

The State of Node for the Enterprise

Report, June 2015

Contents

3 THE ESSENCE OF NODE

- 3 JavaScript on the Server
- 4 Asynchronous Programming
- 4 Module-driven Development
- 5 Small Core, Vibrant Ecosystem
- 7 What Node Brings to the Enterprise

9 KEY SUCCESS FACTORS FOR NODE IMPLEMENTATIONS

- 9 Embracing code modularization, sharing and collaboration
- 10 Building an application infrastructure for a true SOA environment
- 11 Removing communication and organizational barriers
- 12 Elevating Node to a high-priority platform

13 WHAT NODESOURCE BRINGS TO NODE

- 13 Node Strategy Services
- 14 The Node Maturity Curve
- 16 Tooling
- 17 Training
- 18 Support from NodeSource

The Essence of Node

The growing popularity and adoption of Node (Node.js and io.js) is the result of a confluence of factors. First, there is an increasing need by companies of all sizes to quickly build fast, scalable, distributed web applications. Second, large enterprises are finding that there is inherent risk associated with running large, complex monolithic applications due to the difficulty and cost of tuning, maintaining, patching and debugging them. Third, software developers are looking to develop a skill set that prepares them for the new market reality of agile practices, continuous integration and delivery, cloud-scale application design and a highly mobile and demanding user base.

Node.js provides a module-driven, highly scalable approach to application design, development and deployment that enables development teams to put the most desirable elements of agile methodology into practice. This can greatly reduce risk and raise developer productivity—raising developer morale in the process.

In order to understand how these gains can be achieved, it is important to understand what makes Node so unique and powerful.

WHAT IS NODE?



Node (Node.js and io.js) is an open source, cross-platform runtime environment for server-side and networking applications. Node applications are written in JavaScript.

JavaScript on the Server

At the most basic level, Node is a platform for running JavaScript on the server. There are some key benefits associated with this model. First, there is already a large JavaScript developer community that is building some of the world's most exciting web applications.

Second, JavaScript consistently ranks among the most popular languages.¹ “Developer joy” is a common theme for Node and this largely relates to JavaScript’s approachability and the high levels of productivity it affords.

“By 2017, JavaScript will be the most in-demand language skill in application development.”

Forrester Research, 2014

JavaScript on the server with Node further establishes it as the language of the web and its new uses on the server are helping to shape the future of the language itself.

¹ [The RedMonk Programming Language Rankings: January 2015](#)

Asynchronous Programming

Node requires developers to embrace a different mindset in the form of asynchronous programming—as opposed to the traditional serial, or sequential programming. By treating I/O as a special class of operation, developers must design highly performant applications by default, but it does come with the burden of adjusting the way the programmer thinks about the execution of their code. Node is single-threaded by nature, and while some consider this a weakness, it should instead be embraced as a win for application design.

Applications built with Node are built for predictable scalability; the idiomatic design patterns adopted by Node programmers confer robust scalability by nature without the overhead required by complicated synchronization mechanisms.

JavaScript—and Node by extension—was designed for the web and the browser, where nothing is synchronous. Asynchronous JavaScript programs can perform many complex, parallel tasks in the browser. Node takes asynchronicity to the extreme on the server, making it the perfect choice for I/O-heavy and highly concurrent applications.

I/O Is Expensive

CLASS	OPERATION	TIME COST
MEMORY	L1 Cache Reference	1ns
	L2 Cache Reference	4ns
	Main Memory Reference	100 ns
I/O	SSD Random Read	16,000 ns
	Round-trip in Same Datacenter	500,000 ns
	Physical Disk Seek	4,000,000 ns
	Round-trip from AU to US	150,000,000 ns

Module-driven Development

Node is a highly productive platform that is modular by nature and has a track record as a key component in an agile technology stack. Node embraces the new mantra of “*Throw-awayability*” that is becoming pervasive in the services oriented software design world. Node encourages developers to think in terms of creating small services that can be easily replaced or updated when necessary.

