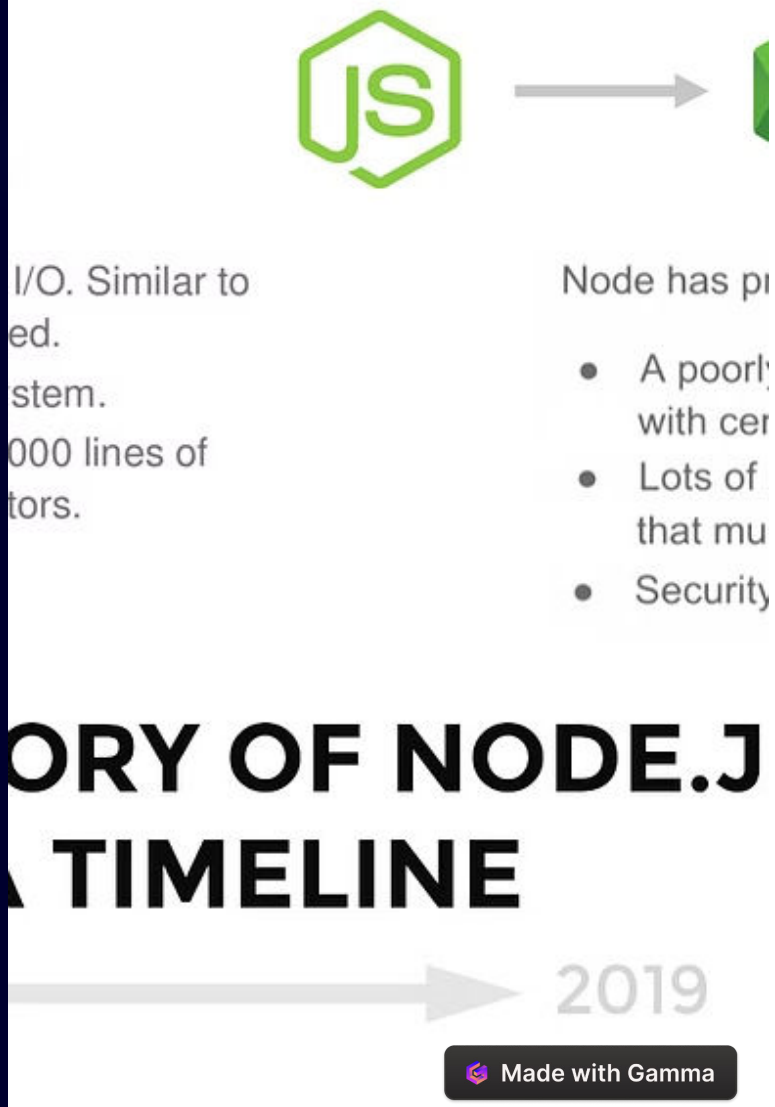
Historia y Evolución de Node.js

Node.js es una plataforma de desarrollo en el lado del servidor, basada en el motor JavaScript V8 de Google Chrome. Fue creado por Ryan Dahl en 2009, con el objetivo principal de proporcionar una manera de ejecutar código JavaScript en el servidor, fuera de un navegador web.

by Carlos Azaustre



Inicio de Node.js

1

Creación por Ryan Dahl

Ryan Dahl creó Node.js en 2009 para proporcionar una manera de ejecutar código JavaScript en el servidor. La necesidad de manejar múltiples conexiones simultáneas de forma eficiente fue la motivación detrás de su creación.

