

## Instrument .NET Applications AppDynamics Pro Documentation Version 4.0.x

Instrument .NET Applications .....	3
.NET Supported Environments .....	5
Install the .NET Agent .....	9
Configure the .NET Agent .....	11
Enable SSL for .NET .....	15
Instrument Windows Services and Standalone Applications .....	20
Name .NET Tiers .....	24
Automatically Name .NET Nodes .....	26
Unattended Installation for .NET .....	27
Upgrade the .NET Agent .....	32
Resolve .NET Agent Installation and Configuration Issues .....	36
Uninstall the .NET Agent .....	47
Install AppDynamics for Windows Azure with NuGet .....	48
Register for AppDynamics for Windows Azure .....	55
Administer the .NET Agent .....	60
.NET Agent Directory Structure .....	61
.NET Agent Configuration Properties .....	63
Configure Multiple Business Application Support for .NET .....	78
Disable Instrumentation for an IIS Application Pool .....	85
Configure Application Domain Monitoring .....	85
Instrument the DefaultDomain for Standalone Applications .....	90
Configure the .NET Machine Agent .....	92
Enable Monitoring for Windows Performance Counters .....	94
Enable Correlation for .NET Remoting .....	95
Enable Thread Correlation for .NET .....	98
Enable Instrumentation for WCF Data Services .....	99
Configure Machine Snapshots for .NET .....	100

# Instrument .NET Applications

**On this page:**

- [Before You Begin](#)
- [Instrument Your CLR with the .NET Agent](#)

**Related pages:**

- [AppDynamics Essentials](#)
- [Install the .NET Agent](#)
- [AppDynamics for Windows Azure with NuGet](#)

**Watch the video:**

[Quick Install: .NET Agent](#)

For AppDynamics to gather performance data about your .NET applications, you must install the .NET Agent (agent) on the servers where the applications run. You only need to install the agent once per server even if you want to monitor multiple applications on one server. The agent instruments your application and sends performance data back to the AppDynamics Controller.

The instructions on this page help you install the agent for IIS applications using the Agent Download wizard in the Controller.

- If you downloaded the agent from the [AppDynamics download zone](#), see [Install the .NET Agent](#).
- For Windows services and standalone applications, see [Instrument Windows Services and Standalone Applications](#).
- If you're using Windows Azure Web Roles or Worker Roles, see [Install AppDynamics for Windows Azure with NuGet](#).

## Before You Begin

1. Confirm you have access to a controller, the web application where you monitor your application performance:
  - If you use a SaaS controller, AppDynamics sent you the controller host in your Welcome Email.
  - If you use the on-premise controller, you supplied the host and port at install time.
2. Verify you've enabled COM+ on your system. See "Verify COM+ services are enabled" on [Resolve .NET Agent Installation and Configuration Issues](#).

## Instrument Your CLR with the .NET Agent

There are four steps to instrument your CLR and begin monitoring:

1. [Download](#): Use the Agent Download Wizard to configure and download the agent.
2. [Install](#): Extract the installer files and run the Installer.bat batch file.
3. [Apply Load](#): Apply load to activate instrumentation.
4. [View Your Application](#): Log on to the Controller to monitor application performance.

## Download the .NET Agent

The Agent Download Wizard walks you through configuration steps and helps you download the agent.

## Install the agent on your app server

After you download the agent, install it to your app server. The final window of the Agent Download Wizard includes brief instructions for installing the agent.

1. Extract the dotNetAgent-Portal-<architecture>-<version>.zip file.
2. Launch an elevated command prompt with full administrator privileges.
3. Execute the Installer.bat file.

The batch file installs the .NET Agent agent and starts the AppDynamics Agent Coordinator service.

4. Restart IIS.

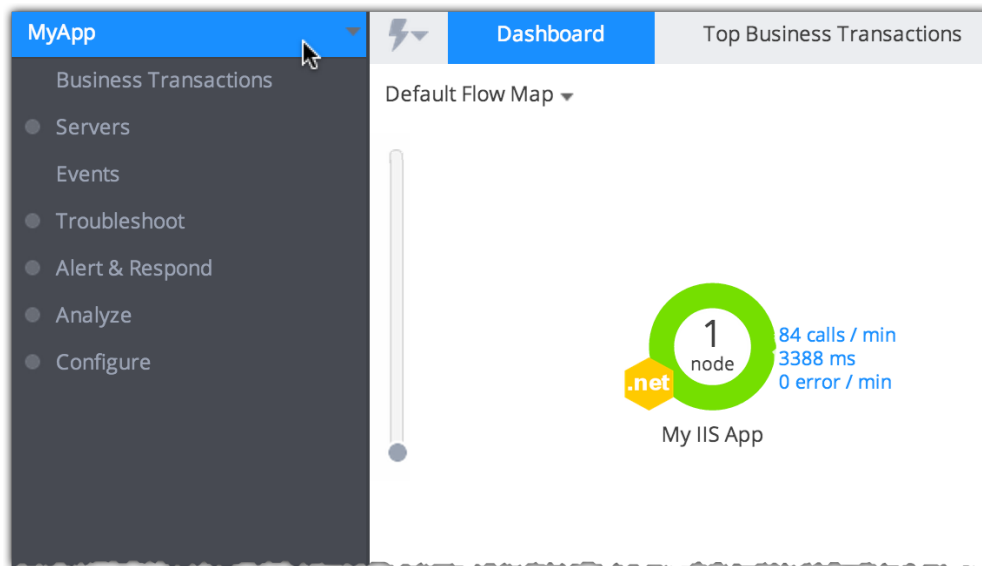
## Apply load to your application

If you are instrumenting a production application, this will happen with customer interaction. Otherwise, create some test load on your application. The agent instruments the application code and reports metrics back to the Controller.

## View your application

Log on to the AppDynamics Controller to see your application in action.

From here, you can install more agents or you can begin monitoring your application. See [AppDynamics Essentials](#).



## .NET Supported Environments

### On this page:

- [Supported Runtime Environments](#)
- [Automatically Discovered Business Transactions](#)
- [Remote Service Detection](#)
- [Supported Windows Azure Remote Services](#)
- [Data Storage Detection](#)

### Related pages:

- [Web EUEM Supported Environments](#)
- [Supported Environments and Versions](#)

## Supported Platform Matrix for the .NET Agent

### Supported Runtime Environments

This section lists the environments where the .NET Agent does some automatic discovery after little or no configuration.

#### OS Versions

- Microsoft\* Windows\* Server 2003 (32-bit and 64-bit)
- Microsoft Windows Server 2008 (32-bit and 64-bit)
- Microsoft Windows Server 2008 R2
- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2

#### Microsoft .NET Frameworks

- Microsoft .NET Framework versions 2.0, 3.0, 3.5, 4.0, 4.5

#### Runtime Environments

- Microsoft IIS versions 6.0, 7.0, 7.5, 8.0, 8.5
- Microsoft SharePoint 2010, 2013 as services running inside IIS
- Managed Windows Services
- Managed Standalone Applications

#### Microsoft Windows Azure

- Windows Azure Cloud Services (Web Roles and Worker Roles)

#### Unsupported Frameworks

- Microsoft .NET versions 1.0, 1.1
- Unmanaged native code
- Windows Azure Web Sites

#### Automatically Discovered Business Transactions

The .NET Agent discovers BTs for the following frameworks by default. The agent enables detection without additional configuration.

Type	Custom Configuration Options	Downstream Correlation
ASP.NET	Yes	Yes
ASP.NET MVC 2 ASP.NET MVC 3 ASP.NET MVC 4 ASP.NET MVC 5	Yes	Yes
.NET Remoting	No	Requires configuration. See <a href="#">Enable Correlation for .NET Remoting</a> .
Windows Communication Foundation (WCF)	No	Yes
Web Services including SOAP	No	Yes
<b>Message Queues</b>		
Apache ActiveMQ NMS framework and related MQs	No	Yes
IBM WebSphere MQ	No	Yes

RabbitMQ	Requires configuration. See <a href="#">Monitor RabbitMQ Backends for .NET</a> .	Yes
TIBCO Enterprise Message Service	No	Yes
TIBCO Rendezvous	No	Yes

The App Agent for .NET automatically discovers entry points for ASP.NET web forms with the Async property set to "true" in the [Page directive](#).

## Supported Loggers for the .NET Agent

- Log4Net
- NLog
- System Trace
- Windows Event Log

If you are using a different logger, see [Configure a Custom Logger](#).

## Remote Service Detection

The .NET Agent automatically detects the following remote service types. The agent enables detection by default. You don't need to perform extra configuration.

Type	Custom Configuration Options	Async Detection †	Downstream Correlation
Directory Services, including LDAP	No	No	N/A
HTTP	Yes	Requires configuration. See <a href="#">Monitor Async Backends for .NET</a> .	Yes
.NET Remoting	Yes	No	Requires configuration. See <a href="#">Enable Correlation for .NET Remoting</a> .
WCF	Yes	Requires configuration. See <a href="#">Monitor Async Backends for .NET</a> .	Yes
WCF Data Services	Yes	No	No

Web Services, including SOAP	Yes	Requires configuration. See <a href="#">Monitor Async Backends for .NET.</a>	Yes
<b>Message Queues</b>			
Apache ActiveMQ NMS framework and related MQs	Yes	No	Yes
IBM WebSphere MQ (IBM XMS)	Yes	No	Yes
Microsoft Message Queuing (MSMQ)	Yes	Requires configuration. See <a href="#">MSMQ Backends for .NET.</a>	Requires configuration. See <a href="#">MSMQ Backends for .NET.</a>
Microsoft Service Bus / Windows Azure Service Bus	No	No	Yes
RabbitMQ	Requires configuration. See <a href="#">Monitor RabbitMQ Backends for .NET.</a>	No	Yes
	Yes	No	Yes
TIBCO Rendezvous	Yes	No	Yes
Windows Azure Queue	No	No	No

† The agent discovers asynchronous transactions for the Microsoft .NET 4.5 framework. See [Monitor Async Backends for .NET.](#) for details.

### Supported Windows Azure Remote Services

Type	Configuration can be customized	Downstream Correlation
Azure Blob	No	No
Azure Queue	No	No
Microsoft Service Bus	No	Yes



## Data Storage Detection

The .NET Agent automatically detects the following data storage types. The agent enables detection by default. You don't need to perform extra configuration.

Type	Configuration Can Be Customized	Async Detection †	AppD for Databases?
ADO.NET (see supported clients below)	Yes	<i>New in 4.0.5,</i> Yes	No
Windows Azure Blob Storage	No	No	No

† The agent discovers asynchronous transactions for the Microsoft .NET 4.5 framework. See [Monitor Async Backends for .NET](#) for details.

## Supported ADO.NET Clients

AppDynamics can monitor any ADO.NET client version and type. Clients we've tested include the following:

Database Name	Database Version	Client Type
Oracle	10, 11, 12	ODP.NET
Oracle	10, 11, 12	Microsoft Provider for Oracle
MySQL	5.x	Connector/Net and ADO.NET
Microsoft SQL Server *	2005, 2008, 2012	ADO.NET

\* *Microsoft*, *SQL Server*, and *Windows* are registered trademarks of Microsoft Corporation in the United States and other countries.

## Install the .NET Agent

### On this page:

- [Installation Overview](#)
- [Requirements](#)
- [Install the .NET Agent](#)
- [Configure the .NET Agent](#)

### Related pages:

- [Configure the .NET Agent](#)
- [Unattended Installation for .NET](#)
- [Upgrade the .NET Agent](#)

### Watch the video:

To monitor IIS applications, Windows services, or standalone applications, install the .NET Agent once on each machine that hosts managed .NET applications. At start up, the agent initializes an individual instance of itself for each application running in the CLR.

This topic describes a new installation for the .NET Agent.

- To install from the command line, see [Unattended Installation for .NET](#).
- To upgrade, see [Upgrade the .NET Agent](#).

## Installation Overview

1. [Install the the agent](#).
2. [Configure the agent](#).
3. Restart instrumented applications.
  - For IIS, the configuration utility gives you the option to restart IIS or not.
  - If you don't restart IIS, monitoring doesn't begin until the next time IIS restarts.
  - You must restart Windows services and standalone applications manually.

