Portada que este de la siguiente manera:

Título del trabajo

Nombre del autor

Mes y año

Nombre de institución

Nombre del departamento

Nombre de la asignatura

Abstract o resumen

Tabla de contenidos (indice)

Lista de tablas

Lista de figuras

Marco teórico o Contenido (aplicar Titulos 1, Titulos 2, etc…)

Lista de referencias o bibliografía

Apendice (aquí van las tablas y figuras)

¿Qué es node.js?

Es un entorno de ejecución para JavaScript construido con el motor v8 de Chrome

Recordar que los CPU solo interpretan lenguaje máquina, entonces estos motores lo que hacen es convertir el codigo en JavaScript a lenguaje maquina

Cada navegador tiene su propio motor de JavaScript, por ejemplo Microsoft Edge tiene un motor de JavaScript que se llama Chakra; Firefox tiene un motor de JavaScript que se llama Spidermonkey y Google Chrome tiene un motor de JavaScript que se llama v8

¿Qué es un entorno de ejecución?

Del inglés Runtime System o Runtime Environment es una capa encima del sistema operativo que ejecuta una pieza de soto, esta capa es responsable de seguir como consume la memoria, como accede a las variables, de como ha de pasar los parámetros entre los diferentes procedimientos, además sabe cuando y como debe ejecutar el garbage collector

En el año 2009 Ryan Dahl creó node.js para poder programar en JavaScript sin necesidad de utilizar un navegador, basándose del motor de Google Chrome v8

¿Qué es Chrome v8?

Es un motor opersource de JavaScript desarrollado por the chromium Project para los navegadores Chrome y chromium. Chrome v8 gracias a su tecnología Just in Time Compiler compila codigo JavaScript originalmente JavaScript no fue concebido como un lenguaje compilado sino interpretado

Node.js es el resultado de tomar el motor de JavaScript Chrome v8 para crear un entorno de ejecución y así ejecutar JavaScript del lado del servidor

# Node.js

¿Qué es Node.js?

Node.js es un entorno de tiempo de ejecución de JavaScript que se utiliza para crear aplicaciones escalables del lado del servidor y de red a través de servidores privados virtuales. Ofrece operaciones de entrada/salida (E/S) no bloqueantes y está construido según una arquitectura asincrónica basada en eventos para ayudar a los desarrolladores a crear diversos proyectos de forma eficiente y sencilla.

Fuente de este texto: [~~https://www.hostinger.es/tutoriales/que-es-node-js~~](https://www.hostinger.es/tutoriales/que-es-node-js)

**Historial de cambios importantes**

Una de las características destacadas de Node.js son sus versiones de Node.js de Long Term Support (LTS), que brindan estabilidad y soporte a largo plazo para los desarrolladores. Estas versiones se mantienen durante un período de tiempo prolongado y se centran en la corrección de errores y en la seguridad, lo que es fundamental en entornos de producción.

Node.js 4.x

La versión 4.x de Node.js fue lanzada en 2015 y marcó el comienzo de la estabilidad en la plataforma. Esta es una de las versiones de Node.js que introdujo características clave, como el soporte para ECMAScript 6 (también conocido como ES2015) y mejoras significativas en el rendimiento y la escalabilidad. Node.js 4.x fue una de las versiones de Node.js más importante para muchos desarrolladores que buscaban adoptar Node.js en entornos de producción.

Node.js 6.x

Lanzada en 2016, la versión 6.x de Node.js trajo consigo mejoras significativas en el rendimiento y estabilidad. También incluyó el soporte para más características de ES2015, lo que permitió a los desarrolladores aprovechar las últimas capacidades de JavaScript. Esta, entre las versiones de Node.js, fue un hito importante para aquellos que buscaban un entorno confiable para construir aplicaciones web escalables y de alto rendimiento.