By adopting a module-driven approach, Node developers can deconstruct the functionality of large monolithic applications and redesign them as a series of Node

modules, bundled together to form a collection of services. This establishes an elegant simplicity in building scalable application functionality that improves both business and developer agility and leads to greater code-reuse.

Having development teams focusing on developing modules enables them to:

- Maintain focus on essential functionality
- Better test, validate and document that functionality
- More easily share and collaborate with other teams

Small Core, Vibrant Ecosystem

The main risk in large monolithic applications and traditional development methods is the natural *mission creep* of applications and the development environments that build them. Over time this results in feature rich but bloated products.

“By 2016, the vast majority of mainstream IT organizations will leverage nontrivial elements of OSS (directly or indirectly) in mission-critical IT solutions.”

Gartner

Node avoids this scenario by creating a small core of essential functionality that is studiously defended and constantly debated by the

Node community. This pushes experimentation to the edges and encourages a vibrant ecosystem and development culture. It also ensures that spurious functionality does not become a permanent part of the Node environment. This ethos also extends to Node-style development, with developers constantly thinking about how to keep modules small and focused and splitting apart functionality where the “*do one thing well*” rule is broken.

New Models in Open Source

Most IT organizations now realize the increasing difficulty of modernizing their approach to development and operations without embracing open source technology to some degree. Gartner predicts that by 2016, “the vast majority of mainstream IT organizations will leverage non-trivial elements of open source software” in mission-critical solutions.

Node was born in the new era of open source, embodied by the GitHub model of developer interaction, collaboration and governance. Both Node and GitHub allow for a public and private open source development platform that empowers a growing, global community of developers.

Enterprises adopting Node are generally seeking to capture some of the obvious benefits of open source development practices and ethos. The culture and development



workflow of Node encourages greater collaboration, communication and code-sharing—engineering practices that often go missing as enterprises grow and change over time.

Node.js and io.js

The world of Node has been evolving very quickly and perhaps the largest shift for Node in the last year has been the forking of the Node.js project to create io.js. Like Node.js, io.js is built on Google's V8 JavaScript engine, which also powers the Chrome browser. The io.js project was created for two main reasons. First, to provide faster and more predictable releases cycles to quicken the introduction of new features - some of which have been in the works for some time. And second, to continue development of Node under an open governance model², as opposed to corporate stewardship³. One of the stated goals of io.js is to maintain compatibility with the npm ecosystem which assures that io.js will run most of what Node.js runs today. Currently, the differences between the two environments are small and don't impact most users or applications.

NodeSource are contributors to both projects and maintain an active, leadership presence in both Node.js and io.js. NodeSource has played a key role in the reunification of these two efforts under the new Node Foundation which will soon begin releasing a new version of Node.js based on both codebases.

Naming Conventions: Node vs Node.js vs io.js

Node				Node.js Foundation
NODE	NODE.JS	IO.JS	THE NODE FOUNDATION	
General	Specific	Specific	Specific	
Node refers generally to the Node.js and io.js technologies and communities.	Node.js refers to the current Node.js distribution governed by the Node.js corporate governance model led by Joyent, Inc.	io.js refers to a fork in the Node.js project by contributors who wished to create a Node.js project with an open governance model and a more rapid development cycle.	The new body formed that has reunited the two groups and will steward future releases of Node.js, and possibly io.js.	

² [GitHub: io.js Project Governance](#)

³ [io.js FAQ](#)

What Node Brings to the Enterprise

The following are the primary reasons that enterprise developers are embracing Node as a first-class citizen in their technology stack.

SOA for the Cloud Era

After a decade of aspirations for a Service Oriented Architecture, companies with large, monolithic code bases are seeking to break these apps up into smaller, decoupled and more nimble services. This “unbundling of everything⁴” better exploits the cloud architecture deployed in most modern data centers. Node has proven its utility in this area, not only as a technology choice, but for the development methodologies it encourages.