## Requirements

The following Windows services must be enabled and running:

**Microsoft Distributed Transaction Coordinator (MSDTC):** MSDTC must run under the "NT Authority\NetworkServices" account. See "Verify MSDTC" on [Resolve .NET Agent Installation and Configuration Issues](#).

**COM+:** See "Verify COM+ Services are enabled" on [Resolve .NET Agent Installation and Configuration Issues](#).

### Windows Management Instrumentation

## Install the .NET Agent

1. Download the MSI installer package from the [AppDynamics Download Zone](#).
2. Run the MSI installer package.
3. Read the **End User Agreement** and click to accept. Click **Next**.
4. Optionally change the destination directory for the App Agent for .NET and click **Install**.  
By default, the agent installs to the following directory:

%ProgramFiles%\AppDynamics\AppDynamics .NET Agent

5. Click **Yes** on the **User Account Control** window to allow the installer to make changes to the computer.  
If the current account does not have administrator privileges, the installer prompts you to supply the password for an administrator account.
6. Wait for the installation to complete.
7. For new installs, AppDynamics recommends you launch the AppDynamics Agent

Configuration utility.

If you encounter problems installing, see [Resolve .NET Agent Installation and Configuration Issues](#).

## Configure the .NET Agent

Launch the AppDynamics Agent Configuration utility to configure the .NET Agent.

- For **IIS applications**, see [Configure the .NET Agent](#).
- For **Windows services** or **standalone applications**, see [Instrument Windows Services and Standalone Applications](#).

## Configure the .NET Agent

### On this page:

- [Prerequisites](#)
- [Configure the .NET Agent](#)

### Related pages:

- [Install the .NET Agent](#)
- [Name .NET Tiers](#)
- [Instrument Windows Services and Standalone Applications](#)
- [Unattended Installation for .NET](#)

### Watch the video:



.NET Agent Manual Configuration

[https://appdynamics-static.com/education/video/dotNETAgentManualInstallationandConfiguration/dotNETAgentManualInstallationandConfiguration\\_player.html](https://appdynamics-static.com/education/video/dotNETAgentManualInstallationandConfiguration/dotNETAgentManualInstallationandConfiguration_player.html)

Configure the .NET Agent according to the types of applications you want to monitor:

- For IIS applications, use the configuration utility with either automatic or manual tier naming. See the instructions in this topic.
- For Windows services or standalone applications, use the configuration utility, then manually update the config.xml. See [Instrument Windows Services and Standalone Applications](#).

**i** It is possible to instrument any combination of IIS Applications, Windows services, and standalone applications on a single server. Run the configuration utility to configure IIS, then follow the instructions for Windows services and Standalone applications.

Use the .NET Agent Configuration Utility to configure the agent just after installation, or to make changes to existing agent configurations. The utility configures the agent for one machine at a time.

## Prerequisites

- [Install the .NET Agent](#).
- Uninstall any pre-existing profiler, such as Ant, VS 2010 Performance Tools, or others. The utility alerts you if it finds a pre-existing profiler.
- [Decide to name tiers automatically or manually](#).

**i** The configuration utility must restart IIS to apply configurations. The utility offers you the option to restart IIS or not. If you choose not to restart, configurations apply the next time IIS restarts.

## File System Security Settings

The following Windows accounts require specific file system permissions:

- The account you use to run your web application as defined by its application pool or the Windows service account.
- The account you use to run the AppDynamics Agent Coordinator, by default the Local System account.

The required permissions are as follows:

- **Write** permission to the .NET App Agent logs directory:  
For agent version 3.7.8 or later, the default is as follows:  
**Windows Server 2008 and later:** %ProgramData%\AppDynamics\DotNetAgent\Logs  
**Windows Server 2003:** %AllUsersProfile%\Application Data\AppData\DotNetAgent\Logs  
For agent version 3.7.7 or earlier, the default is C:\Appdynamics\Logs.
- **Read** and **Execute** permissions to the .NET App Agent install directory, by default C:\Program Files\AppData\DotNetAgent
- **Read** and **Execute** permissions the web application installation directory, for example C:\inetpub\wwwroot\myapp

## Configure the .NET Agent

1. In the Windows menu, click **AppDynamics -> .NET Agent -> AppDynamics Agent Configuration**.

If the "Warning: 3rd Party Profiler installed" message displays, click **Yes** to exit and uninstall any pre-existing profiler.

✓ Check the registry to make sure that the uninstall process cleaned up the registry entries. Use the warning message to identify any undeleted profiler environment variables.

2. When the configuration utility detects legacy agent configurations from version 3.7.7 or earlier, it displays the **Upgrade Configuration** window.

- Answer **Yes** to remove legacy configurations. See "Clean up legacy configurations" on [Upgrade the .NET Agent](#).

⚠ Removing legacy configurations modifies web.config files causing IIS to restart affected applications.

- Answer **No** to leave legacy configurations in place.

3. When the utility discovers no further profiler conflicts or after any configuration clean up it displays the welcome window.

4. Click **Next** to advance to **Log directory permissions**.

(Optional) If you want to change the default location of the log directory, click **Change** and select a new location.

ℹ The default logs directories are as follows:

**Windows Server 2008 and later:** %ProgramData%\AppDynamics\DotNetAgent\Logs

**Windows Server 2003:** %AllUsersProfile%\Application  
Data\AppData\DotNetAgent\Logs

5. If needed, add accounts for log directory permissions. Click **Add**. If you get a warning message, make sure that the account is valid on the system.

6. Click **Next** and the wizard confirms the list of accounts.

7. Click **Next** to advance to **Controller Configuration**.

8. Enter the Controller access information and credentials.

- The .NET Agent configuration utility only supports configuration of one Controller and business application per server.
  - Use tiers to organize different applications you instrument on a single server.
  - Or manually configure support for multiple business applications, see [Configure Multiple Business Application Support for .NET](#).
- For a SaaS Controller, enter the server name or IP, port number, account name, and access key as provided to you by AppDynamics.
- For an on-premise Controller, if you haven't already installed it, cancel this installation and see [Install the Controller](#). Otherwise enter the server name and port number of an existing Controller.
- For a secure connection, click **Enable SSL**.
  - ℹ The Controller must use a trusted certificate.
- If needed, fill in the proxy information. Proxies that use authentication require additional configuration.

9. Click **Test Controller Connection** to verify the connection.

10. Click **Next** to advance to the **Application Configuration**

AppDynamics retrieves existing business application information from the Controller and displays it in the left column. Controller connection status displays on the right.

11. Click **Existing Applications from the Controller** to select business applications from the Controller.


If you haven't defined business applications in the Controller, the utility displays an empty list.

Click **New Application** to define a new business application. Be careful about spellings and capitalization and note down the exact name.


 Ampersands are not supported in application names.

12. Click **Next** to advance to **Assign IIS applications to tiers**.

#### Automatically name tiers

1. In the **Assign IIS applications to tiers** window click **Automatic**.
2. If prompted, click **OK** to confirm Automatic configuration.  
The configuration utility summarizes the configuration settings.
3. By default when you click **Next** the configuration utility restarts IIS.  
 If you do not want to apply the configuration right away, uncheck the box. The Configuration Utility saves the information and applies it the next time you restart IIS.
4. If you proceed and click **Next**, the configuration utility logs its activities, including stopping and restarting IIS, and reports any problems. Review the summary for any issues in red font. Green font indicates the more interesting logged events. The summary shows any Warnings (W) or Errors (E).
5. When there are no errors, click **Next**.
6. Click **Done** to close the Configuration Utility.

#### Manually name tiers

1. In the **Assign IIS applications to tiers** window click **Manual**, then click **Next**.
2. Assign IIS Applications to AppDynamics tiers.  
Select a tier on the right and click a business application on the left. The utility highlights the assigned tier in boldface.  
 For large IIS installations, use the Max IIS tree depth pulldown to display all the projects. A large tree depth may take some time to view.  
To create new tiers, enter a name and click **Add Tier**.
3. When you are done click **Next**. AppDynamics displays a configuration summary.
4. On the **Configuration Summary** window, uncheck **Restart IIS** if you don't want to immediately restart IIS.  
You may restart later to apply your changes, or they will take effect after a reboot.
5. If you proceed and click **Next**, the Configuration Utility logs its activities, including stopping and restarting IIS, and reports any problems.
6. Review the configuration log summary.  
As it applies the configuration, AppDynamics generates a log of the configuration activities and displays a summary. Review the summary for any issues in red font. Green font indicates the more interesting logged events. The summary shows any Warnings (W) or Errors (E).
7. Click **Next**. The wizard completes.

For troubleshooting information see [Resolve .NET Agent Installation and Configuration Issues](#).

## Enable SSL for .NET

### On this page:

- [Prerequisites](#)
- [Enable SSL](#)
- [Establish Trust for the Controller's SSL Certificate](#)
- [Troubleshoot Communication Issues](#)

### Related pages:

- [Security](#)
- [Administer the .NET Agent](#)
- [.NET Agent Configuration Properties](#)

This topic covers how to configure the .NET Agent (the agent) to connect to the Controller using SSL. It assumes that you use a SaaS Controller or have configured the on-premise Controller to use SSL.

### Prerequisites

Before you configure the agent to enable SSL, gather the following information:

- Identify if the Controller is SaaS or on-premise.
- Identify the Controller SSL port.
  - For SaaS Controllers the SSL port is 443.
  - For on-premise Controllers the default SSL port is 8181, but you may configure the Controller to listen for SSL on another port.
- Identify the signature method for the Controller's SSL certificate:
  - A publicly known certificate authority (CA) signed the certificate. This applies for Verisign, Thawte, and other commercial CAs.
  - A CA internal to your organization signed the certificate. Some companies maintain internal certificate authorities to manage trust and encryption within their domain.
  - The Controller uses a self-signed certificate.

### Enable SSL

There are two ways to update the SSL settings for the agent. You can use the [AppDynamics Agent Configuration Utility](#). Otherwise, edit the settings directly in the [config.xml](#), see [Administer the .NET Agent](#).

When you enable SSL for the .NET Agent, you automatically enable SSL for the .NET Machine Agent.

#### To configure SSL using the AppDynamics Agent Configuration utility

1. Launch the AppDynamics Agent Configuration utility.
2. In the **Controller Configuration** window, set the **Port Number** to the SSL port for the

Controller.

- For a **SAAS Controller**, set the **Port Number** to 443.
- For an **on-premise Controller**, set the **Port Number** to the on-premise SSL port. The default is 8181.

3. Click **Enable SSL**.

This example demonstrates connection to an on-premise Controller listening for SSL on port 8181:

**.NET Agent Configuration**

**Controller Configuration**

Please enter controller configuration and then click on the "Test controller connection" button to test the connection

Controller configuration

**Server(Name/IP):** mycontroller.mycompany.com

**Port Number:** 8181 ☒ Enable SSL

For multi-tenant controller:

Account Name:

Account Access Key:

☐ Use proxy:

Proxy address:

Proxy port:

Test Controller connection

Controller connection status:

Mandatory settings are in **Bold**

< Back Next > Cancel

1. Set the Controller **Port Number** to the SSL port.

2. Click **Enable SSL**.

4. Click **Next** and proceed with the rest of the windows to complete the configuration.

5. Restart instrumented applications: IIS applications or application pools, Windows services, or standalone applications.

If you use automatic tier configuration, restart IIS. For example, open a command prompt and run:

```
iisreset
```

Upon restart the agent connects with the Controller via SSL.

To configure SSL in the config.xml

1. Open the config.xml file as administrator. See [Administer the .NET Agent](#).

2. Update the SSL settings. See [Controller Element](#).

- Set the **Controller port attribute** to the on-premise SSL port. The default is 8181. See [Cont](#)



[roller port attribute](#).

- Set the **Controller SSL attribute** to "true". See [Controller ssl attribute](#).

3. Save your changes.

4. Restart the AppDynamics.Agent.Coordinator service.

5. Restart instrumented applications: IIS applications or application pools, Windows services, or standalone applications.

