

INTRODUCCIÓN A NODE.JS



ÍNDICE



1. ¿Qué es Node.js? - Definición y propósito.
2. Evolución de JavaScript - De cliente a servidor.
3. Primera etapa: DHTML (años 90).
4. Segunda etapa: AJAX y librerías (jQuery, Prototype).
5. Tercera etapa: Frameworks de frontend (Angular, React).
6. Cuarta etapa: Expansión al servidor (Node.js).
7. Motor V8 de Google Chrome – Características y papel en Node.js.
8. Requisitos de JavaScript en el servidor – Acceso a ficheros, BD, peticiones.
9. Consecuencia: Un solo lenguaje para todo el desarrollo web.
10. Características principales de Node.js – Asincronía, rapidez, escalabilidad.
11. Empresas que usan Node.js – Netflix, PayPal, Uber, Microsoft.
12. Instalación de Node.js – Windows, Mac, Linux.
13. Gestor de paquetes npm – Definición y funciones.
14. Archivo package.json – Configuración de proyectos Node.
15. Instalación de módulos locales y globales.

¿QUÉ ES NODE.JS? - DEFINICIÓN Y PROPÓSITO

Node.js es un entorno de ejecución de JavaScript en el lado del servidor basado en el motor V8 de Google Chrome, que permite usar JS fuera del navegador para construir aplicaciones rápidas y escalables.

- Entorno en servidor, no un framework: ejecuta JavaScript “del lado del servidor” usando V8.
- Basado en V8: compila JS a código máquina para alta velocidad y recolección de basura eficiente.
- API asíncrona y dirigida por eventos: no bloquea el hilo principal; maneja I/O mientras tu app sigue respondiendo.
- Modelo monohilo, altamente escalable: atiende muchas peticiones con pocos recursos gracias al modelo event-driven.
- Un lenguaje para todo (full-stack JS): mismo lenguaje en cliente y servidor.

EVOLUCIÓN DE JAVASCRIPT - DE CLIENTE A SERVIDOR.

JavaScript ha pasado de ser un lenguaje sólo para el navegador a convertirse en la misma pieza central del desarrollo «full-stack», gracias a etapas sucesivas (DHTML → AJAX → Frameworks → Node.js).

JavaScript empezó siendo pequeño y centrado en el navegador; con AJAX se hizo más interactivo; los frameworks le dieron estructura; y finalmente Node.js lo trajo al servidor, convirtiendo JS en un lenguaje completo para todo el stack.

PRIMERA ETAPA: DHTML (AÑOS 90).

DHTML introdujo las primeras capacidades dinámicas en las páginas web: validaciones, manipulación del DOM y cambios en la interfaz sin servidor complicado.

JS permitió que las páginas dejaran de ser estáticas: se podían validar formularios antes de enviarlos o mostrar/ocultar partes de la página. Pero la comunicación servidor-cliente todavía exigía recargar la página completa.

- Validación de formularios en el cliente.
- Apertura/gestión de ventanas emergentes.
- Exploración y modificación del DOM para cambiar contenido visible.
- Limitaciones: sin comunicación asíncrona con el servidor (recarga total).

SEGUNDA ETAPA: AJAX Y LIBRERÍAS (JQUERY, PROTOTYPE).

Con AJAX se abrió la puerta a la comunicación asíncrona: actualizar partes de la página sin recarga completa; librerías como Prototype y jQuery facilitaron y normalizaron estas operaciones.

- AJAX: peticiones HTTP en background para refrescar fragmentos de la vista.
- Librerías (Prototype → jQuery) simplificaron DOM + AJAX, abordando diferencias entre navegadores.
- Resultado: interfaces más reactivas y experiencias de usuario más fluidas.

