

What is Node.js? The JavaScript runtime explained

Node.js is a lean, fast, cross-platform JavaScript runtime environment that is useful for both servers and desktop applications

By Martin Heller

Contributing Editor, InfoWorld

APR 6, 2020 3:00 AM PDT

Scalability, latency, and throughput are key performance indicators for web servers. Keeping the latency low and the throughput high while scaling up and out is not easy. Node.js is a JavaScript runtime environment that achieves low latency and high throughput by taking a “non-blocking” approach to serving requests. In other words, Node.js wastes no time or resources on waiting for I/O requests to return.

In the traditional approach to creating web servers, for each incoming request or connection the server *spawns* a new *thread of execution* or even *forks* a new *process* to handle the request and send a response. Conceptually, this makes perfect sense, but in practice it incurs a great deal of overhead.

[[Also on InfoWorld: The 10 best JavaScript editors reviewed](#)]

While spawning *threads* incurs less memory and CPU overhead than forking *processes*, it can still be inefficient. The presence of a large number of threads can cause a heavily loaded system to spend precious cycles on thread scheduling and context switching, which adds latency and imposes limits on scalability and throughput.

Node.js takes a different approach. It runs a single-threaded event loop registered with the system to handle connections, and each new connection causes a JavaScript *callback function* to fire. The callback function can handle requests with non-blocking I/O calls, and if necessary can spawn threads from a pool to execute blocking or CPU-intensive operations and to load-balance across CPU cores. Node's approach to scaling with callback functions requires less memory to handle more connections than most competitive architectures that scale with threads, including Apache HTTP Server, the various Java application servers, IIS and ASP.NET, and Ruby on Rails.

Node.js turns out to be quite useful for desktop applications in addition to servers. Also note that Node applications aren't limited to pure JavaScript. You can use any language that transpiles to JavaScript, for example TypeScript and CoffeeScript. Node.js incorporates the Google Chrome V8 JavaScript engine, which supports ECMAScript 2015 (ES6) syntax without any need for an ES6-to-ES5 transpiler such as Babel.

Much of Node's utility comes from its large package library, which is accessible from the npm command. NPM, the Node package manager, is part of the standard Node.js installation, although it has its own website.

Some JavaScript history

In 1995 Brendan Eich, then a contractor to Netscape, created the JavaScript language to run in Web browsers—in 10 days, as the story goes. JavaScript was initially intended to enable animations and other manipulations of the browser document object model (DOM). A version of JavaScript for the Netscape Enterprise Server was introduced shortly afterwards.

RECOMMENDED WHITEPAPERS

 The Cloud Connectivity Cookbook

 ROI of SD-WAN-as-a-Service



IDC: Why Data Strategy is the Glue for App Modernization

The name JavaScript was chosen for marketing purposes, as Sun's Java language was widely hyped at the time. In fact, the JavaScript language was actually based primarily on the Scheme and Self languages, with superficial Java-like semantics.



SponsoredPost Sponsored by ReliaQuest
Looking Ahead for Security Control
Testing: The Role of Attack Simulations

Initially, many programmers dismissed JavaScript as useless for “real work” because its interpreter ran an order of magnitude more slowly than compiled languages. That changed as several research efforts aimed at making JavaScript faster began to bear fruit. Most prominently, the open-source Google Chrome V8 JavaScript engine, which does just-in-time compilation, inlining, and dynamic code optimization, can actually outperform C++ code for some loads, and outperforms Python for most use cases.

The JavaScript-based Node.js platform was introduced in 2009, by Ryan Dahl, for Linux and MacOS, as a more scalable alternative to the Apache HTTP Server. NPM, written by Isaac Schlueter, launched in 2010. A native Windows version of Node.js debuted in 2011.

Joyent owned, governed, and supported the Node.js development effort for many years. In 2015, the Node.js project was turned over to the Node.js Foundation, and became governed by the foundation’s technical steering committee. Node.js was also embraced as a Linux Foundation Collaborative Project. In 2019, the Node.js Foundation and JS Foundation merged to form the OpenJS Foundation.

