IIS Web Servers (Web Apps and Web Services)

IIS will be setup to include "Application Development Features", which refers to the .NET Framework in general, and ASP.NET in particular. The IIS best practices documentation at Microsoft should be used to configure and maintain the IIS web servers.

Web servers require Windows IIS 10.x to be installed, and ASP.NET to be hosted under IIS. The installation of the full .NET Framework 4.8 is therefore a prerequisite. Refer to Microsoft's documentation for .NET Framework and IIS installation instructions.

To add IIS to Windows 10, under Setup/Apps/Optional Features/More Windows Features select both .NET Framework 4.8 Advanced Features and Internet Information Services. To add IIS to Windows Server, under Server Manager/Manage/Add Roles and Features/Role-based or feature-based installation, select both Web Server/Web Server and Web Server/Management Tools.

*Note: IIS running on Windows 10 is limited to 20 concurrent connections, which is fine for development computers. However, Windows Server Standard Edition or better should be used for the Test, QA and Prod environments (even for relatively small systems).

DGP mainly uses the default IIS installation options (from Windows 10 Optional Features):

- Web Management Tools
  - IIS Management Console
- World Wide Web Services
  - Application Development Features
    - .NET Extensibility 4.8.x
    - ASP.NET 4.8.x
    - ISAPI Extensions
    - ISAPI Filters
  - Common HTTP Features
    - Default Document
    - Directory Browsing
    - HTTP Errors
    - Static Content

- Health and Diagnostics
  - HTTP Logging
- Performance Features
  - Dynamic Content Compression
  - Static Content Compression
- Security
  - Request Filtering

Additional options can be selected as necessary, but the list above includes most of the basics for the IIS web server.

The default web site created by IIS is fine to host DGP web apps. However, in the Test, QA and Prod environments, each application should be configured to use its own separate app domain. This can help to insure that problems or restarts for one web app do not spread to other web apps and web services, etc. Logging is the other main issue. In general, logging should be turned off most of the time, and only enabled as needed. Forgetting to disable logging can quickly fill up storage with enormous log files from any web apps that see heavy traffic.

[https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/iis-best-practices/ba-p/1241577]

IIS Best Practices

By

Cenk Iscan

Published Mar 20 2020 06:20 AM 83.1K Views

It has been almost eight years since I first wrote a blog on IIS best practices. During this time, several new versions of IIS have arrived, some reached end of lifecycle; we were introduced a new development platform called .NET Core; a new HTTP version… And after eight more years of experience on a variety of customers and environments, finally I thought it was time for an update. Hope you enjoy it :)

See here for the old version.

For a very long time, I have been asked for a document on IIS best practices. There are some blogs/articles on the Internet, but I could not find a complete one. Actually, the main problem here is that there cannot be "best practices" for a web server. A web server is just a hosting platform for applications, and, each and every application has its own needs. Therefore, in many cases, you will not have one universal best practice.

Having these said, I tried to gather a list of things one should check while configuring an IIS server (and an application on IIS). I should say that these are my own thoughts based on my own experience. It may be possible that you will find some resources mentioning just the opposite of what I say. For such cases, consider your applications specific needs and decide accordingly.

Most of the below mentioned settings or recommendations apply to the web applications hosted on cloud solutions (like Azure Web Apps) and some will already be in-place. Especially some of the performance best practices listed below would help you keep your cloud costs as low as possible.

One more thing before we get to my recommendations: Keep in mind that web servers are just hosting platforms for your applications. They can only be as secure and as performant as your code. Therefore, do not expect the below list to address and resolve all your problems.

Application pool configuration

- Create one application pool for each application. You might consider grouping applications to pools if there are too many applications and/or sites (hundreds, thousands) hosted, for administration and management purposes only.
- Use Integrated mode. If your application does not work in Integrated mode, use Classic mode until you resolve the issue.

Application Pools <applicationPools>

https://docs.microsoft.com/en-us/iis/configuration/system.applicationhost/applicationpools/#overview

- Use 64bit application pools if you do not have a limitation by any of the modules your application uses. Even if you have such a limitation, try to resolve it.

enable32BitAppOnWin64

https://docs.microsoft.com/en-us/iis/configuration/system.applicationhost/applicationpools/applicati...