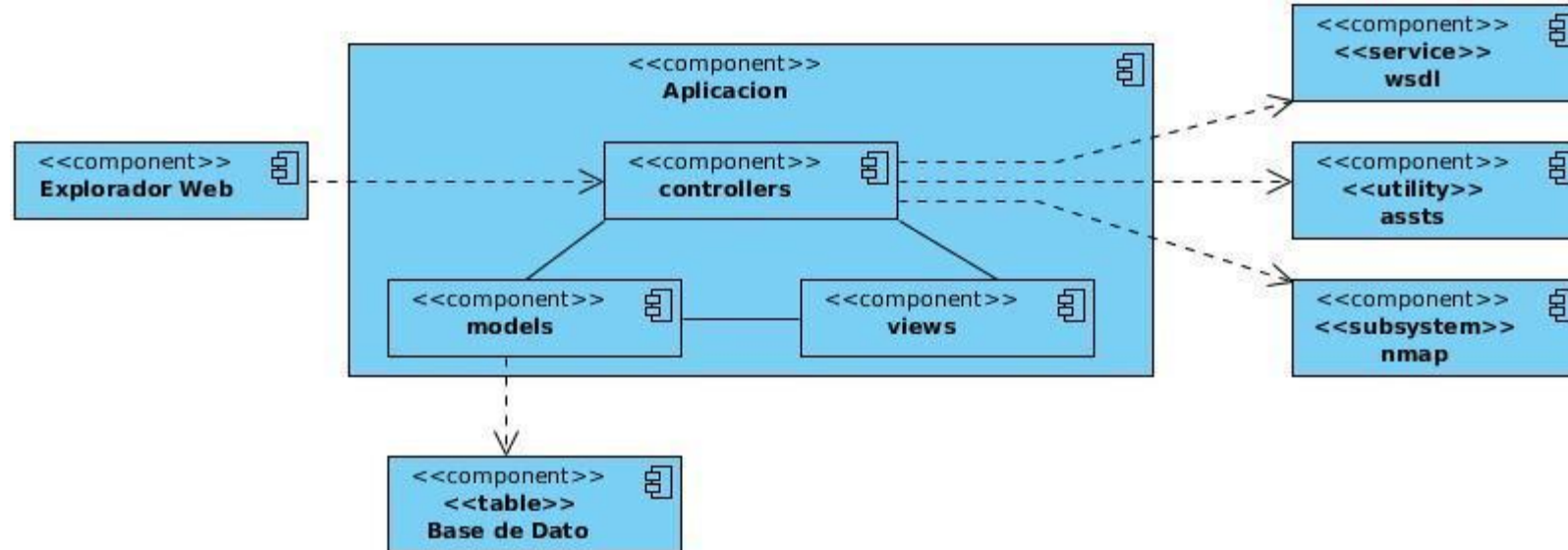
```
// (concepto) petición AJAX con jQuery$.ajax({ url:
'/datos', success: data => $('#zona').html(data) });
```

AJAX permitió que sólo una parte de la página se actualizase al recibir datos del servidor. Las librerías como jQuery hicieron que esa técnica fuera accesible para la mayoría de desarrolladores, evitando las incompatibilidades entre navegadores

TERCERA ETAPA: FRAMEWORKS DE FRONTEND (ANGULAR, REACT).

Los frameworks introdujeron estructuras y patrones claros para aplicaciones complejas: componentes, gestión de estado y renderizado eficiente.

