netguru

Should you use Node.js in your application?

Quick Guide for CEOs

Date: 13th Sep 2017

Table of contents

Choosing backend technology is one of the most important decisions that every CEO and CTO have to make. It determines how fast a product can be shipped to market, what is the total cost and how big pain maintenance will be.

JavaScript has been one of the most popular client-side programming languages and a commonly used front-end web development tool. However, it has also gained ground in different areas of application and on distinct platforms, such as React Native, Appcelerator Titanium, Apache Cordova/PhoneGap, NativeScript, and Node.js, which is totally different from other commonly used JavaScript frameworks.

Node.js is an application runtime environment that allows you to write server-side applications in Javascript. Thanks to its unique I/O model, it excels at the sort of scalable and real-time situations we are increasingly demanding of our servers. It's also lightweight, efficient, and the ability to use Javascript on both frontend and backend opens new possibilities. It comes as no surprise that so many big companies have leveraged it in production, including Walmart, Netflix, Medium, LinkedIn or Groupon.

Why so many big players chose Node.js to use at the back-end? Here are the top benefits of that environment, but also some drawbacks you should consider before you make the choice. Wrong decision may cost you money, so choose wisely.

When to Use Node.js

Internet of Things

Since 2012, when the popularity of IoT rose dramatically, Node.js has become one of the preferred solutions for enterprises and organizations seeking to develop their private and public IoT systems. The most obvious advantage of Node.js as a back-end for such networks is its ability to process multiple concurrent requests and events emitted by thousands or even millions of devices on the network. The avalanche of requests and data coming from IoT devices does not block Node.js servers thanks to their event-driven architecture and asynchronous processing suitable for I/O-heavy operations on the IoT network. This makes Node.js fast as an application layer between these devices and databases used to store data originating from them.

In addition, IoT developers working in data-intensive scenarios can leverage the low resource requirements of Node.js. Low memory requirements allow for the easy integration of Node.js as software into single-board controllers such as Arduino, widely used for building digital devices that make up IoT systems. Finally, the Node community has been an early adopter of the IoT technology, creating over 80 packages for Arduino controllers and multiple packages for the Pebble and Fitbit wearable devices widely used in IoT systems.

Real-Time Chats

Node.js provides all basic functionalities for building real-time chats of any complexity. In particular, Node has a powerful Event API that facilitates creating certain kinds of objects (“emitters”) that periodically emit named events “listened” by event handlers. Thanks to this functionality, Node.js makes it easy to implement server-side events and push notifications widely used in instant messaging and other real-time applications.

Node’s event-based architecture also works well with the WebSockets protocol that facilitates a fast two-way exchange of messages between the client and the server via one open connection. By installing WebSockets libraries on the server and the client side, you can implement real-time messaging that has lower overheads and latency, and faster data transfer than most other, more conventional, solutions.

In Node, you have excellent support for WebSockets via such libraries as socket.io, ws, or websocket-node, thanks to which you can easily deploy efficient real-time chats and applications. With socket.io, for example, all you have to do to create a basic live chat is to install the socket.io library on

the server and the client, and create event emitters and broadcasters that will push messages across the WebSockets open connection. This basic functionality can be achieved with just a few lines of code.

Complex Single-Page Applications

Node.js is a great fit for Single-Page Applications (SPAs) thanks to its efficient handling of asynchronous calls and heavy I/O workloads characteristic of these applications. Node.js's event loop allows to "delay" multiple concurrent requests from the client, which ensures smooth transitions between views and seamless data updates. Also, Node.js works well with data-driven SPAs, where the server acts as a backend that provides data to the client whereas the client does all the HTML rendering.

Also, Node.js is good for SPAs because it is written in the same language (JavaScript) as many popular JavaScript frameworks (Ember, Meteor, React, Angular) used in building SPAs. Since both Node.js and browsers use JavaScript, there is less context switching between them, and developers can use the same data and language structures and modular approaches both on the server and the client side. This results in faster development and better maintainability of your SPAs. The above advantages of Node.js have been leveraged by such famous SPAs as Netflix, Linkedin, and Medium, to name a few.