2

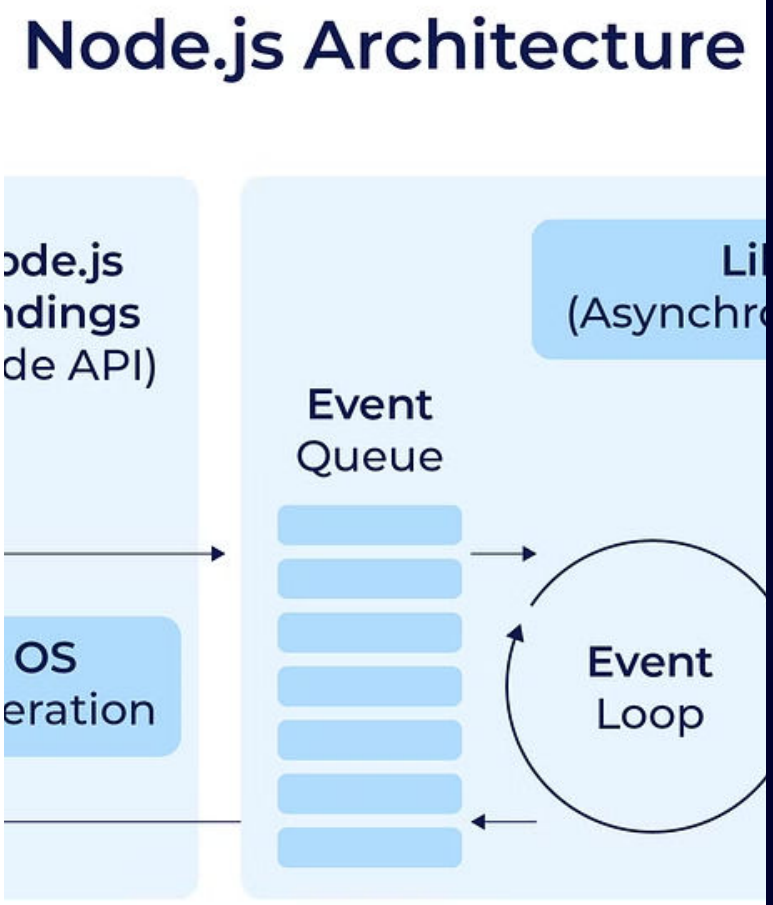
Motivación para su Creación

Las tecnologías de servidor tradicionales, basadas en hilos, no gestionaban de la mejor manera la necesidad de manejar múltiples conexiones simultáneas de forma eficiente, lo que llevó a la creación de Node.js.

3

Punto de Inflexión

La introducción de Node.js marcó un punto de inflexión en el desarrollo de aplicaciones web al ofrecer un entorno de ejecución basado en eventos y no bloqueante.



Node.js Architecture

Node.js is built on the Chrome V8 engine, enabling the execution of JavaScript code on the server. Its architecture, based on non-blocking event-driven operations, makes it ideal for developing applications that require high scalability and the ability to handle numerous simultaneous connections or requests.

by Carlos Azaustre

Características Distintivas

1 Modelo de Operaciones de Entrada/Salida No Bloqueante

Node.js utiliza un modelo de operaciones de entrada/salida no bloqueante, lo que significa que continúa ejecutando el resto del código mientras maneja el resultado de las operaciones de entrada/salida mediante callbacks. Esto contribuye a una mayor eficiencia y rendimiento, especialmente en aplicaciones web en tiempo real.

Beneficios de Node.js

Escalabilidad

Node.js es altamente escalable, lo que permite a las aplicaciones manejar un gran número de conexiones simultáneas de forma eficiente.

Rendimiento

Gracias a su arquitectura no bloqueante, Node.js ofrece un rendimiento excepcional, especialmente en aplicaciones de tiempo real.

Facilidad de Desarrollo

La capacidad de utilizar JavaScript tanto en el frontend como en el backend simplifica el desarrollo de aplicaciones web completas.

