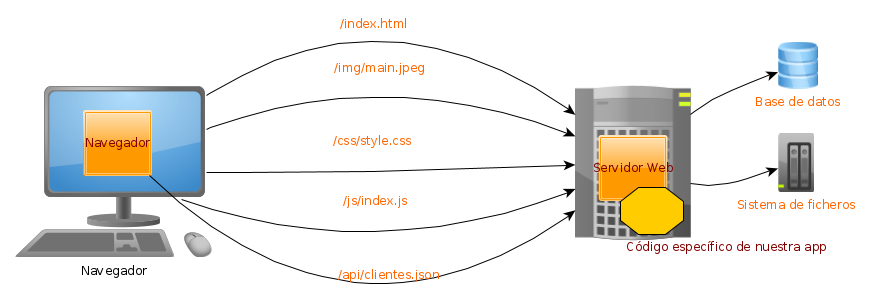
APUNTES LOREZO DESPLIEGUE

**La web son páginas** (En formato HTML, Imagen, JSON, XML ,etc) **que se interconectan entre ellas por enlaces** (urls).

**El navegador solicita por TCP/IP el recurso a obtener**. El **formato** de como solicita ese recurso **es por el protocolo HTTP**. **Y el servidor** de TCP/IP **responde** **usando el protocolo** HTTP devolviendo los datos.



En el host servidor se estaba ejecutando el servidor TCP/IP que entiende el protocolo HTTP. A ese software lo llamaremos servidor Web. El servidor es “personalizable” permitiendo que se ejecute código específico para nuestra aplicación. Ese código específico es el código de servidor que se suele escribir en Java, PHP, NodeJS, etc.

Por lo tanto en una app web hay 3 tipos de código:

* Código de visualización: HTML, CSS, Imágenes, Videos, etc.
* Código en el cliente: El JavaScript que se ejecuta en el navegador
* Código en el servidor: El programa específico de la aplicación que se ejecuta en el servidor Web

## Conceptos

* **VPS (Virtual Private Server):** Una máquina virtual que alquilas a una empresa.
* **Empresa de Hosting:** Empresa que alquila los Host sean tanto un VPSs como una máquina real
* **Balanceador de carga:** Software que le llega una petición y la redirige a otro Host de entre varios para no sobrecargar ningún Host o evitar enviarlo a un host que no funciona
* **Escalabilidad:** Diseñar la aplicación de forma que se alquilen o se desalquilen Hosts (VPS o máquinas reales) en función de la carga del sistema
* **Tolerancia a fallos:** Diseñar la aplicación de forma que aunque un host falle, la aplicación siga funcionando
* **IAAS (Infraestructura como servicio) :** Si la empresa de Hosting solo se ofrece el Host y nosotros nos tenemos que instalar todo el software, incluyendo el sistema operativo y administrarlo todo.
* **PAAS (Plataforma como servicio):** Si la empresa de Hosting nos ofrece el Host pero tambien software generico ya instalado como el Sistema Operativo, Servidor Web, Servidor de Correo, Balanceador de Carga , etc. En este caso nosotros solo debemos instalar el código específico de nuestra aplicación.En este caso aunque nos ofrecen un host ya que en algún sitio debe estar la app, realmente nos están ofreciendo el servidor web donde instalar nuestra app. En el caso de PASS, no tenemos que administrar nosotros ni el Sistema Operativo ni el servidor.
* **SAAS (Software como Servicio):** Como desarrolladores nunca usamos un SASS ya que la empresa de hosting ya ofrece hasta la aplicación instalada. Un ejemplo sería Google con “Google Docs” , Microsoft con su “MS Ofice 365”, DropBox , etc. que ya ofrecen todo para el usuario final.

### **IAAS vs PAAS**

El IAAS es mas versátil ya que solo nos ofrecen el ordenador y nosotros nos montamos todo como queremos. El problema es que es mas complicado todo de hacer y tenemos que administrarlo todo: Sistema operativo, servidor web y aplicación

Por otro lado en el PAAS, solo nos tenemos que preocupar de nuestra aplicación lo que hace que sea mas sencillo. El problema es que ya no hay tanta versatilidad, ya que debemos ceñirnos al entorno que nos ofrece la empresa

## NodeJS

NodeJS (o simplemente node) es un lenguaje de programación basado en JavaScript. Al ser un lenguaje interpretado, su forma de trabajar es mas similar a BASH que a Java. ¿Porque vamos a usar NodeJS? Porque muchas herramientas para el desarrollo web están hechas en node. Además del lenguaje de programación, node incluye un gestor de paquetes similar al de Linux (como apt) que se llama “npm” (Node package manager ).