Ease of Use

JavaScript is quickly becoming the de facto language of the web, with Node as the primary server-side platform for this trend. JavaScript’s ubiquitous nature and an abundance of rich development tools lowers barriers for developers crossing over from the frontend to the frontend-backend where Node finds its natural home.

Performance

Node represents an attractive compromise between performance concerns and platform approachability for developers. While there are other platforms able to match or exceed the performance of Node, none are as approachable for such a large portion of developers as Node via its reliance on JavaScript. Additionally, Node’s asynchronous programming model encourages performant development practices that are either difficult or risky in other languages and platforms. Performance was cited as the top reason for using Node is a recent developer survey⁵.

“We’re used to working in JavaScript all day long. Having Node just makes it feel like a very natural extension of our work environment”

Alex Liu, Node developer at Netflix

Attract and Retain Developers

Enterprises seek to follow the example set by Walmart, PayPal, Netflix and Intuit in connecting with the vibrant open source community and positioning themselves as an attractive employer for top talent.

⁴ [The Unbundling of Everything](#), TechTarget, 2015

⁵ [What is Node.js used for: The 2015 Node.js Overview Report](#), RisingStack Engineering, 2015

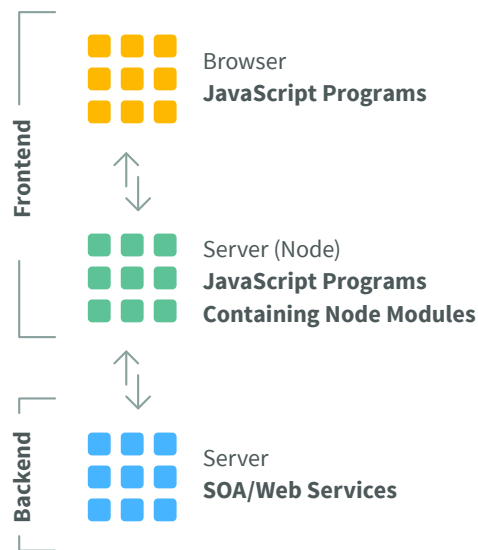
Collaboration

Large enterprises tend to have internal organizational or technological silos that can represent an unnecessary business risk. These barriers need to be removed so that engineering knowledge and personnel can cross between business units and within internal business units. Node's module-driven approach encourages the creation of small teams of developers to solve atomic problems of functionality that interact with those of other teams. This encourages greater interaction and cooperation between developers.

Sidebar: The Frontend Backend

Over the last few years, one of the areas of substantial growth in Node development is the notion of the *frontend backend*. A frontend backend is an architectural tier added to a system to specifically serve frontend resources (templates, html, css , etc.) in front of a legacy system or API service tier.

Rich client teams who have been building exciting, dynamic JavaScript experiences have run up against problems from building large, monolithic structures that naturally result from traditional top-down programming. The result is poor performance and scalability and frustration for end users. Front end-developers must also rapidly iterate on the customer experience to keep users engaged.



It has become clear that the application frontend needs a lightweight, dynamic back-end to deliver the scale and response times needed. Thus, the frontend backend is where Node has created a whole new way for rich client teams to operate.

A frontend backend empowers frontend teams to quickly evolve the user experience to respond to rapidly changing conditions on the ground – such as news items, social happenings, and sporting or cultural events – all while being able to think and operate in the familiar JavaScript mindset.

Key Success Factors for Node Implementations

Short-term focus on the following areas can increase the likelihood of successful Node adoption.

- Embracing code modularization, sharing and collaboration
- Building an infrastructure for a true SOA environment
- Communication and organizational barriers
- Elevating Node to a high-priority platform

Embracing code modularization, sharing and collaboration

Node teams are tasked with building small services of bundled functionality as one of the first steps on the road towards a true service oriented architecture. Some of these services are intended to augment or replace existing monolithic code bases.

NPM

npm is the package manager for JavaScript and Node.js. It manages dependencies for JavaScript and Node.js applications.