Node.js 8.x

La versión 8.x de Node.js, lanzada en 2017, presentó mejoras adicionales en el rendimiento y la estabilidad. Uno de los aspectos más destacados de esta versión fue la introducción del motor V8 actualizado, lo que conllevó una administración de memoria mejor y un tiempo de ejecución más rápido. Además, se incluyeron características clave, como Async/Await, que simplificaron la escritura de código asíncrono en Node.js.

Node.js 12.x

Esta es una de las versiones de Node.js lanzada en 2019, la versión 12.x de Node.js trajo consigo importantes mejoras de rendimiento y estabilidad. Se destacó por la inclusión de la función de tiempo de espera (timers) más eficiente y mejoras significativas en el rendimiento del módulo HTTP. Además, se realizaron mejoras en la administración de memoria, lo que permitió a los desarrolladores construir aplicaciones más eficientes y escalables.

Fuente de este texto: [~~https://keepcoding.io/blog/recorrido-por-versiones-de-node-js/#Versiones\_de\_Nodejs\_Long\_Term\_Support\_LTS~~](https://keepcoding.io/blog/recorrido-por-versiones-de-node-js/#Versiones_de_Nodejs_Long_Term_Support_LTS)



~~Fuente de esta tabla:~~ [~~https://es.wikipedia.org/wiki/Node.js~~](https://es.wikipedia.org/wiki/Node.js)

En Node.js 18, una API de recuperación global experimental está disponible de forma predeterminada. La implementación proviene de undici y está inspirada en node-fetch, que originalmente se basó en undici-fetch. La implementación se esfuerza por cumplir lo más posible con las especificaciones, pero algunos aspectos requerirían un entorno de navegador y, por lo tanto, se omiten.

La API seguirá siendo experimental hasta que se agregue más cobertura de prueba y los contribuyentes hayan verificado que la API implementa la mayor cantidad de especificaciones posible. Si desea participar, visite el repositorio nodejs/undici y busque problemas etiquetados con fetch.

Fuente original de este texto: [~~https://nodejs.org/en/blog/announcements/v18-release-announce~~](https://nodejs.org/en/blog/announcements/v18-release-announce)

**Características principales**

Un Javascript "sin restricciones"

Con NodeJS tenemos un "Javascript sin restricciones", tal como afirma @\_alejandromg, ya que todo se ejecuta en el servidor y no tenemos que preocuparnos de si nuestro código será compatible o no con distintos clientes. Todo lo que escribas en Node JS y te funcione en tu servidor, estarás seguro que funcionará bien, sea cual sea el sistema que se conecte, porque toda la ejecución de código del servidor se queda aislada en el servidor.

Nota: Este detalle de fiabilidad y compatibilidad, o lo que ha llamado el autor como Javascript "sin restricciones" (por estar ejecutado en un ambiente seguro, del lado del servidor) no deja de ser una ventaja de todos los lenguajes del lado del servidor, como PHP, ASP.NET, JSP, etc.

Pero no sólo eso, el Javascript original tiene algunas estructuras de control que realmente no se utilizan en el día a día, pero que realmente existen y están disponibles en NodeJS. En algunas ocasiones resulta especialmente útil alguna de las mejoras de Javascript en temas como la herencia.

Programación Asíncrona

Éste es un concepto que algunas personas no consiguen entender a la primera y que ahora toma especial importancia, dado que NodeJS fue pensado desde el primer momento para potenciar los beneficios de la programación asíncrona.

Imaginemos que un programa tiene un fragmento de código que tarda cinco segundos en resolverse. En la mayoría de los lenguajes de programación precedentes, durante todo ese tiempo el hilo de ejecución se encuentra ocupado, esperando a que pasen esos cinco segundos, o los que sea, antes de continuar con las siguientes instrucciones. En la programación asíncrona eres capaz de liberar el proceso de modo que los recursos se quedan disponibles para hacer otras cosas durante el tiempo de espera.