Real-Time Collaboration Tools

As in the case of the real-time chats, Node's asynchronous and event-based architecture is a great fit for collaboration apps. In these applications, many events and I/O requests occur concurrently. For example, several users can edit the same paragraph, comment, post messages, and attach media. Changes to one piece of content might be applied only after a cascade of events, where each step depends on the previous one.

Node's WebSockets and Event API will ensure that heavy I/O operations performed by many users do not make the server hang and that all server-side events and data are sent back to the client on time. By emitting push notifications to the client, Node.js will also instantly update the collaboration environment so that all users have a single and coherent representation of the application. This is precisely the reason why the team of the project management application Trello uses the Node.js stack. The engineering team of Trello decided that Node.js would be great to instantly propagate a lot of updates and hold a lot of open connections, thanks to its event-driven and non-blocking architecture. Among other real-time collaboration apps built on Node.js, we should also mention Yammer, a freemium social networking service facilitating private communication in enterprises.

Streaming apps

Unlike remote server apps, in application streaming, the program is executed on the end user's local machine. Application streaming allows for downloading parts of the application on demand without overloading the server and the local computer. Initially, only certain parts of the application needed for bootstrap are downloaded, whereas the remainder can be downloaded in the background if needed. When the application is completely downloaded, it can function without any network connection at all. In case you want to save some data in your account, the application can initiate server requests. Similarly, server events can update your local application without too much network traffic overhead.

Node.js is excellent for the development of such streaming applications thanks to its native Stream API. In particular, Node.js has an interface of readable and writable streams that can be processed and monitored very efficiently. Stream instances are basically Unix pipes that allow transmitting parts of the app's executable code to the local machine while keeping a connection open for new components to download on demand. Streams allow users to pipe requests to each other, and stream data directly to its final destination. As a bonus, streams do not require caching and temporary data – just an open connection to stream application data from one place to another.

Microservices Architecture

Node.js is an excellent solution for developing microservices and creating easy-to-use APIs to connect them. In particular, the Node.js repository features Express and Koa frameworks, which make it easy to mount several server instances for each microservice and design routing addresses for them. Node.js with Express allows for creating highly flexible modules responsible for specific parts of your application.

In addition, Node.js can be easily integrated with Docker and will thus allow you to encapsulate microservices in hermetic containers to avoid any conflicts between the application development environments used in each of them. Using Node.js for microservices also benefits from Node's lightweight requirements. Node.js with microservices significantly reduces application deployment time and enhances efficiency, maintainability, and scalability of your applications. Microservices architecture also helps manage the division of labor in your engineering teams efficiently, enabling them to work on specific tasks without affecting other parts of your application.