# **2. Administración de Servidores Web**

## Tipos de Servidores Web

Hay dos grandes tipos de servidores web:

* Servidores Web externos: Son programas completos que hacen de servidor Web. Una vez instalados/ejecutados se añade el código específico de la aplicación en la carpeta del servidor que indique la documentación
* Servidores web integrados (o librerías de Servidores web): Se hace una aplicación (por ejemplo en Java) y se añade como una librería (un JAR) , el código del Servidor Web.

## Servidores Web externos

Para ver como funcionan los servidores Web Externos vamos a explicar dos de ellos:

* Apache HTTP Server Project
* Nginx

### Apache HTTP Server Project

Este servidor Web es un proyecto de una fundación sin ánimo de lucro llamada [Apache Software Fundation](https://www.apache.org/). El proyecto del servidor web se llama [Apache HTTP Server Project](https://httpd.apache.org/) por lo que a veces se confunde el nombre del servidor con el nombre de la fundación.

* **Instalacion**:Para usar el servidor Apache solo hay que descargarselo y ejecutar el programa “/bin/httpd.exe”
* **Despliegue**:Para desplegar páginas web en Apache solo hay que copiarlas en la carpeta “htdocs”
* **Configuración/administración**:Para configurar/administrar el servidor hay que modificar los ficheros de la carpeta “conf”

### nginx

nginx es un servidor web que podemos descargar desde <http://nginx.org/>.

* **Instalacion**:Para usar el servidor nginx solo hay que descargarselo y ejecutar el programa “nginx.exe”
* **Despliegue**:Para desplegar páginas web en nginx solo hay que copiarlas en la carpeta “html”
* **Configuración/administración**:Para configurar/administrar el servidor hay que modificar los ficheros de la carpeta “conf”

