RisingStack

NODE.JS IS ENTERPRISE-READY

Report, Q4 2015

TABLE OF CONTENTS

HOW NODE.JS CAN SOLVE COMMON CHALLENGES IN ENGINEERING TEAMS; AND HOW TO ENABLE NODE.JS ON AN ENTERPRISE ENVIRONMENT TO SHIP BETTER PRODUCTS FASTER

COMMON CHALLENGES

The computer science era is constantly shifting and is happening at a pace that makes it very hard to keep up with the changes. New technologies arise, but not all technologies are going to get long term support. It is hard to tell what the future holds for our applications, but we must ensure that they are working as intended. This also has a downside: our application might end up obsolete after long years of hard work.

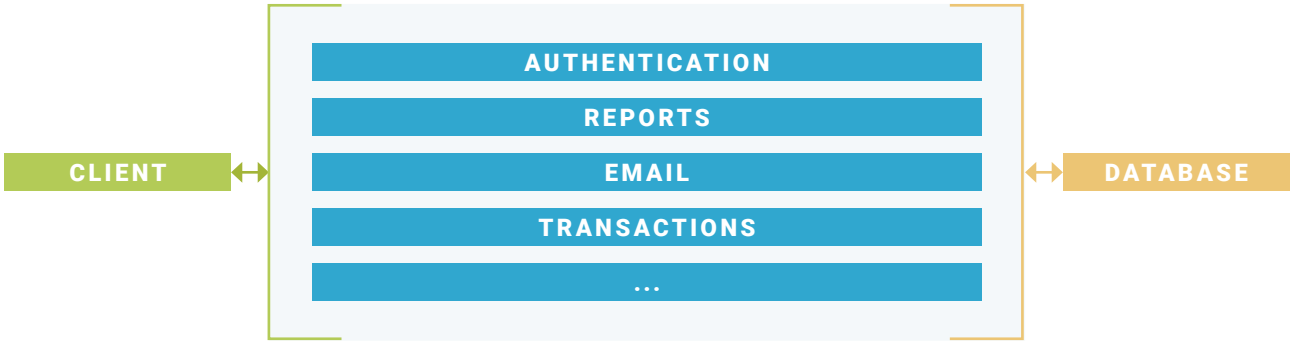
Most software projects start with solving one problem. Then comes another one and the project continues growing, without you even noticing it and without the team even being able to cope with it. This is how monoliths are built. Every new feature gets added to the existing application making it more and more complex. After a certain complexity, it becomes difficult to handle even the smallest changes in the system.

THE MONOLITH

Monolithic applications are self-contained and independent from other computing applications, which means that every feature is bound together in a single system that can perform every step needed to complete a particular function.

A cross-functional team is a group of people with expertise in different areas working together to fulfil a business need.

This brings up a whole list of problems.

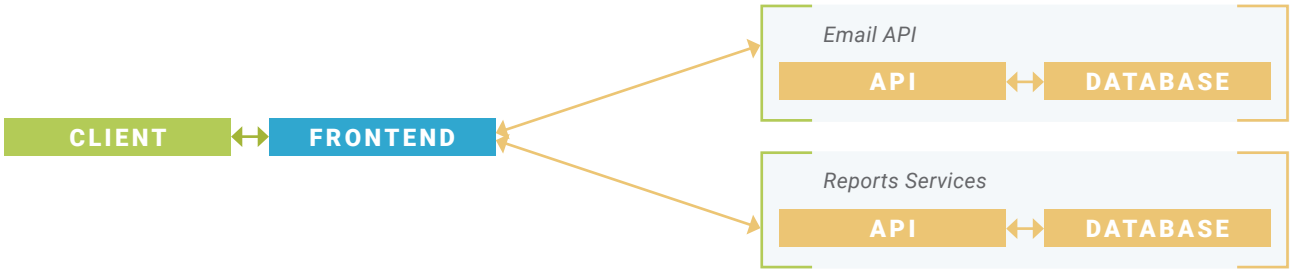


SCALING

Imagine a banking application that handles transactions, email and report generation. The application consists of these three theoretically equal parts, but in reality, each of them serves a different load. Most people may just want to send transactions and do nothing else.