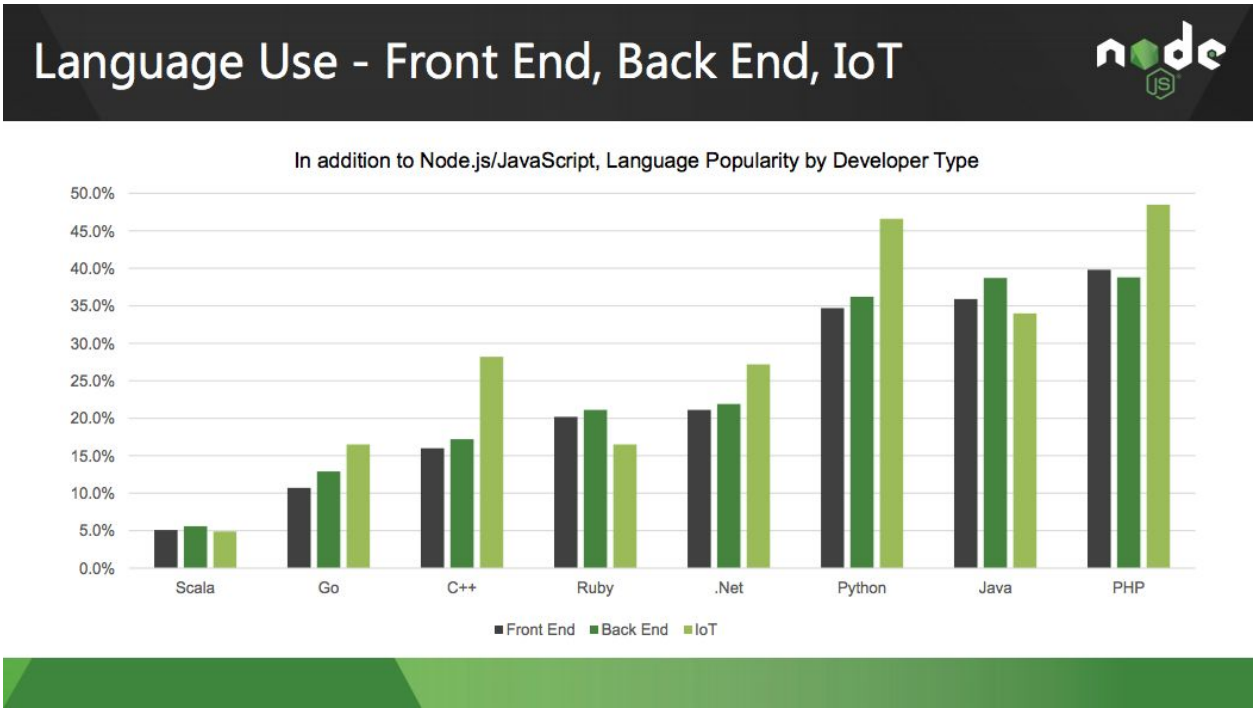
These benefits have been successfully leveraged by PayPal, world's leading online payment system, which has used Node.js to power its microservices architecture since 2013. PayPal modularized its application stack and split the development process into many microservices, and thus organized their teams to work on them more efficiently. PayPal was able to scale Node.js so that

multiple teams could work on the same project. The results of this transition were stunning. PayPal's Node.js app could be built twice as fast and with fewer people. The company has managed to reduce its code base and improve performance, where a single core Node app could handle twice as more rps (requests per second) than 5 Java apps used by PayPal before.

Pros of Using Node.js for Back-end Development

Easy to Learn

According to Node.js's 2016 User Survey Javascript is one of the most popular programming languages for front-end development. Nearly every front-end developer is familiar with this universal language. Therefore, it is much easier for them to switch to using Node.js at the back-end. It requires less effort and less time to learn and work with, even for a junior Javascript programmer.



Freedom in Building Apps

While Ruby on Rails is a framework that imposes rules and guidelines of developing software in a particular way, Node.js gives you much more space and freedom in doing it your own way. Node.js is completely unopinionated, meaning you start building everything from scratch. It can execute basic tasks, but gives you only the bare minimum from a fresh install, making it less restricted.

Active Community

The Node.js community is a very active and vibrant group of developers who contribute to constant improvement of Node.js. Thanks to the cooperation of JavaScript programmers and their input to the community you get access to a ton of ready solutions, codes in Github and many more possibilities. Even though, it is still at a relatively early stage of development, the community is dynamically evolving and its members go the extra mile to provide others with best and reliable solutions.

Simultaneous Request Handling

Node.js provides the non-blocking IO system that lets you process numerous requests concurrently. The system makes simultaneous request handling much better than in other languages like Ruby or Python. Incoming requests are queued up and executed sequentially in a fast manner. In effect your app will take up much less system RAM, achieve high scalability levels and in a result will be perform faster.

