# Azure Development of NodeJS Applications

## Overview

This document serves as a brief overview of the challenges presented by administration, developing and deploying a NodeJS applications in an Azure environment. The development resources making up the team consisted of seasoned Microsoft Stack veterans with cross functional development and Azure cloud experience.

## Contact Information

|  |  |
| --- | --- |
| [Company] | https://[company].com |
| Christopher Barr | chris.barr@[company].com |

## Solution Overview

The NodeJS application referenced in this document is part of a larger real time IOT and data analytics solution, as described below:

|  |  |
| --- | --- |
| Azure Event Hubs | Event hubs to receive and store data for 24-48 hours |
| Streaming Analytic Jobs | Jobs to process raw data to staged data, then analyze staged data |
| Azure Functions | Function trigger on event hubs to send an event with information about the data to a socket server. |
| NodeJS Web Application | Socket Server - receives event data and broadcasts to clients (NodeJS, socketio, expressjs) |
| Static Web Application | Subscriber of socket server events to display data in real time charts |

## Challenges

Understanding the role ASP.NET & IIS play in regards to serving up a NodeJS Web Application hosted on Azure App Service.

Struggles encountered with the solution presented above came when attempting to deploy our NodeJS Socket Server to Azure App Service. Our hope was we could create a new app service, set the NodeJS version in Application Settings then FTP our files up and being serving our HTML and broadcasting events. We were sadly mistaken… what ensued was well over two weeks of googling and struggling to find Microsoft Documentation on how ASP.NET and IIS host NodeJS Web Applications.

A few pain points:

* ASP.NET/IIS host NodeJS and are exposed by ports 80 or 443, and route incoming requests to the underlying NodeJS application via named pipes. We found this out not by reading documentation, but by attempting to debug our application and console.log the port it was being hosted on. (We had our app hardcoded to point to port 80 and not process.env.PORT || 80 - and did not see any documentation anywhere stating that ASP.NET will set the environment port for us)
* ASP.NET requires a web.config file with some interesting settings not documented by Microsoft. We found out about this web.config file by standing up an old “NodeJS Empty Web App” template from the Marketplace instead of a regular Azure App Service. This is where we learned our NodeJS entry file needs to be named server.js (or at least we could change the setting here) - along with quite a few other settings we could modify at our leisure. But we were unable to find any Microsoft documentation on this.
* Mostly all Microsoft documentation in regards to deployment of NodeJS web applications is via kudo, zipdeploy and/or azure CLI. All three of these perform some magic behind the scenes and set up the web.config file based on analyzing package.json file… there does not appear to be any in-depth overview of exactly what is being analyzed here to ensure the NodeJS application is ‘setup’ correctly after deployment. After much googling of the kudo project we were able to figure this out, eventually.

## Suggestions

* Allow easier, straight forward FTP of NodeJS application to App Service without the confusing web.config setup. Drag drop files over FTP and just start serving.
* Improve the process chain of Azure Continuous Delivery + VSTS Configuration + Kudu behind the scenes magic. It can be very jarring for non Microsoft Developer (or event MS Developer) to setup continuous delivery in Azure, only to see they are booted out to VSTS and required to setup a user account there and further customize the build process in a tool that, lets be honest, is not as simple to use as other build and deployment applications.
* Improve the documentation around how ASP.NET/IIS host NodeJS web applications and the relationships between them.