If the user base grows large enough, more computing power needs to be added to the servers and this brings up the problem of scaling. In a monolithic service, everything has to be scaled together: if one part of the system is getting more traffic than it can handle, the whole service has to be scaled up as one, and the resources assigned to the not so busy parts are basically wasted.

With smaller services, this would not be a problem at all. With a microservice based architecture, the parts that need help can be scaled separately, meanwhile allowing the less often used parts to run on less powerful machines. As you will spend less money on the infrastructure, you will have extra budget to hire more competent people to build your product faster.



DEPLOYMENT

Those who have to work with monolithic systems know the real pain of deploying a large codebase. If we split our services into smaller parts, we are left with only small portions of our codebase that we have to deploy at once. This allows faster deployments, and also less effort has to be put into deployment.

ORGANIZATION

Coordinating a large team can be hard; smaller teams and codebases are much easier to handle. Working with micro-services allows the splitting of the team into smaller groups and assigning them specific services that they are responsible for.

These small teams have to take care of the service during its whole lifecycle including development, database-management, hosting, monitoring, and deployment. This can be achieved by building cross-functional teams; this eliminates the common communication barriers that one has to overcome in an organization.

MOVING FORWARD

The microservice pattern is an approach to develop a single application as a suite of small services.

When we start implementing a microservice-oriented architecture, the services can be built separately from each other. The software can be built up from multiple smaller pieces that work together to fulfil the business needs.

This allows us to always choose the right tool for the job. If we can define technology agnostic communication channels between our services, they can be implemented in any language in use. One service could run on the JVM, while others could be implemented in Node.js or Python. We are not restricted to any technology anymore, and it makes possible to experiment with new technologies and break a legacy monolith into small, manageable pieces.

A rule of thumb is that a microservice should be small enough to be rewritable in about 2 weeks time, by a small team. This way, if we are not satisfied with the quality or the performance, it will not take long to replace it, reducing the risk of development.

By adding small services, we can reduce not just the risk of the development, but the costs associated with the operation of the infrastructure; better performance for less money.

What makes Node.js the best option out there to build small services? Let's find out!

NODE.JS IN A NUTSHELL

Node.js became the go-to technology in the past years for both startups and enterprises. Why did that happen? What is so appealing about using Node?

Haven't seen a response time over 8ms so far today - Jason Pincin, Walmart.

Some will argue that Node.js increased the productivity of their teams; some will say their application became blazing fast. Some will say their developers are happier.

In this chapter we are going to do a short intro to Node.js starting with the maturity of the project, and finish with the main principles behind it.

NODE.JS IS ENTERPRISE-READY

Using Node.js has many advantages: it can boost productivity of the teams; it can increase the performance of your applications, or, simply, keep your teams happy. But when we are saying Node.js is enterprise ready, we mean a lot more: Node.js is now a part of the Linux Foundation and therefore has long-term support.

BENEFITS OF USING NODE

PRODUCTIVITY

When PayPal started using Node.js, they reported a 100% increase in productivity compared to the previous Java stack. How is that even possible?

First of all, npm has an incredible amount of modules that can be used instantly. This saves a lot of development effort for you; there is no need to reinvent the wheel.

Secondly, as Node.js applications are written using JavaScript, front-end developers can also easily understand what is going on, and make changes if necessary. This saves valuable time, again, as developers will use the same language on the entire stack. In addition to tools like Browserify or Webpack, you can use the very same modules in the frontend, as you use in the backend.

PERFORMANCE

In 2014, on Black Friday, 1.5 billion dollars were spent online in the US

on a single day. It is crucial to keep up with such traffic - Walmart, one of the biggest retailers using Node.js served 500 million page views on Black Friday, without a hitch.

PayPal had the same results when it came to performance.

> *35% decrease in the average response time for the same page. This resulted in the pages being served 200ms faster — something users will definitely notice.*

DEVELOPER HAPPINESS

Finding top talent in 2016 will be harder than ever - the possibility of using cutting edge technologies on a daily basis can help recruit and retain the best developers.

LONG-TERM SUPPORT

After the Node.js and io.js merged, something great happened: Node.js finally got a LTS (long-term support) working group. What does it mean for Node.js and its releases?