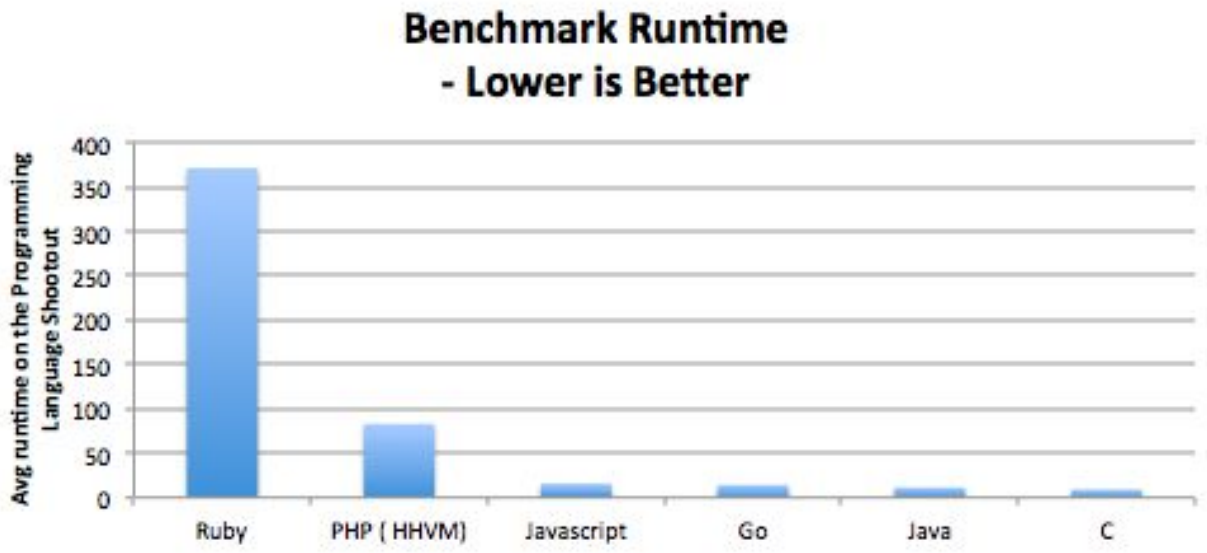
Fast Server-side Solution

When we think about the strengths of Node.js, the first thing that comes to mind is its innovative application of asynchronous and event-based programming and non-blocking I/O maximizing the usage of a single CPU and computer memory. Thanks to this architecture, Node.js servers can process more concurrent requests than conventional multi-threaded servers. Node.js event loop, which does not block program execution under I/O-heavy workflow, improves runtime performance making Node one of the fastest server-side solutions around.

One Language on Both the Front- and the Back-end

Node.js is written in Javascript, a programming language that dominates front-end development. Some of the most popular front-end frameworks, such as Ember, React, and Angular, leverage power and flexibility of JS to create dynamic Web 2.0 applications. With Node.js installed

server-side, developers can use the same programming language across the web stack. The same language at the front end and the back end means you need a smaller and more efficient team, which can communicate better and as result deliver tasks much faster.



Scalable Solution

Node.js is a very scalable solution too. Node clusters and workers are abstractions that can spawn additional Node.js processes depending on the workload of your web application. Limited only by the number of CPUs at their disposal, you can easily scale your Node applications to fully functional enterprise solutions.

10 Top Companies that Used Node.js in Production

Node.js has numerous advantages over other technologies and probably for that reason many big players came to use Node.js in their applications. What are these famous companies that trusted Node.js in creating their apps? We have some high profile examples.