If you use Automatic configuration, restart IIS. For example, open a command prompt and run:

```
iisreset
```

Upon restart the agent connects with the Controller via SSL.

#### Sample SaaS SSL config.xml configuration

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <controller host="mycompany.saas.appdynamics.com" port="443" ssl="true"
enable_tls12="true">
    <application name="MyDotNetApplication" />
  </controller>
  ...
</appdynamics-agent>
```

#### Sample on-premise SSL config.xml configuration

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8181" ssl="true"
enable_tls12="true">
    <application name="MyDotNetApplication" />
  </controller>
  ...
</appdynamics-agent>
```

#### Establish Trust for the Controller's SSL Certificate

The .NET Agent requires that the Common Name (CN) on the Controller certificate match the DNS name of the Controller. Additionally, certificates for the root CA that signed the Controller's SSL certificate must reside in the **Windows Trusted Root Certification Authorities** store for the **Local Computer**.

#### Certificates signed by a publicly known Certificate Authority

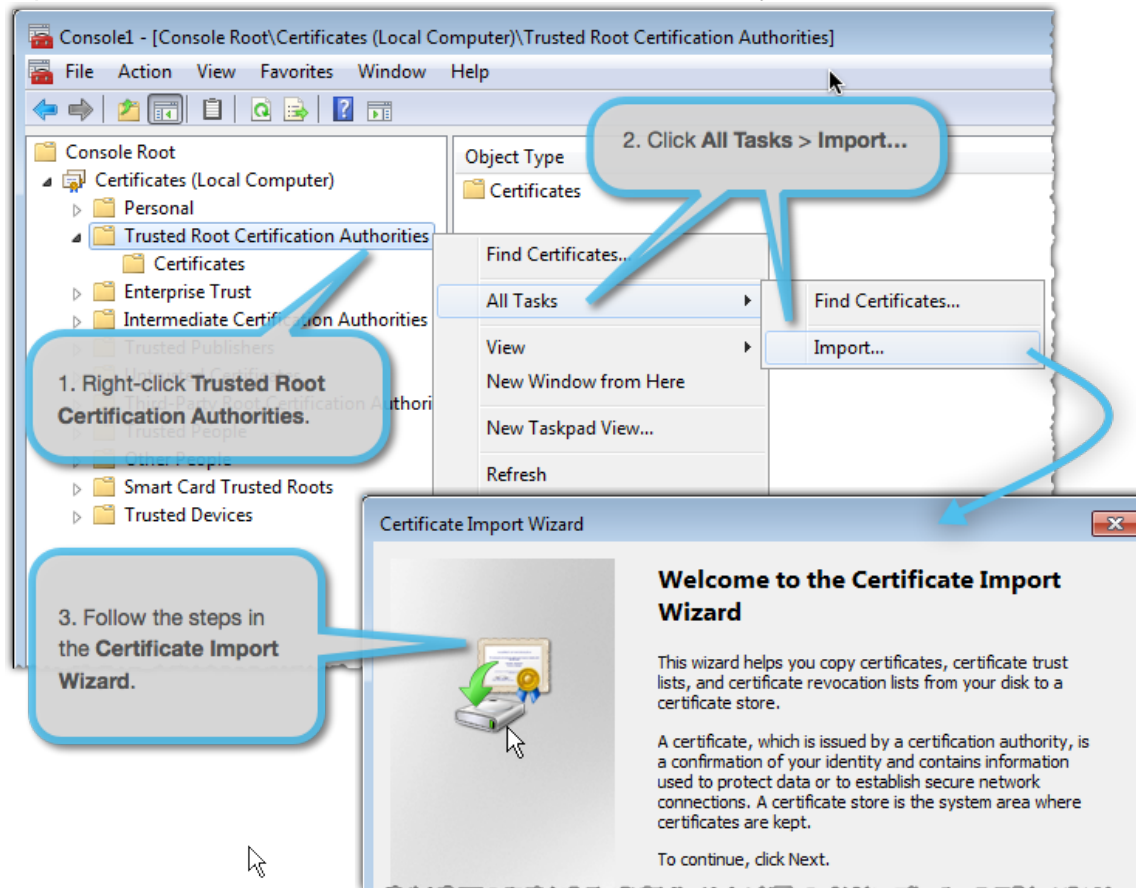
The root certificates for most publicly trusted CA signing authorities, such as Verisign, Thawte, and

other commercial CAs, are in the Trusted Root Certification Authorities store by default.

#### Certificates signed by an Internal Certificate Authority

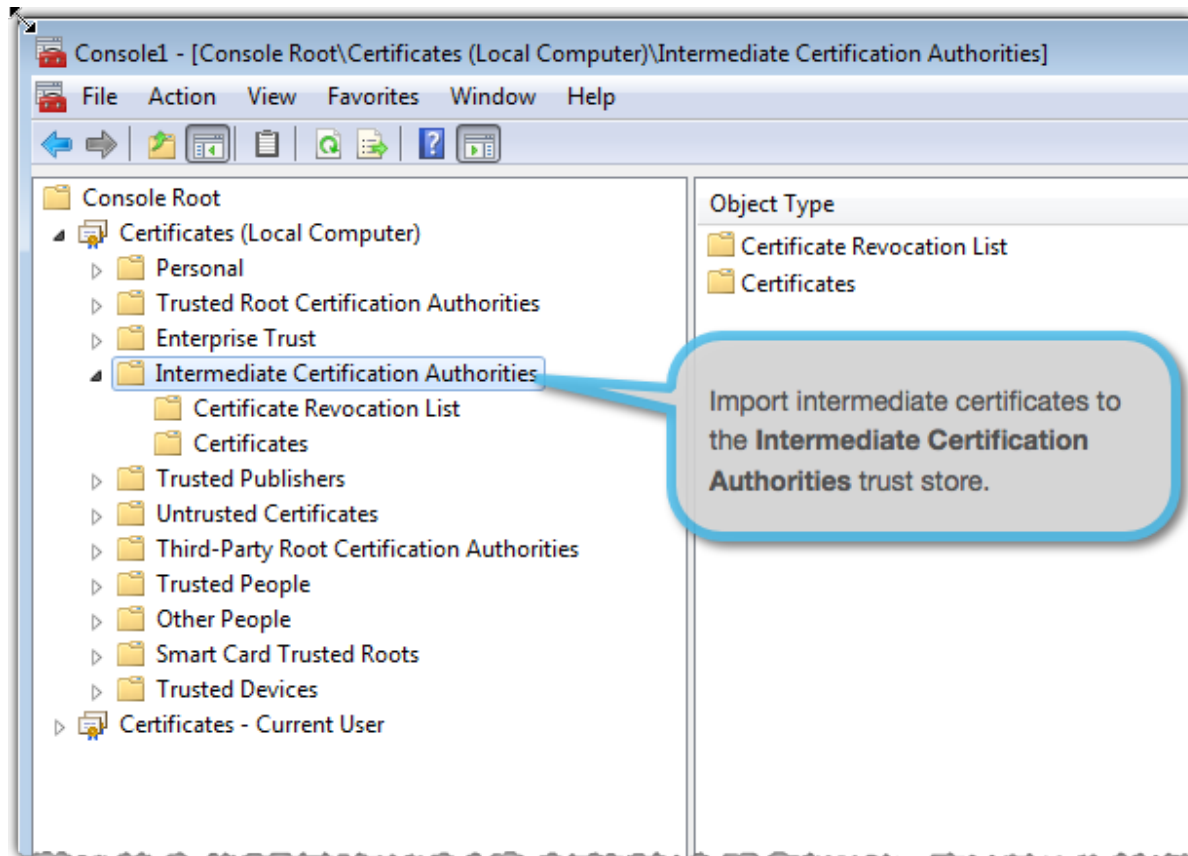
If your organization uses internal CA to sign certificates, you may need to obtain the root CA certificate from your internal security management resource. To import the root certificate, see [Adding certificates to the Trusted Root Certification Authorities store for a local computer](#).

This example shows how to use the Certificate snap-in for the Microsoft Management Console to import a certificate for a Trusted Root Certification Authority:



**i** If an intermediate CA signed the Controller's certificate, you must import the certificate for the intermediate CA in addition to the one for the root CA that signed the intermediate CA's certificate. If your controller is publicly accessible, you can use a certificate checker to identify the certificates required to complete the trust chain. See [the certificate checker from Thawte](#).

This examples shows the **Intermediate Certification Authorities** store:



### Self-Signed Certificates

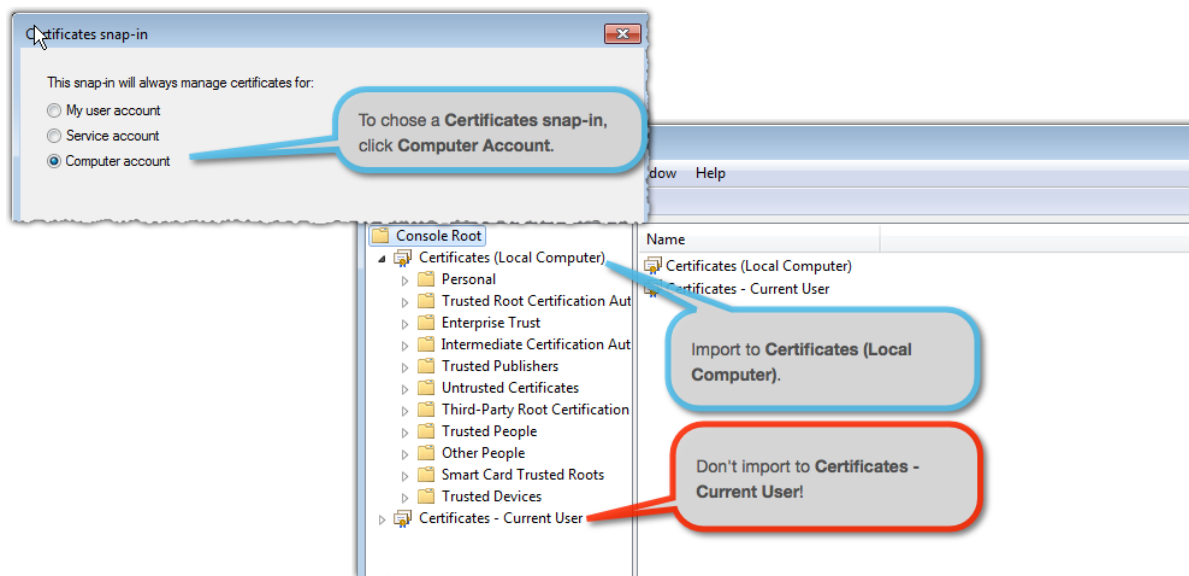
The App Agent for .NET does not support self-signed certificates. In order to implement SSL, the Controller must use a certificate signed by a trusted CA signing authority or an internal trusted root CA. See [Security](#).

### Troubleshooting Tips

- By default the .NET Agent encrypts communications with the controller using TLS 1.2. If you are unwilling or unable to use TLS 1.2, set the Controller enable TLS 1.2 attribute to "false". See "Controller Element" on [.NET Agent Configuration Properties](#).

*New in 4.0.4:* When you enable SSL, the agent secures communication to the Controller using the protocols set for `ServicePointManager.SecurityProtocol` in your application. Set the Controller enable TLS 1.2 attribute to "true" to add TLS 1.2 as the first option in the list of protocols. This affects all secure communications from your application, not just requests to the AppDynamics Controller. See "Controller Element" on [.NET Agent Configuration Properties](#).

- If you imported certificates for a root or intermediate CA, verify the certificate store where you imported them. Import them to **Certificates (Local Computer)**.



- The AppDynamics SaaS Controller uses certificates signed by Thawte. In some cases, SaaS customers must import the Thawte root certificates into the **Windows Trusted Root Certification Authorities** store.
- In some cases system administrators set up group policies that require external certificates be imported to the **Third-Party Root Certification Authorities** store. If importing the certificate for the root CA to the Windows Trusted Certification Authorities store doesn't work, try the Third-Party Root Certification Authorities store.