Arquitectura Basada en Eventos

1

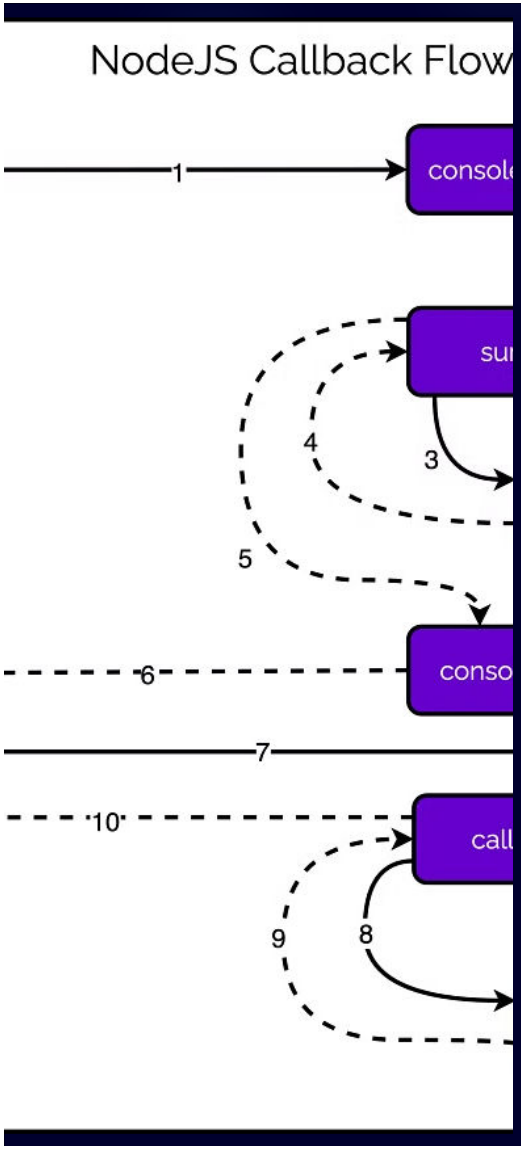
Concepto Central

Node.js utiliza una arquitectura basada en eventos, donde las operaciones son desencadenadas por eventos específicos y manejadas de manera asincrónica.

2

Ventaja Clave

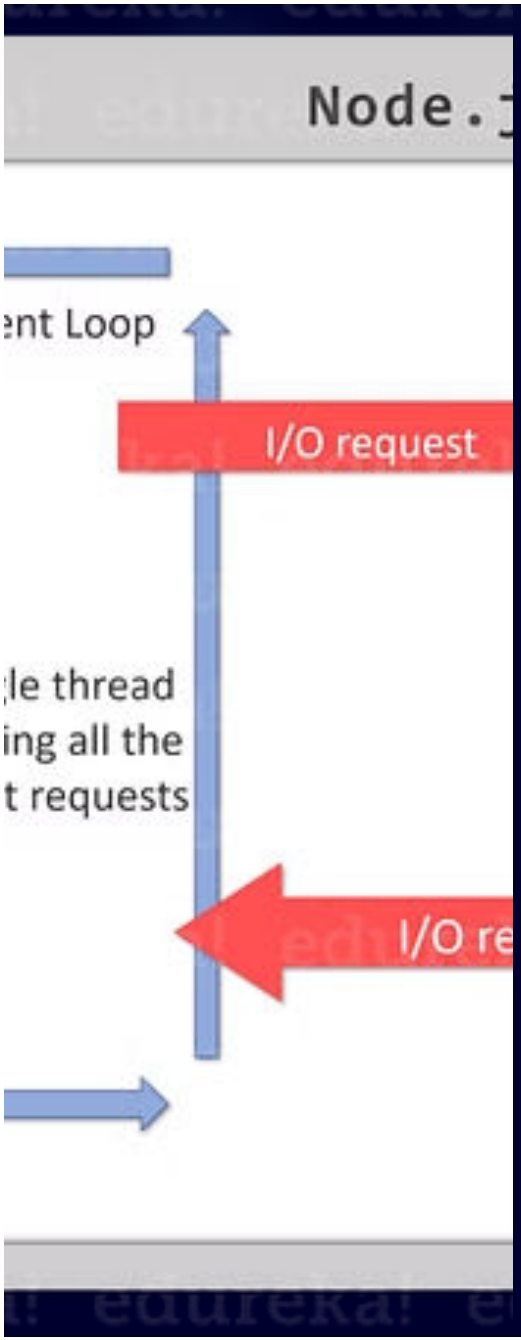
Esta arquitectura permite que Node.js sea altamente eficiente al manejar múltiples operaciones de forma simultánea sin bloquear el hilo de ejecución principal.