# 3. Despliegue Automatizado

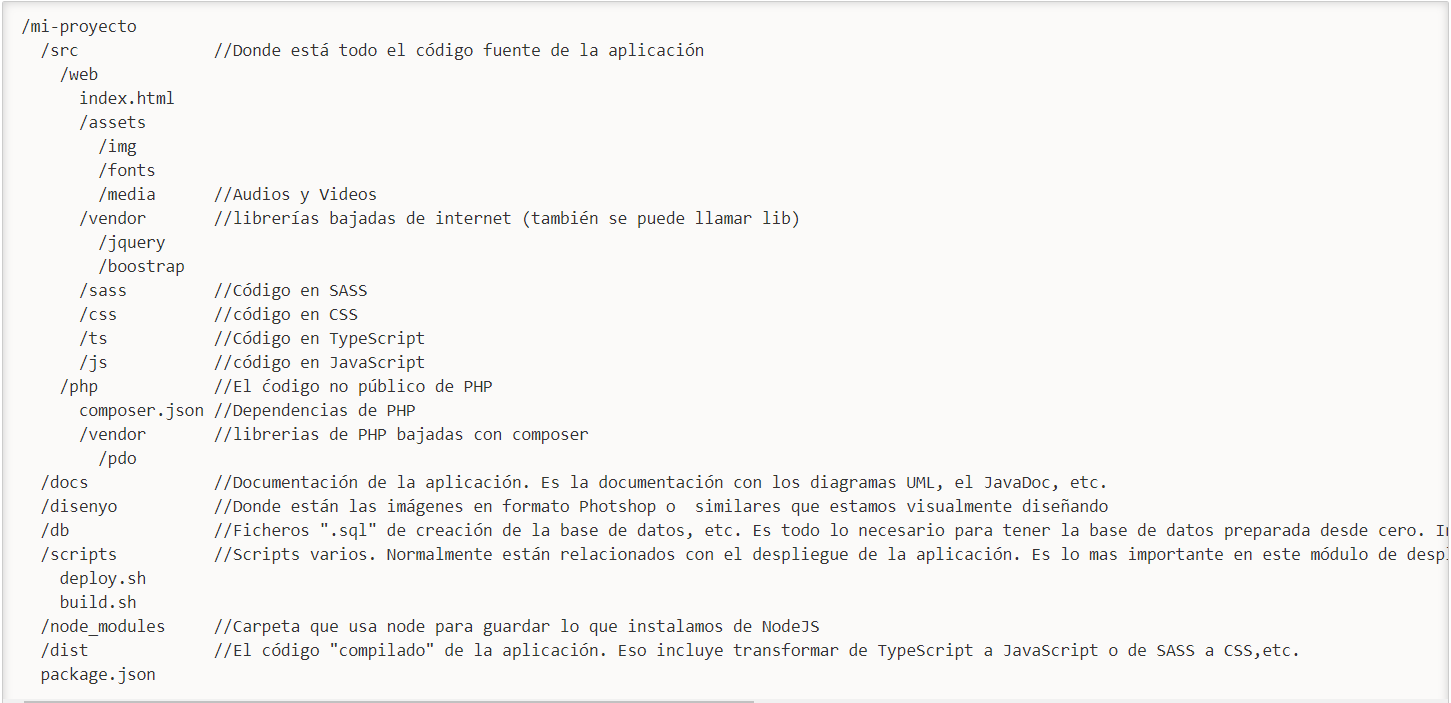
## Despliegue Automatizado

En el tema anterior hemos visto como **desplegar manualmente en los servidores Apache y Nginx**. En este tema vamos a ver como hacer lo mismo pero de forma automatizada. Automatizar tareas es fundamental ya que se minimizan los errores que se cometen. Por otro lado el despliegue puede llegar a hacerse decenas de veces al día, por lo que obviamente debe estar automatizado.

Podemos pensar que instalar (desplegar) decenas de veces al día una aplicación es un poco exagerado pero en sitios web como Amazon ésto si que tiene sentido. Si en la aplicación de amazón hay un error que suele afectar al 0,1% de sus ventas. ¿Cuanto dinero pierden diariamente por tener ese error? Y en este tipo de aplicaciones se suelen estar constantemente realizando cambios y corrección de error , así que ¿porque no desplegarla en cuanto está corregido un error o añadida la nueva funcionalidad? Esto en la literatura informática se llama **Continuous delivery**

## Estructura de carpetas

Para poder automatizar el despliegue , lo primero es tener una estructura definida de nuestro proyecto y de nuestro servidor. Durante el curso vamos a usar este esquema tanto en este módulo como en los otros módulos. El esquema he intentado que sea lo mas fiel posible a la realidad de las empresas aunque he intentado simplificarlo. Sin embargo cada empresa tendrá la estructura que se adecue al proyecto que estén desarrollando.



Hay que fijarse que en git no se debe guardar únicamente la carpeta “src” del código fuente sino que debe guardarse la carpeta “mi-proyecto” completa. Aunque se puede hacer alguna excepción con “dist” o “node-modules”. El motivo de guardar la carpeta “mi-proyecto” es debido a que realmente todas esas carpetas son las que forman el proyecto, y no únicamente la carpeta “src”.

## Carpeta dist

La carpeta dist es la mas importante para el módulo de despliegue ya que en ella estará el proyecto tal y como se va a desplegar. Podemos pensar que debe ser únicamente una copia de la carpeta “src” ya que el código JavaScript o el código PHP no se compilan pero si que se pueden hacer muchos cambios en esta carpeta. Veamos algunos ejemplos:

* Minimizar el código: Hacer que ocupe menos el código JavaScript o CSS. Se hace para que las páginas web se carguen antes. Y suele consistir en quitar los comentarios, los retornos de carro, etc.
* Ofuscar el código: Para evitar que alguien pueda copiar nuestro código se suele, por ejemplo , cambiar el nombre de las variables por nombres sin sentido. Normalmente se minimiza y se ofusca a la vez.
* Transpilar: Es transformar el código fuente de un lenguaje en otro. Se usa:
  + TypeScript a JavaScript: TypeScript es un lenguaje con mayor tipado que JavaScript y se suele usar en vez de JavaScript. Como los navegadores no soportan TypeScript hay que convertirlo (transpilarlo) de TypeScript a JavaScript.
  + SASS a CSS: SASS es un lenguaje similar a CSS pero con mayores funcionalidades. Como los navegadores no soportan SASS hay que convertirlo (transpilarlo) de SASS a CSS. En SASS también se suele decir transformar el código de SASS a CSS en vez de decir “transpilar”
* Etc.

