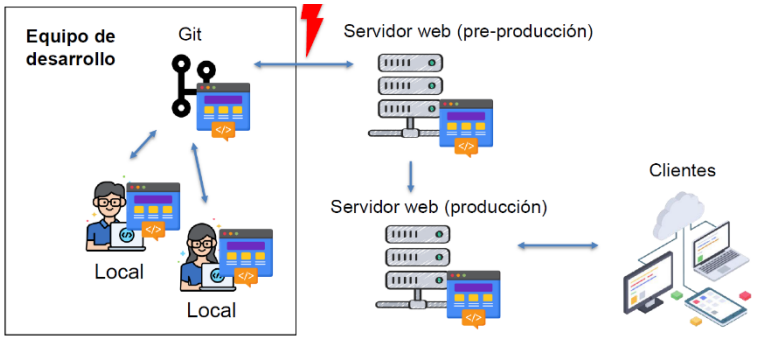
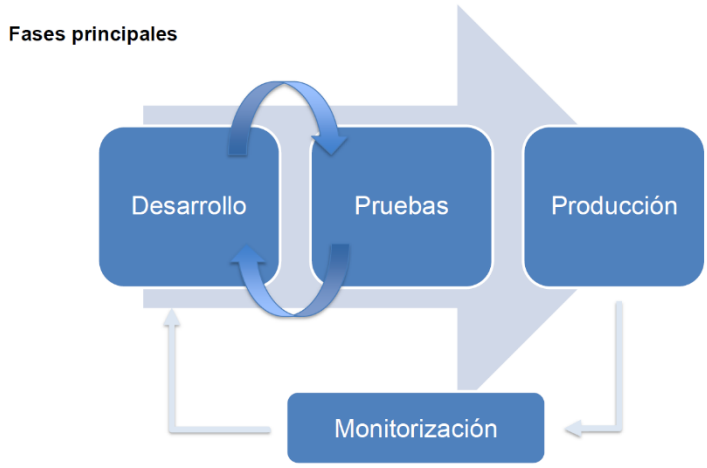
DESPLIEGUE DE APLICACIONES WEB

Despliegue Web

Desplegar: Instalar una aplicación en un servidor web con el objetivo de que esté disponible para otros usuarios (incluidos otros desarrolladores).





Buenas practicas

Elaborar un plan de despliegue: procedimientos, acciones y horarios.

Respetar los entornos de desarrollo locales, colaborativos y de pre-producción.

Evaluar las diferencias entre el entorno de desarrollo y el entorno real.

Testear la aplicación en cada entorno.

Utilizar sistemas de control de versiones (Git).

Utilizar ramas para cada funcionalidad o grupo de funcionalidades.

Utilizar el desarrollo en local en la medida de lo posible.

Implementar mecanismos de monitorización y respuesta rápida.

Prepararse para imprevistos (las cosas suelen romperse).

Aplicaciones Web

Cuando se introduce una dirección en el navegador...

1. Ordenador conectado por módem a un ISP (Telefónica, Vodafone...).
2. Escribimos una URL (dirección) en el navegador.
3. El ISP recibe la URL y la traduce a una dirección IP (protocolo DNS).
4. El navegador solicita una petición HTTP al servidor que "escucha" en la dirección IP para que le envíe el recurso (por ejemplo, una página web).
5. El servidor acepta (mensaje "**200 OK**") la petición y envía el recurso como paquetes de datos (protocolo TCP). Puede dar otros mensajes.
6. El navegador recibe la página y la recorre en busca de elementos que necesite para completarla (por ejemplo, imágenes).
7. El navegador realiza nuevas peticiones al servidor (al mismo o a otros) para obtener cada elemento de la página.
8. El navegador muestra la página completa.

La aplicación web desde el lado del servidor

Petición:

- Servidor escucha en puerto (p.ej. 80 -Apache-)
- Comprobaciones de autenticación (si procede)
- Ejecución de scripts de servidor (p.ej. PHP)
- Consulta (si procede) a base de datos (DBMS, p.ej. MySQL)
- Construcción (dinámica) del contenido
- Preparación de contenido HTML, CSS y JS

Respuesta:

- Petición aceptada (código HTTP 2XX, devuelve contenido)
- Petición redirigida (código HTTP 3XX)
- Petición denegada (código HTTP 4XX)
- Error de servidor (código HTTP 5XX)

La aplicación web: estructura en capas:

Servidor web	• Comunicación TCP (p.ej. puerto 80, 443) • Respuesta estática (fichero HTML, CSS, imágenes, etc.) • Ejecución de un programa (script) • Ejemplos: Apache, Nginx, Tomcat, NodeJS,…
Capa de aplicaciones	• Web dinámica: ejecución de un programa para generar el contenido solicitado por el navegador web • Lenguajes compilados: Java, C, C++, C#,… • Lenguajes interpretados (scripts): PHP, Python, Perl,…
SGBD (DBMS) Sistema gestor de BDD	• Persistencia de la información (más allá de los ficheros) • Estructuración y escalabilidad • SGBD relacional (SQL): MySQL, MariaDB, PostreSQL,… • SGBD no relacional: MongoDB,…

Virtualización

Tecnología para crear una o varias representaciones virtuales de un servidor (máquina virtual) dentro de una máquina física.