LTS releases are going to happen once in every year, at the same time of the year. Each release is going to be maintained for a period of 18 months; after that it is going to stay in maintenance mode for the next 12 months.

Once a release enters LTS, it means a feature freeze: only bug fixes, security updates, and performance improvements are going to be added. Once it enters maintenance mode, only critical fixes will be made.

SUCCESSFUL ADOPTIONS IN THE ENTERPRISE

LINKEDIN

LinkedIn's mobile application was originally powered by Ruby on Rails, but it is now one of their biggest Node.js applications in production. The original application used to make several sequential calls for a single page, with every thread handling a single request.

Node.js enabled the team to move to a model where the client only makes a single request for a page. They also could simplify the code and move to stateless servers.

One of LinkedIn's biggest win was the 10:1 ratio reduction in the

number of machines used to host their services.

NETFLIX

Netflix did not migrate everything to Node.js at first. Context switching between the UI and the backend was often difficult, because the backend infrastructure was based on Java. The UI team wanted to work with a language that was already familiar to them.

Using Node.js enabled quicker iterations, gave a productivity boost to the engineers and improved user experience for the customers.

DOW JONES

There was a small team at Dow Jones called “Engineering Productivity” that started implementing Node.js at the firm. The team had to handle a tough task: transforming 100 C# developers into Node.js developers.

Frameworks and best practices were already in use. A small team of experienced Node.js developers created a series of classes about them and started to teach the C# team. They also helped to debug and review the code.

This resulted in the dominance of JavaScript over the entire stack. Dow Jones was able to release 2x-4x faster, with around 1/10th the amount of code than before.

NEW YORK TIMES

The video team at the New York Times uses Node.js for both the video API and maintaining the videos. There was no main internal API for consumption of video or playlist data when Times Video was created. They needed to update the application every single time when an editor created or updated video or playlist metadata using JSON files.

The team tested Node.js for its non-blocking I/O Model capability and they experienced a huge performance boost compared to their PHP application.

ASYNCHRONOUS PROGRAMMING

In computer programming, asynchronous events occur independent of the main program flow. Asynchronous actions are actions executed in a non-blocking way, allowing the main program to continue processing.

THE COST OF I/O

Input and output (I/O) operations on a computer can be extremely slow compared to the processing of data. An I/O device can incorporate mechanical parts that physically move: this can be slower than the alternation of electric current. For example, during a disk operation that takes ten milliseconds to perform, a processor clocked at one gigahertz could have performed ten million instruction-processing cycles.

It is possible to start an operation and then perform other operations that do not depend on each other. This approach is called asynchronous programming. This way, all available resources can be fully utilised. Many operating system functions implement asynchronous I/O at many different levels. Asynchronous I/O is used to improve throughput, latency, and/or responsiveness.

ASYNCHRONICITY IN JAVASCRIPT

JavaScript is asynchronous **by default**. That means, no workaround is needed to achieve asynchronous functionality.

The main advantage of JavaScript compared to other languages is that functions are first class citizens of the language. They are objects (just like everything else), and can be passed around to other functions and called whenever needed. If you pass a function to another function (known as higher order function) as a parameter, within the function you can call it when you are finished with your job. No return values; only calling another function with the values; these functions are called callbacks.

Callbacks are widely used in the JavaScript world to achieve asynchronicity. For programmers who are used other languages like Java, C# or PHP, it may be hard to fully grasp callback function calls at first. From the early ages of JavaScript, callbacks was the most widely spread feature allowing asynchronous code execution.

PROMISES

In Node.js, most of the core libraries have callback interfaces, but in later versions of Node.js, the V8 execution engine has a vast variety of tools that eliminate the so called 'callback hell'. First, there is a built in language feature called 'Promise', that is quite similar to Java's implementation of Futures and C# Tasks. A promise represents a future value that can be returned at any time after the call was made. Promises can replace callbacks, and they have several benefits over callbacks.

THE FUTURE: ES2015 AND ES2016

Another handy feature of the execution engine is generators. Generators allow the program to pause the execution of a function call. They enable lazy computation of sequences with items calculated on-demand as they are needed.

