

09/01/2015 • 8 minutes to read

## In this article

[Node.js](#)

[Azure Command-Line Tools](#)

[Deploy to Azure](#)

[Wrapping Up](#)

---

September 2015

Volume 30 Number 9

# The Working Programmer - How To Be MEAN: Node.js

By [Ted Neward](#) | September 2015



Microsoft has been adopting technology from across the software spectrum as part of its rebranding and “re-relevancing.” One of the technologies it has adopted is Node.js. This gives developers a golden opportunity to use one of the more popular full-stack software groupings on the Node.js platform known as MEAN: MongoDB, Express, AngularJS and Node.js.


In the last installment ([msdn.microsoft.com/magazine/mt185576](https://msdn.microsoft.com/magazine/mt185576)), I got the basic Node.js parts up and running. In this one, I’ll spin up a simple Node “Hello World” Web endpoint, and deploy it to a Microsoft Azure Web site. Over the next several installments, I’ll slowly build my way through the MEAN stack, working from the ground up.

As discussed in the previous article, there are a lot of places where I could swap out parts of the MEAN stack for something else—DocumentDB for MongoDB, ASP.NET WebAPI for Node.js and ASP.NET MVC for Express, or BackboneJS (or any of a whole host of other JavaScript Single-Page-Application frameworks) for AngularJS—but none of the alternatives enjoys the popularity that MEAN currently holds (at least among JavaScript aficionados).

# Node.js

Node.js is fundamentally just “JavaScript on the server.” Yes, there’s a different programming approach to Node.js that tries to deal with concurrent execution. Instead of block on a call, the programmer passes in a function literal to be invoked when the operation is completed. This lets the developer to think of code as single-threaded, even though multiple threads are in use below the surface. On the whole, though, the largest distinction of Node.js is you use JavaScript to build the server components, instead of C#, Java or Ruby. In that sense, it’s really just a change of scenery—not an entirely different universe.

The simplest Node.js application is, of course, the ubiquitous “Hello, World,” which you can easily write using the built-in console object:

XML	 Copy
<pre>console.log("Howdy, NodeJS!");</pre>	

Assuming this goes into a file called `hello.js` in the current directory, you’d run it using the node utility at the command-line using `node hello.js`. Or you could let Node.js infer the filename extension by simply running “node hello.” Either way, Node greets you in the traditional way.

Like most programming platforms, Node.js has its own set of libraries and APIs out of the box. As I pointed out last time, Node.js uses the `require` convention to reference an installed library. This traps the returned object into a local variable of the same name. So, for example, if I want to write a simple HTTP server that will effectively give me the same greeting over the HTTP protocol, I can put the following into a simple `helloHTTP.js` file:

JavaScript	 Copy
<pre>var http = require('http'); var port = process.env.PORT    3000; http.createServer(function(req, res) {   res.writeHead(200, { 'Content-Type': 'text/plain' });   res.end('Hello World\n'); }).listen(port);</pre>	

The `require` line finds the “http” library within the Node.js installation, and stores it to the `http` object in dependency injection. This is a standard Node.js convention, and should be held as fairly sacrosanct. The second line uses the built-in `process` object to access the surrounding environment. In this case, it uses the “env” object within the `process` object to determine whether an environment variable named `PORT` is set to

anything. If it is, I'll use that as the port on which to run the server. Otherwise, I'll use the default port 3000. Many Node.js frameworks prefer port 3000 as the default, for obscure historical and cultural reasons.


The nature of Node.js programming becomes clearer in the next few lines. I use the `http` object to create an HTTP server. The sole parameter is a function literal that takes a `req` (HTTP request) object and a `res` (HTTP response) object as parameters, and uses `res` to write the HTTP response back. This idiom is ubiquitous throughout all levels of the Node.js stack.

This is one of those "you love it or you hate it" kinds of things. You'll be seeing more of this in the articles to come. So if this isn't clear, spend some time experimenting. The returned object from `createServer` is then bound to the desired port using the `listen` call. Lo and behold, you have a running HTTP server you can easily run using "node helloHTTP" and pointing a browser at `http://localhost:3000`.