### Troubleshoot Communication Issues

If you have verified all prerequisites, but have communication issues try the following:

- Verify the default ciphers are enabled in Windows Server:  
Check the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Security Providers\SCHANNEL\Ciphers\ registry key. If subkeys exist, your operations team may have disabled certain ciphers. Please contact support for assistance.
- If you are installing on Windows Server 2003 and the Certificate Authority is running on Windows Server 2008, you must install a hotfix from Microsoft: <https://support.microsoft.com/en-us/kb/968730> . Otherwise you may see the following error: Controller communication failed. Details: The underlying connection was closed. Could not establish trust relationship for the SSL/TLS secure channel.

## Instrument Windows Services and Standalone Applications

### On this page:

- [Prerequisites](#)
- [Manually Configure the .NET Agent](#)
- [Sample Configuration File](#)

### Related pages:

- [Configure the .NET Agent](#)


- [Install the .NET Agent](#)
- [.NET Agent Configuration Properties](#)

By default, the .NET Agent only instruments IIS applications. Edit the .NET Agent config.xml configuration file to enable the agent for Windows services and standalone applications.

It is possible to instrument any combination of IIS applications, Windows services, and standalone applications on a single server.


### Prerequisites

1. [Install the .NET Agent](#).
2. [Run the AppDynamics Agent Configuration](#) utility to perform basic configuration tasks such as controller connectivity and to generate a config.xml.

 If you have previously instrumented app agents for IIS applications, don't run the configuration utility. You already have a config.xml.

Use the configuration utility to do the following:

- Change the location of the Logs directory and assign permissions.
- Configure and test connectivity to the Controller.
- Set the Business Application for the agent.

 If you are not instrumenting IIS, choose **Manual** for the method of tier generation and assignment. Don't assign any tiers for any IIS applications. This disables instrumentation for all IIS applications.

### Manually Configure the .NET Agent

Once you have configured the Controller properties for the .NET Agent, instrument your Windows service or standalone application by adding an XML element for it to the config.xml.

1. Edit the config.xml file as an administrator. See [Administer the .NET Agent](#).

If you haven't instrumented IIS applications, the file contains minimal configurations for Controller connectivity and the machine agent. Verify the Controller properties and the Business Application name.

▼ [Show minimal configuration without IIS](#)

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <controller host="mycontroller.example.com" port="8090"
ssl="false">
    <application name="My Business Application" />
  </controller>
  <machine-agent />
  <app-agents>
    <IIS>
      <applications />
    </IIS>
  </app-agents>
</appdynamics-agent>
```

If you have already instrumented IIS applications, you see their configurations under the IIS element.

2. Add the Standalone Applications block as a child of the App Agents element. Create a Standalone Application element for each Windows service or standalone application you want to instrument.
  - Use the Tier element to assign the instrumented application to a tier in the Controller. See [.NET Agent Configuration Properties](#).
  - Set the Standalone Application element executable attribute to one of the following:
    - Executable name. For example, `MyStandaloneApp.exe` or `MyWindowsService.exe`. The file extension is optional, so `MyStandaloneApp` also works.
    - Full path to the executable. For example, `C:\Program Files\MyApplication\MyStandaloneApp.exe`.
    - Partial path to the executable. For example, `MyApplication\MyStandaloneApp.exe`.
    - ✔ Use the full or partial path to the executable when you want to assign different tiers to separate instances of the same executable running from different paths.
  - To differentiate between two instances of the same executable, specify any unique portion of the application's command line, such as an argument, in the Standalone Application command-line attribute.

```
<standalone-applications>
  <standalone-application
executable="MyStandaloneApp.exe">
    <tier name="Standalone Tier 1" />
  </standalone-application>
  <!-- Instrument a standalone application using a partial
path. -->
  <standalone-application
executable="MyApplication\MyOtherStandaloneApp.exe">
    <tier name="Standalone Tier 2" />
  </standalone-application>
  <!-- Instrument a Windows service using arguments. -->
  <!-- The following example matches the command
"MyWindowsService.exe -d -x -r". -->
  <standalone-application
executable="MyWindowsService.exe" command-line="-x">
    <tier name="Windows Service Tier" />
  </standalone-application>
</standalone-applications>
```

3. Restart the AppDynamics.Agent.Coordinator
4. Restart the Windows service or standalone application.

**i** If your Windows service or standalone application doesn't implement an [auto-detected framework](#), you must configure a [POCO entry point](#) for a class/method in your service for the agent to begin instrumentation.

### Sample Configuration File

This sample config.xml demonstrates instrumentation for a Windows service and standalone application:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090"
ssl="false">
    <application name="My Business Application" />
  </controller>
  <machine-agent />
  <app-agents>
    <IIS>
      <applications />
    </IIS>
    <standalone-applications>
      <standalone-application executable="MyWindowsService.exe"
command-line="-x">
        <tier name="Windows Service Tier" />
      </standalone-application>
      <standalone-application executable="MyStandaloneApp.exe">
        <tier name="Standalone Tier" />
      </standalone-application>
    </standalone-applications>
  </app-agents>
</appdynamics-agent>
```

## Name .NET Tiers

### On this page:

- [Name IIS Tiers Automatically](#)
- [Name Azure Tiers Automatically](#)
- [Name IIS Tiers Manually](#)
- [Name Windows Service or Standalone Application Tiers](#)

### Related pages:

- [Configure the .NET Agent](#)
- [Automatically Name .NET Nodes](#)

In AppDynamics, a tier represents a key service in your application environment. Such services may include ASP.NET front ends, WCF services, .NET web services, or standalone applications. On flow maps, tiers show up as the primary visual indicators of the component services of your business application. Therefore, you want to name your tiers so your users find them logical and easy to understand.

The .NET Agent includes the AppDynamics Agent Configuration Utility (configuration utility) to map IIS sites to tiers, see [Configure the .NET Agent](#). Manual configuration options require you to edit



the config.xml file, see "Where to Configure Agent Properties" on [Administer the .NET Agent](#).

### Name IIS Tiers Automatically

Name IIS tiers automatically using the configuration utility. Choose **Automatic** on the **Assign IIS applications to tiers** window. The .NET Agent instruments all IIS sites and names the tiers using the following scheme:

`IIS site/app`

The agent omits app when the application is the root application for the IIS site.

✔ Use this option in the following cases:

- You are new to AppDynamics and the .NET Agent.
- You want to instrument all IIS applications and your team can easily understand IIS application names.

### Name Azure Tiers Automatically

The .NET Agent automatically names Azure tiers using the following scheme:

`Azure role name`

If you want to customize tier naming for Azure, see **Edit the config.xml directly** under [Name IIS Tiers Manually](#).

### Name IIS Tiers Manually

The .NET Agent offers two options for naming IIS tiers manually.

**Use the configuration utility:** On the **Assign IIS applications to tiers** window, choose **Manual**. The configuration utility lets you create new tiers and assign IIS applications to tiers.

✔ Use this option in the following cases:

- You don't want to instrument all IIS applications on the server.
- You want custom tier names.
- You want to combine multiple applications into a single tier.

**Edit the config.xml directly:** For each IIS site you want to instrument, add an application element as a child element of the IIS Applications element in the config.xml. You can specify a static IIS site name or a regular expression which is helpful for variable site names like Azure site names. For the full syntax and an example, see "IIS Applications Element" on [.NET Agent Configuration Properties](#).

For an example of Azure tier naming using a regular expression, see "Customize Tier Names for Azure Roles" on [Install AppDynamics for Windows Azure with NuGet](#).

✔ Use this option to customize Azure tier names or in cases where it is not feasible to use the configuration utility.

### Name Windows Service or Standalone Application Tiers

Name the tiers for each instrumented Windows service or standalone application manually in the

config.xml. See the following instructions:

- [Instrument Windows Services and Standalone Applications](#)

## Automatically Name .NET Nodes

**On this page:**

- [Name IIS Nodes Automatically](#)
- [Name Windows Service or Standalone Application Nodes Automatically](#)

**Related pages:**

- [Configure the .NET Agent](#)
- [Name .NET Tiers](#)

By default the AppDynamics .NET Agent (agent) automatically names nodes using a combination of the Windows machine name, the tier name, and the name of the .NET application.

**i** If you are upgrading from the .NET Agent version 3.7.7 or earlier, the new node naming convention takes effect upon restart. Nodes named under the old scheme become historical nodes with no correlation to new nodes.

### Name IIS Nodes Automatically

The agent names IIS nodes as follows:

```
<machine NetBIOS name>-<tier>-<IIS site>/<app>
```

The agent omits app when the application is the root application for the IIS site.

The agent omits tier when the tier name is the same as the IIS site name.

For example:

```
WIN-86M7CEJO6P5-Order Server-OrderSvc
```

WIN-86M7CEJO6P5 is the machine NetBIOS name.

Order Server is the tier name.

OrderSvc is the IIS site name. The application is the site root, so the agent omits the application name.

For another example:

```
WIN-86M7CEJO6P5-Order Server-Store/ProcessOrder
```

Store is the IIS site name.

ProcessOrder is the application name within the site.

**i** Different .NET versions of the same application have their own versions of the CLR and run on independent processes. Therefore the agent identifies the two processes as different nodes.

### IIS Web Gardens

The syntax for web gardens is the same as IIS Nodes, except that the agent appends a zero-based process index to differentiate the worker processes.

```
<machine NetBIOS name>-<tier>-<IIS site>/<app>-<process index>
```

When IIS first launches web garden processes, the agent assigns a sequential index to each process. However when IIS recycles a process, the agent reuses the available index freed by the terminated process. Therefore there is no correlation between the index sequence and the chronological start of the process.

Sometimes you may see more nodes than the maximum number of worker processes. This can happen when a long-running request prevents a process from shutting down before its replacement launches.

### **Name Windows Service or Standalone Application Nodes Automatically**

The agent names Windows service and standalone application nodes as follows:

```
<machine NetBIOS name>-<tier>-<Windows service name or executable name>
```

The agent omits tier when the tier name is the same as the service name or executable name.

For example:

```
WIN-86M7CEJO6P5-MyWindowsService
```

WIN-86M7CEJO6P5 is the machine name.

MyWindowsService is the the Windows service name.

For another example:

```
WIN-86M7CEJO6P5-MyStandaloneApp.exe
```

WIN-86M7CEJO6P5 is the machine name.

MyStandaloneApp.exe is the executable file name.

## **Unattended Installation for .NET**

### **On this page:**

- [Create a Setup Configuration File](#)
- [Sample Setup Configuration File](#)
- [Unattended Installation](#)
- [Setup Configuration File Properties](#)

### **Related pages:**

- [Configure the .NET Agent](#)
- [.NET Agent Configuration Properties](#)
- [Instrument Windows Services and Standalone](#)

## Applications

The .NET Agent provides a command-line unattended installation procedure for cases when you have multiple servers that require the same AppDynamics configuration. Using unattended installation, you only need to configure once, then use the command line scripts to automate installation and instrumentation on multiple servers

**i** For Windows Server 2003, unattended installation of the .NET Agent requires Windows Server 2003 SP2.

For more detail about how to install and configure the agent manually see [Install the .NET Agent](#).

### Create a Setup Configuration File

The .NET Agent MSI installer package allows you to specify the path to a setup configuration file to perform an unattended installation. The setup configuration file contains all the properties you need to enable instrumentation for your .NET applications.

You must run the .NET MSI installer package on one machine before you can use the AppDynamics Agent Configuration utility to create a setup configuration file. See [Install the .NET Agent](#).

Setup configuration files created in previous versions of the AppDynamics Agent Configuration utility work with the 4.0 installer.

1. Launch the AppDynamics Agent Configuration utility from the command line. Use the **-s** parameter to specify the setup configuration file destination.

```
AppDynamics.Agent.Winston.exe -s <path to setup configuration file>
```

For example:

```
c:\Program Files\AppDynamics\AppDynamics .NET  
Agent\AppDynamics.Agent.Winston.exe -s  
"c:\temp\configurationSavedSetupConfiguration.xml"
```

2. Go through the configuration wizard normally.

The configuration utility saves the setup configuration file to the path you specified.

