**Application Pools** <applicationPools>

**Overview**

The <applicationPools> element contains configuration settings for all application pools running on your Internet Information Services (IIS) 7 server. An application pool defines a group of one or more worker processes, configured with common settings that serve requests to one or more applications that are assigned to that application pool. Because application pools allow a set of Web applications to share one or more similarly configured worker processes, they provide a convenient way to isolate a set of Web applications from other Web applications on the server computer. Process boundaries separate each worker process; therefore, application problems in one application pool do not affect Web sites or applications in other application pools. Application pools significantly increase both the reliability and manageability of your Web infrastructure.

You can choose to use the default application pool provided by IIS on install, or you can create your own application pool. You can run as many application pools on your IIS 7 server as you need, though this can affect server performance. Application pools can contain one or more worker processes. Each worker process represents work being done for a Web site, Web application, or Web service. You can create a Web garden by enabling multiple worker processes to run in a single application pool.

In IIS 7, each application pool uses one of two .NET integration modes for running ASP.NET applications: **Integrated** or **Classic**. The .NET integration mode defined for the application pool determines how IIS processes an incoming request to the sites, applications and Web services that run in that application pool.

* **Integrated** mode allows IIS to process requests in the application pool by using the IIS 7 integrated pipeline. This allows ASP.NET modules to participate in IIS request processing regardless of the type of resource requested. Using integrated mode makes available features of the ASP.NET 2.0 request pipeline available to requests for static content, as well as ASP, PHP and other content types. By default, IIS 7 application pools run in this mode.
* **Classic** mode uses the IIS 6.0 processing pipeline for hosting ASP.NET applications. In this mode, requests are processed initially through IIS 7 modules, and ASP.NET requests are further processed by the aspnet\_isapi.dll. The ASP.NET processing pipeline is separate from the IIS 7 processing pipeline, and the ASP.NET request processing pipeline features are not available to other resource types. This also means that an ASP.NET request must pass through authentication and authorization modules in both process models. While this is not as efficient as Integrated mode, it does allow you to run applications developed using ASP.NET version 1.1 on an IIS 7 server without modifying the application to run in Integrated mode.

#### New in IIS 7.5

Starting in IIS 7.5, you can configure an application to start automatically by using the **managedRuntimeLoader**, **CLRConfigFile**, and **startMode** attributes of the <add> element. These attributes configure, respectively, the name of the managed DLL that provides runtime loading for your application, the common language runtime configuration file for the application, and the startup type for the application.

Also new in IIS 7.5 is a new ApplicationPoolIdentity type for the **identityType** attribute of the [<processModel>](http://www.iis.net/configreference/system.applicationhost/applicationpools/add/processmodel) element. This new identity type is now the default process identity for applications, and makes it possible to set the security for your content areas to allow access for a specific application pool. To do so, you would set your security using the name of an application pool by using syntax like "IIS AppPool\DefaultAppPool." This identity is created dynamically, thereby dramatically reducing the surface attack area of your server.

## Compatibility

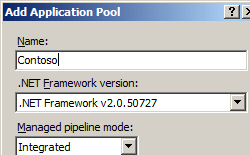
|  |  |
| --- | --- |
| Version | Notes |
| IIS 7.5 | The <add> element of the <applicationPools> element was updated in IIS 7.5 to include attributes that allow you to preload applications by using the **managedRuntimeLoader**, **CLRConfigFile**, and **startMode** attributes, and to run applications using the new **ApplicationPoolIdentity**. |
| IIS 7.0 | The <applicationPools> element was introduced in IIS 7.0. |
| IIS 6.0 | The <applicationPools> element replaces the IIS 6.0 **IIsApplicationPools** metabase object. |

## Setup

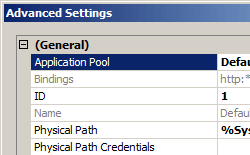
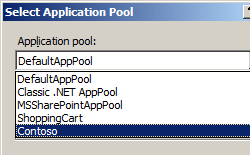
The <applicationPools> collection is included in the default installation of IIS 7.

