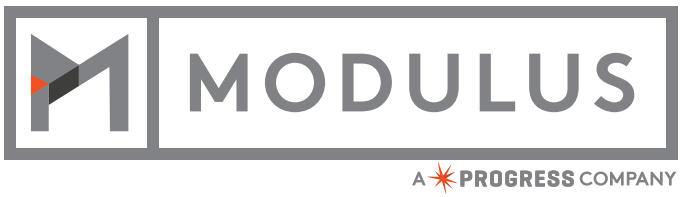

CRITICAL CONSIDERATIONS WHEN BUILDING ENTERPRISE NODE.JS APPLICATIONS

Dave Anderson & Greg Neiheisel / differential.io

OCTOBER 2014

CRITICAL CONSIDERATIONS WHEN BUILDING ENTERPRISE NODE.JS APPLICATIONS

Develop agile, competitive and secure applications with Node

Dave Anderson & Greg Neiheisel / differential.io

INTRODUCTION

Node.js is rapidly becoming the technology of choice for the enterprise, with use growing exponentially over the past two years. Industry leaders like Walmart and PayPal are building enterprise Node applications to optimize performance and achieve a competitive edge.

Node is particularly well suited for enterprise applications, because unlike traditional development methods, Node composes applications from small modules that are then connected together. Traditional monolithic applications become rigid over time, making them difficult and time consuming to change in response to new business requirements, evolving market trends, and emerging technologies. With Node, organizations can develop a group of small applications versus one large application, enabling a change to be made or new functionality to be added without modifications deep inside the entire code-base.

The business benefits of this approach are substantial, including:

- Rapid innovation
- Accelerated time to value and time to market
- Flexibility to support fluctuating compliance, regulatory, and security needs
- High performance within high-transaction environments
- Ease of maintenance
- Increased developer productivity

This white paper will take a deeper look into why Node is ideal for building enterprise applications, and how Node can help your organization develop agile, competitive, and secure applications.

DRIVING BUSINESS AGILITY WITH NODE

How is building an enterprise application different from building a regular application? The primary difference is that the enterprise has processes and procedures that have to be followed, whereas smaller companies and groups can rely on a more ad-hoc approach.

Developing enterprise applications requires planning. However, enterprise development groups must also be flexible, constantly adapting to internal and external factors. As a result, the enterprise planning process has evolved to accommodate these dual, and seemingly contradictory imperatives. Today, planning for enterprise application development is more concerned with making sure that business priorities—such as compliance and security—are rigorously addressed versus adherence to plan for its own sake.

The good news is that most innovative companies have moved from a Waterfall project management approach for building applications to an Agile development approach. Rather than designing and building applications from the top down—one project at a time—these innovative companies have groups of developers working on different parts of the application simultaneously. Node fully complements Agile development, enabling organizations to rapidly build applications while maintaining the agility to modify them to meet ever-changing business needs.

The flexibility afforded by Node means companies are not constrained, limited, or compromised by their technology environment. Rather, Node empowers them to quickly react to business and market change and ensure compliance with constantly changing corporate governance policies and regulatory compliance mandates.

FACTORS AND BEST PRACTICES TO CONSIDER WHEN BUILDING NODE APPLICATIONS

Horizontal Scaling is Necessary to Address a Large User Base

As the Internet continually expands, a site's ability to respond to massive surges in visitor traffic has become a pressing concern. Horizontal scalability provides the ability to increase capacity on the fly while ensuring system uptime.

Today's preferred web application frameworks tend to move processing to the client and use very lightweight server processes. More client processing makes perfect sense; if work can be offloaded from the server, the server can service more clients. More clients per server generally translate into lower costs, and fewer servers for the same number of clients is great for the bottom line. But this has to be tempered somewhat. If the client needs more data sent down the wire in order to perform the offloaded calculations, there may be diminishing returns. Balance is key.

The other side of the scaling issue is stateless communication. When the client sends a request for service, it's advantageous for the least loaded server to handle it. If there's no state associated with the request, any server will do. That's best case. Sometimes the client and a server need to have an extended conversation; however, the stateless communication doesn't fit the problem. In such cases, the best things to do are to load balance, keep connections open, and send lots of little chunks of conversation instead of large amounts of data all at once that might impede the server's ability to handle other requests.

Node handles both cases with ease, taking care of much of what's going on behind the scenes for developers. In fact, building enhanced communications using Streams makes moving sizeable data from server to client even more straightforward. Readable streams are sources of data, writable streams are destinations, duplex streams can read and write, and they all deal with messages broken into chunks. They encapsulate the client-server data transfer issues effectively and efficiently.

Proven Frameworks Should Be Used to Ensure Infrastructural Quality

Before Node, the JavaScript language was tied to browsers. Node can be thought of as an ecosystem that enables applications to be written in JavaScript and run independent of web browsers. This means that Node enabled the development of JavaScript application frameworks. These frameworks establish the high-level architecture of the web applications built upon them, laying out how to do things as well as what can (and cannot) be done.

Node does not suffer from lack of great frameworks for building web applications. The first extremely useful web application framework on Node was Express, which remains extremely popular. Inspired by the Sinatra framework for Ruby, it focuses on creating an application that accepts requests from a port on the server using code that generates a response when the request matches a particular pattern. Express is a no-frills framework; it's bare-bones http. Authentication isn't even part of Express; other Node packages supply authentication.

