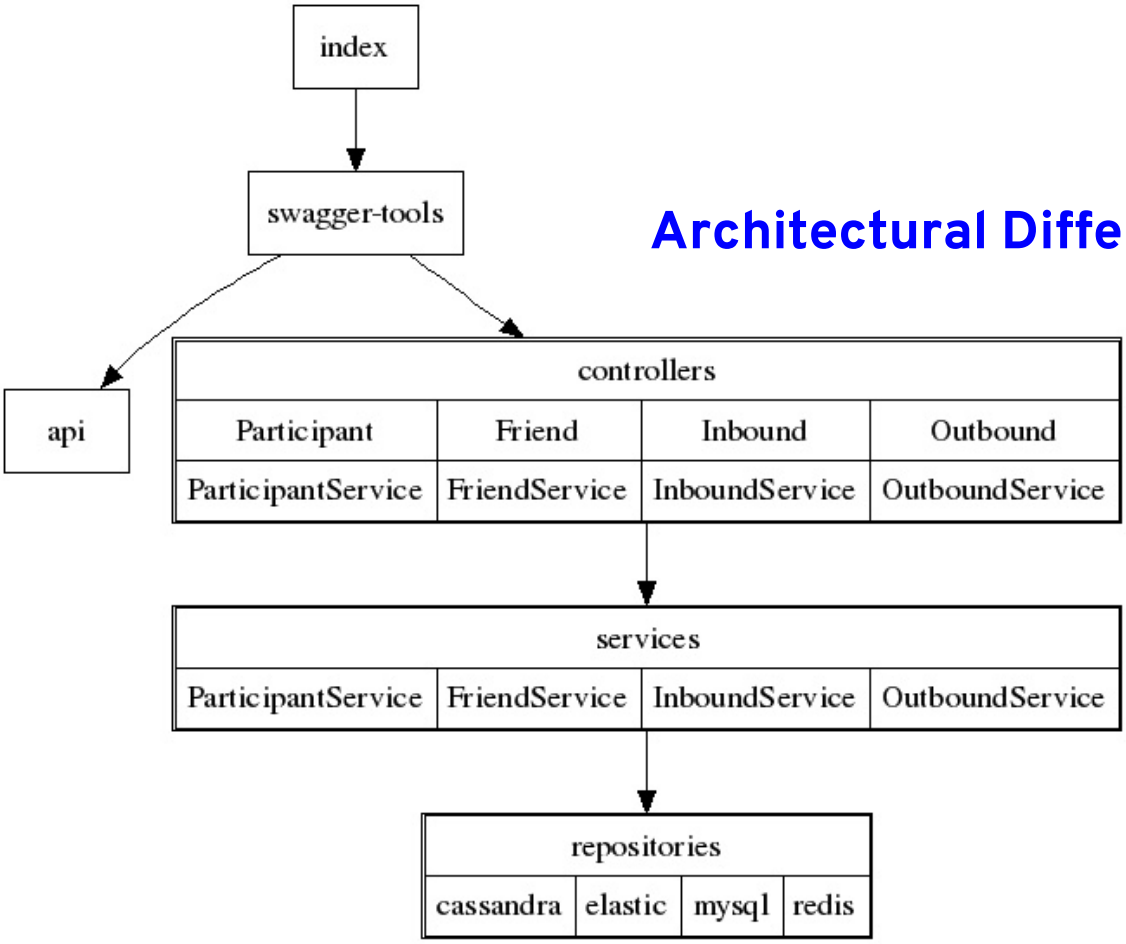
Node.js vs DropWizard

by Glenn Engstrand

There has been a lot of engineering interest in Node.js (i.e. server-side javascript), especially in small start-up technology companies. There is also a lot of ideological conflict between those engineers, who advocate for Node.js, and those who prefer Java.

So much that I thought it would be a good time to explore Node.js by implementing my usual rudimentary news feed micro-service in Node.js then comparing that to the DropWizard version of the same micro-service. This blog documents my findings in how these two technologies are different.