A nivel de comportamiento, es como tener una réplica de un equipo físico.

Maquinas virtuales

Ventajas:

- ✓ Permiten desplegar aplicaciones web en local o en la nube.
- ✓ Disponer de varios sistemas operativos y versiones (actuales/anteriores).
- ✓ Facilitar el desarrollo y las pruebas de software.
- ✓ Crear copias de seguridad.
- ✓ Fáciles de manejar y mantener.

Desventajas:

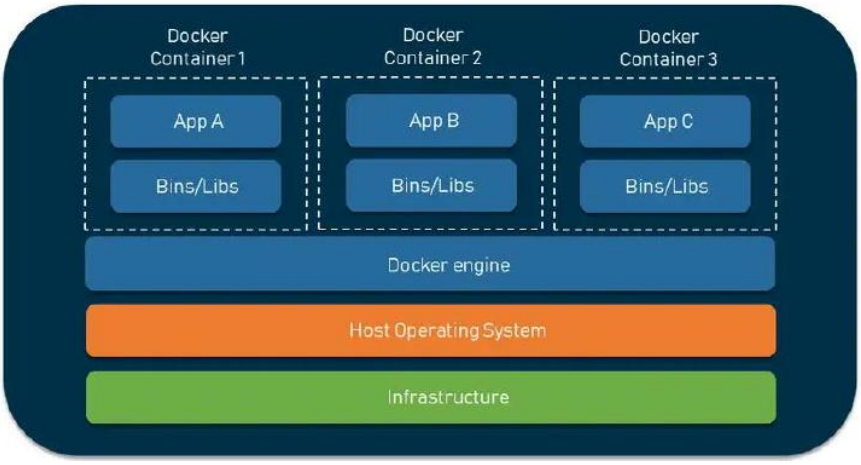
- × Consumo de recursos (CPU, disco duro, RAM).
- × Problemas de eficiencia.

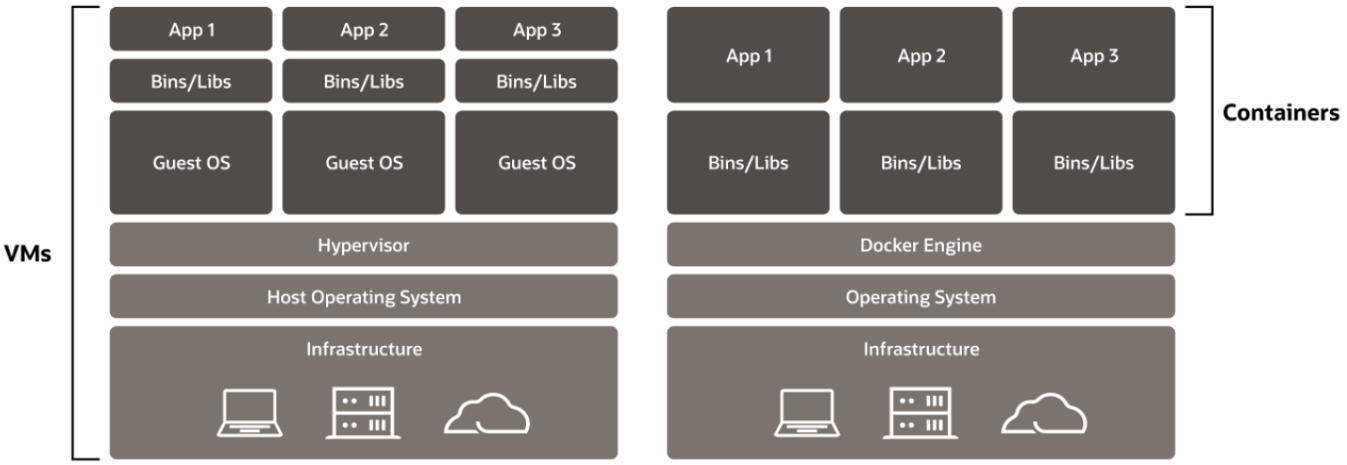
¿Y si solo nos interesa virtualizar el SW, obviando el HW? → **Contenedores**

Un contenedor es una unidad de SW que empaqueta el código de una aplicación y todas sus dependencias (Docker).

Se ejecuta en el sistema operativo huésped (host), donde debe haberse instalado previamente una aplicación específica (Docker Engine).

No son persistentes, se ejecutan a partir de una imagen.





Virtual Machines

- Each virtual machine (VM) includes the app, the necessary binaries and libraries and an **entire guest operating system**

Containers

- Containers include the app and all of its dependencies, but **share the kernel** with other containers.
- Run as an isolated process in userspace on the host operating system.
- Not tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud.

Control de Versiones (GIT)

Git es un sistema de control de versiones distribuido que permite gestionar y hacer un seguimiento de cambios en el código.

Desarrollo colaborativo: Facilita la colaboración de múltiples desarrolladores, que pueden trabajar de manera simultánea sin sobrescribir el trabajo de los demás. Cada desarrollador trabaja en una característica (rama) o revisión y posteriormente se integra de manera ordenada.

