Best practices and troubleshooting guide for node applicatio

11/09/2017 12 minutes to read Contributors

In this article

In this article, you learn best practices and troubleshooting steps for node applications running on Azure Web Apps (with iisnode).

> **Warning**
>
> Use caution when using troubleshooting steps on your production site. Recommendation is to troubleshoot your app on a non-production setup for examp slot with your production slot.

IISNODE configuration

This schema file shows all the settings that you can configure for iisnode. Some of the settings that are useful for your application:

nodeProcessCountPerApplication

This setting controls the number of node processes that are launched per IIS application. The default value is 1. You can launch as many node.exes as your VM 0 for most applications so you can use all of the vCPUs on your machine. Node.exe is single-threaded so one node.exe consumes a maximum of 1 vCPU. To ge use all vCPUs.

nodeProcessCommandLine

This setting controls the path to the node.exe. You can set this value to point to your node.exe version.

maxConcurrentRequestsPerProcess

This setting controls the maximum number of concurrent requests sent by iisnode to each node.exe. On Azure Web Apps, the default value is Infinite. When n configure the value depending on how many requests your application receives and how fast your application processes each request.

maxNamedPipeConnectionRetry

This setting controls the maximum number of times iisnode retries making the connection on the named pipe to send the requests to node.exe. This setting ir total timeout of each request within iisnode. The default value is 200 on Azure Web Apps. Total Timeout in seconds = (maxNamedPipeConnectionRetry * nam

namedPipeConnectionRetryDelay

This setting controls the amount of time (in ms) iisnode waits between each retry to send the request to node.exe over the named pipe. The default value is 25 namedPipeConnectionRetryDelay) / 1000

By default, the total timeout in iisnode on Azure Web Apps is 200 * 250 ms = 50 seconds.

logDirectory

This setting controls the directory where iisnode logs stdout/stderr. The default value is iisnode, which is relative to the main script directory (directory where r

debuggerExtensionDll

This setting controls what version of node-inspector iisnode uses when debugging your node application. Currently, iisnode-inspector-0.7.3.dll and iisnode-ins value is iisnode-inspector-0.7.3.dll. The iisnode-inspector-0.7.3.dll version uses node-inspector-0.7.3 and uses web sockets. Enable web sockets on your Azure http://ranjithblogs.azurewebsites.net/?p=98 for more details on how to configure iisnode to use the new node-inspector.

flushResponse

The default behavior of IIS is that it buffers response data up to 4 MB before flushing, or until the end of the response, whichever comes first. iisnode offers a c the response entity body as soon as iisnode receives it from node.exe, you need to set the iisnode/@flushResponse attribute in web.config to 'true':

```
XML

<configuration>
    <system.webServer>
        <!-- ... -->
        <iisnode flushResponse="true" />
    </system.webServer>
</configuration>
```

Enable the flushing of every fragment of the response entity body adds performance overhead that reduces the throughput of the system by ~5% (as of v0.1.1 response streaming (for example, using the `<location>` element in the web.config)

In addition to this, for streaming applications, you must also set responseBufferLimit of your iisnode handler to 0.

```
XML

<handlers>
    <add name="iisnode" path="app.js" verb="\*" modules="iisnode" responseBufferLimit="0"/>
</handlers>
```

watchedFiles

A semi-colon separated list of files that are watched for changes. Any change to a file causes the application to recycle. Each entry consists of an optional direc directory where the main application entry point is located. Wild cards are allowed in the file name portion only. The default value is `*.js;web.config`

recycleSignalEnabled

The default value is false. If enabled, your node application can connect to a named pipe (environment variable IISNODE_CONTROL_PIPE) and send a "recycle"

idlePageOutTimePeriod

The default value is 0, which means this feature is disabled. When set to some value greater than 0, iisnode will page out all its child processes every 'idlePage what page out means. This setting is useful for applications that consume a high amount of memory and want to page out memory to disk occasionally to fre

> ⊗ **Warning**
>
> Use caution when enabling the following configuration settings on production applications. The recommendation is to not enable them on live production a

debugHeaderEnabled

The default value is false. If set to true, iisnode adds an HTTP response header `iisnode-debug` to every HTTP response it sends the `iisnode-debug` header valu obtained by looking at the URL fragment, however, a visualization is available by opening the URL in a browser.

loggingEnabled

This setting controls the logging of stdout and stderr by iisnode. Iisnode captures stdout/stderr from node processes it launches and writes to the directory sp application writes logs to the file system and depending on the amount of logging done by the application, there could be performance implications.

devErrorsEnabled

The default value is false. When set to true, iisnode displays the HTTP status code and Win32 error code on your browser. The win32 code is helpful in debugg

debuggingEnabled (do not enable on live production site)

This setting controls debugging feature. Iisnode is integrated with node-inspector. By enabling this setting, you enable debugging of your node application. U 'debuggerVirtualDir' directory on the first debug request to your node application. You can load the node-inspector by sending a request to http://yoursite/se 'debuggerPathSegment' setting. By default, debuggerPathSegment='debug'. You can set `debuggerPathSegment` to a GUID, for example, so that it is more diffic

Read Debug node.js applications on Windows for more details on debugging.