## How To

### How to create a new application pool

1. Open **Internet Information Services (IIS) Manager**:
   * If you are using Windows Server 2008 or Windows Server 2008 R2:
     + On the taskbar, click **Start**, point to **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
   * If you are using Windows Vista or Windows 7:
     + On the taskbar, click **Start**, and then click **Control Panel**.
     + Double-click **Administrative Tools**, and then double-click **Internet Information Services (IIS) Manager**.
2. In the **Connections** pane, expand the server name, and then click **Application Pools**.
3. In the **Actions** pane, click **Add Application Pool...**.
4. In the **Add Application Pool** dialog box, enter the name of the application pool in the **Name:** box, in the **.NET Framework version:** drop-down list select the .NET Framework version your site or application uses, in the **Managed pipeline mode:** drop-down list select **Integrated** or **Classic**, and then click **OK**.   
   [](http://i3.iis.net/images/configreference/applicationPools_add_1.png?cdn_id=2014-01-17-002)

### How to configure the application pool for an existing site or application

1. In the **Connections** pane, expand **Sites**, and then navigate to the Web site or application you want to add to the application pool.
2. In the **Actions** pane, click **Advanced Settings...**
3. In the **General** section of the **Advanced Settings** dialog box, click the **Application Pool** entry, and then click the ellipses button.  
   [](http://i1.iis.net/images/configreference/applicationPools_add_2.png?cdn_id=2014-01-17-002)
4. In the **Select Application Pool** dialog box, select the application pool from the **Application pool:** drop-down box, click **OK**, and then click **OK** again.  
   [](http://i1.iis.net/images/configreference/applicationPools_add_3.png?cdn_id=2014-01-17-002)

## Configuration

The <applicationPools> element contains a collection of <add> elements. Each element in the collection configures an application pool running on IIS 7. At a minimum, each <add> element contains a **name** attribute that identifies the application pool to GUI and command-line management tools. The <add> element contains child elements that configure the process model, CPU, and recycling settings for the application pool.

The <applicationPools> element also contains the <applicationPoolDefaults> element, which defines default settings for all application pools on the IIS 7 server. To change default application pool settings across the server, you can edit the <applicationPoolDefaults> element. When you create a new application pool, the configuration settings for that application pool override the defaults set in the <applicationPoolDefaults> element.

### Attributes

None.

### Child Elements

|  |  |
| --- | --- |
| Element | Description |
| [add](http://www.iis.net/configreference/system.applicationhost/applicationpools/add) | Adds an application pool to the applicationPools section. |
| [applicationPoolDefaults](http://www.iis.net/configreference/system.applicationhost/applicationpools/applicationpooldefaults) | Required string attribute.   Configures the default settings for all application pools in an applicationPools section. |

### Configuration Sample

The following configuration sample uses the application pool <add> and <applicationPools> elements to define the standard application pools and defaults for IIS.

<applicationPools>

<add name="DefaultAppPool" />

<add name="Classic .NET AppPool" managedPipelineMode="Classic" />

<applicationPoolDefaults>

<processModel identityType="NetworkService" />

</applicationPoolDefaults>

</applicationPools>

## Sample Code

The following examples add an application pool named Contoso, and set the managed pipeline mode to Integrated.

### AppCmd.exe

appcmd.exe set config -section:system.applicationHost/applicationPools /+"[name='Contoso',autoStart='True',managedPipelineMode='Integrated']" /commit:apphost

**Note**: You must be sure to set the **commit** parameter to apphost when you use AppCmd.exe to configure these settings. This commits the configuration settings to the appropriate location section in the ApplicationHost.config file.