[[Also on InfoWorld: The 6 best JavaScript IDEs reviewed](#)]

Basic Node.js architecture

At a high level, Node.js combines the Google V8 JavaScript engine, a single-threaded non-blocking event loop, and a low-level I/O API. The stripped-down example code shown below illustrates the basic HTTP server pattern, using ES6 arrow functions (anonymous Lambda functions declared using the fat arrow operator, =>) for the callbacks.

SponsoredPost Sponsored by Adobe



The Future of Work: Four Things You Don't Want to Overlook

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

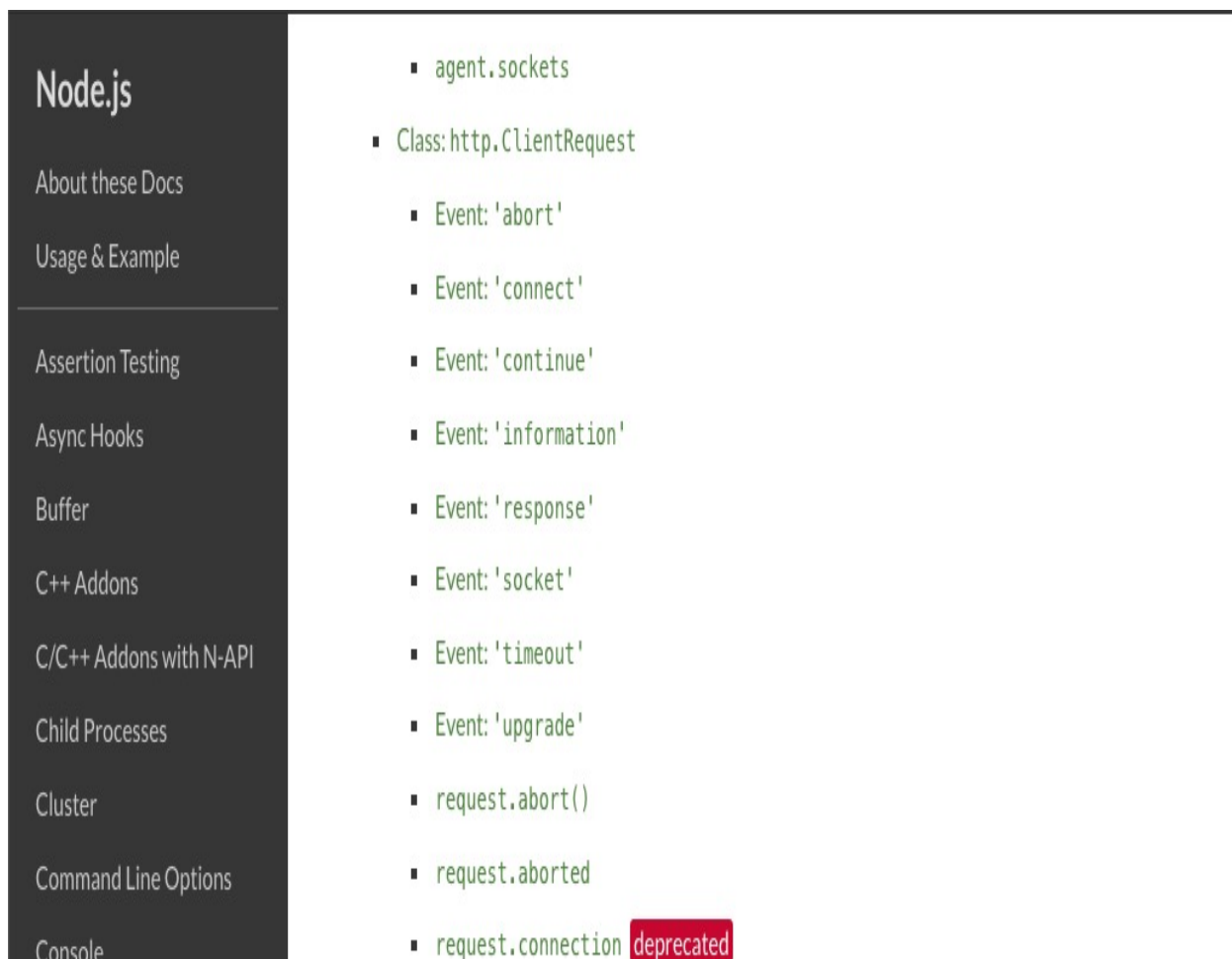
The beginning of the code loads the HTTP module, sets the server `hostname` variable to `localhost` (127.0.0.1), and sets the `port` variable to 3000. Then it creates a server and a callback function, in this case a fat arrow function that always returns the same response to any request: `statusCode` 200 (success), content type plain text, and a text response of "Hello World\n". Finally, it tells the server to listen on `localhost` port 3000 (via a socket) and defines a callback to print a log message on the console when the server has started listening. If you run this code in a terminal or console

using the `node` command and then browse to `localhost:3000` using any Web browser on the same machine, you'll see "Hello World" in your browser. To stop the server, press `Control-C` in the terminal window.