Preventing Mini-monoliths

There exists a very real risk of these teams tending towards building “mini-monoliths”. This is reasonable given the backgrounds of the developers and the learning curve for Node, and developers need to be given latitude to experiment with architectures as long as the direction is towards smaller and modular code bases.

Plan for an npm Registry

With any new development method, tooling is an obvious area for concern. One of the indispensable tools for establishing an advanced Node capability is *npm*. *npm* is the package manager for Node and is used to manage dependencies for Node and JavaScript programs.

There is a public npm registry that enables developers to publish their modules and encourages sharing in the Node community. But the *npm registry* also allows for private modules through an access control mechanism as well as an enterprise version that lives behind the firewall.

Thus, it is important to implement a solid private npm solution that serves the needs of engineers and operations and also meets higher-level business concerns, such as regulatory compliance.

Introduce GitHub Early

Node was born of the GitHub era and the development practices encouraged by Node make GitHub an important for developers. GitHub should be considered a key tool for modularization, sharing and collaboration. GitHub encapsulates current best practices for open source collaboration more than any other solution that exists today. It is also a means of capturing some of the value of open source when applied to an *internal open source* model.

Developers can also be encouraged to seek other means of modularization where they are not comfortable with jumping to an extreme form of it. One such approach is provided by the linklocal⁶ tool which is being increasingly used to build internally-modularized code bases so that a project doesn't necessarily require separate repositories for each of its small pieces. Often this tool is used as a mid-way step before committing to full modularity.

Building an application infrastructure for a true SOA environment

Many organizations who are at early stages of adoption of distributed web service development struggle with the relative newness of a SOA approach. There can be a lack of supporting SOA infrastructure, a lack of tooling for Node in particular and a lack of experience in the architectural decisions of individual development teams.

Application infrastructure for SOA is often lacking because the needs of SOA require different runtime services than monolithic applications. For instance, there is often no concept of a service registry or service discovery mechanism, there are usually no conventions around inter-service communication except for a rough agreement on REST, or REST-like interfaces, and even these are rarely well versioned and are instead are tightly coupled between services.

An ideal SOA infrastructure would allow services to follow their own upgrade path and

KEY ELEMENTS OF GITHUB FOR NODE PROJECTS

STRUCTURE: The use of Organizations and Teams provides structure and order as the usage expands.

COLLABORATION: The use of Pull Requests, Issues and other tools related to *collaborating around code*. GitHub is an excellent platform for code reviews and discussions around the specifics of implementation details and should become a natural home for developers because of this.

DOCUMENTATION: Documentation via README.md and other Markdown files within code repositories. Developers will naturally tend toward keeping embedded documentation updated, with documentation that exists outside of a code repository becoming lower priority.

Rather than fighting this, organizations should attempt to treat GitHub repositories as a primary source of code documentation and find means to integrate this source of truth with other internal tools rather than forcing developers, architects and managers to attempt to synchronize documentation across the various platforms.

MULTIPLE REPOSITORIES: Developers should be encouraged to create more than one repository per project. There should be no expectation that a single project/service will occupy a single repository. Node-style development encourages splitting code bases into many, self-contained components. Each of these components may or may not be sharable outside of a given project, the important aspect of modularization is the internal decoupling of concerns within a codebase.

Along with the code, repositories should also contain documentation and a full, targeted test suite for each component. These application components can then be published to the internal npm and re-assembled by the "glue layer" forming the core of the project.

PAYPAL EMBRACES NODE

In his recent engineering blog at Paypal, UI Architecture lead Jeff Harrell stated that "our web applications are moving away from Java and onto JavaScript and Node.js".

He described a scenario where two dev teams - one coding in Java, the other using JavaScript and Node - were working simultaneously on Paypal's account overview page, "one of the most trafficked apps on the website."

The resulting JavaScript/Node solution was created built almost twice as fast with fewer people, written with 33% fewer lines of code and was constructed with 40% fewer files.