Un ejemplo claro de esto es una llamada a un servicio web, o una consulta a la base de datos. Una vez realizada la solicitud generalmente pasará un tiempo hasta que se obtenga la respuesta. Ese tiempo, por corto que sea, dejaría un proceso esperando en la programación tradicional y en la asíncrona simplemente se libera. En NodeJS, o en Javascript en general, cuando esa espera ha terminado y se ha recibido la respuesta, se retomará la ejecución del código. Para definir las acciones a realizar (código a ejecutar) cuando se haya terminado esa espera, se especifica el código mediante funciones llamadas habitualmente "callbacks". Esas funciones contendrán las líneas de código que ejecutar al final de esos procesos de espera, y una vez se ha recibido la respuesta.

La filosofía detrás de Node.JS es hacer programas que no bloqueen la línea de ejecución de código con respecto a entradas y salidas, de modo que los ciclos de procesamiento se queden disponibles durante la espera. Por eso todas las APIs de NodeJS usan callbacks de manera intensiva para definir las acciones a ejecutar después de cada operación I/O, que se procesan cuando las entradas o salidas se han completado.

Nota: Estos callbacks probablemente ya los hayas usado un montón de veces si tienes experiencia con Javascript del lado del cliente, porque se usan en funciones muy habituales como setTimeout(), que también está disponible en NodeJS. Librerías Javascript como jQuery también las usan de manera intensiva.

Realmente Javascript es primeramente síncrono y ejecuta las líneas de código una detrás de otra. Por ese motivo, como resultado de ejecución del código anterior, primero veremos el mensaje "hola" en la consola, luego el mensaje "ya!" y por último, cuando el fichero terminó su lectura, veremos el mensaje "archivo".

Por la forma de ejecutarse el código se puede entender la programación asíncrona. La segunda instrucción (que hace la lectura del archivo) tarda un rato en ejecutarse y en ella indicamos además una función con un console.log ("archivo"), esa es la función callback que se ejecutará solamente cuando termine la lectura del archivo. Por ese detalle, lo último que aparecerá en pantalla es el contenido del fichero. Este detalle es de extrema importancia para entender la programación con NodeJS.

Problema del código piramidal

El uso intensivo de callbacks en la programación asíncrona produce el poco deseable efecto de código piramidal, también conocido habitualmente como "código Espagueti" o "callback hell". Al utilizarse los callbacks, se meten unas funciones dentro de otras y se va entrando en niveles de profundidad que hacen un código menos sencillo de entender visualmente y, por tanto de mantener durante la vida de las aplicaciones.

La solución es hacer un esfuerzo adicional por estructurar nuestro código. Básicamente se trata de modularizar el código, escribiendo cada función aparte e indicando solamente su nombre cuando se define el callback. Podrías incluso definir las funciones en archivos aparte y requiriéndolas con require("./ruta/al/archivo") en el código de tu aplicación. Todo esto lo veremos con detalle en numerosos ejemplos en el Manual de NodeJS, por lo que no te debes de preocupar mucho si todavía no lo entiendes.

Al conseguir niveles de indentación menos profundos estamos ordenando el código, con lo que será más sencillo de entender y también más fácil de encontrar posibles errores. Además, a la larga conseguirás que sea más escalable y puedas extenderlo en el futuro o mantenerlo por cualquier cuestión.

Nota: También en ES6 existen otras herramientas como las promesas que te ayudan a mantener tu código ordenado con la programación asíncrona y el uso de callbacks

Programación orientada a eventos (POE)

Conocemos la programación orientada a eventos porque la hemos utilizado en Javascript para escribir aplicaciones del lado del cliente. Estamos acostumbrados al sistema, que en NodeJS es algo distinto, aunque sigue el mismo concepto.

En Javascript del lado del cliente tenemos objetos como "window" o "document" pero en Node.JS no existen, pues estamos en el lado del servidor.