- Use ApplicationPoolIdentity as identity

Application Pool Identities

http://learn.iis.net/page.aspx/624/application-pool-identities/

- If your server is domain-joined and if your application needs to access resources over network, consider using "group managed service accounts" (gMSA):

Group Managed Service Accounts Overview

https://docs.microsoft.com/en-us/windows-server/security/group-managed-service-accounts/group-manage...

What's New for Managed Service Accounts

https://docs.microsoft.com/en-us/windows-server/security/group-managed-service-accounts/what-s-new-f...

- Do not leave recycling configuration as default:
  - Idle Time-out (minutes):
    - Either set this to 0 (zero)
    - Or set "Idle time-out action" to "Suspend"
  - Regular time interval (minutes): 0 (set to zero)
  - Specific Times: Set to the least used time of the day:

Recycling Settings for an Application Pool <recycling>
https://docs.microsoft.com/en-us/iis/configuration/system.applicationHost/applicationPools/add/recyc...

If you are confident that your application does not have any resource leak issues and if your application pool is recycled during deployments every now and then, you might consider totally disabling recycling.

- Set "logEventOnRecycle" for all reasons: Time, Requests, Schedule, Memory, IsapiUnhealthy, OnDemand, ConfigChange, and PrivateMemory:

logEventOnRecycle

https://docs.microsoft.com/en-us/iis/configuration/system.applicationhost/applicationpools/add/recyc...

- Do not set any private memory limit unless troubleshooting specific scenarios. If you do set it, turn it off as soon as possible.
- Do not ever set virtual memory limit. Reserved memory usage is meant with "virtual bytes" and especially in 64bit environments, processes reserve huge amounts of memory (that is not an actual reservation, but it is a different story).
- Consider increasing "Maximum Worker Processes" (web gardening) number - with caution:
  - Make sure that your application is sessionless or uses "out-of-process" session state
  - Make sure that your application does not cache too much data (as it would be cached in each process which would then cause performance degradation)
- Keep "Rapid-fail protection" enabled, but makes changes according to application pool settings, i.e. "Maximum Worker Processes"
  - Make sure that your load balancer, if there is one, and "Service Unavailable Response type" setting of your application pools are set accordingly.

Failure Settings for an Application Pool <failure>

https://docs.microsoft.com/en-us/iis/configuration/system.applicationhost/applicationpools/add/failu...

I. Logging

- Do not ever turn logging off, and:
  - Keep your log files (IIS, HttpErr and any application log you might have) on a non-system disk/partition
  - Turn on as many fields as possible (at least defaults + sc-bytes and cs-bytes)

HTTP Logging <httpLogging>

https://docs.microsoft.com/en-us/iis/configuration/system.webserver/httplogging

- Monitor disk usage as the logs might grow quite fast
  - Schedule scripts to archive older logs

- Use "custom fields" if you need to log extra fields. If you have an NLB device which NATs (causing the client IP to be hidden from IIS servers), add the real client IP as a custom field:
  - Do not use "advanced logging" module as it is a very old module and there may be some issues with it, like memory leaks.
  - If you are hosting service applications (web services or WCF) consider adding method names to headers (like SOAPAction header) and log them in IIS logs using custom fields.

Adding Custom Fields to a Log File for a Site <add>

https://docs.microsoft.com/en-us/iis/configuration/system.applicationhost/sites/site/logfile/customf...

- IIS logs provide us with invaluable data on general health and performance of our applications. Periodically analyze them to have an idea on the performance baseline and trends.
  - Although the latest version was released in 2005, Log Parser is still a very powerful tool for log analysis

Log Parser 2.2

https://www.microsoft.com/en-us/download/details.aspx?id=24659

- You can use Azure Monitor to collect and analyze IIS logs

Collect IIS logs in Azure Monitor

https://docs.microsoft.com/en-us/azure/azure-monitor/platform/data-sources-iis-logs

- Also, keep an eye on HttpErr logs as they might contain some details on some specific problems:

Error logging in HTTP APIs

https://support.microsoft.com/en-us/help/820729/error-logging-in-http-apis

- Configure Windows Error Reporting to collect full user dumps when worker processes (w3wp.exe) crash:
  - Note that WER might sometimes not be able to collect memory dumps of .NET applications, but for most scenarios, it would be best to have this setting

Collecting User-Mode Dumps

https://docs.microsoft.com/en-us/windows/win32/wer/collecting-user-mode-dumps

II. Content

- Keep your content on a non-system disk, if possible, on a disk/partition other than the one you keep your logs.
- Configure static and dynamic compression - if CPU usage and content types permit

HTTP Compression <httpCompression>
https://docs.microsoft.com/en-us/iis/configuration/system.webServer/httpCompression/

- Configure antivirus applications to exclude at least:
  - Content folders
  - Temporary ASP.NET Files folder (C:\Windows\Framework[64]\vX.X.XXXXX\)
  - Any temp folder if used

Configure Windows Defender Antivirus exclusions on Windows Server

https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-antivirus/confi...