**i** The configuration utility only configures instrumentation for IIS applications.

3. Optional: To perform unattended installation for Windows services or for standalone applications, you must edit the setup configuration file manually. See [Instrument Windows Services and Standalone Applications](#).

### Sample Setup Configuration File

The following example shows a setup configuration file that instruments two IIS Applications (MainBC and SampleHTTPService), a Windows service (BasicWindowsService), and a standalone application (MyStandaloneApp.exe).

The configuration file sets the log directory as C:\ProgramData\AppDynamics\DotNetAgent\Logs

and grants write permission to four accounts.

```
<winston>
  <logFileDirectory
directory="C:\ProgramData\AppDynamics\DotNetAgent\Logs" />
  <logFileFolderAccessPermissions defaultAccountsEnabled="false">
    <account name="NT AUTHORITY\LOCAL SERVICE" displayName="LOCAL
SERVICE" />
    <account name="NT AUTHORITY\SYSTEM" displayName="SYSTEM" />
    <account name="NT AUTHORITY\NETWORK SERVICE" displayName="NETWORK
SERVICE" />
    <account name="IIS_IUSRS" displayName="ApplicationPool Identity"
/>
  </logFileFolderAccessPermissions>
  <appdynamics-agent
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <controller host="mycontroller.mycompany.com" port="8090"
ssl="false">
      <application name="My Business Application" />
    </controller>
    <machine-agent />
    <app-agents>
      <IIS>
        <applications>
          <application path="/" site="MainBC">
            <tier name="Main Site" />
          </application>
          <application path="/" site="SampleHTTPService">
            <tier name="HTTP Services" />
          </application>
        </applications>
      </IIS>
      <standalone-applications>
        <standalone-application executable="MyStandaloneApp.exe">
          <tier name="Standalone App" />
        </standalone-application>
        <standalone-application executable="MyWindowsService.exe">
          <tier name="Windows Service" />
        </standalone-application>
      </standalone-applications>
    </app-agents>
  </appdynamics-agent>
</winston>
```

## Unattended Installation

### Requirements


**Microsoft Distributed Transaction Coordinator (MSDTC):** MSDTC must run under the "NT Authority\NetworkServices" account. See "Verify MSDTC" on [Resolve .NET Agent Installation and Configuration Issues](#).

**COM+:** See "Verify COM+ Services are enabled" on [Resolve .NET Agent Installation and Configuration Issues](#).


## Windows Management Instrumentation

### Perform unattended installation


1. Uninstall any existing agent.

 If you fail to uninstall the previously installed agent and try to install the same agent version, the installer package may remove the AppDynamics Agent Coordinator service. In this case, uninstall and proceed with the unattended installation as normal.

2. Launch an elevated command prompt with full administrator privileges. See [Start a Command Prompt as an Administrator](#).

 Logging on to Windows as a member of the Administrators group does not grant sufficient permissions to run the installer.

3. Run the agent MSI installer package from the elevated command prompt. Use the **AD\_SetupFile** parameter to pass the absolute file path to the setup configuration file.

 The AD\_SetupFile parameter accepts paths with spaces in the path name. In previous versions spaces were not allowed.

```
msiexec /i dotNetAgentSetup64.msi /q /norestart /lv
%TEMP%\AgentInstaller.log AD_SetupFile=<absolute path to setup config.xml>
```

-  Optionally use the **INSTALLDIR** parameter to customize the agent installation directory.

```
INSTALLDIR=<custom agent install directory>
```

For example:

```
msiexec /i dotNetAgentSetup64.msi /q /norestart /lv
%TEMP%\AgentInstaller.log AD_SetupFile=C:\temp\SetupConfig.xml
INSTALLDIR=D:\AppDynamics
```

4. Start the AppDynamics.Agent.Coordinator.

```
net start AppDynamics.Agent.Coordinator
```

5. Restart applications you have instrumented: IIS, Windows services, or standalone applications.

For example, to restart IIS:

```
iisreset
```

## Setup Configuration File Properties

### Winston element

The Winston element is the root element for the configuration file.

**Required element:** <winston>

### Log File Directory element

The Log File Directory element is a child element of the Winston element. Use the **directory** attribute to specify the log directory. If you omit the Log File Directory element, we use the default directory:

**Windows Server 2008 and later:** %ProgramData%\AppDynamics\DotNetAgent\Logs

**Windows Server 2003:** %AllUsersProfile%\Application  
Data\AppDataDynamics\DotNetAgent\Logs

**Optional element:** <logFileDirectory

directory="C:\ProgramData\AppDataDynamics\DotNetAgent\Logs" />

### Log File Folder Access Permissions element

The Log File Folder Access Permissions is a child element of the Winston element. Unless you set the **default accounts enabled** attribute to false, we grant write access to the logs folder for the default accounts:

- LOCAL SERVICE
- SYSTEM
- NETWORK SERVICE
- ApplicationPool Identity

**Optional element:** <logFileFolderAccessPermissions

defaultAccountsEnabled="false">

### Account element

The Account element is a child element of the Log File Folder Access Permissions element. Create an **Account** element for the Windows account you use to run your application.

Set the **name** attribute to the name of account you use to run your application: the account for the application pool for IIS or the Windows service account.

The **display name** attribute is a user-friendly name you choose for the account. The display name shows up in log entries about assigning permissions for the account.

**Optional element:** <account name="MyAppPoolIdentity" displayName="Custom  
ApplicationPool Identity" />

For example, if you run a Windows service under a domain account:

```
<account name="MYDOMAIN\service_acct" displayName="Domain Service Account" />
```

#### AppDynamics Agent element

The AppDynamics Agent element is a child of the Winston element. It follows the same format as the config.xml to define the agent configuration for all your .NET applications. See [.NET Agent Configuration Properties](#).

**Required element:** <appdynamics-agent

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xsd="http://www.w3.org/2001/XMLSchema">

## Upgrade the .NET Agent

### On this page:

- [Upgrade the .NET Agent from Version 3.8.2 or Newer](#)
- [Upgrade the .NET Agent from Version 3.7.8 to 3.8.1](#)
- [Upgrade the .NET Agent from Version 3.7.7 or Earlier](#)

### Related pages:

- [Agent - Controller Compatibility Matrix](#)
- [Install the .NET Agent](#)
- [Configure the .NET Agent](#)
- [.NET Agent Directory Structure](#)
- [Resolve .NET Agent Installation and Configuration Issues](#)

This topic describes how to upgrade to the .NET Agent (agent) version 4.0. The instructions vary based upon your current version of the .NET Agent.

### Upgrade the .NET Agent from Version 3.8.2 or Newer

You don't need to uninstall the old agent first when you upgrade from the .NET Agent version 3.8.2 or newer.

1. Stop w3wp processes for instrumented IIS applications. Stop instrumented Windows services or standalone applications.
2. Install the new agent. See [Install the .NET Agent](#).
3. Restart the AppDynamics Agent Coordinator service:

```
net stop AppDynamics.Agent.Coordinator
net start AppDynamics.Agent.Coordinator
```
4. Restart IIS:
  - Launch the AppDynamics Agent Configuration utility and click **Restart IIS** on the Configuration Summary window.
  - OR
  - Execute `iisreset` from the command line.Restart Windows services and standalone applications.



### Upgrade the .NET Agent from Version 3.7.8 to 3.8.1

The MSI installer package for the new version of the .NET Agent installs the updated agent files and maintains all legacy configurations. After you complete the installation, start the AppDynamics.Agent.Coordinator service and instrumented applications to finish the upgrade.

#### Uninstall the old version of the agent

1. Stop IIS, instrumented Windows services, and instrumented standalone applications.

✓ If you shut down IIS but continue to see active IIS Worker Processes, check the Application Pools pane in the IIS Manager and stop any started application pools.

⚠ Failing to stop instrumented applications before uninstalling the .NET Agent may require you to reboot the machine.

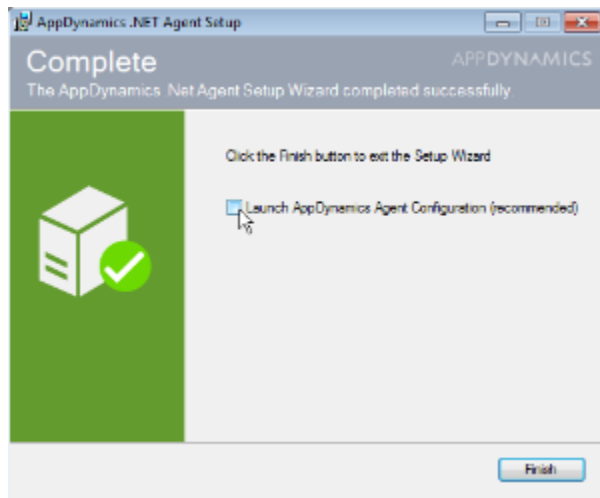
2. Stop the **AppDynamics.Agent.Coordinator** service.
3. In the Control Panel, select Add/Remove Programs. Remove the **AppDynamics .NET Agent**.

✓ In some cases another process interferes with the .NET Agent uninstallation process by locking the profiler.dll. If uninstallation fails, use a utility such as [Process Explorer](#) to see if a process is using profiler.dll. If so, terminate the process. Otherwise try rebooting the machine. Then retry the uninstallation.

#### Install the new version of the agent

See [Install the .NET Agent](#).

✓ Don't launch AppDynamics Agent Configuration when the installer completes unless you want to make changes to the existing configuration. The installer maintains existing agent configurations.



#### Resume monitoring

1. Start the **AppDynamics.Agent.Coordinator** service.  
`net start AppDynamics.Agent.Coordinator`
2. Start IIS, instrumented Windows services, and instrumented standalone applications.

## Upgrade the .NET Agent from Version 3.7.7 or Earlier

Identify the right upgrade path based upon the method of tier naming and assignment (manual or automatic) and the type of application you instrument:

- If you use manual tier naming and assignment, the installer package upgrades configurations for IIS applications and Windows services.
- If you used automatic tier naming and assignment, run the configuration utility to update configurations.
- If you used standalone applications with 3.7.7 or earlier, follow the steps for standalone applications on [Instrument Windows Services and Standalone Applications](#).

The MSI installer package for the new version installs the updated agent files. After installing, you may need to run the configuration utility to update your configuration and optionally remove legacy configurations. Finally, restart the AppDynamics.Agent.Coordinator service and instrumented applications.

## Uninstall the old version of the .NET Agent

1. Stop IIS and instrumented Windows services.

✓ If you shut down IIS but continue to see active IIS Worker Processes, check the Application Pools pane in the IIS Manager and stop any started application pools.

⚠ Failing to stop instrumented applications before uninstalling the App Agent for .NET may require you to reboot the machine.

2. Stop the **AppDynamics.Agent.Coordinator** service.
3. In the Control Panel, select Add/Remove Programs. Remove the **AppDynamics .NET Agent**.

✓ In some cases another process interferes with the .NET Agent uninstallation process by locking the profiler.dll. If uninstallation fails, use a utility such as [Process Explorer](#) to see if a process is using profiler.dll. If so, terminate the process. Otherwise try rebooting the machine. Then retry the uninstallation.

## Install the new version of the .NET Agent

See [Install the .NET Agent](#).

If you used the following environment variables with the earlier version, the MSI installer migrates the configurations to the new configuration file:

- AppDynamicsAgent\_CallGraphOptions
- AppDynamicsAgent\_DisableAppPools
- AppDynamicsAgent\_EnableInProcesses
- AppDynamicsAgent\_IgnoreCLREnv
- AppDynamicsAgent\_Profiler\_Classes

## Configure the .NET Agent

Configure the agent based on your method of tier generation and assignment: automatic or manual.

i The .NET Agent configuration utility only supports configuration of one Controller per server. To

configure multiple business applications, see [Configure Multiple Business Application Support for .NET](#).