Scenarios and recommendations/troubleshooting

My node application is making excessive outbound calls

Many applications would want to make outbound connections as part of their regular operation. For example, when a request comes in, your node app would process the request. You would want to use a keep alive agent when making http or https calls. You could use the agentkeepalive module as your keep alive a

The agentkeepalive module ensures that sockets are reused on your Azure webapp VM. Creating a new socket on each outbound request adds overhead to y requests ensures that your application doesn't exceed the maxSockets that are allocated per VM. The recommendation on Azure Web Apps is to set the agent 40 maxSockets/instance) 160 sockets per VM.

Example agentKeepALive configuration:

NodeJS

```
var keepaliveAgent = new Agent({
    maxSockets: 40,
    maxFreeSockets: 10,
    timeout: 60000,
    keepAliveTimeout: 300000
});
```

Important

This example assumes you have 4 node.exe running on your VM. If you have a different number of node.exe running on the VM, you must modify the maxS

My node application is consuming too much CPU

You may receive a recommendation from Azure Web Apps on your portal about high cpu consumption. You can also set up monitors to watch for certain met check the MAX values for CPU so you don't miss the peak values. If you believe your application is consuming too much CPU and you cannot explain why, you

Profiling your node application on Azure Web Apps with V8-Profiler

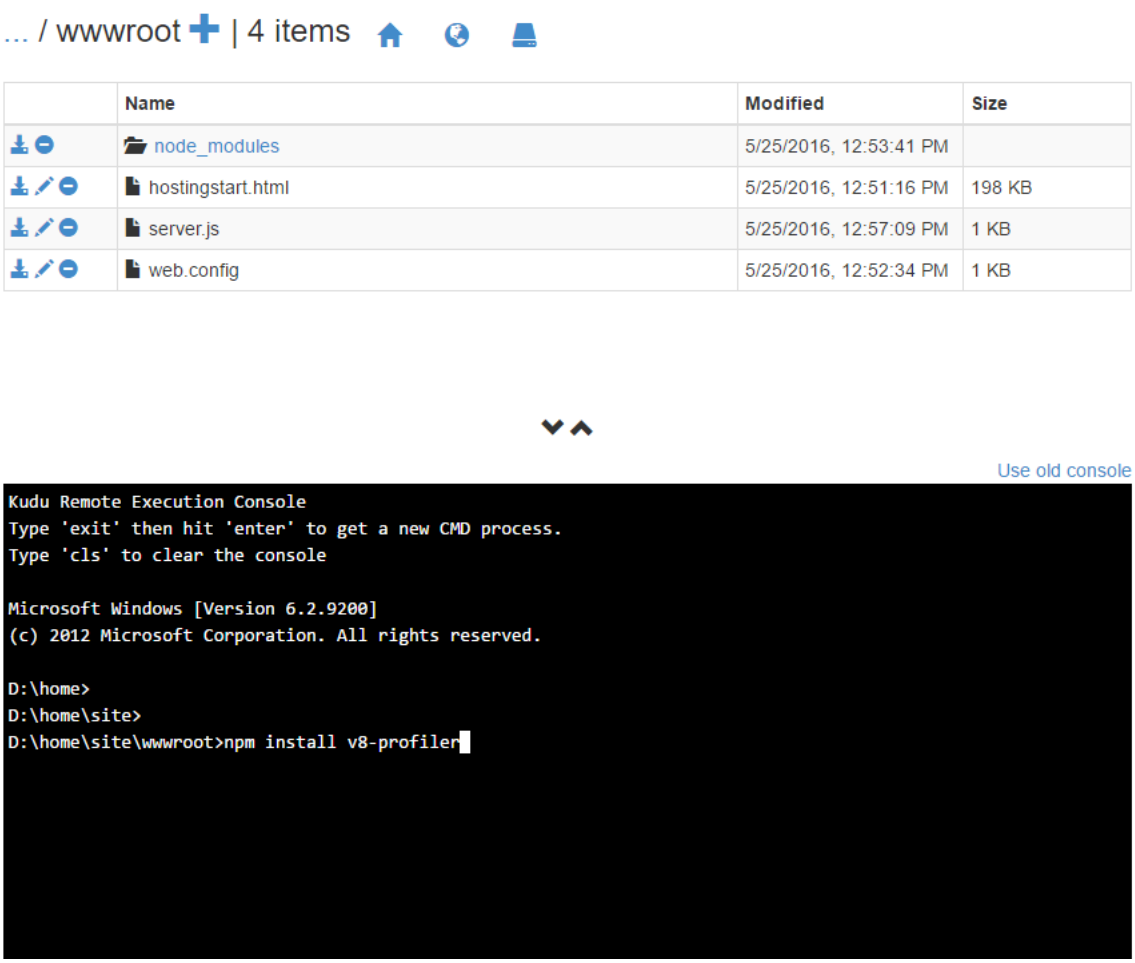
For example, let's say you have a hello world app that you want to profile as follows:

NodeJS

```
var http = require('http');
function WriteConsoleLog() {
    for(var i=0;i<99999;++i) {
        console.log('hello world');
    }
}

function HandleRequest() {
    WriteConsoleLog();
}

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    HandleRequest();
    res.end('Hello world!');
}).listen(process.env.PORT);
```

Go to the Debug Console site https://yoursite.scm.azurewebsites.net/DebugConsole

Go into your site/wwwroot directory. You see a command prompt as shown in the following example:



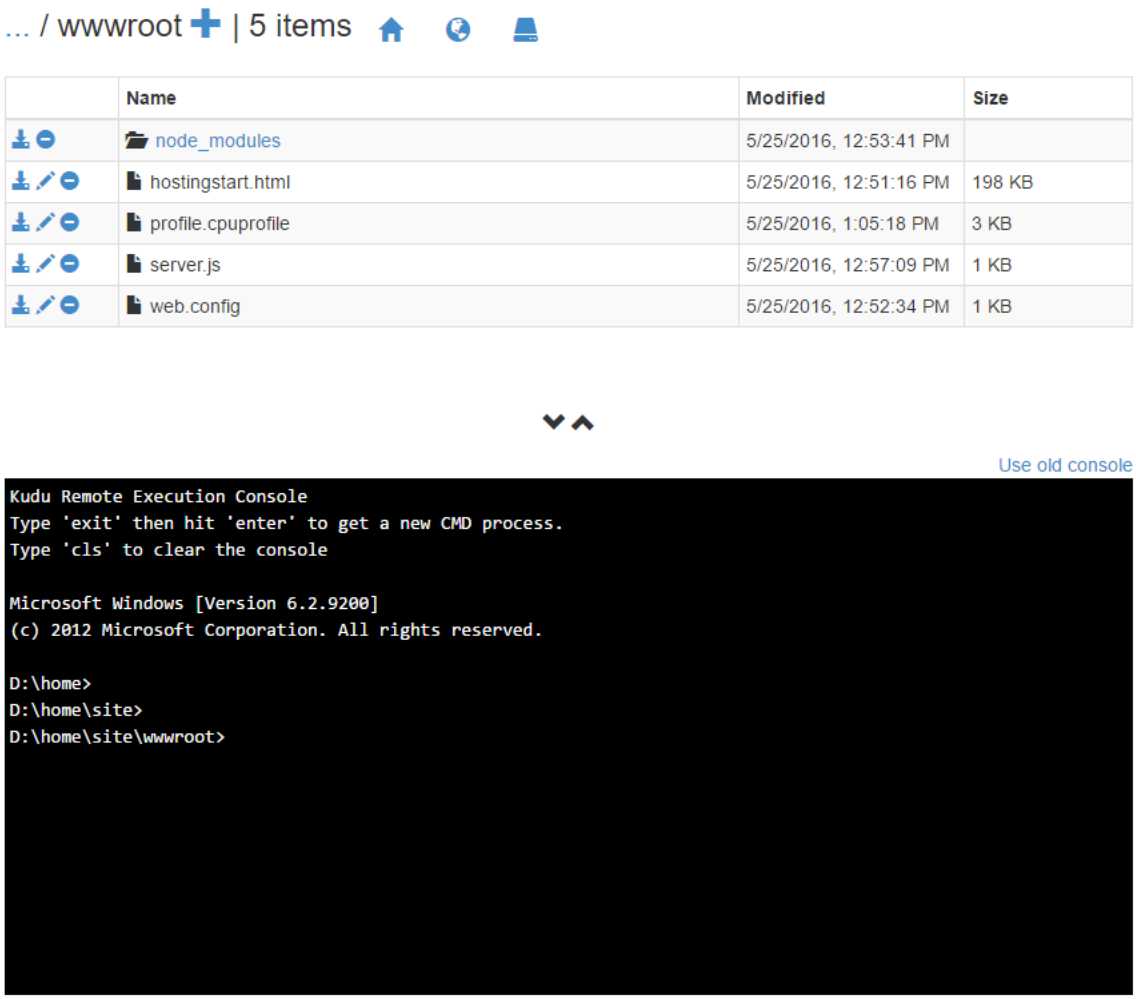
Run the command `npm install v8-profiler`.