Eventos que podremos captar en el servidor serán diferentes, como "uncaughtError", que se produce cuando se encuentra un error por el cual un proceso ya no pueda continuar. El evento "data" es cuando vienen datos por un stream. El evento "request" sobre un servidor también se puede detectar y ejecutar cosas cuando se produzca ese evento.

Volvemos a insistir, NodeJS sería un equivalente a PHP, JSP, ASP.NET y entonces todo lo que sean eventos de Node, serán cosas que ocurran en el lado del servidor, en diferencia con los eventos de Javascript común que son del lado del cliente y ocurren en el navegador.

Fuente del contenido: [~~https://desarrolloweb.com/articulos/caracteristicas-nodejs.html~~](https://desarrolloweb.com/articulos/caracteristicas-nodejs.html)

**Ventajas y desventajas**

Ventajas:

Rendimiento y eficacia

El uso de menor cantidad de recursos y la ejecución de procesos sin bloqueo convierte a NodeJs en un entorno con gran rapidez y eficacia.

Sencillez

NodeJs es el entorno que utiliza JavaScript, un lenguaje con una estructura ya establecida y además, sencillo de aprender, y lo mejor es que como programador, seguramente ya manejas JavaScript, por lo que no tienes que aprender un nuevo lenguaje de programación.

Soporte y comunidad

El crecimiento y uso de NodeJs en conjunto con la creación de comunidades en GitHub y StackOverFlow ha hecho que la documentación disponible incremente y por consiguiente, la comunidad de NodeJs se ha mantenido activa en pro de mantener y mejorar el entorno.

NodeJs es Open source

Es un software de código libre, esto quiere decir que no es necesario poseer ninguna licencia para su uso.

Escalable

Al poder procesar tantas conexiones de forma simultánea, NodeJs es el entorno ideal para crear aplicaciones en red escalables con un elevado rendimiento.

Alto rendimiento

Esto ocurre sobre todo en proyectos que se están ejecutando en tiempo real. NodeJs ofrece un rendimiento extraordinario.

Gran utilidad

NodeJs te permite crear la parte back-end y front-end y hasta una app móvil con tan solo este lenguaje.

Desventajas

Módulos en desarrollo

Aún cuando NodeJs posee un sistema llamado NPM (Node Package Manager) de módulos o dependencias, pueden resultar pocos si los comparas con otros sistemas.

Curva de aprendizaje

NodeJs dista mucho de otros sistemas y requiere de más líneas de código, cosa que puede ser mucho más complicada si estás acostumbrado a trabajar con otros sistemas o lenguajes, como, por ejemplo: PHP.

Compatibilidad

Algunos de los planes de alojamiento web no son compatibles con NodeJS, por lo que, en algunas ocasiones para utilizarlo, necesitarás un hosting especializado en NodeJS.

Fuente del contenido: [~~https://es.linkedin.com/pulse/ventajas-y-desventajas-de-aprender-programar-con-~~](https://es.linkedin.com/pulse/ventajas-y-desventajas-de-aprender-programar-con-)

**Entornos de uso**

Herramientas de colaboración en tiempo real

Node.js es un potente motor para crear herramientas de colaboración en tiempo real que van desde aplicaciones de coworking hasta gestión de proyectos, videoconferencias y audioconferencias, y edición de documentos en colaboración.

Esto explica por qué Node.js es la base de muchas herramientas de colaboración populares propiedad de grandes empresas tecnológicas, como Trello para la gestión de proyectos y Slack para los chats de grupo y las comunicaciones de equipos remotos.

La arquitectura asíncrona y basada en eventos de Node.js proporciona la base y encaja perfectamente con la construcción y el escalado de aplicaciones colaborativas.

Aplicaciones de una sola pagina

Las aplicaciones de una sola página (SPA) no son conceptos nuevos en el desarrollo web. Es una palabra de moda que describe un enfoque en el que toda la aplicación cabe en una sola página, con una mejor experiencia para el desarrollador.