Note that every call made in this example is asynchronous and non-blocking. The callback functions are invoked in response to events. The `createServer` callback handles a client request event and returns a response. The `listen` callback handles the listening event.

The Node.js library

As you can see at the left side the figure below, Node.js has a large range of functionality in its library. The HTTP module we used in the sample code earlier contains both client and server classes, as you can see at the right side of the figure. The HTTPS server functionality using TLS or SSL lives in a separate module.



The image shows a screenshot of the Node.js documentation website. On the left is a dark sidebar with a white 'Node.js' logo at the top. Below the logo are several menu items: 'About these Docs', 'Usage & Example', 'Assertion Testing', 'Async Hooks', 'Buffer', 'C++ Addons', 'C/C++ Addons with N-API', 'Child Processes', 'Cluster', 'Command Line Options', and 'Console'. The main content area on the right is white and displays a list of modules and classes. The list includes: 'agent.sockets', 'Class: http.ClientRequest', 'Event: 'abort'', 'Event: 'connect'', 'Event: 'continue'', 'Event: 'information'', 'Event: 'response'', 'Event: 'socket'', 'Event: 'timeout'', 'Event: 'upgrade'', 'request.abort()', 'request.aborted', and 'request.connection' with a red 'deprecated' badge next to it.

- `agent.sockets`
- Class: `http.ClientRequest`
 - Event: `'abort'`
 - Event: `'connect'`
 - Event: `'continue'`
 - Event: `'information'`
 - Event: `'response'`
 - Event: `'socket'`
 - Event: `'timeout'`
 - Event: `'upgrade'`
 - `request.abort()`
 - `request.aborted`
 - `request.connection` **deprecated**

[Crypto](#)[Debugger](#)[Deprecated APIs](#)[DNS](#)[Domain](#)[ECMAScript Modules](#)[Errors](#)[Events](#)[File System](#)[Globals](#)[HTTP](#)[HTTP/2](#)[HTTPS](#)[Inspector](#)[Internationalization](#)[Modules](#)[Net](#)[OS](#)[Path](#)[Performance Hooks](#)[Policies](#)[Process](#)[Punycode](#)[Query Strings](#)

- `request.end([data[, encoding]][, callback])`
- `request.finished` **deprecated**
- `request.flushHeaders()`
- `request.getHeader(name)`
- `request.maxHeadersCount`
- `request.path`
- `request.removeHeader(name)`
- `request.reusedSocket`
- `request.setHeader(name, value)`
- `request.setNoDelay([noDelay])`
- `request.setSocketKeepAlive([enable][, initialDelay])`
- `request.setTimeout(timeout[, callback])`
- `request.socket`
- `request.writableEnded`
- `request.writableFinished`
- `request.write(chunk[, encoding][, callback])`
- Class: `http.Server`
 - Event: `'checkContinue'`
 - Event: `'checkExpectation'`
 - Event: `'clientError'`
 - Event: `'close'`
 - Event: `'connect'`
 - Event: `'connection'`
 - Event: `'request'`

One inherent problem with a single-threaded event loop is a lack of vertical scaling, since the event loop thread will only use a single CPU core. Meanwhile, modern CPU chips often expose eight or more cores, and modern server racks often have multiple CPU chips. A single-threaded application won't take full advantage of the 24-plus cores in a robust server rack.

You can fix that, although it does take some additional programming. To begin with, Node.js can spawn child processes and maintain pipes between the parent and children, similarly to the way the system `popen(3)` call works, using `child_process.spawn()` and related methods.