Netflix

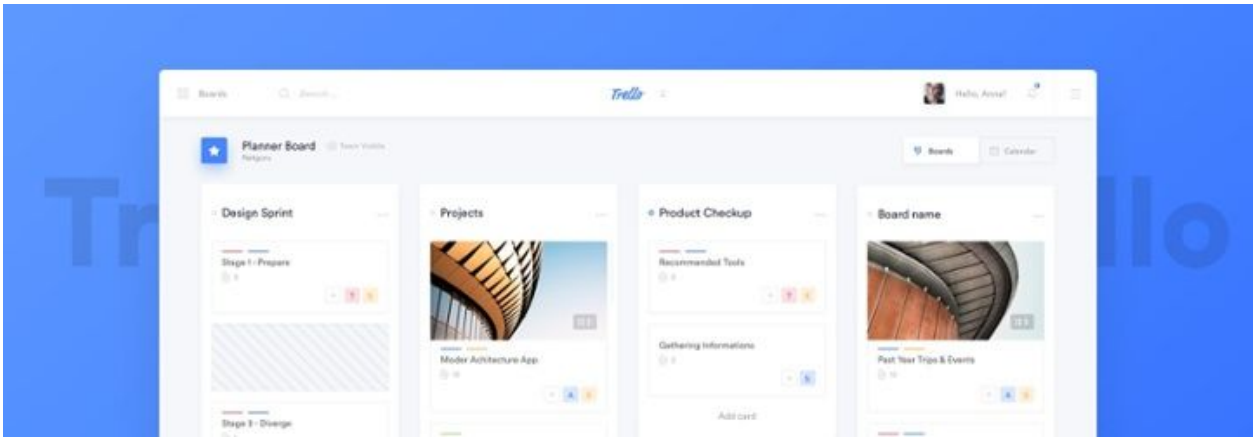
Netflix is the world's leading provider of streaming media and video-on-demand. It's a data-driven platform that uses a massive amount of A/B testing to build rich experience for its 93 million subscribers worldwide. Great numbers of unique packages every push cycle create a problem of conditional dependencies and app scalability. That's why the company decided to leverage the lightweight and fast Node.js. One of the most important results of this was a 70-percent reduction in startup time.

You can learn more about their work with Node.js from this presentation.

Trello

Trello is a project management app that we embrace in our daily work at Netguru. The server side of Trello was built in Node.js. An event-driven, non-blocking server was a good solution for an instant propagation of updates, which required holding a lot of open connections. Node.js also became useful when the company was prototyping a tool for a single-page app. It was a quick way for them to get started and make sure that the everything was going in the right direction.

Read about the whole tech stack in Trello on Frog Creek blog.



PayPal

PayPal, a worldwide online payments system, has also moved their backend development from Java to JavaScript and Node.js. Beforehand, the engineering teams at the company were divided into those who code for the browser and those who code for the application layer, and it didn't work perfectly. Then, full-stack engineers came to the rescue, but that model wasn't ideal too. Adopting Node.js solved their problems, as it allowed for writing the browser and the server applications in the same programming language – JavaScript. As a resulted, the unified team is able to understand problems at both ends and then more effectively react to customer needs.
If you are curious how PayPal moved to Node.js check out this blog post.

LinkedIn

LinkedIn, the world's biggest business and employment-oriented social networking service, also trusted Node.js, and last year they moved their mobile app backend from Ruby on Rails to Node.js. Even though at that time it was still a very immature environment, it proved to be a smart move for the company. The new app is two to ten times faster than its predecessor, and it is also extremely lightweight. On top of that, the development was quite quick.
See how they made a change in a blog post at VB.