Manejo de versiones: Historial de cambios, comparación de versiones, reversión a una versión anterior y mantenimiento de diferentes versiones (desarrollo/pruebas/producción).

Automatización de despliegues: Facilita la integración y entrega continua (CI/CD), haciendo que cada cambio en el código desencadene automáticamente procesos de prueba, integración y despliegue en producción.

¿Git o GitHub? Git es la tecnología de control de versiones, mientras que GitHub es básicamente un sistema de almacenamiento en la nube para gestionar repositorios Git. Otro servicio similar a GitHub es GitLab.

¿Como funciona? 4 áreas diferenciadas

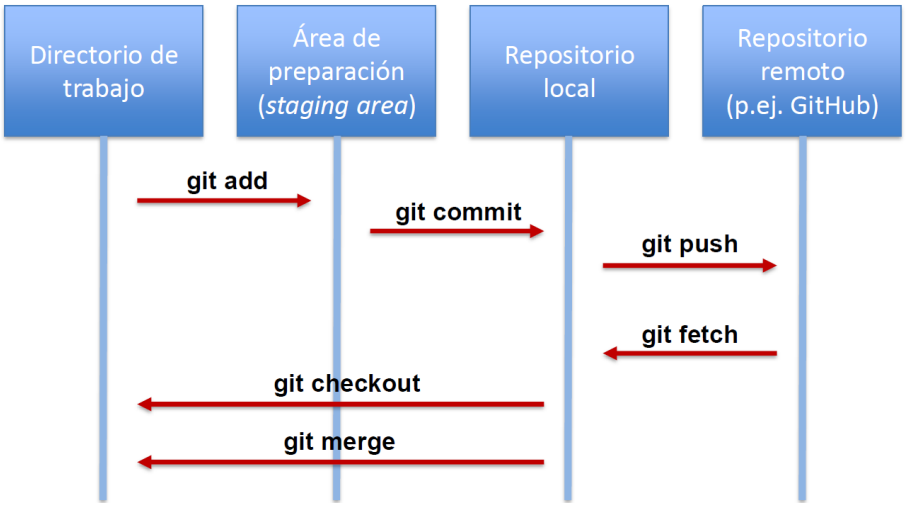
Trabajamos en nuestro código en el **directorio de trabajo** del IDE (Visual Studio Code, Sublime, Eclipse, PHPStorm, etc.).

Añadimos a la **staging area** los ficheros cuyas versiones queramos controlar (no tienen por qué ser todos los del directorio).

Proporcionamos una descripción de los cambios que se han hecho en esos ficheros (p.ej. “fichero nuevo”, “cambios en fichero X” ...) y se mandan (*commit*) al **repositorio local**.

El repositorio local y el **repositorio remoto** se sincronizan para que esté siempre disponible el control de versiones (última versión y versiones anteriores) para nosotros y nuestros colaboradores.

El paso de código de un área a otra se realiza con comandos (flujo de trabajo):



Documentación

Las herramientas de documentación son necesarias para que los desarrolladores puedan entender, utilizar y mantener el código desarrollado (propio o de otros). Todos los lenguajes de programación tienen herramientas que permiten generar documentación (habitualmente como páginas webs) a partir de anotaciones realizadas en el propio código.

Para documentar APIs se puede utilizar OpenAPI/Swagger, una especificación de formato para describir los endpoints, métodos, parámetros, respuestas y cualquier otra información relevante de una API. Se puede integrar en cualquier lenguaje de programación utilizando las bibliotecas correspondientes.

Conclusiones

Desplegar es básicamente alojar una aplicación en un servidor web para que los clientes la puedan utilizar.

Una aplicación web es accesible a través de un navegador web (cliente), al que un servidor proporciona vistas (páginas web con HTML+CSS+JS) de manera dinámica en función de las solicitudes que envía el cliente.

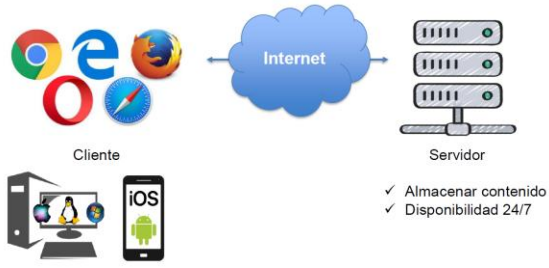
El servidor crea las vistas ejecutando cierto código (interpretado -PHP, Python, Perl- o compilado -Java-).

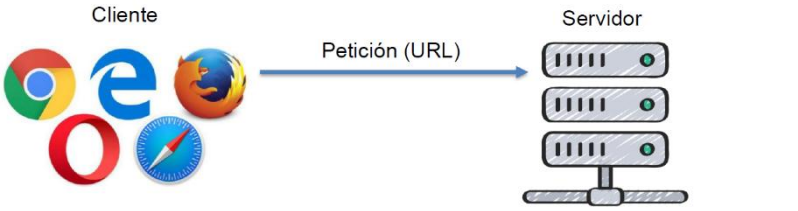
Desarrollar una aplicación web requiere componentes y herramientas para su correcto funcionamiento, uso y mantenimiento (servidor, librerías, frameworks, bases de datos, dependencias, control de versiones, documentación, etc.).