⁶ [linklocal project on GitHub](#)

dependencies to be expressed by versioning and discovery with a strong decoupling between different moving pieces of the larger application.

Removing communication and organizational barriers

Within many organizations there exist many communication mediums across various teams, most of which are not compatible with each other. Communication without friction is essential to increase collaboration and enable architects, managers, engineers and operations staff to communicate across business units.

Engineering resources and incentives must be aligned such that project teams, front line support, DevOps and IT operations teams have a clear charter and an understanding of their respective roles in achieving an advanced Node capability.

Some of the objectives developing an effective model are:

- **Understanding operational needs** – IT leadership must understand operations needs and concerns that are somewhat separate from project development concerns, including logging, monitoring, health checks, production configuration and secrets management.
- **Recognizing overlapping concerns** – Project teams have different needs in development environments for overlapping concerns, including logging, monitoring and configuration. These need to be reconciled with deployment concerns.
- **Providing incentives for shared solutions** – Project teams are often strongly incentivised by existing organizational structures to produce solutions specifically for the project at hand and not to develop or support shared solutions. There is little reason, other than personal interest, to take on shared projects since they could lead to additional responsibilities outside the scope of their normal tasking. It is important to incentivise developers to join and lead shared solutions.
- **Managing skills and resources** – As project teams continue to expand and invest in Node development resources, they can quickly overtake other teams in terms of engineering numbers and expertise. Particularly as projects are put into production, the engineers on these teams will likely accrue skills and understanding that go beyond what is available to other teams. The best approach is to allow Node engineering resources and skills to filter organically through the company so as to avoid skill silos.
- **Managing tooling** – As project teams expand and invest in Node development resources, additional tooling to support these efforts will be required and will therefore impact the budget. Establishing a controlled pattern of investment in Node tooling will ensure uninterrupted progress during the transition.

Elevating Node to a high-priority platform

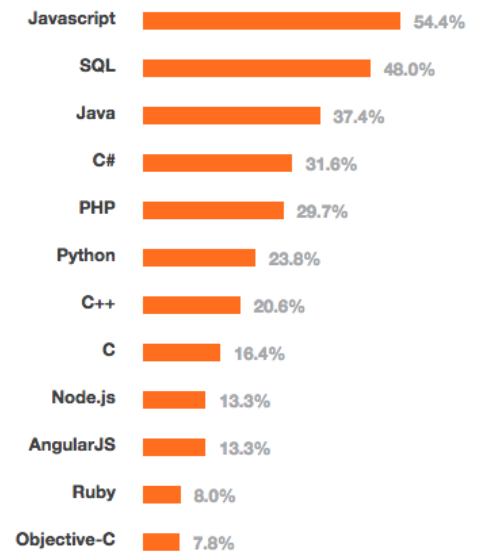
Making Node a peer technology to traditional Java, C++ or .NET environments is essential to getting the most value out of the platform. The newness of Node to many operations and security teams can slow the progress of Node projects and delay the typically high return on investment that Node projects offer.

It is vital that operations teams become comfortable with regularly updating their deployed version of Node.js in production. Some of the required steps toward this goal are:

- Security and operations teams should tune in to release announcements for Node.js, the Node Security Project and also Node-related announcements via the *Common Vulnerabilities and Exposures (CVE)*⁷ system where security problems with Node.js itself and some important parts of the Node ecosystem are reported. The *Node.js blog*⁸ should be monitored for new releases. The *Node Security Project*⁹ should be monitored for ongoing Node ecosystem vulnerability announcements, and these cross-referenced with code in production.
- Node platform vulnerabilities should be treated in the same way as vulnerabilities reported for the JVM, the CLR or RHEL, even where Node is not exposed directly to the internet. While it is common for new technologies to be treated as second-class citizens while the organization catches up, security cannot be allowed to lag behind.
- A solid smoke test suite or plan should be developed in cooperation between development and operations teams to sanity-check upgrades to components of Node deployments. Operations teams should have a means to perform basic validation of the success of upgrades in order to gain confidence in deploying and upgrading an evolutionary platform like Node.