#### **Configure the agent using automatic tier generation and assignment**

If you used automatic configuration with the earlier version of the .NET Agent, run the configuration utility to configure the agent:

1. Use the .NET Agent Configuration utility to reconfigure instrumentation for IIS applications. Choose **Automatic** for the method of tier generation and assignment. See [Configure the .NET Agent](#).
2. Configure instrumentation for Windows services manually. See [Instrument Windows Services and Standalone Applications](#).

#### **Configure the agent using manual tier generation and assignment**

For agents using manual tier generation and assignment, the installer package migrates the configurations for IIS applications and for Windows services to the config.xml. At this stage, the configuration for IIS applications and Windows services is complete.

**i** If you choose not to launch the configuration utility and clean up legacy configurations, restart the **AppDynamics.Agent.Coordinator** service.

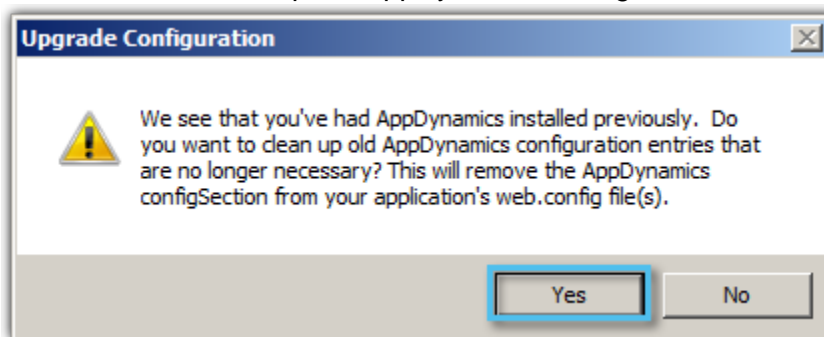
```
net stop AppDynamics.Agent.Coordinator
net start AppDynamics.Agent.Coordinator
```

#### **Clean up legacy configurations**

You can clean up legacy configurations by launching the AppDynamics Agent Configuration utility. When the utility detects agent settings from a previous version, it offers you the option to clean up.

**!** The clean up procedures modifies the web.config files causing an IIS restart.

1. Launch the AppDynamics Agent Configuration utility.
2. Answer **Yes** to clean up old AppDynamics configurations.



The utility removes the following configurations:

- AppDynamics configSections from web.config files for IIS applications and from application.config files for Windows services.
- Environment variables:
  - AppDynamicsAgent\_IgnoreCLREnv
  - AppDynamicsAgent\_CallGraphOptions
  - AppDynamicsAgent\_EnableInProcesses

- AppDynamicsAgent\_DisableAppPools
  - AppDynamicsAgent\_Profiler\_Classes
3. Proceed through the wizard normally.
    - Verify or update the log directory and grant write permissions to it.
    - Verify the controller connection information.
    - Verify or update manual tier assignment.

#### *Resume monitoring*

Start IIS and instrumented Windows services.

## Resolve .NET Agent Installation and Configuration Issues

### On this page:

- [Verify Agent Controller Communication](#)
- [Check Internet Explorer proxy settings](#)
- [Checklist for Resolving .NET Agent Installation Issues](#)
- [Resolve .NET Agent Installation Issues](#)
- [Resolve Configuration Errors](#)
- [Resolve Log Issues](#)

### Related pages:

- [Install the .NET Agent](#)
- [Configure the .NET Agent](#)

### Verify Agent Controller Communication

Use the AppDynamics UI to verify that the agent is able to connect to the Controller.

1. In a browser open:

```
http://<controller-host>:<controller-port>/controller
```

If you can't connect to the controller in Internet Explorer, see [Check Internet Explorer proxy settings](#).

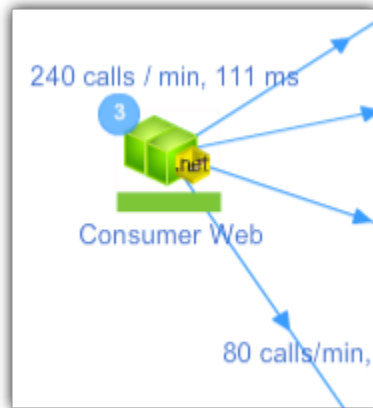
2. Log in to the AppDynamics UI.
3. Select the application to open the Application Dashboard.
4. In the left navigation panel click **Servers -> App Servers** and open the Health tab.

The Health tab lists the tiers, their nodes, and App Agent Status. When an agent successfully reports to the Controller, you see an "up" arrow symbol.

For details see [Connect the Controller and Agents](#).

- When deploying multiple agents for the same tier, determine whether you get the correct number of nodes reporting into the same tier.
- After sending a request to your web application, data should appear on the AppDynamics UI. The agents should be displayed in the Application Flow Map of the Application

Dashboard.



If no data appears after a few minutes:

- Verify that the Agent is writing its log files:  
**Windows Server 2008 and later:** %ProgramData%\AppDynamics\DotNetAgent\Logs\AgentLog.txt  
**Windows Server 2003:** %AllUsersProfile%\Application Data\AppData\DotNetAgent\Logs\AgentLog.txt
- If the log file exists, open it and review it for errors.
- If the log file doesn't exist, run the Windows Event Viewer and see the application messages.
- If there are no AppDynamics event messages, look for messages from the .NET Runtime.

#### Check Internet Explorer proxy settings

**i** This section is only for resolving issues connecting to the AppDynamics Controller when step 1 of "Verify Agent Controller Connection" above fails. To configure the .NET Agent to work through a proxy, see "Controller Element" on [.NET Agent Configuration Properties](#).

Misconfigured proxy settings in Internet Explorer may cause the .NET Agent to fail to connect to the controller. If **Test Controller connection** fails on the Controller Configuration window in the AppDynamics Agent Configuration utility, do the following:

1. Verify the Controller host and port settings are correct.
2. In Internet Explorer, open:

```
http://<controller-host>:<controller-port>/controller
```

3. If the connection also fails in Internet Explorer, check the proxy settings. See [Change IE Proxy Settings](#).
4. Correct or remove any incorrect proxy settings.

### Checklist for Resolving .NET Agent Installation Issues

	Item	Notes
✓	Run the installer as Administrator.	Verify Administrative privileges
✓	Verify that COM+ is enabled.	Verify COM+ services are enabled
✓	Verify that MSDTC is enabled and that it is running under the correct account.	Verify MSDTC is enabled and running under the correct account
✓	Verify permissions for Agent directory.	Verify that the .NET Agent directory has the correct permissions based on the site's application pool identity.
✓	Verify that the Agent is compatible with the Controller.	Agent - Controller Compatibility Matrix
✓	Verify the correct settings in the config.xml: <b>Windows Server 2008 and later:</b> %ProgramData%\AppDynamics\DotNetAgent\Config\config.xml <b>Windows Server 2003:</b> %AllUsersProfile%\Application Data\AppData\DotNetAgent\Config\config.xml	Update the config.xml file to include the .NET Agent Configuration Properties.

### Resolve .NET Agent Installation Issues

If the Agent installation is failing, check the following configurations in your environment:

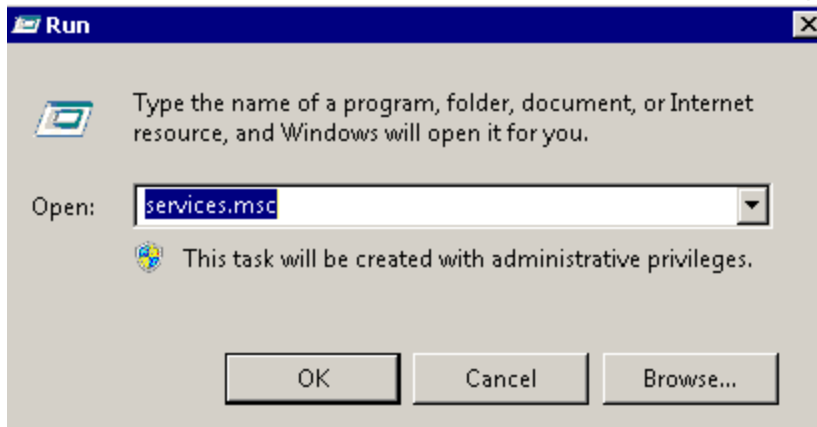
#### Verify administrative privileges

Ensure that you have the administrative privileges when you launch the installer. If the currently user doesn't have sufficient privileges, the installer prompts you for an administrator password.

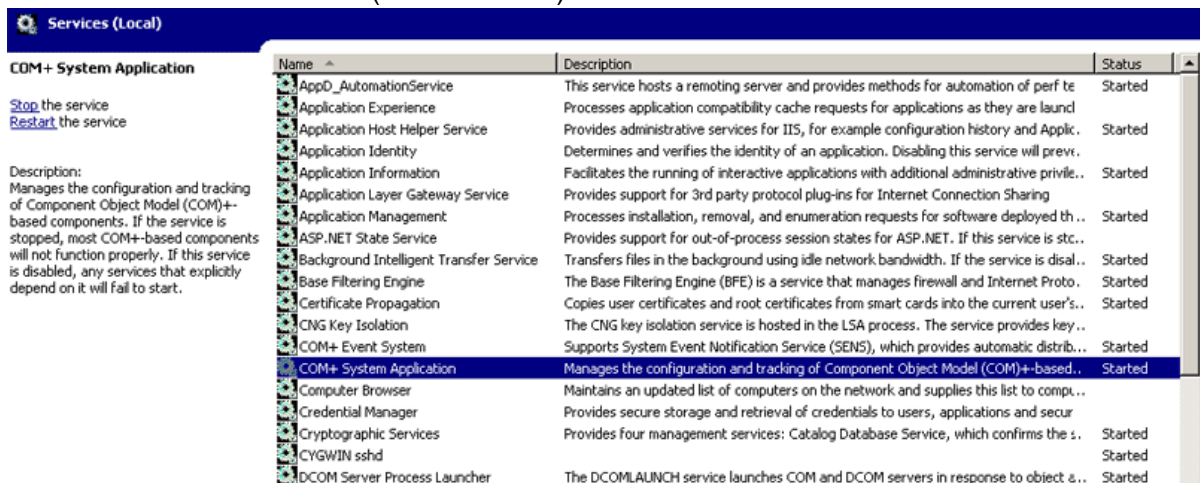
### Verify COM+ services are enabled

If you encounter an error about the Appdynamics.Agent.Coordinator\_service, ensure that the COM+ applications in your system are enabled.

1. From the Windows Start Menu, click **Run**.
2. Enter "services.msc" and click on the "OK" button. Windows opens the Services Console.



3. Enable the COM+ services (if not enabled).



### Verify MSDTC is enabled and running under the correct account

If you encounter an error that MSDTC is not enabled or it is running under the wrong account, launch an elevated command prompt with full administrative privileges and execute the following command:

```
msdtc -install
```

Even if MSTDC is already installed, this command resets the service to run using the "NT Authority\NetworkServices" account.

#### Generate a log for agent installation failures

If installer fails, use the command line utility to launch the installer.

```
msiexec /i $Path_to_the_MSI_File /l*v verbose.log
```

A verbose log for the .NET Agent is created at the same location where you saved the installer file. Send this log to the [AppDynamics Support Team](#).

#### Correct failed installation caused by other APM products

The .NET Agent installation may fail if there are other Application Performance Management (APM) products installed in the same managed environment. Remove the associated "Environment" subkey for certain services for the installed APM products.