This command installs the v8-profiler under node_modules directory and all of its dependencies. Now, edit your server.js to profile your application.

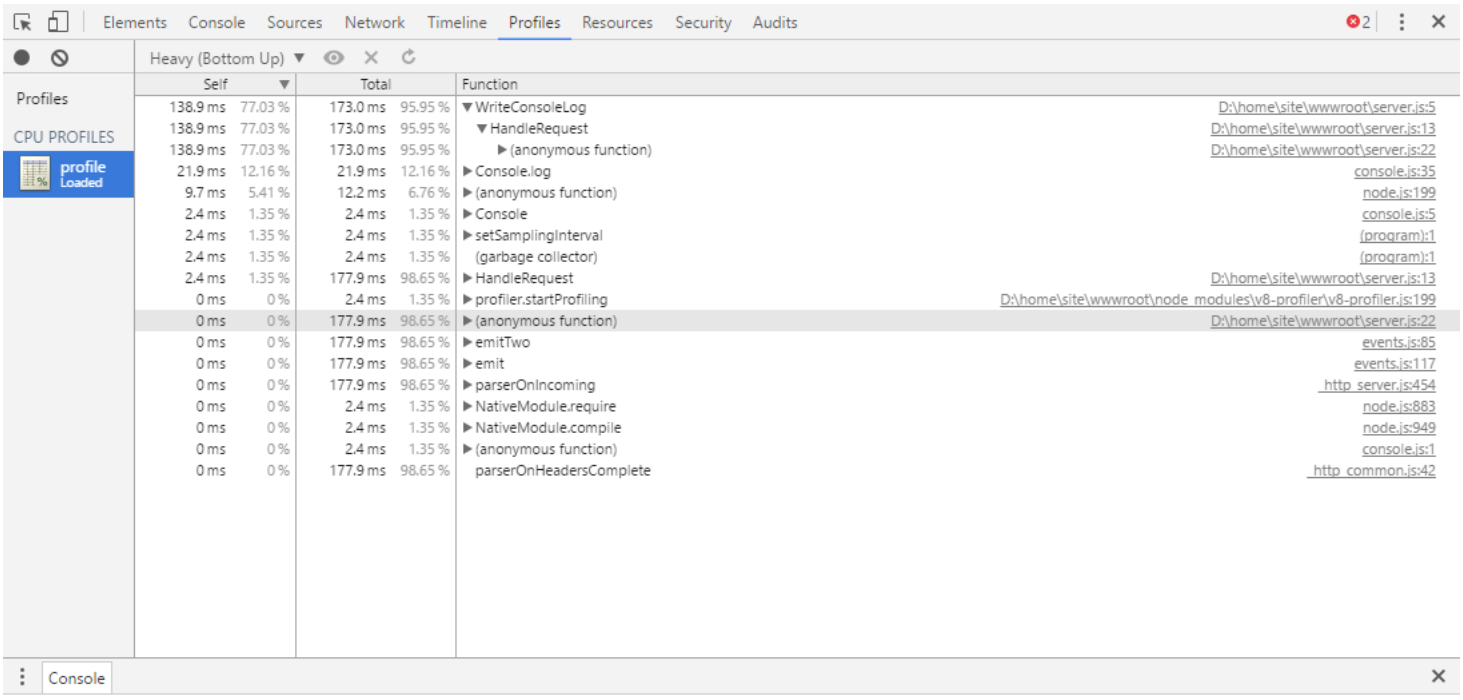
NodeJS

```
var http = require('http');
var profiler = require('v8-profiler');
var fs = require('fs');

function WriteConsoleLog() {
    for(var i=0;i<99999;++i) {
        console.log('hello world');
    }
}

function HandleRequest() {
    profiler.startProfiling('HandleRequest');
    WriteConsoleLog();
    fs.writeFileSync('profile.cpuprofile', JSON.stringify(profiler.stopProfiling('HandleRequest')));
}

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    HandleRequest();
    res.end('Hello world!');
}).listen(process.env.PORT);
```

The preceding code profiles the WriteConsoleLog function and then writes the profile output to the 'profile.cpuprofile' file under your site wwwroot. Send a re under your site wwwroot.