Walmart

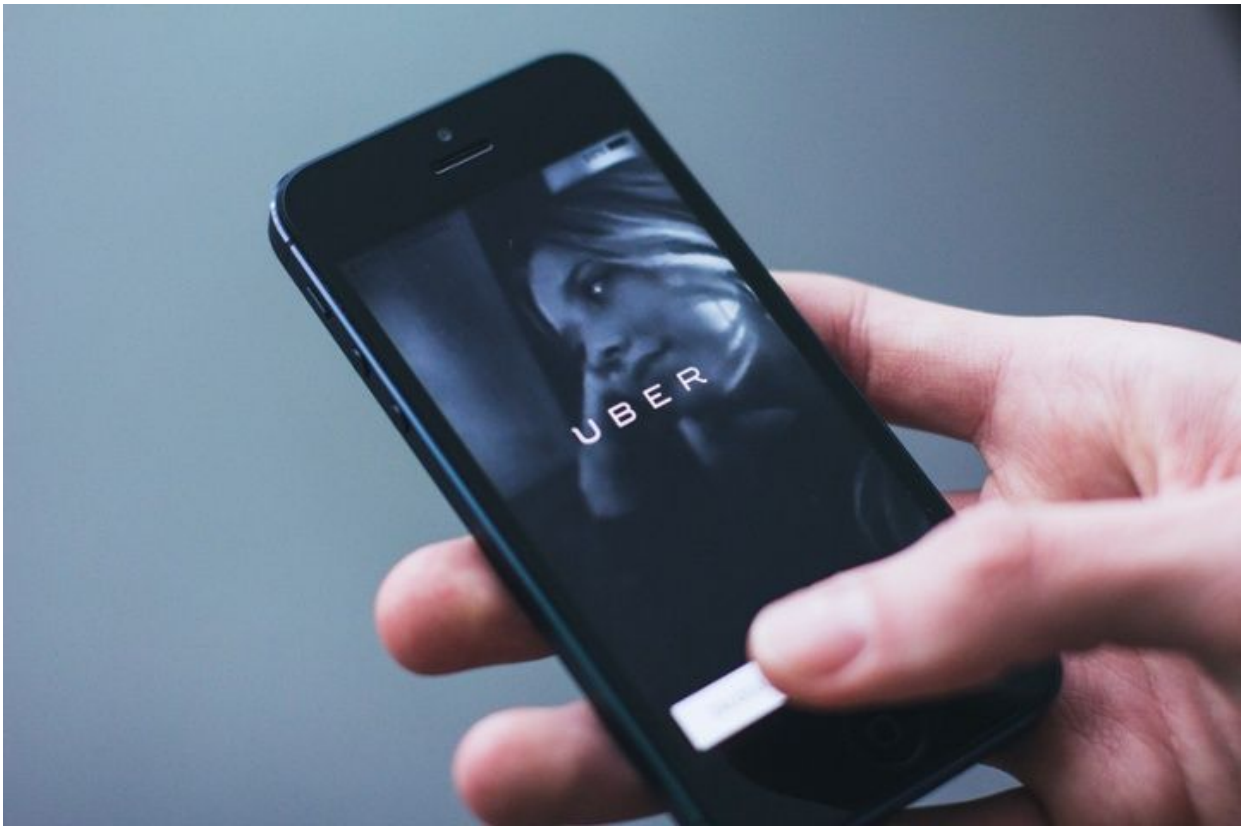
Walmart is the world's largest retailer, and it is now making headway into the online retail market. The giant has jumped on the bandwagon of working with Node.js – a relatively new and very trendy technology despite the risk that was involved in such a move. The company re-engineered the mobile app to provide sophisticated features on the client side. Walmart especially appreciated the Node.js' famous asynchronous I/O and its single-threaded event loop models that can efficiently handle concurrent requests.

Watch the presentation on lessons learned when moving to Node.js to learn more about Walmart's successful move.

Uber

Uber, a platform that connects drivers with customers in need of transportation (and now also food delivery) services, leverage many tools and programming languages in the engineering of their app. Uber's tech stack is evolving constantly, and they've since introduced new technologies that proved more efficient in certain areas. That said, Node.js is still one of the crucial cogs in the company's operation, as it enables scaling up in line with the constantly rising demand for their services.

Read more about Uber's relationship with Node.js.



Medium

Medium is an online publishing platform that uses Node.js for their web servers. Even though at first glance, the web app might seem like a simple HTML page, there is actually much more technology behind it. Medium is a data-driven platform that evolves together with the users and their behaviour. Node.js is particularly useful when it comes to running A/B tests to get a better comprehension of product changes and experiment with new ideas.
Read the confessions of a Medium engineer.

Groupon

Groupon, a popular online deal marketplace operating in many countries worldwide, decided to rebuild their entire web layer on top of Node.js. The initial motivation was the fact that the stack they had been using earlier became hard to maintain. Especially when as a result of their numerous acquisitions they ended up with a bunch of other stacks to manage in different parts of the world. These events led the company to unify the development across all their platforms. In a year-long, project they moved to Node.js, making it one of the largest production deployments of Node.js worldwide.

Read the interview with Adam Geitgey, director of software engineering at Groupon, who shares the story about their transition to Node.js.