##### ***Remove associated "Environment" subkey for W2SVC and WAS services in the registry:***

1. Run Regedit or regedt32.
2. In regedit.exe, locate the following registry keys:  
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\W3SVC`  
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\WAS`
3. Expand the keys.
4. Modify the **Environment** subkey to delete the following values:

```
COMPLUS_ProfAPI_ProfilerCompatibilitySetting=EnableV2Profiler  
COR_ENABLE_PROFILING=1  
COR_PROFILER= {a GUID}
```

5. Restart the services. For more details see [How to restart the W2SVC and WAS services?](#)

#### Resolve Configuration Errors

##### Verify the configuration in the config.xml file

- Ensure that you have correctly configured the config.xml file for the .NET Agent. For more detail, see [.NET Agent Configuration Properties](#).
- If you made manual edits to the config.xml, check the AgentLog.txt and WarnLog.txt for errors. Invalid XML shows in the log as follows:

```
2014-03-13 10:49:18.7199 1232 dllhost 1 1 Error ConfigurationManager Error  
reading the configuration file
```

#### Resolve Log Issues

The .NET Agent writes logs to the following directories:



**Windows Server 2008 and later:** %ProgramData%\AppDynamics\DotNetAgent\Logs

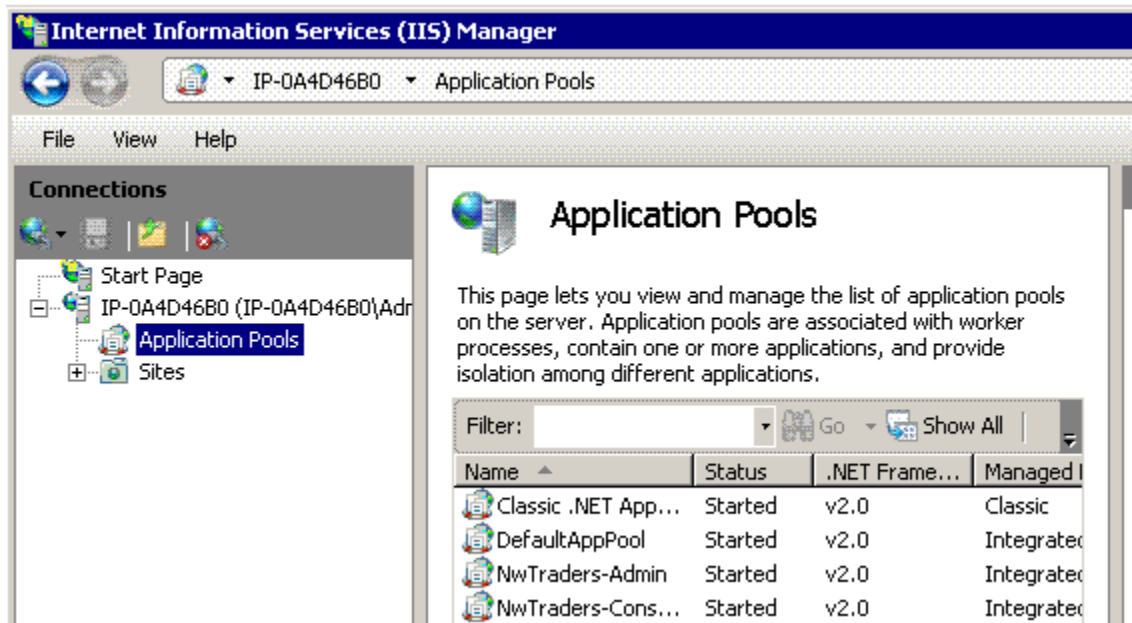
**Windows Server 2003:** %AllUsersProfile%\Application  
Data\AppData\DotNetAgent\Logs

The agent will not generate logs if the agent directory does not have sufficient permissions.

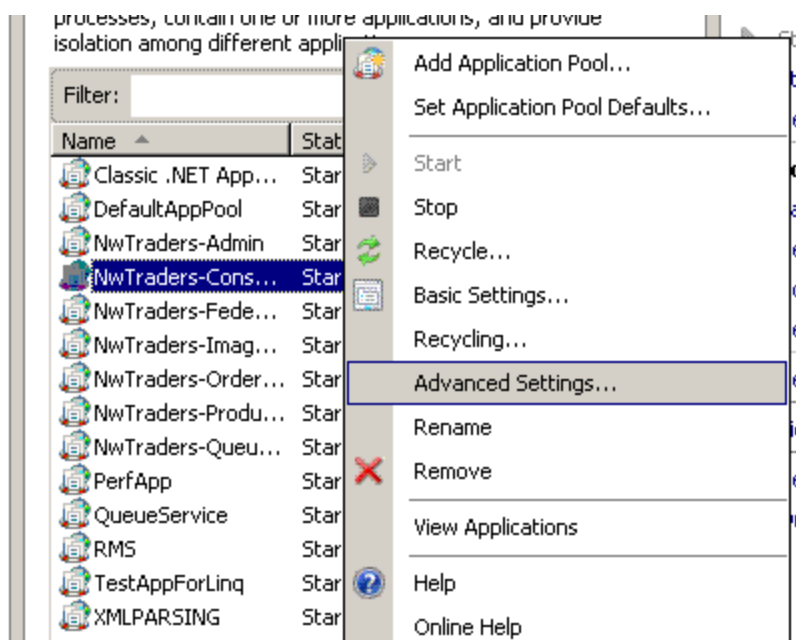
**⚠ IMPORTANT:** If the installation doesn't create the AppDynamics directory, contact [AppDynamics Support Team](#).

Verify that the .NET Agent directory has the correct permissions

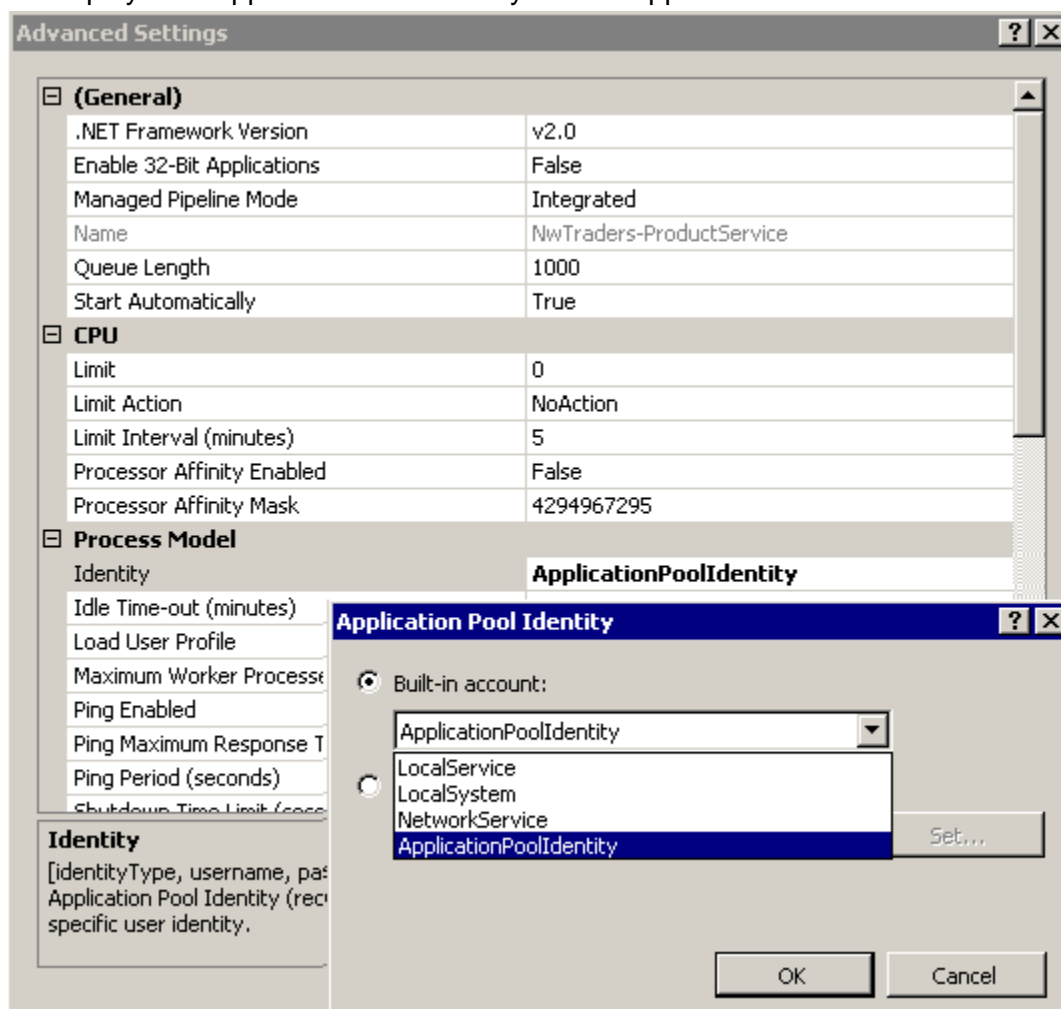
1. Click **IIS -> Application pools**.  
IIS displays the list of application pools for your machine.



2. Right-click on a particular application pool.
3. Click **Advanced Settings**.



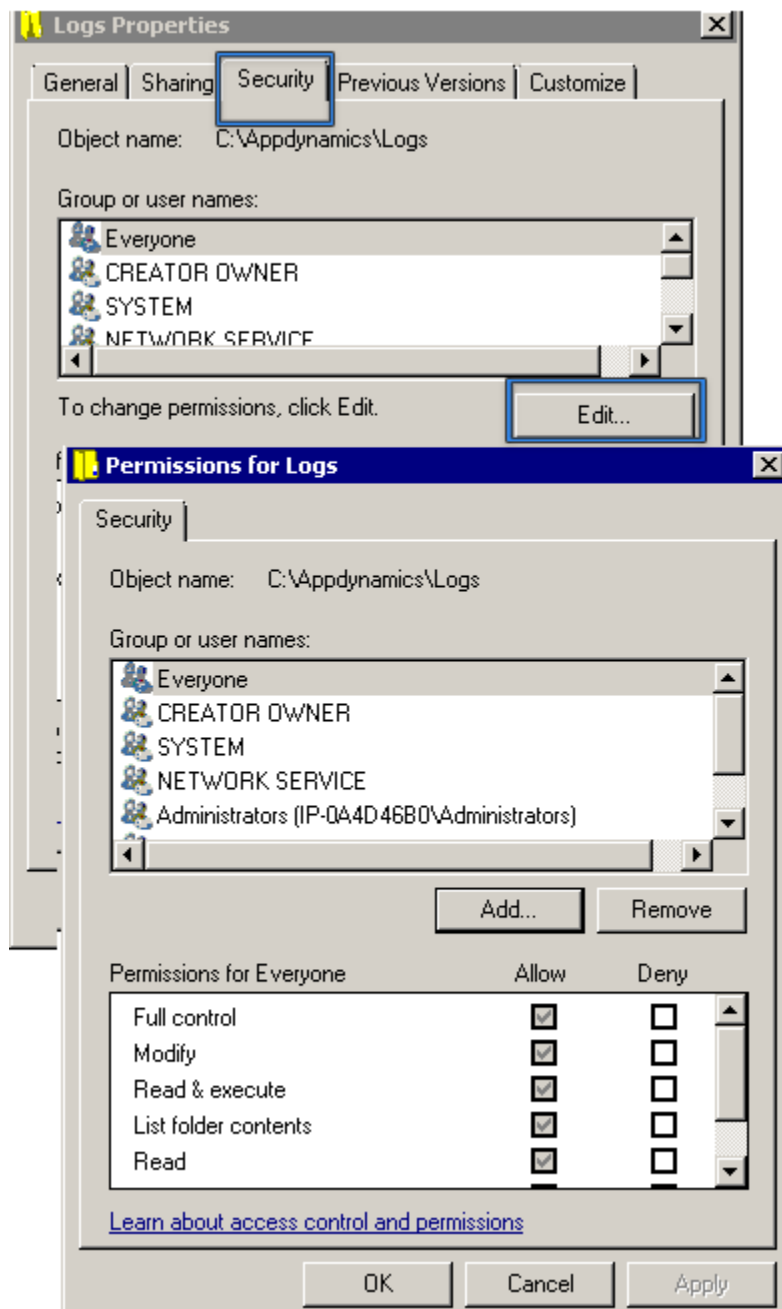
IIS displays the Application Pool Identity for that application.



