Amit Ashwini Follow
VP of Marketing at Zibtek. Former Director of Marketing at CognitiveClouds. I help top startups and companies build remarkable web, mobile and tablet products.
Oct 6, 2017 · 7 min read

What Are The Best Practices For Node.JS Development?

What Are The Best Practices For Node.JS Development

Node.js has played a significant role in the widespread adoption and popularity of JavaScript. The popularity of Node.js, based on Google's V8 JavaScript engine can be attributed to the speed and efficiency it provides the development environment. The platform makes it easy enough to get started on but once you move beyond the core functionalities of your app, knowing how to deal with errors and how to best structure your code can get difficult. How you fix these issues, whether it's a patchwork or based on the right practices makes all the difference between a launch disaster and a rock solid production application. Having said this let's take a look at a few Node.js tips will help you get the most out of your applications.

Begin all projects with NPM init

Since you can organize them a lot better with npm scripts and Node, stop writing bash scripts. Npm's init command, inferring common properties from the working directory, will scaffold out a valid package.json for your project. Most people know NPM as a method of installing dependencies, but it is a lot more than this. Node Package Manager is at the base of almost all the deployment systems for node.js. The new package.json it creates for you, allows you to add a whole lot of metadata to assist others working on the project, by letting them have the same setup as you. This simple, reliable package management has allowed the Node ecosystem to grow extremely well.

Use environment variables

Don't clutter your project with environment-specific config files. Rather, take advantage of environment variables. Utilize environment variables

even for the early stages of a project to ensure there's no leakage of sensitive info, and basically to build the code properly from the beginning. Configuration management is always a matter of a discussion. How does one decouple their code from the databases, services, etc. which it has to use during node.js development, production, and QA? The method advised in Node.js is to use environment variables and look up the values from process.env in your code.

Use a style guide

A style guide will make you a more productive node.js developer since it reduces unessential choice. Most of us have spent the good half of a day changing the spaces to tabs, reformatting the braces to be on other lines, basically trying to make sense of the code when we open a new file from another project we've never worked on before. The problem here is a mixture of opinionated developers and no company standard style guide. It's a lot easier to understand code on a code base when it's written in a consistent style. It further reduces the cognitive overhead of if you should be writing with spaces or tabs. If the style is dictated and enforced then all of sudden, the code base becomes a lot more manageable. Sometimes it's better to go for an existing set of guidelines and follow them, this way you won't have to come out with your own rules either. Be it from Airbnb, Google, jQuery or Standard JS.

What Are The Best Practices for Node.js Development

Familiarize yourself with JavaScript best practices

It's a good idea first to make sure you've learned the best practices of JavaScript, before developing out your Node.js strategy. If you don't have time for a full immersion or you're in need of a review of JavaScript, then some of the benchmarks you should learn are asynchronous programming, scope, data types, function arguments, function & objects in JavaScript, and callbacks.

Go Asynchronous

Node.js is by design single-threaded, which means lots of synchronous components can potentially lock up the entire application.

Synchronous functions in JavaScript block any other code from running until they complete. Nevertheless, the synchronous code makes the flow of your application logic easy to understand. On the other hand, asynchronous structures like promises bring back a lot of that reasoning while keeping your code free from blockages. Even though you could be avoiding synchronous methods in your code, it's still possible to inadvertently use an external library which has a blocking call, and this can severely reduce performance. The best way around this is to use asynchronous APIs in your code, especially in performance critical sections. Keep this in mind, when choosing third party modules as well, to ensure an external library doesn't revert to synchronous calls.

Handle errors

Managing exceptions well is important for any app, and the best way to deal with errors is to use asynchronous structures. For instance, promises provide a .catch() handler which will propagate all errors to be treated, cleanly.

Ensure your app automatically restarts

Okay, so you follow the best practice to handle errors. Sadly, an error from a dependency somehow still brings your app down. This is when the use of a process manager becomes necessary to make sure the app recovers gracefully from a runtime error. Now, if the entire server you're running on goes down, you again need it to restart. Here, you want minimal downtime. You want your application to restart as soon as the server is alive again.

Cluster your app to improve reliability and performance

Node.js runs in a single process, by default. Ideally, you want one process for each CPU core so you can distribute the workload across all the cores. Hence improving the scalability of web apps handling HTTP requests and performance in general. In addition to this, if one worker crashes, the others are still available to handle requests. One thing to bear in mind is, each process is standalone, they don't share resources or memory. The cluster will hand you easy flexibility for the future, even if you're just running a single process on small hardware today. Testing is the right way to understand the ideal number of clustered

processes for your app. However, you can start with the reasonable defaults offered by your platform, with a simple fallback.

Require all your dependencies up front

Always load all your dependencies and configure them upfront. This way, you'll know from the beginning if there is any problem, not four or five hours after your application has gone live in production.

Use Gzip compression

Gzip is a software application used for file decompression and compression. Today, most clients and servers support gzip. The server compresses the response before sending it to the browser when a gzip compatible browser requests a resource, reducing time lag and latency. It can improve the overall performance of your application if you use gzip both when you respond to clients and when you make requests to remote servers.

Keep your code light

It's important to keep your Node.js code base as compact as possible to reduce latency and speed things up. Here are some questions worth asking in the development stage: "Why are we using this framework? Is it worth the overhead?", "Do we really need this module?", "Can we attempt this in a simpler way?" One way to optimize application performance is by minifying and concatenating multiple JS files into one. For instance, if your application has six JavaScript files the browser will make six separate HTTP requests to fetch them. A better approach would be to minify and concatenate those six files into one streamlined one, to avoid the block and wait time.

Employ client side rendering

Due to robust new client side MVC frameworks like BackboneJS and AngularJS, it has become a lot easier for developers to create dynamic, one-page apps. These frameworks expose APIs which send JSON responses, rather than through the server, directly to the client. For every request, this returns an HTML page if you let Node.js render server-side. Making use of client side rendering in your Node.js environment will dramatically reduce latency and save bandwidth.

Only git the significant bits

Most applications are composed of both necessary files and generated files. You should avoid tracking anything that's created when using a source control system like git. It leads to unnecessary bloat and noise in your git history when you track generated files. In fact, since certain dependencies are native and should be compiled, checking them in will render your app less portable since you'll be providing builds from just a single, and perhaps incorrect, architecture.

Application Monitoring

It is critical to get notified when something goes wrong with your application, especially in production applications. You don't want to check your social media feed and see thousands of angry users telling you your servers are down, or your application has been broken for a few hours now. So employing something to monitor and alert you to critical issues or abnormal behavior is important.

Test your code

This point is an obvious one, but seriously though, testing will save you on many occasions. Testing does get in the way of your speed of development. However, once the first few production issues occur on a project with no tests, you'll wish you had run a few tests in the first place. Whatever stage you are on, in a project, it's never too late to introduce testing. Start small, start simple.

Hope this proves useful.

. . .

Originally published on Product Insights Blog from CognitiveClouds: Top ***Node JS Development Company***

This story is published in The Startup, Medium's largest entrepreneurship publication followed by 293,189+ people.

Subscribe to receive our top stories here.