Node.js es excelente para las SPA porque maneja las llamadas asíncronas y las cargas de trabajo de entrada/salida (I/O) pesadas de manera más rápida y eficiente. Para que la transición de datos entre Views y el servidor sea fluida, el bucle de eventos de Node.js puede «retrasar» las múltiples peticiones concurrentes de los clientes, lo que permite un procesamiento fluido.

Node.js es también una excelente opción para construir el backend de un SPA con su enfoque basado en datos. Podemos utilizar frameworks de JavaScript como Express.js, Adonis.js o Koa para desarrollar un backend de un SPA complejo y con muchos datos.

Aplicaciones en tiempo real

La construcción de aplicaciones de red en tiempo real es una de las mejores características de Node.js. Puedes hacer de todo, desde construir aplicaciones de chat en tiempo real, como aplicaciones de mensajería instantánea (IM) y de chat de retransmisión por Internet (IRC), hasta construir complejas aplicaciones en tiempo real.

Node.js proporciona características básicas para construir y desplegar aplicaciones en tiempo real y en red utilizando la API de eventos. Puedes crear un objeto, llamado emisor, que emite eventos con nombre periódicamente y que pueden ser escuchados por los manejadores de eventos.

Gracias a su arquitectura basada en eventos, Node.js funciona eficazmente con el protocolo WebSockets, facilitando la comunicación bidireccional en tiempo real entre el servidor y el cliente.

Node.js tiene un excelente soporte de WebSockets a través de librerías como Socket.io y WebSocket-node, que puedes utilizar para crear y desplegar chats en tiempo real de forma rápida y eficaz.

Fuente del contenido: [~~https://kinsta.com/es/blog/node-js-aplicaciones/~~](https://kinsta.com/es/blog/node-js-aplicaciones/)

# Bun

¿Qué es Bun?

Bun es un proyecto que se está siendo desarrollando con miras a garantizar la compatibilidad con las aplicaciones de servidor escritas para Node.js, que admiten la mayor parte de la API de Node.js, incluidos los módulos y estructuras.

En comparación con Node.js y Deno, que utilizan el motor JavaScript V8, Bun se basa en el marco JavaScriptCore desarrollado por Apple para el navegador Safari. Además del alto rendimiento, el uso de JavaScriptCore también redujo el consumo de memoria.

También se admiten las API web tradicionales, incluidos los objetos, ademas de que se proporciona compatibilidad tanto con los módulos Node.js como con los módulos JavaScript (ESM), creados de acuerdo con la especificación ECMAScript 6 y utilizados en aplicaciones web basadas en navegador. Tal como están las cosas, la mayoría de los paquetes npm creados para Node.js se pueden ejecutar en Bun con poca o ninguna modificación.

Fuente: [~~https://blog.desdelinux.net/bun-una-plataforma-javascript-que-presume-de-ser-mas-rapida-que-deno-y-node-js/~~](https://blog.desdelinux.net/bun-una-plataforma-javascript-que-presume-de-ser-mas-rapida-que-deno-y-node-js/)

**Historial de cambios importantes**

Bun v1.0.28

Lanzada el 19 de febrero de 2024 esta versión corrige 6 errores. Corrige errores que afectan a Prisma y Astro, "node:events", "node:readline" y "node:http2". Corrige un error en Bun Shell relacionado con la redirección estándar y corrige errores en "bun:test" con "test.each" y "describe.only".

Bun v1.0.27

Lanzada el 17 de febrero de 2024 esta versión corrige 72 errores. Bun Shell admite el lanzamiento de códigos de salida distintos de cero, transmite cuerpos de respuesta usando generadores asíncronos, mejora la confiabilidad de "fetch()", "cliente http2", "correcciones de Bun.Glob". Corrige una regresión con "bun --watch en Linux". Mejora la compatibilidad con "Node.js"

Bun v1.0.26

Lanzada el 3 de febrero de 2024 corrige 30 errores agrega soporte para consultas de múltiples declaraciones en "bun:sqlite", hace que "bun --watch sea más confiable en sesiones de ejecución más larga", "Bun.FileSystemRouter" ahora admite más de 64 rutas, corrige un error con "expect ().toStrictEqual()", corrige 2 errores con "error.stack", mejora la compatibilidad con "Node.js"

Bun v1.0.25

Lanzada el 21 de enero corrige 4 errores, agrega "vm.createScript". Corrige un fallo en "fs.readFile", un fallo en "Bun.file().text()", un fallo en IPC y un error del transpilador que involucra iguales sueltos.