Para facilitar el despliegue de la aplicación se emplean técnicas de virtualización, que reproducen el entorno de desarrollo y pruebas en máquinas dedicadas a producción (máquinas virtuales o contenedores).

ARQUITECTURAAS WEB. IMPLANTACION Y ADMINISTRACION DE SERVIDORES

Arquitectura Cliente-Servidor





Partes de una aplicación web:

Frontend: Componentes destinados a la interacción con los usuarios

Backend: Componentes relacionados con los datos, la infraestructura, lógica de negocio, microservicios, APIs, seguridad, escalabilidad, eficiencia, etc.

Aplicación web:
- Recibe y procesa la(s) petición(es)
- Ejecuta código (PHP, Java, JS,…)
- Operaciones CRUD (SQL, NoSQL)
- Devuelve respuesta y contenido
- Renderizado dinámico de webs

CRUD: operaciones sobre bases de dados: crear, leer, actualizar y borrar (***C**reate*, ***R**ead*, ***U**pdate*, ***D**elete*)

NOTA: El frontend de la aplicación se lo “envía” el servidor al navegador (cliente), para que este lo muestre (renderizar) y el usuario pueda interactuar (código Javascript) y continuar realizando peticiones. Debe estar optimizado para el cliente (responsive).

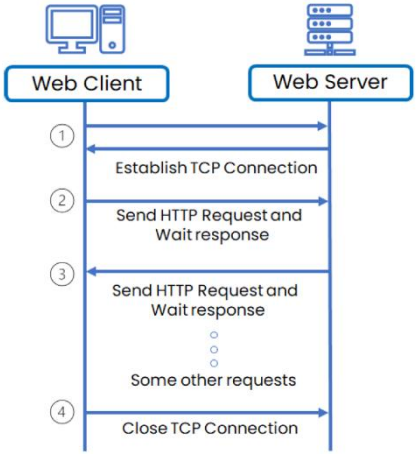
Protocolos de Comunicaciones

HTTP: Hypertext Transfer Protocol (protocolo de transferencia de hipertexto)

Protocolo orientado a transacciones con esquema petición-respuesta entre cliente (user agent, navegadores web) y servidor. El cliente solicita un recurso identificado unívocamente como URL (uniform resource locator).

Tipos de peticiones más utilizadas:

- ✓ GET: Solicita una representación del recurso especificado.
- ✓ POST: Envía datos en la URL de la petición para que sean procesados por el recurso identificado del servidor (creación de un nuevo recurso).
- ✓ PUT: Actualización de un recurso.
- ✓ DELETE: Borrado de un recurso.
- ✓ OPTIONS: Devuelve los métodos HTTP que soporta el servidor para un recurso concreto.



Cliente: peticiones (request)

Servidor: respuestas (response)

Protocolo TCP/IP: la información (peticiones y respuestas) viaja "troceada" en paquetes y debe confirmarse cada paquete que se recibe (ACK - acknowledgement)

Cargar una página implica múltiples peticiones y respuestas (y ACKs)

Códigos de estado de respuesta HTTP

1xx (Información): Petición recibida, continua el proceso.

2xx (Éxito): Petición recibida con éxito, comprendida y aceptada.

3xx (Redirección): Se necesitan más acciones para completar la petición.

4xx (Error de cliente): La sintaxis de la petición no es correcta o no puede completarse.

5xx (Error de servidor): El servidor no pudo completar la petición (aunque sea válida).

Código	Estado	Que hace
100	Continue	El navegador "pregunta" primero al servidor si puede continuar con la petición, especialmente en los casos en que el cuerpo de la petición es grande.
101	Switching Protocols	El servidor acepta un cambio de versión de protocolo HTTP a propuesta del navegador.
102	Proccessing	El servidor responde que aún está procesando una petición del navegador, evitando así que el navegador considere que se ha perdido.
200	Ok	Respuesta estándar para peticiones correctas. Suele ir acompañada de los datos solicitados por el navegador.
201	Created	Petición completada y creación de un nuevo recurso.
202	Accepted	Petición aceptada, aunque no se ha podido completar aún.
203	Non-authoritative Information	Petición se ha completado, pero el contenido procede de otro servidor.
204	No Content	Petición completada, pero la respuesta no tiene contenido.
205	Reset Content	Petición completada, respuesta sin contenidos y el navegador tiene que reinicializar la página desde la que realizó la petición (p.ej. borrar formularios una vez se han enviado).
206	Partial Content	Petición aceptada, pero devuelve solo parte de los contenidos (p.ej. trocear una descarga o retomar una descarga interrumpida).
300	Multiple Choices	Indicar opciones múltiples para el contenido solicitado por el cliente (p.ej. videos con distinta resolución).
301	Moved Permanently	Todas las peticiones se redirigen a otra URL.
302/303	See Other	La respuesta a la petición está en otra URL accesible mediante otra petición GET.
304	Not Modified	El contenido solicitado en la URL no ha sido modificado desde la última vez que se solicitó, con lo que no hace falta reenviarlo (ahorro de tráfico y ancho de banda).
400	Bad Request	La solicitud del cliente tiene errores de sintaxis.
401	Autorization Required	La solicitud requiere de autorización previa por parte del cliente.
403	Forbiden	Tanto si el cliente está autorizado como si no, no tiene privilegios suficientes para acceder al recurso solicitado.
404	Not Found	El servidor no encuentra el recurso solicitado.
405	Method Not Allowed	La petición (GET, POST, PUT...) no es la adecuada para la URL solicitada.
406	Not Acceptable	El servidor no puede devolver los datos en el formato que solicita el cliente en su petición (campo "Accept" de la cabecera de la petición).
408	Request Timeout	El cliente no ha continuado la petición que tenía pendiente.
414	URI Too Long	La URL es demasiado larga (p.ej. una petición GET que debería ser POST).