## Azure Command-Line Tools

From my last article, you'll recall Node.js has a package utility called the Node package manager (npm) you can use to download dependent libraries. You can also use it to download tools you can then use from the command line. This is a subtle, but powerful, aspect of Node.js. It acts as a platform-neutral "playing field equalizer."


You can effectively hide any distinctions between Windows, Mac OS or Linux behind a wall of JavaScript scripts. Microsoft picked up on this a while back, and packaged a set of command-line tools into a Node.js package called `azure-cli`. Installing it is easy with npm:

XML	 Copy
<pre>npm install -g azure-cli</pre>	


The `-g` flag tells npm to install the tools "globally" (meaning they're not tied to the local directory in which the command is run). This makes the resulting package available for use across the entire system. When it's finished, a new command-line utility, `azure`, will be available for use. The `azure-cli` package doesn't give any greater or lesser functionality than using the Azure Portal. The `azure-cli` tool's advantage lies in its ability to script `azure` commands as part of an automated deploy system, for example.

## Deploy to Azure


So, if you want this lovely little greeting to be available to the world over the Internet, you need to create an Azure Web Site as a host. Using the “azure” tool, it’s pretty simple. First, you have to tie the tool in to the account in Azure:

XML	 Copy
<pre>azure account download</pre>	


This will fire up the system’s default browser pointing to the Azure login portal. Use the Azure account credentials to log in. When you’re done, it will automatically download a publishSettings file containing the credential information the azure tool needs, which can be directly imported:

XML	 Copy
<pre>azure account import &lt;filename&gt;</pre>	

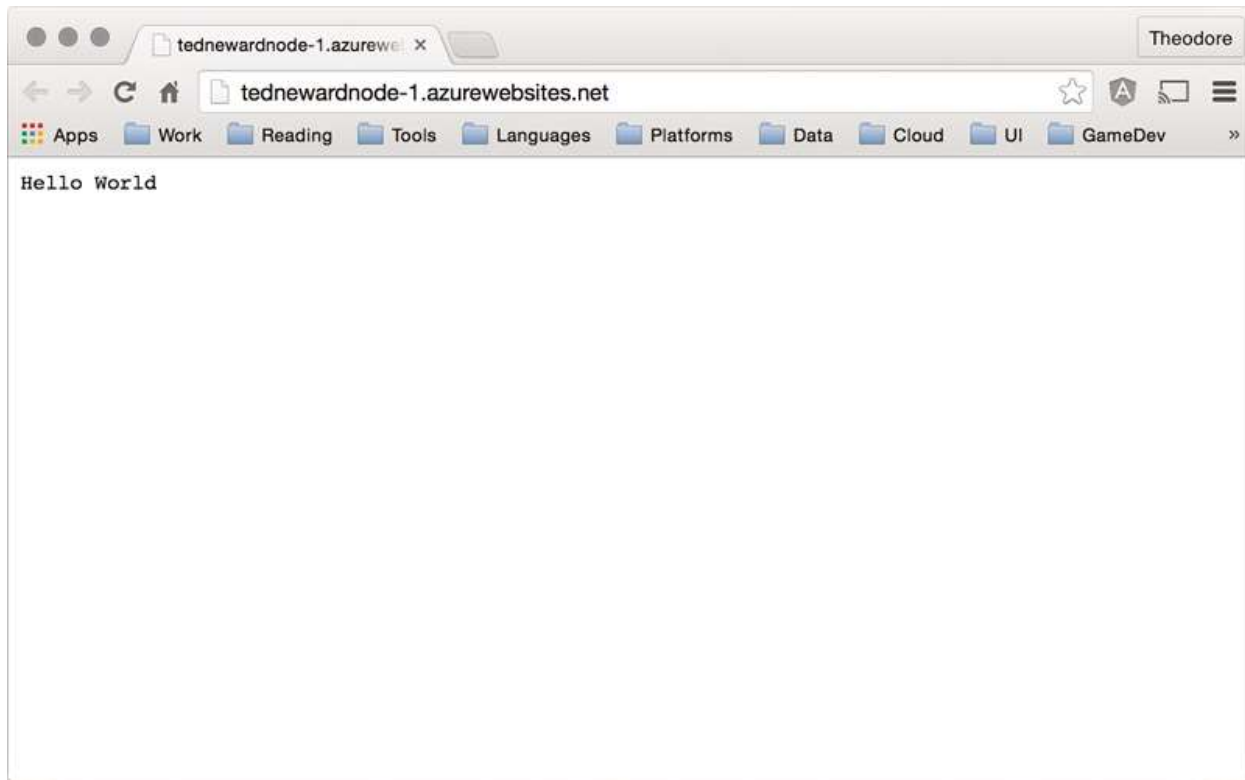
Filename will often be something like “Visual Studio Ultimate with MSDN-4-23-2015-credentials.publishsettings,” depending on the Azure subscription details and the date you download the publishSettings file. Once that’s done, it’s a simple matter to create a new Azure Web site by setting up Git to do deployments to the site:

XML	 Copy
<pre>azure site create -git</pre>	

This will prompt for a site name, and (assuming that name is unique) spin up the appropriate Web site. If all that works, spin up the current directory as a git-initialized local repository. Assuming you’re still in the same directory that holds the helloHTTP.js file from before, you can add it to the Git repository and push it to the Azure cloud:

XML	 Copy
<pre>git add helloHTTP.js git commit -m "Initial commit" git push azure master</pre>	

Git will think about it for a few seconds. Then it will go through a sequence of steps that remain opaque for the moment. When it’s finished, though, Azure will hold the new Node.js code, and you can browse to it, as shown in **Figure 1**.



**Figure 1 Hello, Node World**

This is why the `helloHTTP` code uses either the default port 3000 or the environment variable `PORT` from the surrounding process. On the Azure cloud, `PORT` is set to a value the Azure infrastructure maintains. This is so Microsoft can manage the various service endpoints more efficiently.

That's really the last of the installation steps you need to do. Honestly, it takes longer to read about it than it does to actually run the commands, once you're past the initial setup stages. And even more honestly, any Azure work or exploration will require similar kinds of setup. Azure really represents a next-step platform for many development efforts. In other words, this is something you'll need to know how to do eventually, so you may as well learn about it now.

From here, pretty much everything will be MEAN-related. The only time you'll need to bring Azure details to mind is when you need to configure the environment to point to the MongoDB database server, for example, or when dealing with how Azure interacts with Node.js.

## Wrapping Up

It's worth pointing out that Node.js isn't just an HTTP pipeline. In fact, Node can run any kind of networked application by opening up the right library. The same is true of the Microsoft .NET Framework. However, like the .NET Framework, most Node.js applications are going to be HTTP-based in nature.

The http library in Node.js is pretty low-level. As a result, the Node.js community developed a higher-level library and set of abstractions to make dealing with HTTP-based endpoints easier. This library is called Express. It lets you build what the Node.js community calls “middleware.” That’s what I’ll be looking at next time.

For now, experiment with the Node.js http library, but don’t get too attached, because I’ll be leaving it behind pretty quickly once I get into the next iteration. In the meantime, however ...

Happy coding.

---

**Ted Neward** *is the CTO at iTrellis, a consulting services company. He has written more than 100 articles and authored or co-authored a dozen books, including “Professional F# 2.0” (Wrox, 2010). He is an F# MVP and speaks at conferences around the world. He consults and mentors regularly—reach him at [ted@tedneward.com](mailto:ted@tedneward.com) or [ted@itrellis.com](mailto:ted@itrellis.com) if you’re interested.*

Thanks to the following technical expert for reviewing this article: Shawn Wildermuth  
Shawn Wildermuth is a thirteen-time Microsoft MVP (ASP.NET/IIS), the author of eight books and dozens of articles on software development, and a Pluralsight author with over fifteen courses to his name. He is one of the Wilder Minds (<http://wilder minds.com> ) and can be reached at his blog at <http://wildermuth.com> .