**About Me**

* Solutions architect at MYOB
* Born in Dublin Ireland, moved to Australia in 2006
* 16 years development experience working with Microsoft technologies
* Currently working on a cloud platform for the next generation of MYOB’s accounting products – C#, REST, SQL Server, .NET 4.0

**What Is Node.js**

* Open Source Project , Ryan Dahl creator
* 3 years old, very new technology IISNode only arrived in 2011 a year ago
* gaining traction quickly - LinkedIn, FaceBook, Walmart, EBay all using it
* 10000 lines of c++ code 6mb
* more followers than Ruby on Rails on GitHub

**Graph**

* V8 increases performance by compiling JavaScript to native machine code & employing caching optimization methods

**Event Loop Processing**

* The main event loop uses a single thread to handle multiple concurrent connections, which makes the overhead of Node.JS grow relatively slowly as the number of requests it has to serve increases as there’s no OS thread / process initialization overhead;
* All long-running tasks (network I/O, data access, etc…) are always executed asynchronously on top of worker threads which return the results via callback to the event loop thread; and
* JavaScript’s language features (functions as objects, closures, etc…) and Node’s programming model make this type of asynchronous / concurrent programming much easier to utilize – there’s no thread management, no synchronization mechanisms, and no message-passing nonsense. This eliminates a lot of pitfalls that most developers fall into when attempting to develop concurrent applications.
* IIS has a thread-per-request model, and although it is extremely well-implemented under the hood it still introduces context-switching and associated overhead. Node.JS does not - it just allocates a tiny record onto the heap, farms out work to a pool of green threads (like CLR threads) and then pushes the response to the request via callback to the main event loop which returns the resultant HTTP response. This makes Node ideal for scenarios where you have a large number of requests but a relatively small responses
* **IIS typically 100 threads maximum per processor**

**Node Package Manager**

* very active community- all packages are open sourced
* the challenge is picking the best ones
* **Azure packages** for interacting with Queues, Table Storage, Service Bus and Blob storage
* NPM registry contains over 10,000 open source packages
* Packages are easily installed using a command line tool (node package manger)
* Not all packages are cross platform (some target O/S specific features)
* Beware of dependency hell !

**Common Packages**

Express - web development framework

SocketIO - real time

Underscore - utilities library that gives you ForEach iterators, Map Reduce functionality etc

**Azure SDK for Node.js**

* Really easy to deploy (locally / cloud) using PowerShell cmdlets
* Azure package gives access to
* blob service, table service, queue service, service bus …
* Currently you can’t deploy multiple node apps to the same instance
* You need to push up the source code for all packages you use
* Debugging supported via IISNode & node Inspector

**When Should I use it**

 High-volume, small response-size systems – think of a real-time logging system where you have to write millions of messages an hour and only send small ACK response objects back to the caller – Node.JS is perfectly suited to handle this;

 Real-time Pub/Sub systems – if you have an application that is heavy on notifications and needs to do it in real-time, Node is a great choice given its penchant for getting in and out of requests quickly.