Last but not least, in future versions of the EcmaScript, standard async functions will be introduced. With async functions, synchronous- looking code can be executed asynchronously. It solves most problems of callbacks and hides them from the sight of the programmer, providing the ability to focus more on the features instead of fiddling with small implementation details. This concept comes from the .NET framework 4.5 features with the same syntax, semantics, and name.

THE UNIX PHILOSOPHY APPLIED

The Unix philosophy consists of a few common patterns that can be applied to many different fields of computer programming. It is a set of cultural norms and approaches to develop small, but also very capable applications. The principles have been devised by Ken Thompson, based on the experiences of building the Unix operating system.
According to Doug McIlroy, inventor of the Unix pipe (head of the Bell Labs CSRC) the Unix philosophy is the following:

> *› Write programs that do one thing and do it well.*
> *› Write programs to work together.*
> *› Write programs to handle text streams, because that is a universal interface.*

DO ONE THING AND DO IT WELL

For the Node.js economy, this is the most important principle. What Node.js developers tend to do is to write small, composable pieces and put them together as necessary. This is achieved through the modules provided by the npm registry.

THE NPM REGISTRY

The npm registry consists of hundreds of thousands of modules, and many more are getting added every day. Developers do not have to use the modules as they are: most of the modules on the public registry are hosted on GitHub. Anyone can just take a peek into the source or fork it as needed.

This attitude enhances the developer experience, and makes developing features faster. The developer can focus more on business logic, rather than boilerplates or other small details of the application.

npm has a growing public registry, but now also supports private registries for corporations. This enables code sharing between members of a team, without exposing business logic to the public. One solution for this is npm On-Site, which enables on-premise hosting of modules.

THE WORLD OF OPEN SOURCE

Node.js, with its whole ecosystem, is built upon open source. From small util libraries to whole frameworks, everything is in the public thereby encouraging companies and individuals to take part and make them better through collaboration.

THE ROAD TO NODE

To reach the full potential of developing with Node.js, there are crucial points that must be addressed at an organization level.

These include integration of modern tooling into the development workflow, removing the communication barrier by building cross-functional teams, and finally embracing the new way of building complex applications: the microservices pattern.

EMBRACE MODERN TOOLING

The tools that we use to develop software have become a lot better in the last couple of years.

CODE

The code itself is the main part of our application that will get compiled or interpreted into machine language and will provide the end result of the application for our users. Writing clean and maintainable code is a goal that we all should keep in mind. Well structured code is self documenting; it does not require external documentation tools; therefore, it makes working with it a pleasure.

Nowadays, there are quite a few tools that we can use to manage our codebase.

SOURCE CONTROL

Git was first (and still) used for managing the code of the linux kernel and developed by Linus Torvalds, the father of linux.

It is a well-crafted example of source control management. Git lets the developers work together and make changes to the same code without having to figure out the differences when, later, these changes have to be merged.

It enables us to track changes of the source code and revert to any given change at anytime. This makes the code resistant to failures.

With it you can and should set up a different repository for each service, giving commit rights to only those who are responsible for maintaining the project.

There is on-premise solution for Git, but GitHub and Bitbucket should also be taken into consideration. GitHub offers a superset of Git where new features can be added via pull-requests with which code-reviews can be done.

CONTINUOUS DEPLOYMENT

Customers do not have to wait years for the first look at their software. It is considered a best practice to adapt the methodology called continuous integration and delivery. It means that every change that is made gets to the end user as soon as the change goes through the testing and deployment steps. This is extremely helpful for companies, because of the fast feedback cycles.

There are several programs to make continuous delivery fast and easy. In just a couple of minutes, you can set up a build pipeline that runs your test and creates your build artifacts which then get deployed to your application servers. Codeship, CircleCI, or Jenkins are three examples of this kind of tooling. With tools like these, each and every commit will get checked by your integration tool, and will notify the team members in case of a failing build.

COMMUNICATION

Seamless communication is a key for cross-functional teams. Without proper channels, it is very hard to get things done. It may be frequently the case that your team is split all around the world and cannot walk up to each other to ask questions. This is one of the many use cases that makes chat software viable.

IRC is a viable option when talking about team chats, but for some reason, it is not so popular anymore. These days, projects with easy setup came to the market, HipChat and Slack are two good examples with which you can set up a team chat fast and easy. They can be integrated with CIs, alerts can be sent to them, making them the perfect communication platforms. They also have XMPP and IRC gateways. Unfortunately they do not offer self hosted solutions.