Today, Walmart has thrown its weight behind a framework called hapi. A more ambitious effort than Express, hapi includes more of the standard web application needs within the framework without relying on other packages. With hapi, developers implement applications using the logic in plugins, which sit atop packs of servers, making it easier to build different composite applications out of different plugins to meet enterprise needs.

Another interesting framework is LoopBack, which takes the Ruby on Rails perspective of convention over configuration. LoopBack focuses on letting the developer write less code to get more done, and includes more features, including object-relational mapping and monitoring.

Sails, a Ruby-on-Rails-like MVC framework in JavaScript for Node, includes the ability to map to any database and any front end, and includes a number of code generators as well as an asset pipeline.

Meteor is a very different framework that delivers a native application user experience in web applications (also optimized for personal devices) using mechanisms like reactivity, latency compensation, and isomorphic JavaScript. Meteor allows code to be shared between client and server as needed.

There are a number of good web application frameworks for Node. Some are older and require more coding to use, while others are newer and are more monolithic. Undoubtedly, there will be more of both to come.

All Parts of an Applications Should Be Tested Using Automated Testing Frameworks

Testing software is one of the most important aspects of enterprise software development. Node applications have the same testing requirements as any other application or system.

Because Node is JavaScript, most JavaScript test frameworks are useful for validating the code in a Node system. There are also quite a few testing frameworks that have evolved specifically for the Node ecosystem. Principal among these are Mocha and Jasmine. Both are well featured, allowing tests to be written in the familiar test-driven-development style. They generate good reports that simplify identifying problems. Mocha has been built to run under Node, whereas Jasmine exists outside of Node but has been extended to run under the Node umbrella.

Expresso looks a little less like standard TDD, since it concentrates on making explicit assertions. Should is even less standard, extending objects with should assertions, indicating whether or not things are as expected. NodeUnit is similar, asserting expectations that pass or fail. Finally, JSUnit focuses on testing client side JavaScript in the browser.

To meet the obvious and growing need for rigorous testing, especially for enterprise-class apps, there is no shortage of Node testing frameworks. Automated frameworks will simplify this critical task.

Modern Tools Should Be Used to Identify Problems

There are a number of tools available to help enterprise developers quickly identify and understand bugs to ensure robust and secure applications.

At the top of the list is Node Inspector, maintained and supported by the same people who provide LoopBack. Supplied with the name of the Node app's main JavaScript file, it loads into a Chrome or Opera web browser and provides a significantly complete debugging interface. And because it runs on WebSockets, Node Inspector enables remote debugging of any Node application running on the same machine through port 8080 (similar to Chrome's Developer tools).

Dtrace (dynamic tracepoints) is used for a different kind of debugging. Instead of interacting directly with code, it profiles the code in a very lightweight and unobtrusive way. Imagine attaching a probe to electronics on a circuit board. The probe captures voltage values as the electronics are being used. Dtrace probes into Node applications and captures their state as they run. This perspective helps uncover unexpected flows within the software, pointing to areas that may benefit from streamlining or logic changes.

Node-heapdump captures a snapshot of the heap (used for memory allocation) to help identify leaky memory resources. In the beginning, JavaScript's use of its memory heap was short lived. Web pages were transient, and JavaScript use was born again on each page. When single-page web apps (e.g., Google Mail) hit the scene, leaky memory became an obstacle. Node applications can run for days, weeks or months. A memory leak in such a process can lead to serious issues. Heapdump can help developers locate these issues and fix the leaks.

Application Deployment Processes Should Not Be a Point of Failure

When an enterprise application is ready for production, deployment issues should not be an issue. The process for getting the application up and running on a server (or set of servers) should be fast and easy, particularly since a bug fix may need to be delivered almost immediately.

Modulus specializes in getting Node applications online quickly and simply, eliminating the need for in-house servers and support and significantly reducing the costs. Modulus also supports deployment on-premises, in the cloud, or in hybrid environments. Download the Modulus Getting Started guide for more information.

Production Monitoring Must Be Used to Stay Ahead of Problems

Monitoring a Node application identify performance issues in production. For example, an organization can identify slow application response before it impacts the customer experience, or pinpoint why certain servers continue to fail.

Modulus provides visibility into Node applications through a centralized dashboard, providing real-time intelligence on application performance. Developers can delve into statistics on component usage, processing demands, server status, and much more.

When critical points in the application result in poor performance, a tool like New Relic becomes useful New Relics identifies and traces database and memory usage, as well as particularly slow transactions. Unlike a Dtrace however, there can be significant overhead in this style of monitoring. Data is generated and sent to a New Relic—sometimes a lot of data—so it should be used sparingly. Otherwise, the monitoring itself can become a bottleneck.

Another noteworthy monitoring solution is nodetime, which collects statistics on Node applications and provides centralized Node application profiling and monitoring. The service features performance, resource, and database monitoring, as well as on-demand tools that connect to the production application and provide additional information. But again, with such monitoring, more data must be generated which can also cause bottlenecks. Therefore, balance is important. Use intrusive monitoring carefully to access the information required, but not more than necessary.

MEETING BOTH BUSINESS AND DEVELOPER NEEDS WITH MODULUS

Node really is ready for prime time in the world of enterprise applications. Shrinking the application developer's stack down to one language—JavaScript—and running on both the server and client greatly simplifies delivering functionality to the marketplace through the web.

Modulus allows development of Node applications on a dedicated, enterprise class platform, accelerating their time to value while maintaining the agility to rapidly respond to the needs of the business. The Modulus enterprise platform provides a secure, robust, and feature-rich environment, readily configurable to accommodate compliance and regulatory needs.

For more information, visit https://modulus.io.