Download this file and open it with Chrome F12 Tools. Press F12 on Chrome, then choose the **Profiles** tab. Choose the **Load** button. Select your profile.cpupro



You can see that 95% of the time was consumed by the WriteConsoleLog function. The output also shows you the exact line numbers and source files that cau

My node application is consuming too much memory

If your application is consuming too much memory, you see a notice from Azure Web Apps on your portal about high memory consumption. You can set up n usage on the Azure Portal Dashboard, be sure to check the MAX values for memory so you don't miss the peak values.

Leak detection and Heap Diff for node.js

You could use node-memwatch to help you identify memory leaks. You can install `memwatch` just like v8-profiler and edit your code to capture and diff heaps

My node.exe's are getting killed randomly

There are a few reasons why node.exe is shut down randomly:

1. Your application is throwing uncaught exceptions – Check d:\home\LogFiles\Application\logging-errors.txt file for the details on the exception thrown. T
2. Your application is consuming too much memory, which is affecting other processes from getting started. If the total VM memory is close to 100%, your kills some processes to let other processes get a chance to do some work. To fix this issue, profile your application for memory leaks. If your application r increases the RAM available to the VM).

My node application does not start

If your application is returning 500 Errors when it starts, there could be a few reasons:

1. Node.exe is not present at the correct location. Check nodeProcessCommandLine setting.
2. Main script file is not present at the correct location. Check web.config and make sure the name of the main script file in the handlers section matches th
3. Web.config configuration is not correct – check the settings names/values.
4. Cold Start – Your application is taking too long to start. If your application takes longer than (maxNamedPipeConnectionRetry * namedPipeConnectionRe values of these settings to match your application start time to prevent iisnode from timing out and returning the 500 error.

My node application crashed

Your application is throwing uncaught exceptions – Check `d:\\home\\LogFiles\\Application\\logging-errors.txt` file for the details on the exception thrown. Th

My node application takes too much time to start (Cold Start)

The common cause for long application start times is a high number of files in the node_modules. The application tries to load most of these files when startin Azure Web Apps, loading many files can take time. Some solutions to make this process faster are:

1. Be sure you have a flat dependency structure and no duplicate dependencies by using npm3 to install your modules.
2. Try to lazy load your node_modules and not load all of the modules at application start. To Lazy load modules, the call to require('module') should be ma the first execution of module code.
3. Azure Web Apps offers a feature called local cache. This feature copies your content from the network share to the local disk on the VM. Since the files a

IISNODE http status and substatus

The `cnodeconstants` source file lists all of the possible status/substatus combinations iisnode can return due to an error.

Enable FREB for your application to see the win32 error code (be sure you enable FREB only on non-production sites for performance reasons).

Http Status	Http Substatus	Possible Reason?
500	1000	There was some issue dispatching the request to IISNODE – Check if node.exe was started. Node.exe could have crashed when starting. Check your w
500	1001	- Win32Error 0x2 - App is not responding to the URL. Check the URL rewrite rules or check if your express app has the correct routes defined. - Win32 because the pipe is busy. Check high cpu usage. - Other errors – check if node.exe crashed.
500	1002	Node.exe crashed – check d:\home\LogFiles\logging-errors.txt for stack trace.
500	1003	Pipe configuration Issue – The named pipe configuration is incorrect.
500	1004-1018	There was some error while sending the request or processing the response to/from node.exe. Check if node.exe crashed. check d:\home\LogFiles\log

Http Status	Http Substatus	Possible Reason?
503	1000	Not enough memory to allocate more named pipe connections. Check why your app is consuming so much memory. Check maxConcurrentRequestsP increase this value to prevent this error.
503	1001	Request could not be dispatched to node.exe because the application is recycling. After the application has recycled, requests should be served norm
503	1002	Check win32 error code for actual reason – Request could not be dispatched to a node.exe.
503	1003	Named pipe is too Busy – Verify if node.exe is consuming excessive CPU

NODE.exe has a setting called `NODE_PENDING_PIPE_INSTANCES`. By default, when not deployed on Azure Web Apps, this value is 4. Meaning that node.exe can on Apps, this value is set to 5000. This value should be good enough for most node applications running on Azure Web Apps. You should not see 503.1003 on Az `NODE_PENDING_PIPE_INSTANCES`

More resources

Follow these links to learn more about node.js applications on Azure App Service.

- Get started with Node.js Web Apps in Azure App Service
- How to debug a Node.js web app in Azure App Service
- Using Node.js Modules with Azure applications
- Azure App Service Web Apps: Node.js
- Node.js Developer Center
- Exploring the Super Secret Kudu Debug Console