Alvin Charity

October 18, 2022

Deno vs. Node.js — Five Major Differences You Should Consider

deno

Node.js has long been considered the default JavaScript framework for modern developers. With package management, modules, and the ability to use TypeScript, Node has evolved alongside the JavaScript/TypeScript ecosystem. But when Deno was launched in 2018, developers gained access to a new type of framework that puts TypeScript first and seeks to correct some of the major pitfalls of using Node.

So how do you choose between the frameworks? This article compares Deno and Node in terms of security, third-party packaging, and other factors so you can determine which one is best for your projects.

What Is Deno?

Deno is a newer framework built by Ryan Dahl, who also developed Node.js. Deno was designed to have a better package management system and more secure runtime than Node. Additionally, Dahl wanted to make Deno a more modern platform by forgoing many of the legacy libraries available in Node.

In order to offer a simple, secure runtime, Deno uses web platform standards, ships with a single executable file, and offers built-in development tooling. It has an alternate set of standard modules that are guaranteed to work within the Deno runtime. Deno can also run TypeScript files natively without any additional configuration.

What Is Node?

Described as "an asynchronous event-driven JavaScript runtime," Node is an extremely popular JavaScript runtime, considered the standard by most JavaScript developers. Node can be used to build server-side or desktop applications, and it's been available to JavaScript users since 2009.

Node has been a hit among developers because it offers a module system to use third-party code as a library, as well as the `require()` function to import modules from other files, Node packages, or third-party libraries.

Similarities between Node and Deno

On a basic level, Node and Deno are pretty similar—both offer package management, a system for importing modules and other third-party code, and a solid standard library.

Additionally, both Node and Deno can use TypeScript, which is a popular superset of JavaScript developed by Microsoft. It's slightly more difficult to get started with TypeScript when using Node, though.

Both Deno and Node use the V8 JavaScript engine created by Google. This means that in-browser performance should be just about the same between the two frameworks.

Differences between Node and Deno

Where Node and Deno differ is that Node is built using C and C++, while Deno is built using the newer language Rust. Rust is a multi-paradigm, low-level programming language originally built by Mozilla to be a memory-safe and statically typed alternative to other system level languages. This allows the underlying architecture of the Deno binary to catch internal errors at compile time. These improvements affect Deno's internals only, not the performance of your code. The performance of TypeScript and JavaScript code using Deno is about the same as code running in Node.js.

On a higher level, Node uses npm for package management, while Deno technically does not have a package manager. Unlike Node, Deno can import packages directly from their source—the deno.land CDN, GitHub, or any URL that resolves to a compatible module.

How the Frameworks Compare

Being a newer runtime, Deno is greatly overshadowed by the Node ecosystem. However, Deno has a few distinct advantages over Node and may be a better choice for many projects.

This comparison will point out which areas each framework excels at and which features you might want to avoid. It will focus on the following factors:

- Formatting
- Module system
- Security
- Package management
- TypeScript integration

Formatting

Node.js does not have a built-in formatter or linter. However, you have several options to choose from when you need to add a formatting step to your project. ESLint, JSHint, StandardJS, and Prettier have been available for a while and offer battle-tested options for formatting and linting your code. There are newer options as well; Rome and quick-lint-js offer an improved developer experience and are designed to work out of the box with minimal or no configuration required.

Deno, on the other hand, has a built-in formatter that eliminates the need for an external tool to install and configure. The `deno fmt` command works more similarly to something like ESLint and allows you to auto-format .js, .ts, and .md files by passing the respective filenames to the `deno fmt` command. Formatting in Deno can be handled in your terminal like this:

```
# format markdown and any compatible code samples
deno fmt readme.md

# format a directory of code
deno fmt dist/
```

```
# format stdin
cat squareRoot.ts | deno fmt -
```

Which Is Better?

If you are looking for a formatting command to replace something like ESLint, Deno may be a good choice. It's built into the framework, so you won't need to install any extra tooling, and it can even format code in Markdown code blocks.

Keep in mind that Deno may not have enough formatting rules for your particular project. Still, the methods applied by `deno fmt` make more sense in the context of creating an application or publishing content to the web.

Module System

Node uses CommonJS and ECMAScript modules, which can be imported into other files by using the built-in `require()` function for CommonJS and `import()` for ECMAScript. The specific use of each style of module varies slightly and offers a small degree of interoperability. For instance, you can use `require()` to load ECMAScript modules into your codebase.

In Node, most modules are added to your project via npm. In Deno, however, these modules can be loaded from anywhere. The standard library is loaded using the `import` function, for example:

```
import { assertEquals } from "https://deno.land/std@0.153.0/testing/asserts.ts";
assertEquals("hello", "hello");
```

In addition to the Deno CDN, you can import modules from GitHub or any URL that resolves to a valid package.