Uso de Callbacks

1 Manejo de Resultados

Node.js emplea callbacks para manejar el resultado de operaciones de entrada/salida, lo que permite que el código continúe ejecutándose mientras espera la finalización de dichas operaciones.



Operaciones de Entrada/Salida No Bloqueantes

1

Funcionamiento Continuo

Node.js mantiene el flujo de ejecución al no bloquear el procesamiento mientras se llevan a cabo operaciones de entrada/salida.

2

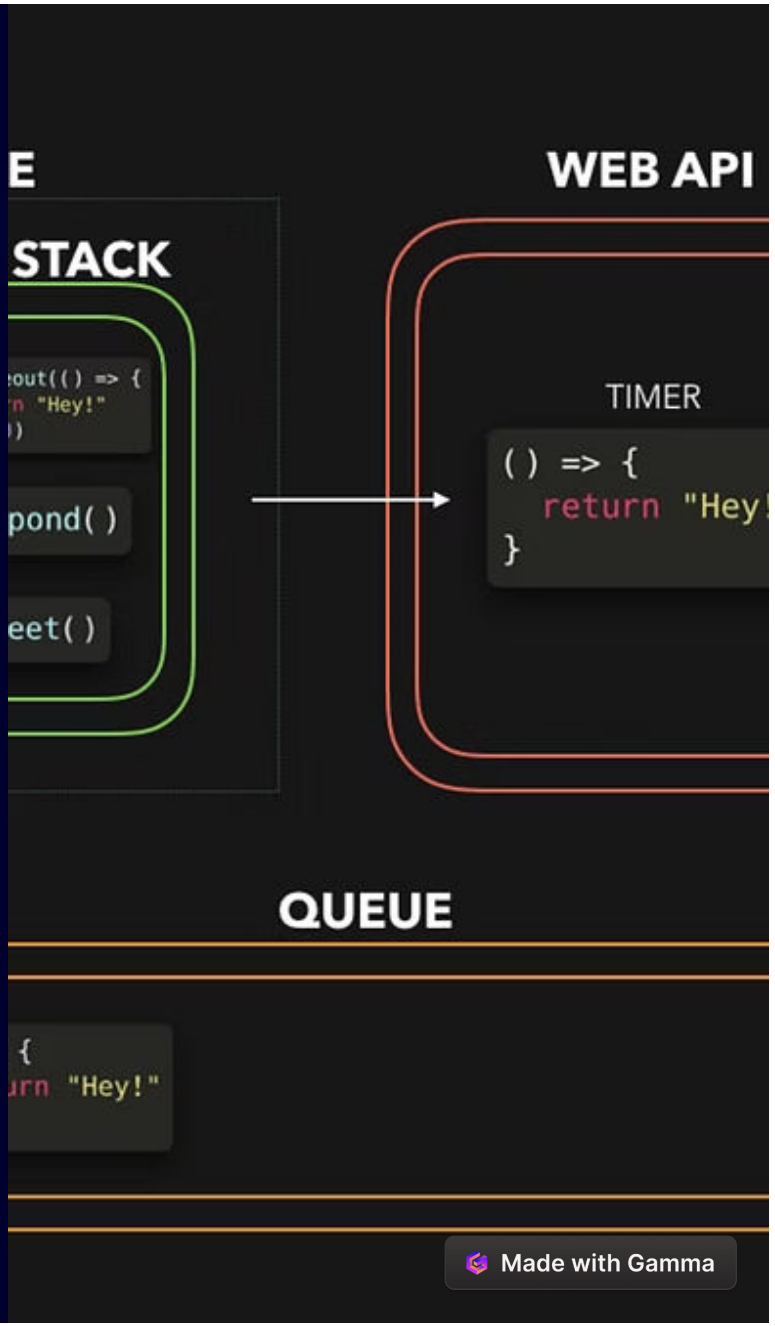
Mayor Eficiencia

Este enfoque contribuye significativamente a la eficiencia del sistema, especialmente en escenarios con un alto volumen de solicitudes concurrentes.