Fuente: [~~https://bun.sh/blog/bun-v1.0.28~~](https://bun.sh/blog/bun-v1.0.28)

Fuente de fechas: [~~https://github.com/oven-sh/bun/releases?page=1~~](https://github.com/oven-sh/bun/releases?page=1)

**Características principales**

Bundler

Un bundler consolida el código JavaScript y sus dependencias en un único archivo͏, optimizando su eficiencia de carga para navegadores o dentro de aplicaciones Node.js. Node.js carece de un bundler͏ nativo, lo que obliga a los desarrolladores a depender de bundlers de terceros como Rollup, Webpack y ͏Parcel cuando manejan código JavaScript.

En cambio, Bun ofrece un bundler incorporado que requiere una configuración mínima y admite diferentes formatos de módulos. También tiene optimizaciones integradas que permiten una agrupación más rápida. Comparativamente, el bundler de Bun supera a los demás, funcionando 1,75 veces más rápido que ͏esbuild, 150 veces más rápido que Parcel, 180 veces más rápido que Rollup + Terser, y 220 veces más rápido que Webpack.

Motor de pruebas

Las pruebas͏ son una parte integral del desarrollo de software, ya que garantizan la funcionalidad del código e identifican posibles problemas antes de la producción. Bun también in͏corpora un motor de pruebas en su conjunto de herramientas.

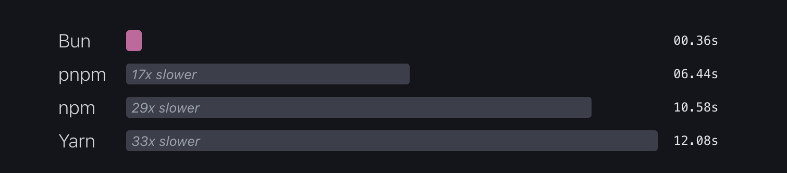
Tra͏dicionalmente, los desarrolladores de Node.js han empleado frameworks de pruebas externos como Jest, que es potente y flexible pero introduce dependencias adicionales y sobrecarga de configuración al proyecto.

Por otro lado, Bun integra su propio motor de pruebas, haciendo hincapié en la velocidad y la compatibilidad. Este enfoque integrado ofrece varias ventajas: Velocidad, compatibilidad, simplicidad.

Gestor de paquetes

El gestor de paquetes compatible con Node.js de Bun es significativamente más rápido que npm, yarn y pnpm. Acelera la velocidad, reduce el uso de disco y minimiza la huella de memoria.

Empleando enlaces simbólicos (symli͏nks), Bun enlaza paquetes͏ para cada proyecto a una ubicación centralizada, eliminando la necesidad de volver a descargar módulos para proyectos posteriores.͏ Aunque el uso de enlaces simbólicos no es totalmente nuevo en los gestores de paquetes, la implementación de Bun supera a las demás.



Fuente: [~~https://kinsta.com/es/blog/bun-sh/#principales-caractersticas-de-bun~~](https://kinsta.com/es/blog/bun-sh/#principales-caractersticas-de-bun)

**Ventajas y desventajas**

Ventajas:

Velocidad

Bun está escrito en Zig, un lenguaje de programación de bajo nivel diseñado para la programación de sistemas, centrándose en el rendimiento, la seguridad y la legibilidad. Diseñado para ser una alternativa moderna a C y C++.