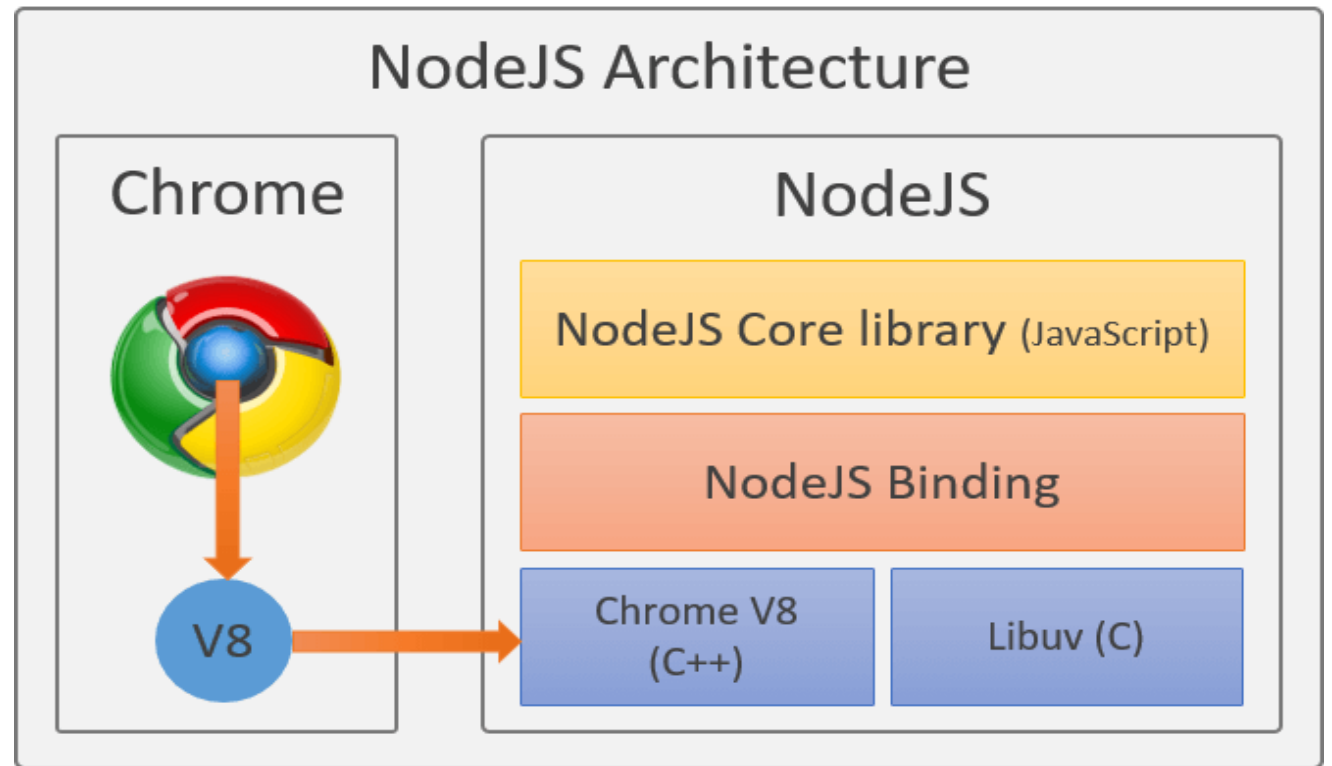
- Componentes reutilizables (UI encapsulada).
- Gestión de vistas y estados (unidireccional o reactivo).
- Herramientas de routing, binding de datos y testing.
- Facilitan la separación de responsabilidades en aplicaciones ricas del lado cliente.



CUARTA ETAPA: EXPANSIÓN AL SERVIDOR (NODE.JS).

Node.js permitió ejecutar JavaScript fuera del navegador, ofreciendo acceso a recursos del sistema y transformando JS en lenguaje completo para la web.