### C#

using System;

using System.Text;

using Microsoft.Web.Administration;

internal static class Sample {

private static void Main() {

using(ServerManager serverManager = new ServerManager()) {

Configuration config = serverManager.GetApplicationHostConfiguration();

ConfigurationSection applicationPoolsSection = config.GetSection("system.applicationHost/applicationPools");

ConfigurationElementCollection applicationPoolsCollection = applicationPoolsSection.GetCollection();

ConfigurationElement addElement = applicationPoolsCollection.CreateElement("add");

addElement["name"] = @"Contoso";

addElement["autoStart"] = true;

addElement["managedPipelineMode"] = @"Integrated";

applicationPoolsCollection.Add(addElement);

serverManager.CommitChanges();

}

}

}

### VB.NET

Imports System

Imports System.Text

Imports Microsoft.Web.Administration

Module Sample

Sub Main()

Dim serverManager As ServerManager = New ServerManager

Dim config As Configuration = serverManager.GetApplicationHostConfiguration

Dim applicationPoolsSection As ConfigurationSection = config.GetSection("system.applicationHost/applicationPools")

Dim applicationPoolsCollection As ConfigurationElementCollection = applicationPoolsSection.GetCollection

Dim addElement As ConfigurationElement = applicationPoolsCollection.CreateElement("add")

addElement("name") = "Contoso"

addElement("autoStart") = True

addElement("managedPipelineMode") = "Integrated"

applicationPoolsCollection.Add(addElement)

serverManager.CommitChanges()

End Sub

End Module

### JavaScript

var adminManager = new ActiveXObject('Microsoft.ApplicationHost.WritableAdminManager');

adminManager.CommitPath = "MACHINE/WEBROOT/APPHOST";

var applicationPoolsSection = adminManager.GetAdminSection("system.applicationHost/applicationPools","MACHINE/WEBROOT/APPHOST");

var applicationPoolsCollection = applicationPoolsSection.Collection;

var addElement = applicationPoolsCollection.CreateNewElement("add");

addElement.Properties.Item("name").Value = "Contoso";

addElement.Properties.Item("autoStart").Value = true;

addElement.Properties.Item("managedPipelineMode").Value = "Integrated";

applicationPoolsCollection.AddElement(addElement);

adminManager.CommitChanges();

### VBScript

Set adminManager = CreateObject("Microsoft.ApplicationHost.WritableAdminManager")

adminManager.CommitPath = "MACHINE/WEBROOT/APPHOST"

Set applicationPoolsSection = adminManager.GetAdminSection("system.applicationHost/applicationPools","MACHINE/WEBROOT/APPHOST")

Set applicationPoolsCollection = applicationPoolsSection.Collection

Set addElement = applicationPoolsCollection.CreateNewElement("add")

addElement.Properties.Item("name").Value = "Contoso"

addElement.Properties.Item("autoStart").Value = True

addElement.Properties.Item("managedPipelineMode").Value = "Integrated"

applicationPoolsCollection.AddElement(addElement)

adminManager.CommitChanges()

**How many web applications per application pool**

<http://blogs.msdn.com/b/carmelop/archive/2013/03/22/how-many-web-applications-per-application-pool.aspx>

Many times during my work activities, I figure out wrong configurations on web servers. Today, I want to discuss here this topic: web applications cannot be managed and configured like traditional windows applications.

When we think to desktop applications like notepad, we know that each time we run it, O.S. defines a new process (notepad.exe in our example).

Is it true for web application? No, a web application needs a host process to be executed, it means a process that carries on the web application and allows its execution. Starting from IIS6 this process is called w3wp.exe.   
Please pay attention, a web application runs on top of an application pool that you define in IIS. An application pool, can be associated to one or more worker process.

A typical mistake is the following: web server administrator defines an application pool and execute many web applications on top of it. I worked with customer who has 20 -30-web applications on top of a single application pool!

This is bad very bad especially for 32 bit process. Let me try to explain why. Every web application needs:

1. Assemblies loaded in memory

2. Memory to calculate the application goal.

In case that you define a single application pool for all web applications, what happens is that all web apps have to load their assemblies and share the same memory. Do you think that this memory is infinite? No, it is not.

In a 32-bit process, by default every application can allocate 2GB of memory and a 32-bit process on a 64 bit machine 4GB. Those values are the maximum ones available by default, but do not except to use all that memory.

