Agile siguen teniendo una gran influencia en las herramientas y las prácticas de DevOps. Muchos de estos métodos, incluidos Scrum y Kanban, han incorporado elementos de la programación Agile. Algunas de estas prácticas están asociadas a una mayor capacidad de respuesta a los continuos cambios en requisitos y necesidades, los requisitos de documentación en forma de casos prácticos, la realización de reuniones diarias para ponerse al día y la incorporación de comunicación continua con los clientes para conocer sus opiniones. En Agile también se estipulan ciclos de desarrollo de software más cortos en lugar de los tradicionales métodos de desarrollo «en cascada» que se prolongaban en el tiempo.

CADENA DE HERRAMIENTAS DE DEVOPS

Los seguidores de las prácticas de DevOps a menudo incorporan a su «cadena de herramientas» de DevOps particular algunas herramientas que se adaptan perfectamente a estos métodos. El objetivo de estas herramientas es tratar de optimizar, acortar y automatizar las diversas etapas del flujo de trabajo de creación de software (o «canalización»). Muchas de estas herramientas también promueven los postulados principales de DevOps, como son la automatización, la colaboración y la integración entre los equipos de desarrollo y operaciones. A continuación se ofrece un ejemplo de herramientas que se emplean en las diversas etapas del ciclo de DevOps.

Planificación. En esta fase se definen los requisitos y valores empresariales. Algunas herramientas de muestra son Jira o Git, con las cuales se puede hacer un seguimiento de los problemas conocidos y llevar a cabo la gestión de los proyectos.

Codificación. Esta fase implica el diseño del software y la creación del código. Algunas herramientas de muestra son GitHub, GitLab, Bitbucket o Stash.

Compilación. En esta fase se gestionan las versiones y las compilaciones del software, y se utilizan herramientas automatizadas que ayudan a compilar y crear paquetes de código para publicarlos después para la producción. Se utilizan repositorios de código fuente o repositorios de paquetes que también «empaquetan» la infraestructura que se necesita para el lanzamiento del producto. Algunas herramientas de muestra son Docker, Ansible, Puppet, Chef, Gradle, Maven o JFrog Artifactory.

Prueba. Esta fase incluye la realización de pruebas continuas (manuales o automatizadas) para garantizar la calidad de la programación. Algunas herramientas de muestra son JUnit, Codeception, Selenium, Vagrant, TestNG o BlazeMeter.

Puesta en marcha. En esta fase se emplean herramientas que ayudan a gestionar, coordinar, programar y automatizar las tareas de producción de las versiones de productos. Algunas herramientas de muestra son Puppet, Chef, Ansible, Jenkins, Kubernetes, OpenShift, OpenStack, Docker o Jira.

Funcionamiento. En esta fase se gestiona el software durante su producción. Algunas herramientas de muestra son Ansible, Puppet, PowerShell, Chef, Salt o Otter.

Supervisión. En esta fase se identifica y recopila información sobre problemas que surgen en una versión de software específica que se encuentra en producción. Algunas herramientas de muestra son New Relic, Datadog, Grafana, Wireshark, Splunk, Nagios o Slack.

PRÁCTICAS DE DEVOPS

Las prácticas de DevOps son un reflejo de la idea de automatización y mejora continuas, y muchas de ellas se centran en una o en varias fases del ciclo de desarrollo. Estas prácticas incluyen lo siguiente:

Desarrollo continuo. Esta práctica abarca las fases de planificación y codificación del ciclo de DevOps. Puede incluir también mecanismos de control de versiones.

Realización de pruebas continuas. Esta práctica incorpora continuas pruebas de código automatizadas y programadas con antelación que se realizan a medida que el código de aplicación se está creando o actualizando. Gracias a estas pruebas, el código pasa antes a la fase de producción.

Integración continua (CI). En esta práctica se combinan herramientas de gestión de configuración (CM) con otras herramientas de pruebas y desarrollo para saber qué cantidad del código que se está creando está listo para pasar a producción. Para ello, debe existir un intercambio fluido de información entre las fases de prueba y de desarrollo que permita identificar y resolver con rapidez problemas en el código.

Entrega continua. Esta práctica automatiza la introducción de cambios en el código para pasar a un entorno de preproducción o de almacenamiento provisional tras la fase de pruebas. Un miembro de la plantilla podría entonces decidir si es conveniente promover estos cambios de código a la fase de producción.

Puesta en marcha continua (CD). Al igual que sucede con la entrega continua, esta práctica automatiza el lanzamiento de código nuevo o modificado a la fase de producción. Una empresa que pone en práctica la puesta en marcha continua podría publicar cambios en código o funciones varias veces al día. Las tecnologías de contenedor, como Docker y Kubernetes, hacen posible esta fase de puesta en marcha continua al ayudar a mantener la coherencia del código entre los diferentes entornos y plataformas de puesta en marcha.

Supervisión continua. Esta práctica implica la supervisión continua del código en la fase de producción y la infraestructura subyacente que la sustenta. A través de un bucle de retroalimentación en el que se notifican errores o problemas, este podría volver a la fase de desarrollo.

Infraestructura como código. Esta práctica se puede utilizar durante varias fases de DevOps para automatizar el aprovisionamiento de la infraestructura que se necesita para publicar el software. Los desarrolladores añaden «código» de infraestructura procedente de las herramientas de desarrollo actuales. Por ejemplo, los desarrolladores podrían crear un volumen de almacenamiento bajo demanda desde Docker, Kubernetes u OpenShift. Gracias a esta práctica, los equipos de operaciones también pueden supervisar las configuraciones de entorno, registrar los cambios y simplificar la reversión de las configuraciones.

VENTAJAS DE DEVOPS

Los partidarios de DevOps describen varias ventajas técnicas y empresariales con las que, en última instancia, se consiguen clientes más satisfechos. Entre algunas de las ventajas de DevOps se incluyen las siguientes:

Una mejor y más rápida entrega de productos

Resolución de problemas en menos tiempo y con menor complejidad

Mejor escalabilidad y disponibilidad

Entornos de funcionamiento más estables

Mejor utilización de los recursos

Mayor automatización

Mayor visibilidad de resultados del sistema

Mayor innovación

1. Endpoint