MOST POPULAR TECHNOLOGIES

As recorded in the 2015 Stack Overflow Developer Survey



⁷ [Common Vulnerabilities and Exposure](#)

⁸ [The Node.js blog](#)

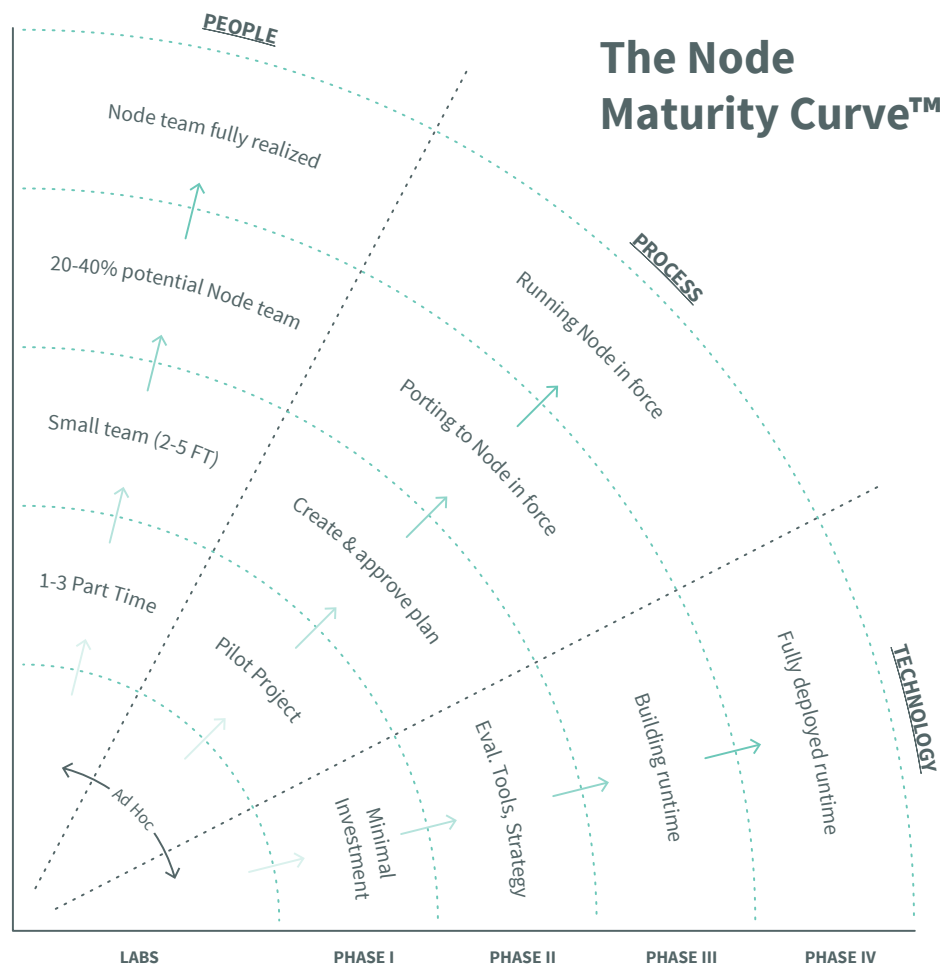
⁹ [The Node Security Project](#)

What NodeSource Brings to Node

The adoption of Node does not provide automatic guarantees regarding an organization's ability to realise the benefits it seeks from the platform. Nor is the path to Node adoption clearly marked and risk-free. The role of NodeSource is to help companies seek out early opportunities and long-term strategies to succeed with Node and to mitigate risk in the process.