- Use "Output caching" whenever possible (be careful doing this as it might cause users to access each other's sessions)

Caching <caching>

https://docs.microsoft.com/en-us/iis/configuration/system.webserver/caching/

- Remove any unused "default document" setting and keep only the one you will use

Default Document <defaultDocument>

https://docs.microsoft.com/en-us/iis/configuration/system.webserver/defaultdocument/

III. Security

- Use end-to-end encryption

- If you have reverse proxy and/or load balancer in front of your web servers, prefer to use SSL-bridging instead of SSL-offloading
- Disable older SSL/TLS versions than TLS 1.2
- Disable weak cypher suits
- SSL/TLS and cypher suit settings are server-wide settings, and IIS supports whatever the OS supports. However, for .NET applications check the below article:

Transport Layer Security (TLS) best practices with the .NET Framework

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/tls

- Add security headers to your applications:

Content Security Policy (CSP)

https://docs.microsoft.com/en-us/microsoft-edge/extensions-chromium/store-policies/csp

HSTS Settings for a Web Site <hsts>

https://docs.microsoft.com/en-us/iis/configuration/system.applicationhost/sites/site/hsts

X-Frame-Options

https://tools.ietf.org/html/rfc7034

OWASP Secure Headers Project

https://www.owasp.org/index.php/OWASP_Secure_Headers_Project#tab=Headers

- Configure "Request Filtering":
  - “Allow unlisted file name extensions": Uncheck (allow only the extensions you will use; add "." to allow extensionless requests)
  - “Allow unlisted verbs": Uncheck (allow only the verbs you will use)
  - Lower "request limits" if possible

Request Filtering <requestFiltering>

https://docs.microsoft.com/en-us/iis/configuration/system.webserver/security/requestfiltering/

- Remove HTTP headers which identifies the server and application. These headers are believed to cause security vulnerability:

removeServerHeader

https://docs.microsoft.com/en-us/iis/configuration/system.webserver/security/requestfiltering/#new-i...

Remove Unwanted HTTP Response Headers

https://techcommunity.microsoft.com/t5/iis-support-blog/remove-unwanted-http-response-headers/ba-p/3...

- Set NTFS permissions on the content folders as needed:
  - Do not give unnecessary permissions to unnecessary users. Remove permissions of Users and other groups. You should consider authentication and impersonation configurations to do this.
  - The content folder should only need "read" and "read and execute" permissions. If your application needs to write something (like logs or temp files) write them to a separate folder (one for each application on the server) and give "write" permission only to that specific folder.
  - Make sure that the folders with write permissions cannot be accessed through HTTP protocol. i.e. make sure that access to that folder is denied by Request Filtering module.

- If using anonymous authentication, set the user to "Application pool identity" to be able to isolate your sites and applications

- Do not store sensitive information in configuration files. Encrypt such fields if you need to have them:

Protecting Connection Strings and Other Configuration Information (C#)

https://docs.microsoft.com/en-us/aspnet/web-forms/overview/data-access/advanced-data-access-scenario...

- Remove any unused modules to reduce attack surface. For example, if you do not specifically need WebDAV, do not install it.
- Consider adding the host names of your web sites to Hosts file to point 127.0.0.1, so that you can test your applications locally on the servers in a web farm environment. This would be the first and the easiest test to eliminate network issues.

IV. Application Performance

- If your .NET/.NET Core application gets burst loads or for some reason you need to recycle your application or application pool during work hours, consider increasing minWorkerThreads and minIoThreads settings:

processModel Element (ASP.NET Settings Schema)

https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/7w2sway1%28v%3dvs.100%29

ProcessModelSection.MinWorkerThreads Property

https://docs.microsoft.com/en-us/dotnet/api/system.web.configuration.processmodelsection.minworkerth...

runtimeconfig.json

https://docs.microsoft.com/en-us/dotnet/core/run-time-config/#runtimeconfigjson

ASP.NET Thread Usage on IIS 7.5, IIS 7.0, and IIS 6.0

https://docs.microsoft.com/en-us/archive/blogs/tmarq/asp-net-thread-usage-on-iis-7-5-iis-7-0-and-iis...