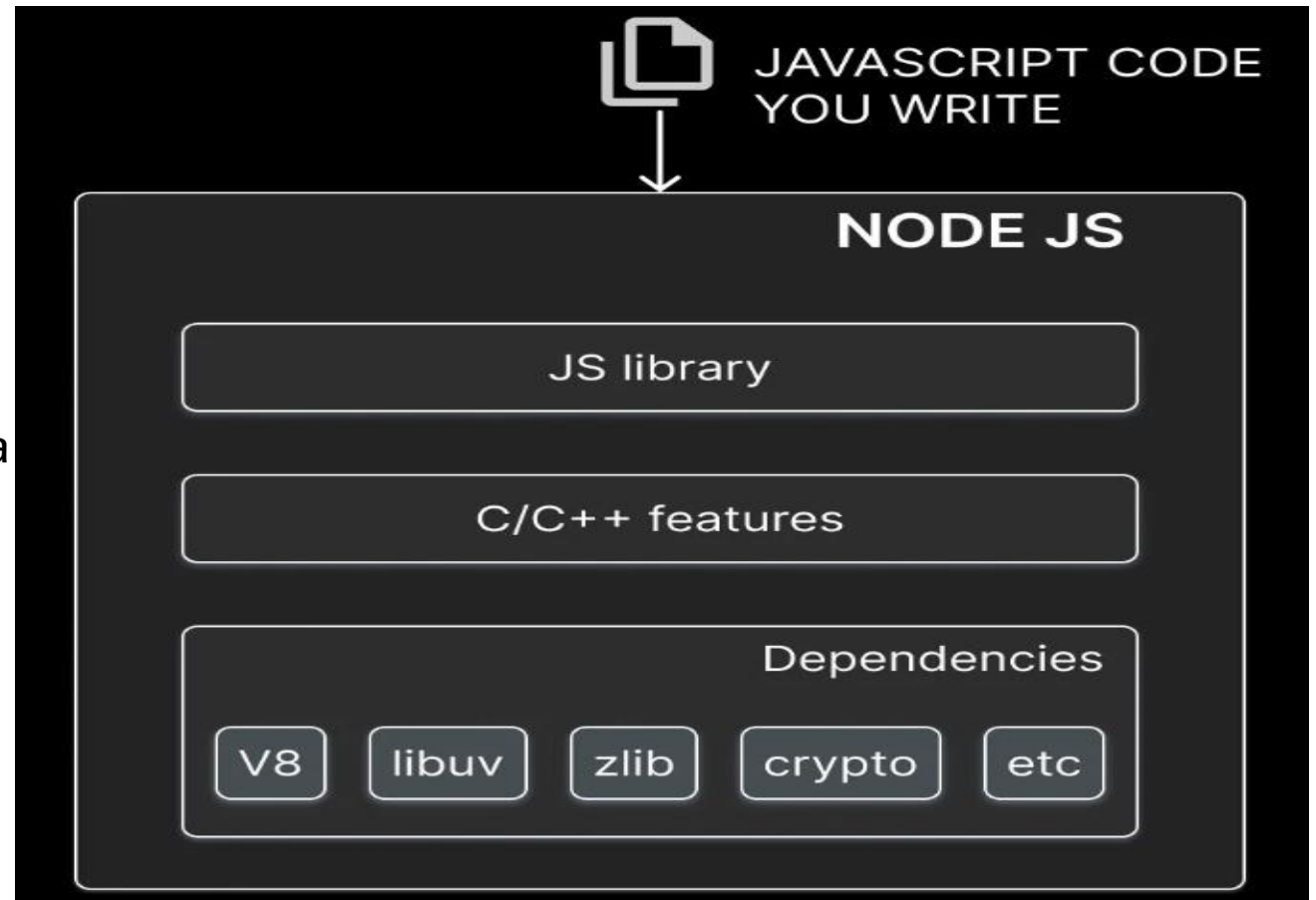
- Node usa V8 para ejecutar JS en servidor.
- Añade APIs y bindings (a través de C++) para acceder a ficheros, red y BD.
- Implica un cambio arquitectónico: el mismo lenguaje para cliente y servidor.



MOTOR V8 – CARACTERÍSTICAS Y PAPEL EN NODE.JS.

V8 es el motor que compila JavaScript a código máquina; en Node.js actúa como núcleo de ejecución sobre el que se montan APIs de sistema.

- Escrito en C++ y open-source.
- Compila JS a código máquina (JIT) para ejecución muy rápida.
- Se encarga también de recolección de basura y gestión de memoria.
- V8 se usa en Chrome/Chromium y está embebido por Node.js; Node añade bindings extras en C++ para I/O y acceso a SO.



REQUISITOS DE JAVASCRIPT EN EL SERVIDOR – ACCESO A FICHEROS, BD, PETICIONES.

Para ser un lenguaje de servidor, JavaScript necesitaba capacidades que ECMAScript no define: acceso a ficheros, conexión a bases de datos, comunicación de red y gestión de peticiones HTTP. Node.js aporta esas capacidades.

- Acceso al sistema de ficheros (leer/escribir, subir archivos).
- Conectividad a bases de datos (drivers y clientes).
- Comunicación en red (sockets, HTTP).
- Aceptar peticiones y enviar respuestas: servidor HTTP y manejo de rutas.

CONSECUENCIA: UN SOLO LENGUAJE PARA TODO EL DESARROLLO WEB.

+

•

Node.js posibilita que la misma sintaxis y paradigmas (JavaScript) se usen en cliente y servidor, simplificando aprendizaje, reutilización y despliegue.