Typically, an Out Of Memory exception occurs before that value. It happens due to internal memory fragmentation!

In case that your application are managed one: ASP.NET to be clear, what happened is that Garbage Collector takes care to clean up the memory. If all web applications share the same memory, this one is under pressure.   
It means that GC runs a lot of time per second in order to provide clean memory to your app. What is the side effect? In server mode, garbage collector requires to stop all threads activities to clean up the memory (this was improved on .NET Fx 4.5, where the collation does not require to stop all threads). What is the side effect?

* Slow performance of your web application
* High impact on CPU time due to GC’s activity.

Furthermore, if one of your web application crashes, all application running on the same pool will be impacted. Nothing more? There are other things to consider:

Threadpool, the number of thread that single process can define is not infinite. So all web application have to share that number/threads. Same thing for connection pool and so on.

Please do not consider this post in the following way: define necessarily one application pool per web application. This is not the goal, IIS gives you the flexibility to host multiple applications on the same application pool. However, when you decide how many applications have to run on top of the same application pool, take care of above considerations.

### Questions on application domains, application pools and unhandled exceptions

<http://blogs.msdn.com/b/tess/archive/2008/08/19/questions-on-application-domains-application-pools-and-unhandled-exceptions.aspx>

From what I read, my understanding is that a website has an app pool associated with it. This app pool leads to the creation of a w3wp.exe process. Inside this app pool/w3wp.exe process, an application domain is created.

Tess:  This is correct.  In IIS you can create different application pools that have different healthmonitoring settings, run under different user contexts etc. and when you create a web site you choose which application pool it will run under.  Each application pool will then spawn it's own w3wp.exe process (or multiple if you have web gardening turned on) when the first request comes in.

The process (w3wp.exe) contains multiple application domains, typically a shared domain, a default domain, a system domain and one application domain per web application (virtual directory marked as application).

An application domain recycling is different than an application pool/process (w3wp.exe) recycling, right?

Tess: Yes.  An appdomain can recycle without recycling the process.  Simplified an appdomain is a process within the process with its own statics (cache/session etc.), but all appdomains in the process share the same GC, threadpool, finalizer thread etc.

An appdomain recycle is triggered by a few things like web.config changes, directory name changes etc.  You can find most of the appdomain recycle reasons in [this post.](http://blogs.msdn.com/tess/archive/2006/08/02/asp-net-case-study-lost-session-variables-and-appdomain-recycles.aspx)  When an appdomain recycles the process stays up, however when the process goes down the appdomains in the process will of course also go down.

Can unhandled exceptions cause the application domain to recycle ?

Tess:  It depends on what you mean by unhandled exceptions.  In ASP.NET there is the concept of unhandled exceptions that are caught by the global error handler or page error handler. i.e. the ones that give you the yellow exception output when you view a page.  They will be listing as unhandled exceptions in eventvwr, but in reality they are handled by the page error handler, and they will neither crash the appdomain or the application/w3wp.exe process.

If on the other hand you have an exception that occurrs on a non-request thread, such as a timer thread or the finalizer thread and you dont have a try/catch block around it, it is really an unhandled exception, and such unhandled exceptions will cause the process to go down and take all the appdomains with it.

What exactly happens inside the application domain when an unhandled exception occurs ?

Tess:  The process shuts down, and the appdomains are unloaded so anything inside it is gone including session vars etc.

Is there a setting in any of the config files that will prevent the application domain from recycling ?

Tess: There is a legacyUnhandledExceptionPolicy see [this post](http://blogs.msdn.com/tess/archive/2006/04/27/584927.aspx) for more info, that will cause a 2.0 process to behave as 1.1, i.e. not shut down the process and instead just quit processing the current thread of execution. I would seriously advice against using it though other than as a temporary measure while you troubleshoot the unhandled exception as it may cause your process to behave erratically, since you don't really know what has processed and what hasn't.  For example if an exception occurrs during finalization you will not know if you have released all the native handles you were supposed to or not.