

Front-end Development 6 min read

Introduction to Bun

Learn about Bun, a new runtime for JavaScript and TypeScript apps that boasts some impressive performance and productivity improvements over Node.js.



Alvin Charity
Published August 23, 2022

Table of contents

Differences between Bun and Node Bun is a new runtime for JavaScript and TypeScript applications. Created by **Jarred Sumner**, Bun has undergone rapid development since its inception in early 2021, and saw its first beta release in July of 2022.

Differences between Bun and Deno

Creating a TypeScript application using the Bun runtime

Conclusion

Bun aims to provide an “all-in-one” runtime for building JavaScript and TypeScript applications. Bun’s built-in bundler replaces the need for Webpack or esbuild, and supports a variety of file-types including Javascript, Typescript, **tsx**, **jsx**, **css**, and **svg**. Bun also includes a built-in web server for use in local development, which eliminates the need for a plugin like webpack-dev-server. Additionally, common web APIs like **fetch** and WebSockets are included in its standard library. Bun also implements the Node module resolution algorithm, which makes migrating applications from Node easier when compared to another alternative runtime like Deno.

Bun advertises some impressive performance gains over Node, with more than 2x performance improvements across several **benchmarks**. Performance gains are achieved via the use of Apple’s **JavaScriptCore** engine, the same JS engine used in Apple Safari, which is used in place of Google’s **V8 engine** utilized by Node and Deno.

Bun can be installed on Linux (x64), MacOS (x64 and Silicon), and Windows Subsystem for Linux.

Note on MacOS support: Bun works best with MacOS 11 or higher. The commands **bun create** and **bun run** will sometimes crash on earlier versions of MacOS. Code in this article was tested on both MacOS 10.13 (High Sierra) and 12.5 (Monterey).

Note on Linux support: Bun on Linux requires **Linux 5.6** or higher.

Smoke Test / Run #4298755

Passed - View Details
2 hours ago

Create Issue

1	Navigate to Linear – A better way to build pr... https://linear.app/	0:00	✓
2	Click on Log in link	0:01	✓
3	Visit Linear https://linear.app/login	0:02	✓
4	Click on Continue with Email button	0:02	✓
5	Select email textbox and enter value tedd@tedd.com reflect.com	0:03	✓

Tired of flaky end-to-end tests?

Create fast and reliable tests for anything that runs in a browser.

[Learn more](#)

Differences between Bun and Node

Node is by-far the most popular server-side JavaScript runtime. Since one of Bun’s design goals is an easy migration path from Node, there are quite a few similarities between these respective runtimes. For instance, Bun’s standard API is mostly binary compatible with Node, and as mentioned earlier, Bun implements the same module resolution algorithm that’s used by Node. However, there are a few major differences:

TypeScript is not a first-class citizen in Node, requiring the use of third-party libraries which can be tricky to configure correctly. By contrast, Bun supports Typescript (e.g. `.ts` or `.tsx` files) will transpile them to JavaScript without any special configuration. Bun uses Apple's JavaScriptCore engine instead of Google's V8 engine. Bun is written with the `zig` programming language rather than `C++` and `C`. This means Bun can take advantage of low-level memory control and lack of hidden control flow. However `zig` itself is a younger programming language and future updates to the language may affect Bun usage or performance. Bun allows you to use npm modules via the `bun install` command, which is an alternative to `npm install`. Bun installations are up to 100 times faster than using npm.

However, Bun has not yet implemented a few key features, such as JavaScript and CSS minification, TypeScript Decorators, and Web Streams. More information about planned features can be found on [the Bun README](#), and a Bun roadmap can be found in [this issue](#).

Differences between Bun and Deno

Deno, another alternative TypeScript/JavaScript runtime created by the original author of Node, seeks to resolve what he considers to be major design flaws of Node and npm. This means that Deno significantly diverges from Node when it comes to both the runtime as well as its approach to package management and module dependency. Bun by contrast is designed more as a "drop-in" replacement for Node. Major differences between Bun and Deno include:

- Modules in Deno do not use npm, and are instead imported directly via URL. There is however an upcoming change that will add [support for npm modules](#). Bun uses Node's module resolution algorithm, albeit with some significant reported performance improvements.
- Deno does not use a `package.json` file, whereas Bun has support for `package.json`, including running tasks via its built-in task runner.
- Node and Bun apps have permission to access network and filesystem resources by default. This is something that Ryan Dahl, the creator of Node and Deno, considers a design flaw of Node, and is why the Deno runtime does not grant network or filesystem access by default. To access the network and filesystem in Deno, developers must use the `Deno.permissions` construct to query, request, or revoke permissions at runtime. [There are no immediate plans](#) to add runtime security checks to Bun. The Bun maintainers plan to add static code analysis to the runtime to try to mitigate potential security issues in apps via automatic dead-code removal.
- Deno is written using the `rust` programming language but continues to rely on the V8 JavaScript engine written with `C++`. Processes such as file reading and writing (or `io`), TCP Networking, and process management are implemented with `rust`. Bun is implemented in the `zig` programming language and utilizes JavaScriptCore as its JS engine.

Creating a TypeScript application using the Bun runtime

Creating a simple TypeScript application with Bun is easy. Start by creating a new Bun project using `bun create` (optional).

```
bun create ./draw-card
```

You can also pass a Github repository in `user/repository_name` format:

```
bun create unforswearing/introduction_to_bun draw-card
```

Create an `index.ts` file containing the following code

```
let api = "https://deckofcardsapi.com/api/deck/new/draw/";

async function drawCard() {
  let cardJson = await fetch(api).then((resp) => resp.json());
  console.log(cardJson.cards[0].value, "of", cardJson.cards[0].suit);
}

drawCard();
```

Finally, use `bun index.ts` to run the `drawCard()` function

```
bun index.ts
```

The above command retrieves and prints a random card from the deck, e.g. `JACK OF HEARTS`.

You can also set up a `package.json` file for running the `index.ts` script. Lets say your `package.json` file contains the following script:

```
{
  "scripts": {
    "drawCard": "bun index.ts"
  }
}
```

You can use `bun run drawCard` to execute the `index.ts` file.

For more complex applications, `bun create` offers behavior specific flags for compatibility with other package managers. For example, if you prefer to use `yarn` for package management, run `bun create -yarn ./project_directory`. [These flags](#) can ease the development process for new Bun users, and allow for more predictable behavior when migrating a Node project to Bun.

Conclusion

In this article you learned about Bun, how Bun compares to Node and Deno, how to migrate a Node project to Bun, and how to create and deploy a TypeScript application using Bun. Bun is a young framework with a very large scope so there is a lot about the software that this article does not cover.

For more information about Bun head over to [bun.sh](#) or visit the [Bun GitHub repository](#). Also be sure to check out the [Awesome Bun repository](#) for additional articles, tools, videos, and more.

Code examples in this article can be found [at this Github repository](#).