- Menos context switching (mismo lenguaje para front y back).
- Reutilización de librerías y modelos de datos entre cliente y servidor.
- Facilita equipos full-stack y prototipado rápido.
- Nota práctica: las APIs no son idénticas (DOM vs FS), por lo que no todo código es intercambiable.

La ventaja estratégica es clara: un único lenguaje reduce la curva de aprendizaje y los posibles malentendidos entre capas, aunque hay que tener presente las diferencias de entorno.

CARACTERÍSTICAS PRINCIPALES DE NODE.JS

Node.js combina un motor rápido (V8) con un modelo event-driven y API no-bloqueante, lo que permite servir muchas conexiones con pocos recursos.

- API asíncrona y dirigida por eventos: callbacks, promesas y async/await.
- Ejecución rápida gracias a V8.
- Modelo monohilo con event loop pero altamente escalable para I/O intensivo.
- Apto para servicios que manejan muchas conexiones concurrentes con bajo overhead.

```
// servidor HTTP no bloqueante (concepto)
const http = require('http');
http.createServer((req, res) => { //
  manejo rápido y no bloqueante
  res.end('OK');
}).listen(3000);
```

EMPRESAS QUE USAN NODE.JS

Node.js ha sido adoptado por grandes empresas internacionales como parte de su stack de producción.

Cada una lo ha empleado para distintas necesidades: APIs, optimización de rendimiento y desarrollo rápido.



NETFLIX

Uber




INSTALACIÓN DE NODE.JS – WINDOWS, MAC, LINUX.

Instalar Node.js es sencillo: descarga desde el sitio oficial o usa repositorios para Linux; se ofrece versión LTS (recomendada) y versión Current.

Ir a nodejs.org y elegir entre LTS (recomendada) o Current.


Windows: paquete .msi — ejecutar instalador y seguir el asistente.

 Windows Installer (.msi)

 Standalone Binary (.zip)

macOS: paquete .pkg — ejecutar y seguir asistente.

 macOS Installer (.pkg)

 Standalone Binary (.gz)

Linux (Debian/Ubuntu): ejemplo en material:

```
apt-get update
apt-get upgrade
apt-get install curlcurl -sL
https://deb.nodesource.com/setup_9.x | bash -
apt-get install -y nodejs
```

GESTOR DE PAQUETES NPM – DEFINICIÓN Y FUNCIONES.

npm es el gestor de paquetes que se instala junto con Node.js y contiene el mayor ecosistema de librerías open-source para JavaScript.

- npm permite instalar paquetes locales por proyecto y paquetes globales.
- Comprobar versión: `npm -v`.
- Registro público en npmjs.com: buscar, ver descargas y publicar paquetes.
- Se usa para gestionar dependencias y scripts del proyecto.

```
→ npm-test npm install sass

added 16 packages, and audited 17 packages in 1s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
→ npm-test
```

ARCHIVO PACKAGE.JSON

package.json es el archivo de metadatos que describe un proyecto Node: nombre, versión, entry point, scripts y dependencias.

- name y version: identifican el paquete.
- main: archivo de entrada (por defecto index.js).
- scripts: comandos que puedes ejecutar con npm run <script>.
- dependencies / devDependencies: paquetes necesarios en producción o desarrollo.

```
{ "name": "PruebaNPM",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": { "test": "echo \"Error: no test specified\" && exit 1" },  
  "keywords": [], "author": "", "license": "ISC"  
}
```


INSTALACIÓN DE MÓDULOS LOCALES Y GLOBALES.

npm permite instalar módulos por proyecto (locales) o en el sistema (globales). Usa globales para herramientas CLI; instala localmente las dependencias de una app.

- Local (en proyecto): `npm install nombre_modulo` → añade a `node_modules` del proyecto.
- Global (sistema): `npm install -g nombre_modulo` → útil para herramientas de línea de comandos (Grunt, JSHint).
- Nota: un módulo instalado globalmente no se importa con `require` en una app a menos que también esté instalado localmente.

```
npm install express (local, para usar en la app)
```

```
npm install -g grunt-cli (global, herramienta CLI)
```