Which Is Better?

Both frameworks allow you to build scripts and applications either with your own code or by including third-party packages from their respective package managers. The flexibility of the Deno module and import system wins out over Node here, as you can pull code from sites like GitHub and any CDN or registry you host for yourself or your team. This allows you to import modules on the fly, without downloading or installing anything.

Security

Node users have the benefit of standard security practices that prevent attacks like cross-site request forgery (CSRF) and cross-site scripting (XSS). These practices include monitoring logs, properly handling exceptions, and validating user input.

Deno has a set of command line flags that can be applied at runtime to allow specific features for individual scripts. These features are deactivated by default, and you must manually apply a flag for any items that may be needed by your script. Flags include:

- `--allow-env` to use variables from your terminal environment
- `--allow-hrtime` for formatting options for printing timestamps
- `--allow-net` to allow network access, which Deno does not allow by default
- `--allow-read` to allow your Deno script to read files
- `--allow-run` to let your scripts execute imported code
- `--allow-write` to allow your Deno script to write files to the system

Which Is Better?

The restrictiveness of Deno's security module takes some time to get used to, but Deno is the top choice for a more secure environment.

Package Management

npm is the default package manager for Node and allows you to download external packages as modules. These modules can exist either locally to your project or globally as a command line application. npm also allows you to add your packages to the registry for others to use in their projects by including a `package.json` file. Installing can look like this:

```
# install a package to the current directory
npm install <package>

# install a package globally
npm install -g <package>
```

Deno does not have a package manager. Instead, it allows you to import modules via its CDN, public or private GitHub repositories, or any site that supports HTTP. This open system means you can pull packages from just about anywhere, which could create security issues for enterprise users. Importing can look like this:

```
// import a local file
import { squareRoot } from './squareRoot.ts';

// import from a CDN
import { squareRoot } from "https://deno.land/x/squareRoot/squareRoot.ts";
```

Which Is Better?

When it comes to ease of use, Deno beats Node. However, the benefit of loading any URL as a module may be outweighed by the possibility that your imports will be altered at their source or in mid-flight. Node may be best for projects where you only need to import the Node standard library and a few other trusted sources.

Still, considering the security issues that sometimes arise with npm packages, Deno is the better choice if you need runtime safety for your application. For absolute certainty about the security of your program, you may need to audit external packages from both npm and Deno.

With npm, you can use the built-in `npm-audit` command that compares your dependencies with a database of known vulnerabilities. The command also lets you fix any issues it finds. Similar tools like Retire.js, Snyk, and Mend Renovate provide deep integration with other languages, frameworks, and runtimes, allowing full code vulnerability coverage for your project. Your use case will determine which tool is the best for your code.

TypeScript Integration

With a bit of work, you can have TypeScript and Node working together in no time. If you have Node and npm installed, just run `npm install -g typescript` to install the TypeScript language globally on your machine. Generate a TypeScript config file by running `tsc --init`. From there, you can use the TypeScript compiler `tsc` to compile your .ts files.

Deno has a TypeScript compiler built into the binary, and with it, you can run any TypeScript file without any extra work. The compiler built into Deno uses the official compiler created by Microsoft and a Rust language library called SWC. This allows Deno to compile, type check, and run your code without converting it to JavaScript first.

Which Is Better?

Working with TypeScript in Deno requires no configuration, so if you are a fan of TypeScript and want to check out a new framework, Deno is right for you. However, if you rely on the Node standard library, Deno's version of the built-ins is not complete. You may find yourself with a difficult-to-trace error in your codebase.

Should You Use Node or Deno?

When it comes to choosing between Node and Deno, there isn't a clear winner. Both systems have their advantages and disadvantages. Node is a much more mature framework that developers have used successfully since 2009, but with that maturity comes security issues that may compromise sensitive applications.

Deno is a newer framework created specifically to fix some issues that developed in Node. Deno has an easier import system and uses TypeScript by default, with no extra effort on your part. But it also may not have the battle-tested stability of Node. Additionally, if you rely on the Node standard library, you may be left wanting more from the Deno standard library.

Having said that, there are a few scenarios where Node is a better choice than Deno:

- You're not interested in using TypeScript or the TypeScript ecosystem.
- The codebase requires specific Node libraries or packages only available on npm.
- Most of your work involves legacy Node projects.

Likewise, there are situations where Deno is the clear winner over Node:

- Your projects are mostly greenfield work where you have the flexibility to choose your tooling.
- You have a legacy project that can benefit from TypeScript features.
- You prefer the flexibility of Deno's package management over Node.

Conclusion

Node and Deno both have a lot to offer for your projects. As the comparison in this article points out, there are situations where one framework is a clear choice over the other. Keep your specific use case and the needs of your organization in mind when you make your selection.