Ebay

Ebay, a multinational e-commerce company, has always been open to new technologies. The company settled on Node.js for two major reasons: they needed an application as real-time as possible to maintain live connections with the server and a solution that could orchestrate a huge number of eBay-specific services that display information on the page. Node.js seemed to be a perfect fit.
Check out some on their tech blog.

NASA

Yes, that's right. NASA use Node.js too. The technology is of a much greater importance than in other applications, because it actually saves lives, keeping astronauts safe during their dangerous space expeditions. After an accident when one of the astronauts nearly died because of the ineffective data hosted in many locations, NASA faced the challenge of moving the data related to the EVA spacesuits to one cloud database in order to reduce the access times. The new system based on Node.js decreased the number of steps in the process from 28 to 7.

Learn more about NASA's astronomical challenge.

Look at the Future

The examples mentioned above are only the tip of the iceberg of all companies that embraced Node.js in production. There are many more big players using Node.js, e.g. Microsoft, Google, Yahoo, Mozilla or Github. The technology creates countless possibilities in development, and we're looking forward to seeing more and more up-and-coming Node.js-based apps conquer the market.

When You Shouldn't Use Node.js

Even though Node.js is – in many cases – the perfect match for building applications, there are still some bad use cases when you shouldn't consider using it.

The first thing that comes to my mind are heavy-computing apps. Node.js is based on an event-driven, non-blocking I/O model and uses only a single CPU core. CPU-heavy operations will just block incoming requests, rendering the biggest advantage of Node.js useless. Therefore, if you consider building some CPU-heavy software, try a different, more suitable technology that will give you better results.

There are more things to consider before kicking off with Node.js. Many packages for Node.js applications are available in npm. The community is vibrant, the technology is maturing, and npm is the largest available repository at the moment. However, packages vary in their quality. Sometimes, you can still encounter issues with packages supported only by individual users and not maintained properly; for instance, when connecting your Node app to obscure or old database system.

Finally, using Node.js is unnecessary for simple HTML or CRUD applications in which you don't need a separate API, and all data comes directly from the server. Your application might be marginally more scalable, but don't expect more traffic coming to your app just because you used Node.js. In such cases, you should stick to proven frameworks such as Ruby on Rails.

Both Rails and Node can achieve the same results. However, there are different situations where each one works best. With Rails, you can prototype and move fast, break things, and get a crud app in an hour.

Node, on the other hand, requires more resources, but it scales better, works great with real-time tasks and is faster overall. Hopefully, the tips above will help you pick the best environment.

Cons of Using Node.js for Your Back-end

Unstable API

One of the biggest disadvantages of Node.js is that it lacks consistency. Node.js' API changes frequently, and the changes are often backward-incompatible. When this happens, programmers are forced to make changes to the existing code base to make it compatible with the latest version of the Node.js API.

More Development Time

The fact that Node.js is unopinionated can also be seen as a drawback by some developers. Ruby on Rails provides you with a lot of directions from a fresh install and guides you into their way of doing things, but with Node.js you basically need to write everything from scratch. It might result in a decrease in productivity, slowing your work down. However, if you cooperate with an experienced team of programmers who have internally developed a good processes for developing and maintaining code, you do not have to worry about efficiency.

Immaturity of Tools

Even though, the core Node.js is stable, many packages in the npm registry (pre-installed node package manager that organises the installation and management of third-party Node.js programs) are still of poor quality or have not been properly documented. As it is mostly an open source ecosystem, numerous tools have not been supervised and so they might lack the quality and fail at meeting coding

standards. The npm's structure makes it harder to spot reliable packages. Therefore, you might need more experienced developers who can find tools that can be trusted.

Less Efficient with CPU Intensive Operations

At the same time, however, Node.js asynchronous and single-threaded models have certain limitations. Yes, Node is good at handling multiple concurrent events and requests. However, it is not as efficient when it comes to CPU intensive operations, such as generating graphics or resizing images. Luckily, a workaround of making a new task queue to manage CPU intensive requests exists, though it requires spawning additional workers, and introducing new layers to your Node.js application.