415	**Unsopported Media Type**	El formato solicitado por el navegador no es entendible por el servidor.
429	**Too Many Requests**	Demasiadas conexiones desde la misma IP cliente.
451	**Unavailable For Legal Reason**	Contenido eliminado por temas legales (p.ej. seguridad, derechos legales, etc.).
500	**Internal Server Error**	Hay un error en el servidor que no depende del servidor (p.ej. hosting).
501	**No Implemented**	El servidor no soporta la petición del cliente.
502	**Bad Gateway**	El servidor funciona como puerta de enlace (gateway) hacia otro servidor y recibe una respuesta errónea por parte de ese otro servidor.
503	**Service Temporaly Unavailable**	El servidor no puede responder a la petición de manera temporal (p.ej. por sobrecarga, mantenimiento, etc.).
504	**Gateway Timeout**	El servidor funciona como puerta de enlace (gateway) hacia otro servidor y no recibe respuesta por parte de ese otro servidor.
505	**HTTP Version Not Supported**	El servidor no soporta la versión de HTTP de la petición del cliente.
507	**Insufficient Storage**	El servidor no dispone de suficiente espacio libre para procesar la petición.

HTTPS

Conexión HTTP cliente-servidor

Se acuerda qué sistema de encriptación (TLS) se va a usar.

El servidor envía su certificado digital (SSL) al cliente (incluye su clave pública).

El cliente valida el certificado y genera una clave de sesión con él y se la envía encriptada al servidor utilizando su clave pública. El servidor recibe la clave y la desencripta con su clave privada.

El resto de los mensajes se encriptan con la clave de sesión.

Algunas consideraciones (el precio de la seguridad):

Necesidad de autoridades certificadoras: Entidades confiables que emiten certificados digitales a personas, organismos, empresas, etc. comprobando previamente ciertos requisitos legales.

Aunque puedes crear tu propio certificado digital (p.ej. en Java), no será admitido como fiable por ningún navegador actual.

Pérdida de eficiencia: Encriptar y desencriptar mensajes implica más consumo de CPU (latencia). Los servicios de caché de servidores intermedios no trabajan de forma tan eficaz, por lo que las actualizaciones de contenido a través de Internet son más lentas.

¿Hay que sustituir completamente HTTP por HTTPS?

Depende de nuestras necesidades de seguridad o de la fase en que se encuentra una aplicación (p.ej. desarrollo inicial).

SSH: ***S****ecure* ***SH****ell* (intérprete de órdenes seguro)

Acceso seguro (encriptado) a máquinas remotas para manejarlas por completo a través de línea de comandos o interfaz gráfica (servidor X).

Copia de datos de forma segura.

Evolución del protocolo Telnet (texto plano ➔ no seguro).

Se basa en el uso de claves

- Criptografía asimétrica para autenticación y establecimiento de la conexión (claves pública y privada).
- Criptografía simétrica (clave acordada) para encriptar/desencriptar los datos que se transfieren a través de la conexión (túnel SSH).

FTP: ***F**ile **T**ransfer **P**rotocol* (protocolo de transferencia de ficheros)

SFTP: ***S**ecure **S**hell **F**ile **T**ransfer **P**rotocol*

Utiliza claves SSH para encriptación

Permite autenticación con usuario/contraseña o con claves SSH

DNS (***D**omain **N**ame **S**ystem*): Sistema de Nombres de Dominio

Protocolo de Internet que traduce nombres "amigables" (p.ej. wikipedia.org) a direcciones IP (p.ej. 185.15.58.224) y viceversa.

Principales registros DNS

Registro A (Address Record): Registro básico de DNS que se utiliza para mapear un nombre de dominio a una dirección IPv4.

wikipedia.org IN A 185.15.58.224

Registro CNAME (Canonical Name Record): Registro que se utiliza como alias y que redirige a otro nombre de dominio (que se denomina nombre canónico). Es útil para tener varios dominios que apuntan al mismo registro A, aunque requiere una consulta adicional para conocer la IP.

Servidores Web Apache y Nginx

Apache y Nginx son los servidores web más populares en la actualidad.

Ambos son de código abierto, gratuitos y compatibles.