Node Strategy Services

With anticipation, the challenges to success and risks to the business can be mitigated. A high-level perspective of the needs around Node development and deployment—along with the possible solutions—can be used to chart a path along a measured maturity curve. The Node Maturity Curve below is helpful in examining how organizations can evolve from a traditional Java, C++ or .NET development shop to a “Node-ready” organization. It is organized along the familiar axes of People, Process and Technology, with an added experimental stage represented as a Lab or developer sandbox.



The Node Maturity Curve

It is important to note that a maturity curve, as applied to Node in particular, may not reflect a serial process along all stages of the curve. Each stage is managed and sequenced differently for every organization and the progress realized can vary from company to company. The Node Maturity Curve is simply a means to identifying and solving business needs, but the ongoing progress should map to those needs being met.

Let's consider each stage:

Labs

For some organizations, new development methods and technologies like Node must go through a rigorous vetting process, while others allow for a more organic method of adoption. Many start with the creation of a sandbox or lab environment from which to launch the first stages of a Node project. This allows for a controlled introduction of Node into existing environments with a minimum of disruption to current operations.

NodeSource can work with development and operations architects to determine how Node fits within current environments and how it can be used to speed time to market for new applications or features and relieve an organization of its design debt.

People

Developers themselves are often the most agile and flexible resources available to IT leaders. Because of the ubiquity of JavaScript developers in enterprise development shops, the first instances of the use of Node often emerge from a particular need for a more robust frontend or greater scale of an existing JavaScript application.

NodeSource can help companies move towards a state of the art Node development team by helping them understand and practice Node's Module-driven development approach, the realities of asynchronous programming and the practical realities of mainstream use of open source technologies.

Process

Establishing efficient, repeatable processes that are lightweight enough to allow agile development and deployment has become a critical priority for engineering and operations teams. Because investment in Node projects can blur the lines between frontend and backend teams, organizations are presented with the challenge of smoothly introducing basic—and later advanced—methods of agile development and the deconstructing of monolithic applications using Node's module-based development methods.

The table below reflects some of the elements NodeSource has identified as critical to a successful transition into a high-functioning Node development culture.

AREA OF CONCERN	NEEDS	SOLUTIONS (PEOPLE, PROCESS, TECHNOLOGY)
ORGANIZATION & CULTURE	• Collaboration	• Training
	• Communication	• Outreach
	• Internal Open Source	• GitHub Enterprise
	• Connect with external open source	• npmE
	• Hiring & retention “cool”	
DEVELOPMENT / ENGINEERING	• Skills	• Training
	• Agility	• Hiring
	• SOA	• GitHub Enterprise
	• Anti-monolith	• Open Source Community Participation
	• Discoverability	• npmE
DEPLOYMENT	• Bundling	• Training
	• Signing	• NodeSource N Ship
	• Security	• npmE
	• Storage	• Adapters for existing tooling
	• CI / CD	
PRODUCTION / POST-MORTEM	• Logging	• Training
	• Server monitoring	• Adapters for existing tooling
	• Application monitoring	
	• Debugging	

Technology

In a sense, technology is the least flexible and agile of resources. A wholesale commitment to a technology platform is a massive investment in time, effort and human resources. Because technology architecture forces organizations to establish standard development and operations processes, business or IT leaders sometimes use it to drive change.

NodeSource adds value here by helping organizations assess where they are on the maturity curve—in terms of architecture, people, process and technology—and then helping them to fully realize a Node development and deployment capability.

Tooling

npm

As enumerated above, an internal npm solution should be assembled for use across engineering teams and for deployment. Such a solution will:

- Allow easy sharing of code with minimal overhead for developers
- Enable seamless access to both internal and external Node packages
- Assist in organization as the number of packages increases, preferably via the use of the npm scoped packages feature
- Allow secure and stable deployment to production, isolated from the public registry
- Meet compliance needs regarding code signing and storage, particularly for deployable bundles
- Provide insight to code and dependencies being published, used and deployed throughout.