- For .NET Core applications, choose to run it "in-process":

In-process hosting model

https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/aspnet-core-module?view=aspnetcore-2.2#...

- If your server has many CPU cores (like more than 16, for instance) consider making changes on GC settings:
    - Consider setting GC heap count to a lower number than the number of CPU core. While deciding heap count, take the number of applications hosted on the server into consideration

<GCHeapCount> element

https://docs.microsoft.com/en-us/dotnet/framework/configure-apps/file-schema/runtime/gcheapcount-ele...

Middle Ground between Server and Workstation GC

https://devblogs.microsoft.com/dotnet/middle-ground-between-server-and-workstation-gc/

- Depending on the number of CPU cores on your server, consider setting CPU affinity of your application pools. For very high CPU consumption issues, leave at least one CPU core unused, so that you can access the server to collect logs to troubleshoot and be able to take actions towards resolution.

smpAffinitized and smpProcessorAffinityMask

https://docs.microsoft.com/en-us/iis/configuration/system.applicationhost/applicationpools/add/cpu

- If your hardware or virtualization environment has NUMA support, consider setting "Maximum number of processes" to zero:

IIS 8.0 Multicore Scaling on NUMA Hardware

https://docs.microsoft.com/en-us/iis/get-started/whats-new-in-iis-8/iis-80-multicore-scaling-on-numa...

- For ASP.NET applications, confirm that they are built in release mode, and that they do not have debug="true" in any web.config file in production

Debug Mode in ASP.NET Applications (the article is very old, but it is still valid for all .NET versions)

https://support.microsoft.com/en-us/help/2580348/debug-mode-in-asp-net-applications

- Make sure tracing is installed (you never know when you would need it) but keep it disabled at both application and IIS level (Failed Request Tracing)

Troubleshooting Failed Requests Using Tracing in IIS 8.5

https://docs.microsoft.com/en-us/iis/troubleshoot/using-failed-request-tracing/troubleshooting-faile...

Trace Failed Requests Logging for a Site <traceFailedRequestsLogging>

https://docs.microsoft.com/en-us/iis/configuration/system.applicationhost/sites/site/tracefailedrequ...

- Set the applications' timeout values as low as technically possible

Set Timeouts Aggressively
https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff647787(v=pandp.10)?redirectedfrom=MSDN#...

- Make settings to make use of HTTP/2:

How do I use HTTP/2?

https://docs.microsoft.com/en-us/iis/get-started/whats-new-in-iis-10/http2-on-iis#how-do-i-use-http2

- Install/copy these tools on the server as you never know when you will need them

PerfView

https://github.com/Microsoft/perfview

Debug Diagnostics Tool

https://www.microsoft.com/en-us/download/details.aspx?id=58210

ProcDump

https://docs.microsoft.com/en-us/sysinternals/downloads/procdump

ProcMon

https://docs.microsoft.com/en-us/sysinternals/downloads/procmon

- Monitoring is the most important thing about performance. Know your applications performance baseline and monitor for any changes:

Azure Monitor overview

https://docs.microsoft.com/en-us/azure/azure-monitor/overview

What is Application Insights?

https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview

- Save the performance counters one should monitor in case of an issue to the desktop or somewhere. The following might be a starting point for ASP.NET applications:

\Processor Information(_Total)\% Processor Time
\.NET CLR Exceptions(w3wp)\# of Exceps Thrown / sec
\.NET CLR Memory(w3wp)\# Gen 0 Collections
\.NET CLR Memory(w3wp)\# Gen 1 Collections
\.NET CLR Memory(w3wp)\# Gen 2 Collections
\.NET CLR Memory(w3wp)\# Induced GC
\.NET CLR Memory(w3wp)\Process ID
\.NET CLR Memory(w3wp)\#Bytes In All Heaps
\ASP.NET\Application Restarts
\ASP.NET\Request Execution Time

\ASP.NET\Requests Current
\ASP.NET\Requests Queued

\ASP.NET Applications(*)\Requests Executing
\ASP.NET Applications(*)\Requests/Sec
\Memory\Available MBytes
\Process(w3wp)\% Processor Time
\Process(w3wp)\ID Process
\Process(w3wp)\Private Bytes

\WAS_W3WP(*)\Total Health Pings.