Además, a diferencia de Node.js y Deno, que utilizan el motor JavaScript V8 de Chrome, utiliza el motor JavaScriptCore, que impulsa Safari.

Además de Zig y JavaScript Core, Bun también utiliza otras técnicas, como un asignador de memoria personalizado optimizado para JavaScript y un compilador justo a tiempo (JIT) para optimizar el código a medida que se ejecuta.

En general, la combinación de Zig, JavaScript Core y otras optimizaciones hacen de Bun un tiempo de ejecución de JavaScript muy rápido en comparación con otros tiempos de ejecución.

Compatibilidad con Node.js

Bun está diseñado para ser un reemplazo directo de Node.js y, como tal, es compatible con todas las API de Node.js.

También tiene todos los módulos integrados de Node.js, como crypto, fs, path, etc. Puede verificar los módulos de Node.js disponibles y no disponibles en la documentación de Bun.js.

Además, Bun es un administrador de paquetes compatible con npm. Esto significa que puedes usar Bun para instalar y administrar paquetes de Node.js usando Bun.

Compatibilidad con TypeScript lista para usar

Bun tiene soporte nativo y perfecto para TypeScript, lo que lo convierte en una excelente opción si prefiere o necesita TypeScript en sus proyectos.

TypeScript, una versión extendida y de tipo estático de JavaScript, introduce funciones avanzadas de lenguaje y escritura estática para mejorar el desarrollo de JavaScript.

Con Bun, no hay necesidad de configuración adicional y no se requieren procedimientos de configuración o compilación adicionales para habilitar la funcionalidad TypeScript.

Desventajas:

Recursos limitados

Bun es todavía relativamente nuevo, lo que significa que la comunidad es actualmente pequeña.

No hay muchos recursos que cubran el desarrollo específico de Bun-js, lo que significa que puede resultarle difícil descubrir cómo utilizar el tiempo de ejecución.

Sin embargo, la documentación de Bun es completa y sirve como un valioso punto de referencia. Si encuentra dificultades, también existe la opción de buscar ayuda a través de su canal Discord.

Soporte para Windows

Actualmente, Bun ofrece soporte limitado para el sistema operativo Windows. En el momento de escribir este artículo, solo se admite el tiempo de ejecución en Windows.

El ejecutor de pruebas, el administrador de paquetes y el paquete aún están en desarrollo y, como tales, no funcionan en Windows.

Fuente: [~~https://blog.back4app.com/es/como-implementar-una-aplicacion-bun/#Ventajas\_de\_Bun~~](https://blog.back4app.com/es/como-implementar-una-aplicacion-bun/#Ventajas_de_Bun)

**Entornos de uso**

Los mismos que node.js con otros como ser:

Aplicaciones basadas en la localización

Al igual que Node.js, la asíncrona de Bun y su capacidad para proporcionar actualizaciones en tiempo real lo convierten en la mejor opción para crear aplicaciones basadas en la localización.

En 2020, aumentó el número de aplicaciones basadas en la localización. La mayoría de estas aplicaciones utilizan Node.js; con la mayor simplicidad y rendimiento de Bun, no es tarde para que el número de aplicaciones que requieren localización que se desarrollen usando el entorno Bun, iguale o supere a las desarrolladas con Node.js.

Aplicaciones de streaming

El streaming de aplicaciones es un proceso de descarga de partes de una aplicación a la carta sin sobrecargar el servidor o la máquina local del usuario.

Con la API de streaming, Bun puede manejar fácilmente flujos de datos en tiempo real, que son la característica principal de una aplicación de streaming. Bun también puede proporcionar una rápida sincronización de datos entre el servidor y el cliente, lo que mejora la experiencia del usuario al minimizar los retrasos.

Bun proporciona una interfaz de flujo legible y escribible que puedes utilizar para procesar flujos de datos de manera eficiente.