CONTAINERIZATION

Containerization is a lightweight alternative to full machine virtualization that involves encapsulating an application in a container with its own operating environment. On the same machine, we can run multiple fully separated processes.

This technique enables immutable deployments without actually having to set up new machines for each deploy.

A company called Docker has an existing and great solution for containers.

BUILDING CROSS-FUNCTIONAL TEAMS

Melvin Conway's law states that "Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure."

This is as true today as it was a half century ago.

A deeper look into what this really means: if you want successful cross-functional teams, the first thing you need to do is to make sure that your organization can adapt. The software you create reinforces the culture of your company.

CROSS-FUNCTIONAL TEAMS

In these teams, people work together to deliver the same business value: a new feature a new project, or even a new product. The teams need to be stable enough to get the job done and in exchange, they can move faster and more efficiently than teams grouped by similar function. The communication is more likely to be oriented around the goal itself and not around the communication or management issues across functional units, making this approach much more effective.

THE REAL BENEFITS OF HAVING CROSS-FUNCTIONAL TEAMS

Functional (or often called 'siloed') departments, often adapt an 'us vs them' thinking against other teams. Instead of better productivity, this is more likely to result in hostility against each other. Working with people from different backgrounds also enables you to view the project from a different point of view. This helps to understand the real reason behind a conflict and resolve (or even prevent) it.

Being able to make their own decisions and work independently within an organization, cross-functional teams are able to ship code faster than functional teams. The teams can focus on improving their cycle time and implement continuous deployment in order to solve the challenges they face almost instantly.

THE SUCCESSFUL EXAMPLE OF NETFLIX

Netflix decided to go with highly aligned and loosely coupled teams. The company set clear, specific, and broadly understood goals. The interactions between teams are focused on strategy and goals, not tactics. Although it requires a large investment in management time in order to be transparent, they feel it is definitely worth it.

Their teams try to keep meetings at the minimum. This is possible because the teams truly trust each other - without requiring layers of approvals. The leaders reach out proactively to help whenever they feel it's appropriate, and do not focus on supervising each task of the team members.

HOW RISINGSTACK CAN HELP

CONSULTING

The architecture of a system is one of the most crucial decisions to get right from the beginning. We help you choose hosting and deployment, provide recommendations related to developing a highly-available application, as well as evaluation of connected services, like databases and file storages.

Also, learning proper patterns and avoiding pitfalls result in a cleaner, more reliable and more maintainable codebase. RisingStack can help you with a comprehensive, hands on review of your codebase to ensure production-ready quality.

SUPPORT

RisingStack offers support plans to ensure that your system remains robust and up-to-date. You get a dedicated senior engineer who will answer all your technical questions and will help you move your business forward.

TRAINING

RisingStack offers diverse trainings for Node.js, from a beginner level to more advanced topics like performance engineering.

GETTING STARTED

JAVASCRIPT FUNDAMENTALS

Node.js at the end of the day is just JavaScript - having a solid understanding of the language is a must when developing with it.

NODE.JS FUNDAMENTALS

Module system, managing dependencies, asynchronous programming - just a few to name the very basics of Node.js that your team is going to learn during this training.

INTERMEDIATE

APPLICATION ARCHITECTURE

Getting the application architecture at first can be challenging - Node.js is no exception. During this training, you will learn how to structure your codebase, which framework to pick, and how to test your applications.

ADVANCED

BUILDING MICROSERVICES WITH NODE

Moving away from a monolithic application to microservices using evolutionary design has its very own challenges: securely integrating new features into the existing application, while keeping or improving the user experience. During this training, your team will learn how to keep the core of the current application, while adding new features as microservices.

OPERATING NODE.JS APPLICATIONS

This training covers how your team can operate Node.js in production and what changes it brings to your infrastructure. It dives into topics like npm, and how it affects the deployment, building artifacts and deploying them, and how to secure and monitor Node.js applications.

RisingStack, Inc.
info@risingstack.com | www.risingstack.com
linkedin.com/company/risingstack | twitter.com/risingstack