The `cluster` module is even more interesting than the `child process` module for creating scalable servers. The `cluster.fork()` method spawns worker processes that share the parent's server ports, using `child_process.spawn()` underneath the covers. The cluster master distributes incoming connections among its workers using, by default, a round-robin algorithm that is sensitive to worker process loads.

Note that Node.js does not provide routing logic. If you want to maintain state across connections in a cluster, you'll need to keep your session and login objects someplace other than worker RAM.

[Also on InfoWorld: TypeScript vs. JavaScript: Understand the differences]

The Node.js package ecosystem

The NPM registry hosts more than 1.2 million packages of free, reusable Node.js code, which makes it the largest software registry in the world. Note that most NPM *packages* (essentially folders or NPM registry items containing a program described by a `package.json` file) contain multiple *modules* (programs that you load with `require` statements). It's easy to confuse the two terms, but in this context they have specific meanings and shouldn't be interchanged.

NPM can manage packages that are local dependencies of a particular project, as well as globally installed JavaScript tools. When used as a dependency manager for a local project, NPM can install, in one command, all the dependencies of a project through the `package.json` file. When used for global installations, NPM often requires system (sudo) privileges.

You don't *have* to use the NPM command line to access the public NPM registry. Other package managers such as Facebook's Yarn offer alternative client-side experiences. You can also search and browse for packages using the NPM website.

Why would you want to use an NPM package? In many cases, installing a package via the NPM command line is the fastest and most convenient to get the latest stable version of a module running in your environment, and is typically less work than cloning the source repository and building an installation from the repository. If you don't want the latest version you can specify a version number to NPM, which is especially useful when one package depends on another package and might break with a newer version of the dependency.

For example, the Express framework, a minimal and flexible Node.js web application framework, provides a robust set of features for building single and multi-page, and hybrid web applications. While the easily clone-able Expresscode repository resides at <https://github.com/expressjs/express> and the Express documentation is at <https://expressjs.com/>, a quick way to start using Express is to install it into an already initialized local working development directory with the `npm` command, for example:

```
$ npm install express --save
```

The `--save` option, which is actually on by default in NPM 5.0 and later, tells the package manager to add the Express module to the dependencies list in the `package.json` file after installation.

Another quick way to start using Express is to install the executable *generator express* (1) globally and then use it to create the application locally in a new working folder:

```
$ npm install -g express-generator@4
$ express /tmp/foo && cd /tmp/foo
```

With that accomplished, you can use NPM to install all of the necessary dependencies and start the server, based on the contents of the package.json file created by the generator:

```
$ npm install
$ npm start
```

It's hard to pick highlights out of the million-plus packages in the NPM, but a few categories stand out. Express is the oldest and most prominent example of Node.js frameworks. Another large category in the NPM repository is JavaScript development utilities, including browserify, a module bundler; bower, the browser package manager; grunt, the JavaScript task runner; and gulp, the streaming build system. Finally, an important category for enterprise Node.js developers is database clients, of which there are more than 8,000, including popular modules such as redis, mongoose, firebase, and pg, the PostgreSQL client.

[Keep up with the latest developments in software development, cloud computing, data analytics, and machine learning with the InfoWorld Daily newsletter]

To summarize, Node.js is a cross-platform JavaScript runtime environment for servers and applications. It is built on a single-threaded, non-blocking event loop, the Google Chrome V8 JavaScript engine, and a low-level I/O API. Various techniques, including the cluster module, allow Node.js apps to scale beyond a single CPU core. Beyond its core functionality, Node.js has inspired an ecosystem of more than a million packages that are registered and versioned in the NPM repository and can be installed using the NPM command line or an alternative such as Yarn.

Martin Heller is a contributing editor and reviewer for InfoWorld. Formerly a web and Windows programming consultant, he developed databases, software, and websites from 1986 to 2010. More recently, he has served as VP of technology and education at Alpha Software and chairman and CEO at Tubifi.

UNITED STATES ▾

Follow



Copyright © 2020 IDG Communications, Inc.

- Stay up to date with InfoWorld’s newsletters for software developers, analysts, database programmers, and data scientists.
- Get expert insights from our member-only Insider articles.

YOU MAY ALSO LIKE

Recommended by



Tim O'Reilly: the golden age of the programmer is over



Microsoft adds a new Linux: CBL-Mariner



The shifting market for PostgreSQL