Por ello , por ejemplo, en la carpeta “src” estaría el código en TypeScript y en la carpeta “dist” estaría en JavaScript correspondiente al TypeScript.

## Tareas de Automatización del despliegue

Veamos ahora un ejemplo de como desplegar una aplicación web en PHP con SASS , TypeScript y que se despliega en Apache.

* Copiar el código de “src” en “dist”
* Transformar el código SASS en CSS dentro de la carpeta “dist”
* Transpilar el código TypeScript en JavaScript dentro de la carpeta “dist”
* Borrar el código SASS de la carpeta “dist”
* Borrar el código TypeScript de la carpeta “dist”
* Copiar el código de la carpeta “dist” a “htdocs”
* Ejecutar el “httpd.exe”

Para hacer todas esta tareas vamos a usar Scripts del sistema operativo , por ejemplo en Linux se usará “Bash” y los Script de npm que vamos a ver mas adelante.

# Scripts npm

Para hacer las tareas que acabamos de indicar, se harán scripts de Bash que se dejarán en la carpeta “/scripts” sin embargo ahí puede haber gran cantidad de Script o podemos querer usar otra carpeta u otro lenguaje. Para unificar todo esto en los proyectos web, vamos a homogeneizar la ejecución de estos scripts mediante los Scripts de npm.

Veamos ahora un ejemplo de Script de transformar SASS a CSS.

* Crear un Script para node dentro del **packaje.json**

|  |  |
| --- | --- |
| 1  2  3 | "scripts": {    "compile-scss": "node-sass  -o ./dist/css ./src/scss"  } |

* Ejecutar un Script de node.

npm run compile-scss

Que ahora todas las “tareas” que tenemos que hacer en el proyecto se pueden ver en el “packaje.json” y siempre se ejecutarán como “**npm run nombre-tarea**”

Veamos ahora tres script que deberíamos siempre tener en cualquier proyecto.

* build: Compila el código de src y lo deja en dist. Eso incluye compilar el TypeScript, el SASS, etc.
* deploy: Compila y despliega el código. Deja el contenido de dist en el servidor Apache en una carpeta de htdocs.
* start: Inicia el servidor apache , Compila y despliega el código con cada cambio. En el siguiente apartado se explica como ejecutar un Script cuando hay un cambio en un fichero.

# **4. El protocolo HTTP**

El protocolo HTTP se usa para enviar y recibir datos en la Web.

## Características

* Sencillo: Es en modo texto y fácil de usar directamente por una persona.
* Extensible: Se pueden enviar más metadatos que los que están por defecto. Ej: Nº de página
* Sin estado: Cada petición es independiente. Eso es un problema en sitios como por ejemplo un carrito de la compra.

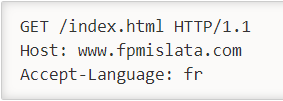
## Ventajas

* Cache: Mejora la velocidad al controlar la cache de las páginas
* Autenticación: Permite identificar a un usuario
* Proxys: Permite de forma transparente usar proxies
* Sesiones: Gracias a las cookies podemos mantener el estado entre peticiones.
* Formatos: Permite indicar el formato de lo que se envía, de lo que se pide y de lo que se retorna.

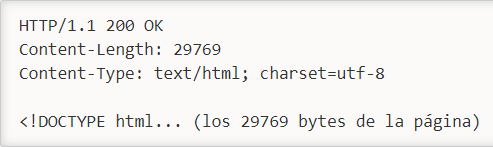
## Formato

El formato lo vamos a explicar sobre un ejemplo.

Una petición HTTP tiene la siguiente forma:



La respuesta del servidor es:



Vamos ahora a explicar la petición

* GET: Es el método por el que se piden los datos. Entre sus valores está: GET, PUT,POST, DELETE.
* /index.html: Es la ruta dentro del servidor del documento que estamos pidiendo
* HTTP/1.1 : La versión del protocolo. Prácticamente siempre es 1.1
* Host: [www.fpmislata.com](http://www.fpmislata.com/): Cabecera llamada Host que indica el nombre del host al que va dirigida la petición.
* Accept-Language: fr: Otra cabecera que indica en que idioma queremos que nos retorne los datos. En este caso es en francés.

Es decir que en una petición HTTP hay una primera línea y luego varias líneas con las cabeceras.