Architectural Differences

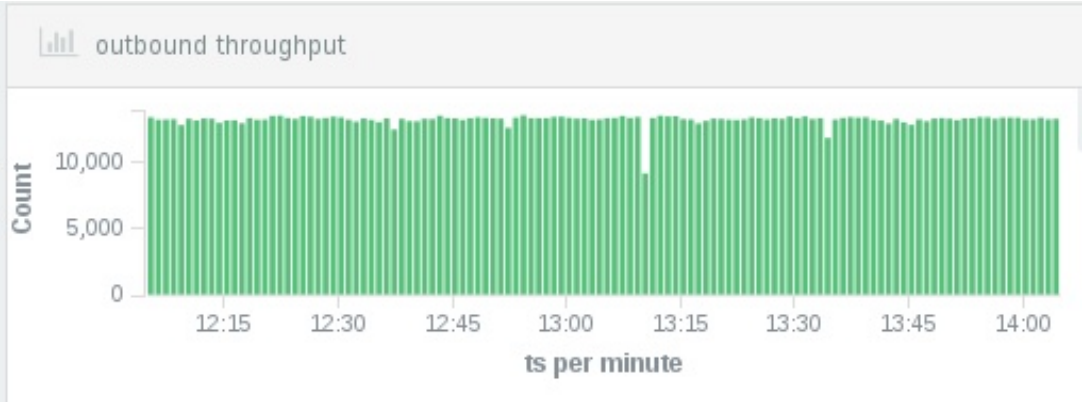
There are a lot of profound architectural differences between a Node.js micro-service and one written in Java. Perhaps the biggest difference is that javascript is dynamically typed whereas Java is statically typed. When you build an application written in Java, you compile the source code into machine code for the Java Virtual Machine.

Part of that compilation process is that the compiler checks to see that identifiers are used in a way that is consistent with their types. You don't get that in javascript at all. This has profound ramifications throughout the entire Software Development Life Cycle from choice of IDE to the amount of code coverage in unit tests.

Node.js uses swagger-tools for the RESTful parts.

Another big difference between Node.js and Java is the threading model. DropWizard integrates with Jetty which is a multi-threaded web service container. Node.js uses a single threaded event loop model where you have lots of little event handlers all running cooperatively on a single thread.

Both the DropWizard and the Node.js micro-service used Swagger to specify the RESTful parts. The difference here is that, with the DropWizard application, the Swagger code generator wrote the Java code which depends on Jersey whereas the Node.js application depends on the swagger-tools project which parses the Swagger YAML specification at time of service start in order to dynamically register all the RESTful parts.



Software Design Differences

That difference in threading model means that the javascript developer must design their functions, that perform I/O requests, in a very specific way. Instead of waiting on I/O then returning the value from that request, the caller passes in a callback routine that gets called in the function with whatever was going to be returned. This callback function also gets called with any errors that occur too.

This has to propagate all the way up the call stack. If a function calls another function that requests I/O, then both functions have to use callbacks instead of returning values. It is unwise to make any assumptions about the ordering of execution of lines of code within a callback and without. Any state that needs to be cleaned up (such as closing connections from connection pools) should be done before calling the callback.

There is a very popular trend in Java development to use Dependency Injection because it is an effective way to afford flexibility without dramatically increasing design complexity. The Node.js community rejects DI. Instead, they use modules and configure their applications using environment variables.

The javascript programming language does support prototype based Object Oriented Programming but it doesn't have quite the relevance in Node.js that it does in Java. The typical Node.js developer most probably consumes objects, such as connections in connection pools, but they don't produce them. Instead, they make use of Node.js's module system. Think of modules as singleton objects.

Coding Differences

The architecture and design differences make Node.js code less flexible than its Java equivalent but it does make it simpler and smaller too.

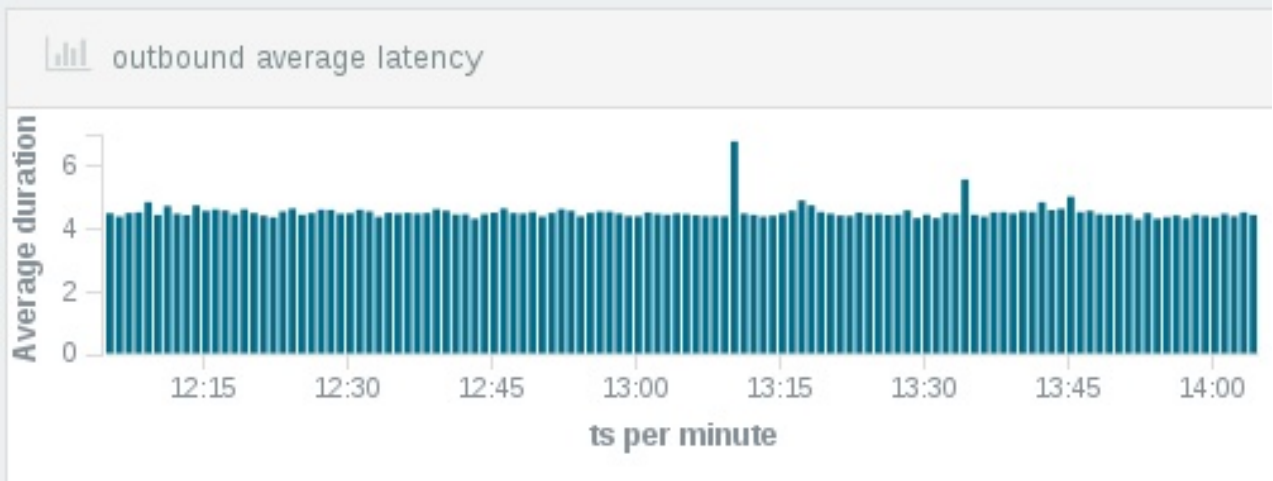
The Node.js implementation of the news feed service is half the number of files as the DropWizard implementation and one fifth the number of lines of code.