El Event Loop en JavaScript y Node.js

El Event Loop es un concepto fundamental para entender cómo JavaScript y Node.js manejan operaciones asíncronas y no bloqueantes. Aunque JavaScript es un lenguaje de programación de un solo hilo, el Event Loop permite la ejecución de operaciones no bloqueantes de manera eficiente, lo que significa que puede manejar múltiples operaciones en el fondo sin interrumpir el flujo principal de ejecución del programa.

by Carlos Azaustre



Componentes del Modelo de Concurrencia Basado en Eventos

Pila de llamadas (Call Stack)

Registra todas las funciones en ejecución y las retira cuando se completan.

Heap

Región de memoria donde se almacenan los objetos en JavaScript y Node.js.

Cola de eventos (Event Queue)

Almacena eventos o callbacks que esperan ser procesados, como eventos del DOM y solicitudes AJAX completadas.

Funcionamiento del Event Loop en JavaScript

1. ## Monitoreo de la Pila de Llamadas

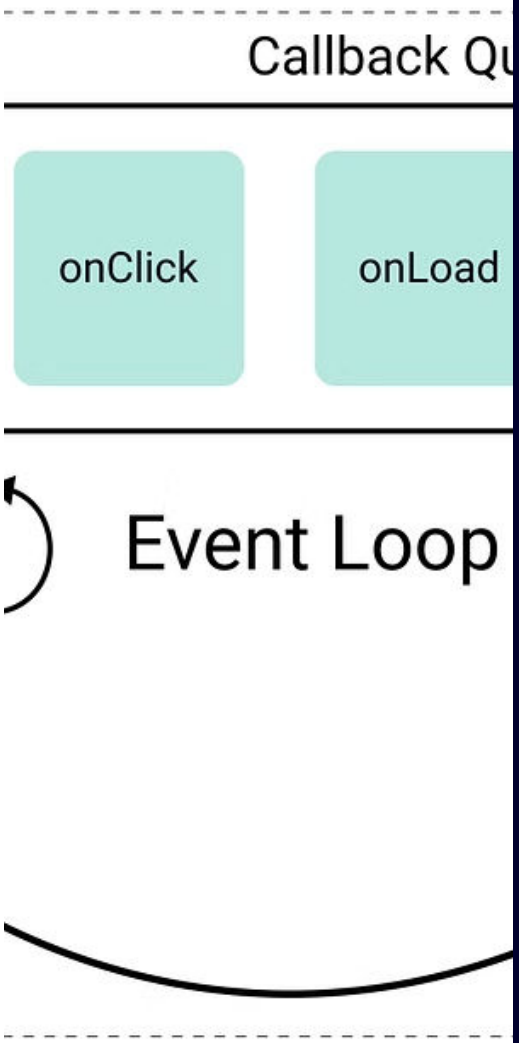
   El Event Loop supervisa la pila de llamadas para ejecutar callbacks cuando está vacía.

2. ## Procesamiento de la Cola de Eventos

   Si la pila de llamadas está vacía, el Event Loop procesará el primer evento de la cola, ejecutando su función de callback.

3. ## Ejecución de Operaciones de I/O no Bloqueantes

   Permite realizar operaciones no bloqueantes, como iniciar una operación y continuar ejecutando otras tareas.



Importancia del Event Loop

Asincronía

El corazón de la asincronía en JavaScript y Node.js.

No Bloqueantes

Permite la ejecución de operaciones no bloqueantes en un entorno de un solo hilo.

Multitarea Eficiente

Gestiona múltiples operaciones simultáneamente, a pesar de su naturaleza de un solo hilo.