Pasemos ahora a explicar la respuesta:

* HTTP/1.1: La versión del protocolo con la que responde. Prácticamente siempre es 1.1
* 200 OK: Si ha sido existosa o no la petición.
* Content-Length: 29769: Cabecera llamada Content-Length que indica las bytes que ocupan los datos que se retornan
* Content-Type: text/html; charset=utf-8:Cabecera llamada Content-Type que indica el formato [MIME type](https://en.wikipedia.org/wiki/Media_type) de los datos que retornan y su codificación. En este caso es en HTML y en formato UTF-8.
* <!DOCTYPE html…: Son finalmente los datos que se han pedido.

## Cabeceras HTTP

Existen muchas cabeceras HTTP, podemos ver un listado en [List of HTTP header fields](https://en.wikipedia.org/wiki/List_of_HTTP_header_fields). Pero aquí solo vamos a ver algunas.

Las cabeceras se dividen entre las que se envían en la petición y las que se retorna en la respuesta.

### **Petición**

Cabeceras que se pueden enviar en la petición

* Accept: El formato [MIME type](https://en.wikipedia.org/wiki/Media_type) en la que queremos que se retornen los datos. Ej: En text/html, en text/xml, application/json , application/pdf, etc. Luego el servidor los retornará en el formato que quiera/pueda
* Accept-Language: El idioma en el que queremos que nos retorne los datos.. Luego el servidor los retornará en el idioma que quiera/pueda.
* Host: El dominio al que se está enviando la petición. Esta cabecera el muy útil ya que permite en un mismo servidor tener alojados varios dominios.
* Content-Type: El formato de los datos que envian al servidor. Ej: En text/html, en text/xml, application/json , application/pdf, etc. Y como están codificado. Normalmente los formatos son utf-8 o ISO-8859-1.

### **Respuesta**

Cabeceras que se pueden enviar en la respuesta

* Content-Type: El formato de los datos que se retorna. Ej: En text/html, en text/xml, application/json , application/pdf, etc. Y como están codificado. Normalmente los formatos son utf-8 o ISO-8859-1. No tiene porque coincidir con Accept.
* Content-Language: El idioma de los datos que se retorna.
* Content-Length: Tamaño en bytes de los datos
* Cache-Control: Cuanto tiempo pueden estar cacheado los datos.

## Estados HTTP

Es estado es lo que indica si una petición HTTP ha tenido éxito o no. Sus principales valores son:

* 200-299: La petición ha tenido éxito
* 300-399: Redirección de los datos.
* 400-499: Los datos que ha enviado el cliente no son correctos
* 500-599: Se ha producido un error en el servidor.

Mas información en [Códigos de estado HTTP](https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP)

De entre todos los código están algunos que solemos ver a menudo:

* 204: La petición no retorna datos.
* 403: Prohibido el acceso al documento
* 404: No encuentra el documento
* 500: Error del servidor

ANEXO

Tema 1

**Desafíos y tareas para desplegar**

* Generación de la aplicación a instalar
  + Obtener el código fuente
  + Compilar el código fuente
  + Probar la aplicación
  + Analizar la calidad del código
* Múltiples programas a instalar
  + Sistema Operativo
  + Servidor Web
  + Base de datos
  + La aplicación
  + Servidor de envío de correos
  + Etc.
* Hosts
  + La base de datos puede estar en un Hosts y el servidor web en otro Hosts, etc.
  + Los Host pueden ser maquinas virtuales o máquinas físicas estando cada una de ellas en proveedores distintos (Amazon AWS, Microsoft Azure, Google Cloud, etc).
* Administración
  + Creación de copias de seguridad
  + Logs
  + Seguridad: Recuperación en caso de perdida de datos por algún fallo
* Rendimiento: Permitir que la aplicación siga funcionando aunque haya un pico de peticiones
  + Añadir nuevas máquinas si hay un pico de peticiones y quitarlas cuando ya no las hay tantas peticiones
  + Balanceo de carga entre todos los servidores de la aplicación
  + Monitorización del rendimiento
  + Alertas de bajo rendimiento
* Fiabilidad: Permitir que la aplicación siga funcionando aunque falle algún Host.
* Microservicios: Una única aplicación se divide en pequeñas micro-aplicaciones, llamadas **microservicios** con lo que se multiplican todas las complicaciones anteriores por el número de microservicios que tengamos