Because there is no compiler in Node.js, there is more dependency on unit tests in order to assure software quality. The biggest source code file, in terms of lines of code, is where the unit tests are implemented. It is twice as big as the second biggest source code file and about 20% of the entire code base.

Stability Differences

Even though Node.js has gotten a lot of traction recently, it is still less stable than Java. I tried with two different libraries but I could never get Node.js code to successfully connect to Kafka. Also, any unhandled error or exception in Node.js terminates the entire process.

I am not sure if this is a Node.js stability problem but I found that I could not ssh back into any EC2 instance once I installed Node.js and NPM (Node Package Manager) on it.



Performance Differences

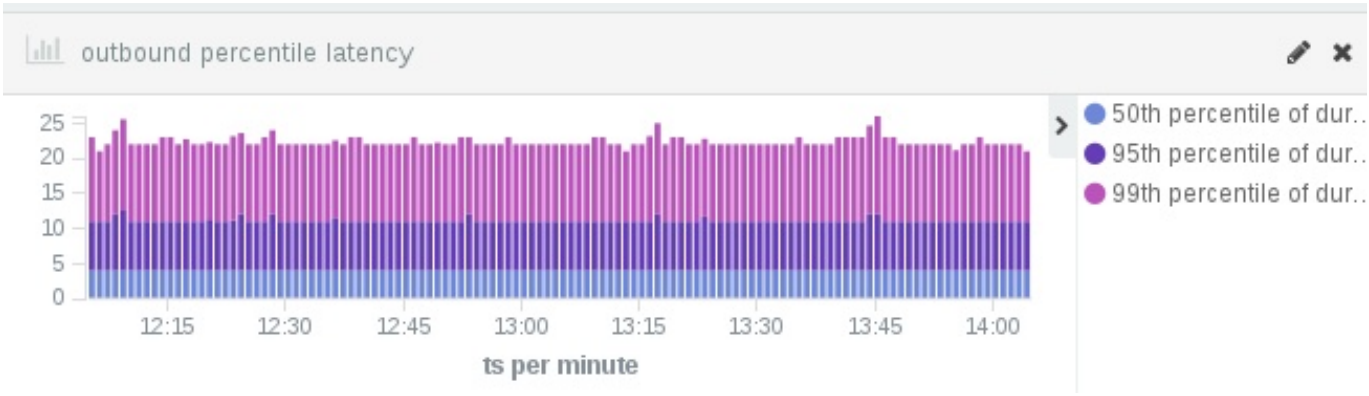
Node.js	duration (ms)
Average:	4.5
Median:	4
95th percentile	7
99th percentile:	11

DropWizard	duration (ms)
Average:	4.8
Median:	4
95th percentile	7
99th percentile:	9

	Throughput (RPM)
Node.js:	13300
DropWizard:	15700

At one fifth the code size, Node.js had comparable latency and 16% lower throughput than DropWizard.

I ran my standard load test on EC2 for the Node.js micro-service and captured the performance relevant results. I then compared those results to a recent load test of the DropWizard micro-service with Kafka configured off. This is what I found. The Node.js service achieved 16% lower throughput than the DropWizard service. Latency for the Node.js service was very comparable to that of the DropWizard service under the same load and configuration conditions.



Is Node.js right for you?

I can see how Node.js has caught on with early stage start-up companies. It is easier to write micro-services very quickly in Node.js and get them running than it is with Java. As companies mature, their focus tends to shift from finding product / market fit to improving economies of scale. This might explain why more established companies prefer Java with its higher scalability.