Apache:

- Primer servidor web consolidado (soporte)
- Arquitectura basada en procesos e hilos para cada conexión (recursos)
- Pérdida de eficiencia en cargas elevadas
- Flexible y personalizable con módulos
- Maneja páginas estáticas y dinámicas desde el propio servidor (PHP/Perl)

Nginx:

- Desarrollado con posterioridad a Apache
- Arquitectura basada en eventos y asincronía (más eficiente)
- Ideal para cargas altas y concurrencia (rendimiento y escalabilidad)
- También es modular, pero menos flexible y personalizable
- Optimizado para servir páginas estáticas muy rápidamente
- Para contenidos dinámicos utiliza otras aplicaciones (Apache, FastCGI...)

Servidores Web Vs Servidores de Aplicaciones

Servidor web: Sirve contenido HTTP (HTML con texto, imágenes, videos, etc.) generado de manera estática (sin interacción del cliente) o dinámica (interacción y ejecución de scripts como PHP, Perl, Python, etc.). Apache, Nginx...

Servidor de aplicaciones: Sirve contenido HTTP dinámico (y también de otros protocolos), que depende de interacciones del usuario y que implica aplicar la lógica de negocio avanzada de la aplicación (ejecución de código, acceso a bases de datos, mensajería, seguridad, etc.). El resultado de esta ejecución de código puede ser o no una página HTML que el servidor web envía al cliente. Tomcat (Java), Gunicorn (Python), Node.js (Javascript), Apache (PHP), Nginx (PHP), Microsoft IIS (.NET)...

Frameworks

Conjunto de herramientas, bibliotecas (librerías) y estructuras que proporcionan la base para el desarrollo de aplicaciones.

Los *frameworks* relacionados con desarrollo web proporcionan normalmente un servidor de aplicaciones embebido que facilita el desarrollo y testeo de la aplicación.

Conclusiones

En aplicaciones web, la arquitectura más común es la denominada **arquitectura cliente-servidor**.

Una aplicación web se puede dividir en dos partes: ***frontend*** y ***backend***.

El **protocolo HTTP** se utiliza para enviar y recibir información entre cliente-servidor.

Las peticiones y respuestas HTTP incluyen cabecera (***header***) y cuerpo (***body***).

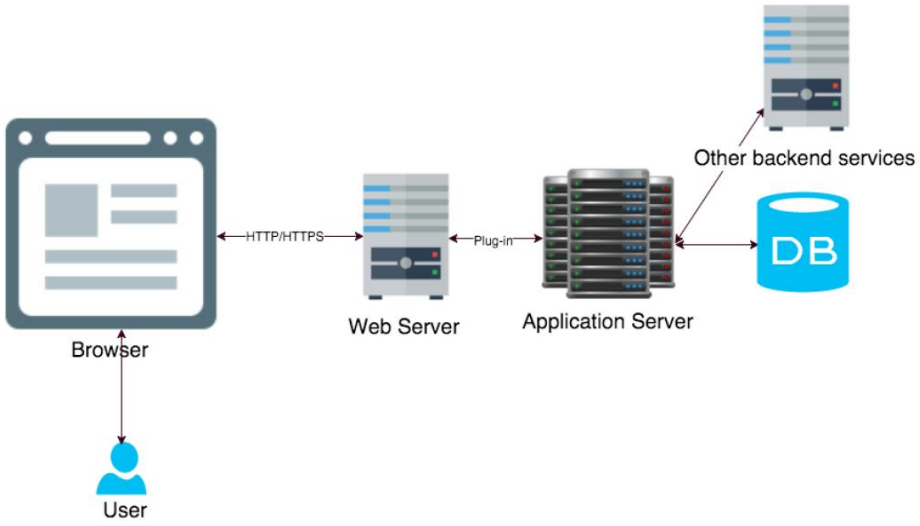
HTTP es un protocolo no seguro: **HTTPS** incorpora cifrado (TLS/SSL) por pares de claves.

El protocolo **SSH** permite administrar equipos (servidores) remotos de forma segura. Permite el uso del terminal y la transferencia segura de ficheros (protocolo **SFTP**). SSH y SFTP también tienen una arquitectura cliente-servidor y facilitan el despliegue de aplicaciones y la administración de servidores.

El protocolo **DNS** permite traducir direcciones web (amigables) por direcciones IP (entendibles por Internet).

SERVIDORES DE APLICACIONES Y DE DATOS

Servidores de Aplicaciones



El servidor de aplicaciones es un intermediario entre el cliente que accede al servidor web y las bases de datos, así como diferentes servicios y aplicaciones empresariales.

Tomcat

Tomcat (Apache Tomcat o Jakarta Tomcat) es un servidor que permite conectar las peticiones HTTP de un servidor web con aplicaciones Java que se ejecuten en el *backend*.

Estas aplicaciones Java se denominan *servlets* y *JSPs* (JavaServer Pages) en el contexto web y serían el equivalente en Java de los scripts PHP.

Estrictamente, Tomcat no es un servidor de aplicaciones, sino un contenedor web de *servlets/JSPs*, aunque en muchos sentidos tiene la misma función.

Funcionamiento básico:

Desarrollo de una aplicación en Java:

Compilación como ejecutable (creación de un fichero WAR)

- Copiar el fichero WAR en la carpeta /webapps del servidor
- El servidor descomprime el fichero WAR y despliegue la aplicación en el puerto indicado (habitualmente 8080).

