What is Node.js?

Node.js is a **free, open-source, cross-platform** JavaScript **runtime environment** that lets developers create servers, web apps, command line tools and scripts. This definition says it all. Let's break it down into pieces.

- **Node.js is free:** unfortunately this can't be any clearer.
- **Node.js is open-source:** This means that the source code for Node.js is publicly available. And it's maintained by contributors from all over the world. The Node.js contribution guide shows you how to contribute.
- **Cross-platform:** Node.js is not dependent on any operating system software. It can work on Linux, macOS, or Windows.
- **Runtime environment:** When you write JavaScript code in your text editor, that code cannot perform any task unless you execute (or run) it. And to run your code, you need a runtime environment.

Why use Node.js?

Node.js provides a runtime environment outside of the browser. It's also built on the Chrome V8 JavaScript engine. This makes it possible to build back-end applications using the same JavaScript programming language you may be familiar with.

In the browser, most of the time what you are doing is interacting with the DOM, or other Web Platform APIs like Cookies. Those do not exist in Node.js, of course. You don't have the document, window and all the other objects that are provided by the browser.

And in the browser, we don't have all the nice APIs that Node.js provides through its modules, like the filesystem access functionality.

Another big difference is that in Node.js you control the environment. Unless you are building an open source application that anyone can deploy anywhere, you know which version of Node.js you will run the application on. Compared to the browser environment, where you don't get the luxury to choose what browser your visitors will use, this is very convenient.

This means that you can write all the modern ES2015+ JavaScript that your Node.js version supports. Since JavaScript moves so fast, but browsers can be a bit slow to upgrade, sometimes on the web you are stuck with using older JavaScript / ECMAScript releases. You can use Babel to transform your code to be ES5-compatible before shipping it to the browser, but in Node.js, you won't need that.

Another difference is that Node.js supports both the CommonJS and ES module systems (since Node.js v12), while in the browser, we are starting to see the ES Modules standard being implemented.

In practice, this means that you can use both require() and import in Node.js, while you are limited to import in the browser.

History

Node.js was initially written by Ryan Dahl in 2009, about 13 years after the introduction of the first server-side JavaScript environment, Netscape's LiveWire Pro Web. The initial release supported only Linux and Mac OS X. Its development and maintenance was led by Dahl and later sponsored by Joyent.

Dahl criticized the limited capability of Apache HTTP Server to handle many (10,000+) concurrent connections, as well as the dominant programming paradigm of sequential programming, in which applications could block entire processes or cause the creation of multiple execution stacks for simultaneous connections.

Dahl demonstrated the project at the inaugural European JSConf on November 8, 2009. Node.js combined Google's V8 JavaScript engine, an event loop, and a low-level I/O API.

In January 2010, a package manager was introduced for the Node.js environment called npm. The package manager allows programmers to publish and share Node.js packages, along with the accompanying source code, and is designed to simplify the installation, update and uninstallation of packages.

In June 2011, Microsoft and Joyent implemented a native Windows version of Node.js. The first Node.js build supporting Windows was released in July 2011.

In January 2012, Dahl yielded management of the project to npm creator Isaac Schlueter. In January 2014, Schlueter announced that Timothy J. Fontaine would lead the project.

In December 2014, Fedor Indutny created io.js, a fork of Node.js created because of dissatisfaction with Joyent's governance as an open-governance alternative with a separate technical committee. The goal was to enable a structure that would be more receptive to community input, including the updating of io.js with the latest Google V8 JavaScript engine releases, diverging from Node.js's approach at that time.

The Node.js Foundation, formed to reconcile Node.js and io.js under a unified banner, was announced in February 2015. The merger was realized in September 2015 with Node.js v0.12 and io.js v3.3 combining into Node v4.0. This merge brought V8 ES6 features into Node.js and started a long-term support release cycle. By 2016, the io.js website recommended returning to Node.js and announced no further io.js releases, effectively ending the fork and solidifying the merger's success.

In 2019, the JS Foundation and Node.js Foundation merged to form the OpenJS Foundation.

On September 6, 2023, Node.js 20.6.0 was released. The update brought the addition of built-in support for .env files, the unflagging of import.meta.resolve, the introduction of a new node:*module* API register for module customization hooks and a new initialize hook. Additionally, the module customization load hook now supports CommonJS, and Node.js C++ add-ons have gained experimental support for cppgc (Oilpan), which is a C++ garbage collection library for V8.

Installation of Node.js

Node.js can be installed in different ways. This post highlights the most common and convenient ones. Official packages for all the major platforms are available at https://nodejs.org/download/.

One very convenient way to install Node.js is through a package manager. In this case, every operating system has its own. Other package managers for macOS, Linux, and Windows are listed in https://nodejs.org/download/package-manager/.

nvm is a popular way to run Node.js. It allows you to easily switch the Node.js version, and install new versions to try and easily rollback if something breaks. It is also very useful to test your code with old Node.js versions.

You might run through some problems. The most commons could be not configuring the environment variables properly or having an incompatible version.

Run Node.js scripts from the command line

The usual way to run a Node.js program is to run the globally available node command (once you install Node.js) and pass the name of the file you want to execute.

If your main Node.js application file is app.js, you can call it by typing: **$ node app.js**.

Other commands include:

$ node --version: to check your version of Node.js

$ node –help: to see eligible commands

Tip: What is V8?

V8 is Google's open source high-performance JavaScript and WebAssembly engine, written in C++. It is used in Chrome and in Node.js, among others. It implements ECMAScript and WebAssembly, and runs on Windows, macOS, and Linux systems that use x64, IA-32, or ARM processors. V8 can be embedded into any C++ application.