~~(x2) Fuente: Fuente del contenido:~~ [~~https://kinsta.com/es/blog/node-js-aplicaciones/~~](https://kinsta.com/es/blog/node-js-aplicaciones/)

# LLRT

¿Qué es LLRT?

LLRT es otro tiempo de ejecución de JavaScript que se centra en las demandas específicas de un entorno sin servidor, que no se satisfacen completamente con opciones existentes como Node.js, Bun y Deno, diseñadas para aplicaciones de propósito general. A diferencia de estos tiempos de ejecución, LLRT no incluye un compilador Just-In-Time (JIT), lo que reduce la complejidad del sistema y el tamaño del tiempo de ejecución al tiempo que conserva recursos de CPU y memoria, lo que conduce a tiempos de inicio de aplicaciones más rápidos.

Fuente: [~~https://www.infoq.com/news/2024/02/aws-llrt-lambda-experimental/#:~:text=LLRT%20is%20another%20JavaScript%20runtime,designed%20for%20general-purpose%20applications~~](https://www.infoq.com/news/2024/02/aws-llrt-lambda-experimental/#:~:text=LLRT%20is%20another%20JavaScript%20runtime,designed%20for%20general-purpose%20applications)~~.~~

**Historial de cambios importantes**

Low Latency Runtime es una tecnología reciente, por lo que el historial de cambios importantes no es muy amplio y novedoso, tanto así que en su repositorio de GitHub solo cuenta con casi 400 commits.

Algunos de las versiones más recientes que han sido subidos y verificadas son:

#190 Fix make release target for aarch64 lanzado el 21 de febrero de 2024

#185 Fix object.toJSON() when stringifying lanzado el 20 de febrero de 2024

#166 Fix/xml parser lanzado el 16 de febrero de 2024

Fuente:[~~https://github.com/awslabs/llrt/commits/main/~~](https://github.com/awslabs/llrt/commits/main/)

**Características principales**

LLRT requiere que uno agrupe su código y dependencias en un único archivo .js y, por lo tanto, elimina todas las búsquedas del sistema de archivos que ocurren durante la fase de resolución del módulo en Node, por lo que la aceleración proviene principalmente de esta limitación.

Además, preempaquetan, precompilan (en código de bytes) y precargan bits de AWS SDK y luego los comparan con un tiempo de ejecución de propósito general como Node o Bun que tiene que cargar las bibliotecas de AWS SDK como archivos JS e interpretarlos al iniciar la aplicación. Apuesto a que de aquí proviene el 99% de los beneficios de rendimiento.

~~(x2) Fuente:~~ [~~https://www.infoq.com/news/2024/02/aws-llrt-lambda-experimental/#:~:text=LLRT%20is%20another%20JavaScript%20runtime,designed%20for%20general-purpose%20applications~~](https://www.infoq.com/news/2024/02/aws-llrt-lambda-experimental/#:~:text=LLRT%20is%20another%20JavaScript%20runtime,designed%20for%20general-purpose%20applications)~~.~~

**Ventajas y desventajas**

Ventaja

LLRT (Low Latency Runtime) es un tiempo de ejecución de JavaScript liviano diseñado para abordar la creciente demanda de aplicaciones sin servidor rápidas y eficientes. LLRT ofrece un inicio hasta 10 veces más rápido y un costo general hasta 2 veces menor en comparación con otros tiempos de ejecución de JavaScript que se ejecutan en AWS Lambda.

Desventaja

Al igual que Bun, LLRT al ser nuevo, implica que la comunidad es actualmente pequeña. Por lo que no hay muchos recursos que cubran el desarrollo específico de LLRT, lo que significa que puede resultarle difícil descubrir cómo utilizarlo.

Fuente: [~~https://github.com/awslabs/llrt~~](https://github.com/awslabs/llrt)

Entornos de uso

Personaliza el tiempo de ejecución en Amazon Linux 2023 y empaqueta el binario de inicio de LLRT junto con tu código JS.

Utiliza AWS SAM para configurar una función Lambda con LLRT.