Compilación como ejecutable (creación de un fichero JAR)

NOTA: Ejecución de aplicaciones Java

Java requiere disponer de una máquina virtual (JVM) instalada en el sistema operativo para poder ejecutar la aplicación.

Fichero JAR: Aplicaciones de propósito general ejecutables por la JVM mediante la instrucción → java -jar miAplicacion.jar

Fichero WAR: Aplicación web preparada para ejecutarse por un servidor Tomcat.

Tomcat → Spring Boot: aplicaciones web y microservicios

Spring Boot

Framework para desarrollo en Java de aplicaciones web.

Integra por defecto un servidor Tomcat.

Múltiples opciones de configuración automatizada.

Facilita la gestión e inyección de dependencias (Maven).

Permite generar un fichero JAR ejecutable con todo lo necesario para desplegar la aplicación en un servidor (solo requiere que esté instalada la JVM).

Permite no incluir el servidor Tomcat y generar un fichero WAR que se ejecute en un servidor Tomcat externo.

Optimizado para la generación de microservicios y APIs.

Node.js y Express

Node.js es un entorno de ejecución de Javascript para servidor.

Permite crear aplicaciones web y dispone de un servidor propio.

Incluye NPM (*Node Package Manager*), que permite gestionar las dependencias de las aplicaciones.

Express es un *framework* que facilita la creación de aplicaciones, APIs, microservicios, etc. sobre Node.js.

Javascript no requiere compilar previamente el código, por lo que se puede ejecutar directamente en servidor o en cliente (en realidad se va compilando en tiempo de *ejecución).*

Servidores de Bases de Datos

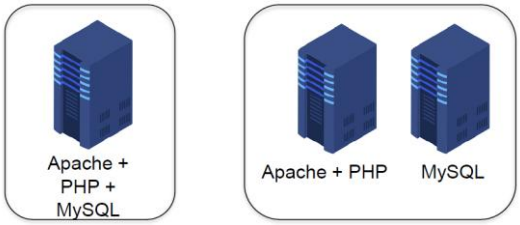
De manera análoga a los servidores web y de aplicaciones que hemos visto en apartados anteriores, un servicio de base de datos se identifica mediante una IP (protocolo IP) y un puerto (protocolo TCP).

Además, las bases de datos deben estar convenientemente protegidas, por lo que para acceder a su contenido será necesario autenticarse con usuario y contraseña.

Así, para conectarse al servicio de bases de datos desde una aplicación se necesita lo siguiente:

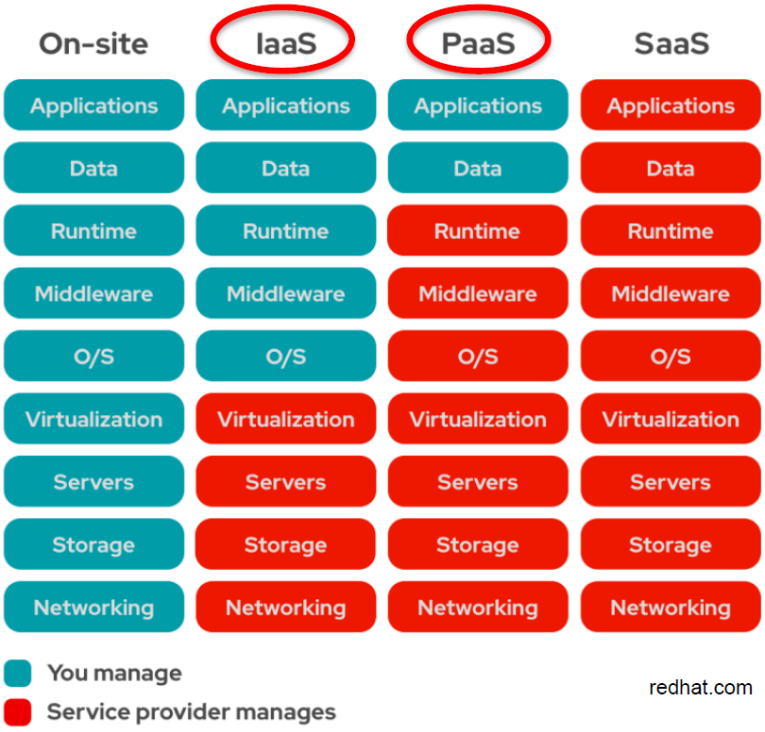
- IP
- Puerto
- Usuario
- Contraseña

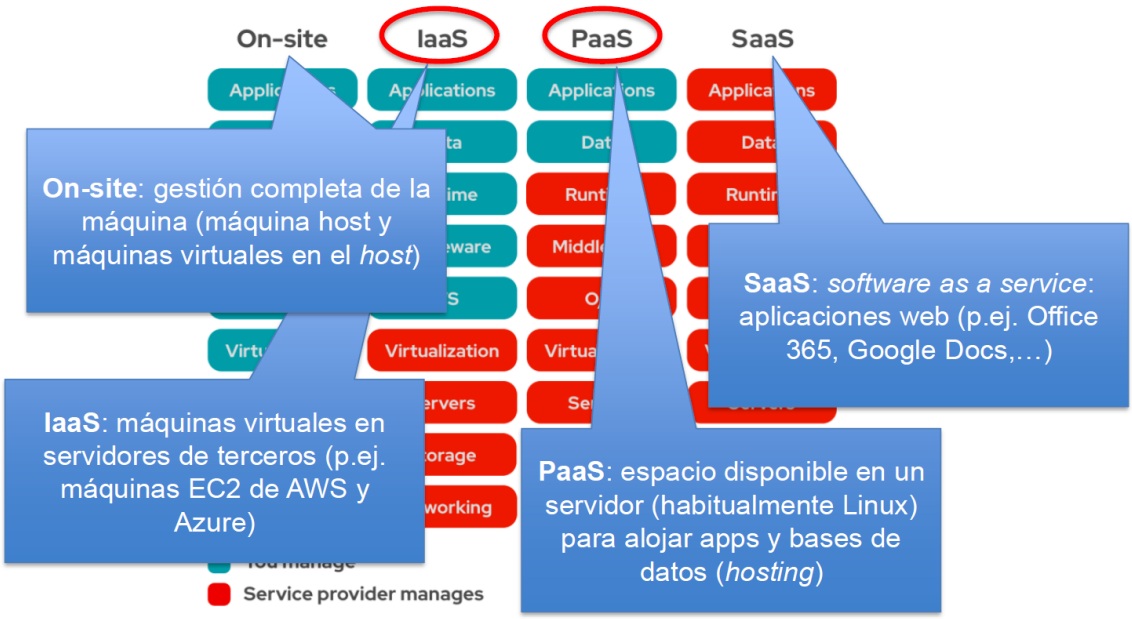
Recordar que los servidores de datos pueden estar alojados en la misma máquina donde estén ejecutándose las aplicaciones web o en otras máquinas.



La decisión de una arquitectura u otra dependerá de nuestras necesidades y/o limitaciones. En cualquier caso, es importante que la administración de la base de datos sea eficiente y segura.

IaaS: Infrastructure as a Service





AWS (Amazon Web Services) proporciona entre sus servicios la posibilidad de crear máquinas virtuales en la nube (EC2).

EC2 (*Elastic Compute Cloud*): permite a los usuarios alquilar computadores virtuales en los cuales pueden ejecutar sus propias aplicaciones.

AWS Academy: Permite disponer de recursos AWS gratuitos para estudiantes.

Learner Lab: Entorno de desarrollo que permite a los estudiantes utilizar diferentes servicios de AWS, como máquinas EC2 y contenedores.

PaaS: Plataform as a Service

GitHub Pages

Ofrece un servidor web para contenido estático, ideal para complementar la descripción de proyectos.

Servicio gratuito:

- ✓ 1GB
- ✓ No permite ejecutar código de aplicaciones (PHP, JS, Java...)
- ✓ Subdominios (DNS)
- ✓ Certificados SSL (HTTPS)
- ✓ Permite redireccionar un dominio propio

InfinityFree

Ofrece una pila LAMP para desplegar aplicaciones web basadas en PHP y MySQL.

Servicio gratuito:

- 5GB
- PHP 8.2
- MySQL 5.7 y MariaDB 10.4
- Subdominios (DNS)
- Certificados SSL (HTTPS)
- Permite redireccionar un dominio propio

Despliegue y configuración de una aplicación web con InfinityFree

DBaaS: Database as a Service

Existe también la opción de tener nuestra base de datos separada de la aplicación en un servidor dedicado a bases de datos. Todas las tecnologías de bases de datos admiten su versión *cloud*.

Ejemplos de proveedores:

- ❖ Aruba Cloud (MySQL, PostgreSQL y MS SQL Server): de pago
- ❖ Amazon RDS (*Relational Database Service*): incluido en AWS Academy Learner Lab
- ❖ Aiven (MySQL, PostgreSQL): gratuito hasta 5 GB
- ❖ MongoDB Atlas (MongoDB): clúster gratuito de 512 MB

El funcionamiento del DBaaS es equivalente al de una conexión local o a una máquina virtual, como hemos visto con anterioridad. Cabe resaltar la necesidad de que el acceso esté debidamente protegido con una contraseña fuerte, ya que los ataques de tipo *ransomware* son frecuentes.

Conclusiones

Servidores de aplicaciones: Más complejo que un servidor web (acceso a bases de datos y múltiples aplicaciones relacionadas con la lógica empresarial, no solo páginas web).

Cada lenguaje de programación tiene el suyo, acompañado con frecuencia de un *framework* que facilita el desarrollo (Tomcat/SpringBoot para Java o Node.js/Express para Javascript).

Los servidores de bases de datos complementan las aplicaciones web y varían en arquitectura y características (SQL: MySQL/MariaDB, PostgreSQL y NoSQL: MongoDB).

Suelen gestionarse con herramientas tipo aplicación web (phpMyAdmin) o de escritorio (MySQL Workbench, HeidiSQL, Compass, etc.).

Las IaaS y las PaaS facilitan el despliegue de aplicaciones en la nube, ya que proporcionan diversos elementos HW y SW configurables que se pueden ajustar a las necesidades de la aplicación, optimizando recursos.