GitHub Enterprise

GitHub Enterprise is the on-premises version of GitHub.com. It makes collaborative coding possible and enjoyable for large-scale enterprise software development teams. GitHub is a developer tool rather than a management or architect tool and developers will find it more natural to manage their projects, including documentation (and possibly issues) with GitHub rather than other tools that are of greater interest to management and architects. As noted above, effort should be made to reduce friction for developers around code and documentation management.

NodeSource's Product Line

NodeSource is growing a set of product offerings that are designed to help enterprise development and operations organizations successfully develop and deploy an advanced Node capability. NodeSource products are designed to bring trust, performance and productivity to the Node datacenter.

NodeSource N|Ship is the first released component and acts as the curator of your Node environment. N|Ship helps organizations ship high quality code with determinism and reliability. It focuses on:

- **Lifecycle** – defining a platform for software lifecycle pipelines that ensure quality. And then, managing the quality of deployed software throughout its lifecycle, even through the archival and decommissioning phases.
- **Tracking** – creating deterministic bundles that define discrete, unique and cryptographically signed Node artifacts.

- **Extensibility** – Providing extensibility via NodeSource cartridges for integration into existing workflows.

NodeSource is developing future product releases that will provide additional intelligence, automation and runtime enhancements to the Node environment. Watch the *NodeSource blog*¹⁰ for further information on upcoming announcements.

Training

NodeSource delivers a broad suite of introductory, intermediate and advanced Node training courses in JavaScript, Node, Testing and npm.

Introductory

NodeSource offers training at an introductory level for engineers needing to quickly skill-up to work with JavaScript and Node:

- **JavaScript for Node** is a course designed to teach the basics of JavaScript, specifically targeting features and design patterns relevant for use in Node.
- **Node Fundamentals** is a course designed to introduce engineers to Node development. A basic level of JavaScript knowledge is assumed and the material covers core Node, an introduction to the Node ecosystem, and basic npm workflows.

Intermediate

Intermediate courses offer more in-depth topics for application architects/developers and operations teams.

- **Node Application Architecture** is designed to help engineers maximise their use of the Node platform, its associated tools and the Node development ethos to design performant, reliable and secure applications. Topics include: Node design patterns, project organization and code modularization, choosing and using appropriate frameworks, testing and documentation as a first-class activity.
- **Node Testing** aims to teach basic Node and asynchronous testing patterns and libraries to developers with a Test-Driven Development (TDD) approach in order to make testing one of the primary activities of development.
- **npm** is designed to introduce basic through to advanced npm workflows with a greater insight in to how npm serves as an integral tool for Node development, rapid project evolution and Node-style code modularization.

¹⁰ [The NodeSource blog](#)

Advanced/Specialized

NodeSource also has advanced multi-day training:

- **Performance Engineering for Node** is conducted by our performance experts and covers performance engineering focused on JavaScript for V8, Node design patterns for performance, advanced debugging and inspection techniques for application tuning and problem discovery. This course has a heavy workshop component and trainees can bring specific problems and learn how to solve them.
- **DevOps for Node** caters to operations teams having to deploy and manage Node applications. This is designed as a stand-alone course for operations staff and covers the basics of the Node runtime model and how it differs from traditional platforms (such as the JVM), npm and its impact on deployment and bundling, deployment / network architectures, security and monitoring. Other special use cases can also be covered such as embedded systems and hardware topics.

NodeSource provides targeted short-courses on advanced or specialized Node topics that go beyond the treatment given in the introductory and intermediate material. Topics including Streams, Buffers and Binary Data and other topics that deepen knowledge and skills in various areas.

Short-courses should be used as a mechanism to help developers increase the depth of their understanding of Node and to move beyond basic patterns to more optimal code that takes full advantage of the platform.

NodeSource can work with you to design a series of specialized short-courses for engineers already on-boarded to Node projects to ensure ongoing professional development.

Support from NodeSource

NodeSource offers world-class support that enables companies to build, iterate and grow both their Node development team and the software itself. Working in conjunction with NodeSource training offerings, NodeSource can help guide and support your organization through each stage of the Node Maturity Curve.