What Are Alternatives To Node.js

Ruby on Rails

In a nutshell, Rails is a full-fledged web framework with a basic web development functionality provided out of the box. The framework is built on Ruby, a powerful and expressive programming language developed by Yukihiro Matsumoto in 1993. Rails is used by such popular sites and companies as GitHub, Shopify, Airbnb, and SoundCloud.

Pros of Ruby on Rails

Based on Best Web Development Practices

RoR creators describe Rails as an opinionated framework because it implements their vision of the best web development practices and solutions. Since RoR values convention over configuration, the framework ships with all necessary modules and libraries. All of them are implemented with MVC (Model-View-Controller) paradigm in mind.

Rich and Well-developed Infrastructure

Rails offers a fully integrated web server solution (Puma web server) and ActiveRecord database that allows to abstract schema and models. Even better, Rails features generators – powerful scripts that allow to rapidly scaffold a RoR application by creating the directory structure, files, controllers, routers, database models, and other common web development nuts and bolts. All this makes Rails suitable for rapid prototyping and deployment of web applications.

Easy Portability Across Various Database Platforms

Similarly, web developer community values Rails for its easy portability across various database platforms. This feature is supported by the Rails database migrations. Active Model that underlies the Rails default database ActiveRecord can easily abstract the differences between various SQL back-ends. Rather than writing schema in pure SQL, migrations allow using easy Ruby DSL syntax to describe changes to tables. As a result, RoR allows creating a database-agnostic schema and models simplifying migration of Rails applications across different database environments.

Cons of Ruby on Rails

Less Flexibility

If your application has unique functionality and demands, it may turn hard to adjust an opinionated framework such as Rails to your product's needs. Since Rails imposes much of authority over the app development process, you may get to the point when adjusting the framework takes more time than building it from scratch, or using a less opinionated one.

Drop in Performance

Since RoR offers a majority of modules out of the box, the framework is heavier and somewhat slower in performance than Node.js. It is also less efficient than Node.js in managing simultaneous requests. The only solution to this is using additional server instances, which consumes additional memory

Slow Debugging

Rails large stack frames and multiple layers of abstraction can make a debugging of RoR applications difficult. Your development team has to spend more time fixing errors and making updates to applications. This may slow down development, testing, and production cycles.

So, How to Make the Right Choice?

Node.js is more suitable for dynamic applications with multiple server requests, and frequent shuffling of data between the client and the server. What kind of applications are these? These are instant messaging apps like chats and collaborative apps (drawing, video conferencing) collectively referred to as RTAs (Real-Time Applications). Event-based Node.js architecture is perfect in handling heavy I/O operations, server requests and data flow in these apps. For the same reason, Node.js is a preferred choice for Single Page Applications (SPAs) that involve heavy client-side processing, rendering, and where the main function of the back-end is to provide a REST API. Similarly, whenever performance and scalability of web applications are of concern, lightweight and fast Node.js outperforms the Rails framework. That is why such companies as Linkedin began using Node.js for performance and scalability reasons

Ruby on Rails, on the other hand, performs better than Node.js in CPU-intensive applications. Node.js is a single-threaded environment that was not designed for CPU intensive operations with graphics, images, and data. Making computations on vast arrays of data may simply block all incoming requests rendering the main advantage of the Node.js useless. If you consider building CPU-heavy applications, Rails is definitely a better option than Node. Rails is also better when the speed of development is critical. With all the modules and generators available out of the box, Rails is very powerful in Rapid Application Development (RAD). Just in few commands, you may have a fully functional prototype that may be amended with additional features later. Node.js repositories also provide generator scripts to speed up the development, but fast prototyping is the philosophy of Rails.

So, when choosing between Node and Rails you should consider a speed of development, performance, and scalability of your application, and the type of use cases it addresses. If your requirement is a fast development of CPU-heavy applications, go with Rails. On the opposite, choose Node.js if you are thinking about RTAs, SPAs, and other I/O heavy solutions. Hopefully, this overview will help you make the right choice of your web development environment for the next project.

Netguru provides fast and reliable software solutions for your business - in both Node.js and Ruby on Rails.