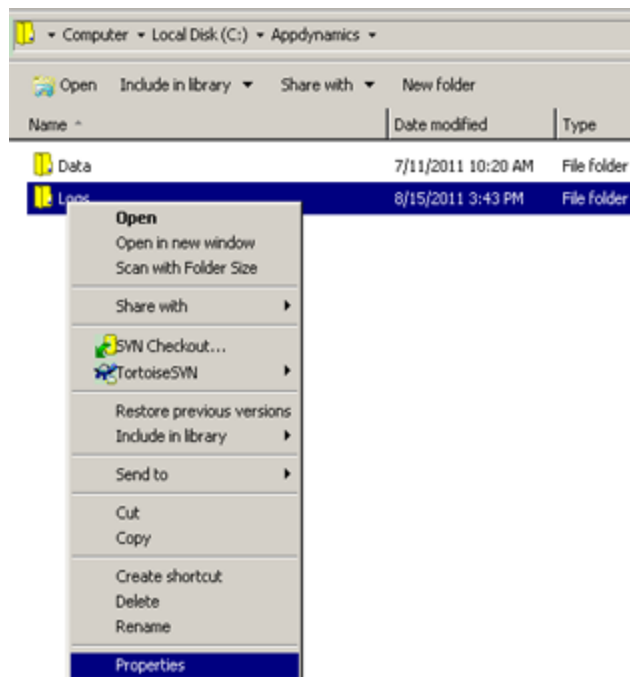
4. Ensure that your Agent Directory also has the same permissions as your site's application

pools.

- Navigate to AppDynamics .NET App Server Agent directory location.
- Right-click on the "logs" directory for the App Server Agent and select **Properties**.

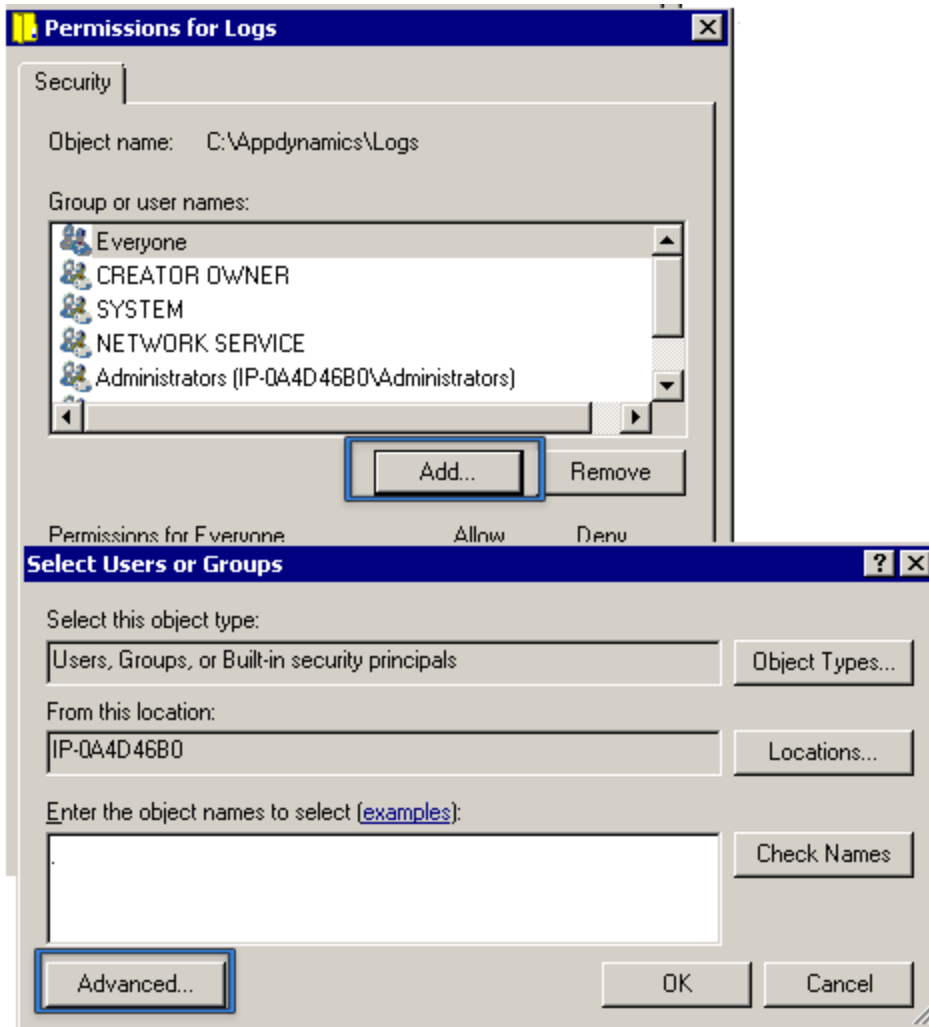


- Click the **Security** tab and verify that the same Application Pool Identity is specified for the .NET Agent directory.



If the Agent's logs directory does not have the required permissions:

1. In the Security tab, click **Edit**.
2. Click **Add** to add new permissions to the Agent directory.
3. Click **Advanced**.



4. Click **Find Now** to find all the users, groups, or built-in security principals on your machine.
5. Select the required group (see the information given below for "**Allowed groups**") from the list and click **OK**.
6. Provide the read and write permissions for the selected user/group/security principal to the Agent directory and click **OK**.
7. Click **Apply**.

#### **Allowed groups for different IIS versions**

For IIS v6.x, following settings are applicable for Application Pool Identities:

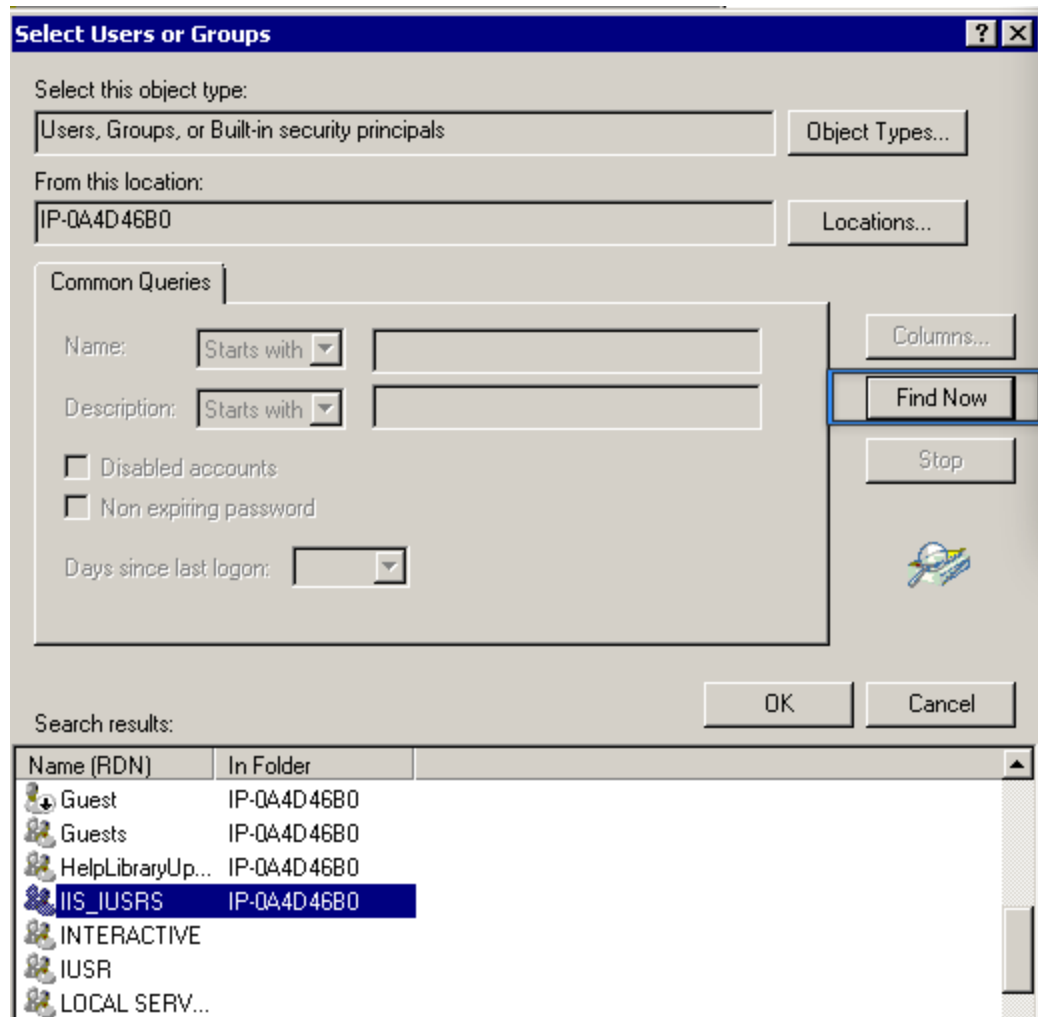
Application Pool Identity	Permission Level
LocalService	LOCAL SERVICE
LocalSystem	SYSTEM
NetworkService	NETWORK SERVICE

Custom Account	Provide the exact name of the account.
----------------	--

For IIS v7.0 and later, following settings are applicable for Application Pool Identities:

Application Pool Identity	Permission Level
LocalService	LOCAL SERVICE
LocalSystem	SYSTEM
NetworkService	NETWORK SERVICE
ApplicationPoolIdentity	Provide the group level permissions for IIS_IUSRS Group (see the screenshot given below).
Custom Account	Provide the exact name of the account.

For example, if your application has the identity "ApplicationPoolIdentity", you must provide the permissions for "IIS\_IUSRS" group to your Agent's directory.



## Uninstall the .NET Agent

### Related pages:

- [Upgrade the .NET Agent](#)
- [Platform Release Notes](#)

This topic describes how to do a complete uninstall of the .NET Agent.

**⚠** Do not follow these instructions if you are doing an upgrade. If you want to upgrade to a new version see [Upgrade the .NET Agent](#).

### Completely uninstall the .NET Agent

1. Stop IIS, instrumented Windows services, and instrumented standalone applications.
  - ⚠** Failing to stop instrumented applications before uninstalling the agent requires you to reboot the machine to complete the uninstall.
2. Stop the **AppDynamics.Agent.Coordinator** service.
3. In the Control Panel, select Add/Remove Programs. Remove the **AppDynamics .NET**

### Agent.

4. The uninstall procedure does not remove configuration files. Delete the configuration directory:

**Windows Server 2008 and later:** %ProgramData%\AppDynamics

**Windows Server 2003:** %AllUsersProfile%\Application Data\AppDataDynamics

5. Verify the uninstall deleted the AppDynamics executables directory. If not, delete it manually:

**Windows Server 2003 and later:** %ProgramFiles%\AppDynamics.

6. Restart IIS, Windows services, and standalone applications.

✓ In some cases another process interferes with the .NET Agent uninstallation process by locking the profiler.dll. If uninstallation fails, use a utility such as [Process Explorer](#) to see if a process is using profiler.dll. If so, terminate the process. Otherwise try rebooting the machine. Then retry the uninstallation.

## Install AppDynamics for Windows Azure with NuGet

### On this page:

- [Requirements](#)
- [Register for an AppDynamics for Windows Azure Account](#)
- [Add the .NET Agent to Your Azure Solution](#)
- [Update Agent Configuration](#)
- [Customize Tier Names for Azure Roles](#)

### Related pages:

- [AppDynamics Essentials](#)
- [APM for .NET](#)

Monitor your Azure cloud solutions with the AppDynamics for Windows Azure NuGet package. Sign up for the AppDynamics add-on in the Windows Azure portal, then enable the AppDynamics agent in your Visual Studio solution.

### Requirements

The AppDynamics for Windows Azure Nuget package requires the following:

- Visual Studio 2010 or later
- A Visual Studio solution to monitor
  - Visual Studio must have the following permissions to the solution:
    - **Read** and **Write** permissions to each **project directory**
    - **Read** and **Write** permissions to each **Visual Studio .NET C# Project** (\*.csproj) file
    - **Read** and **Write** permissions to the **Service Definition** (ServiceDefinition.csdef) file
- Windows Azure SDK
- Windows Azure account
- If the Visual Studio Standalone Profiler is installed, you must uninstall it and remove related



registry entries.

### Register for an AppDynamics for Windows Azure Account

If you haven't registered for an AppDynamics account, see [Register for AppDynamics for Windows Azure](#).

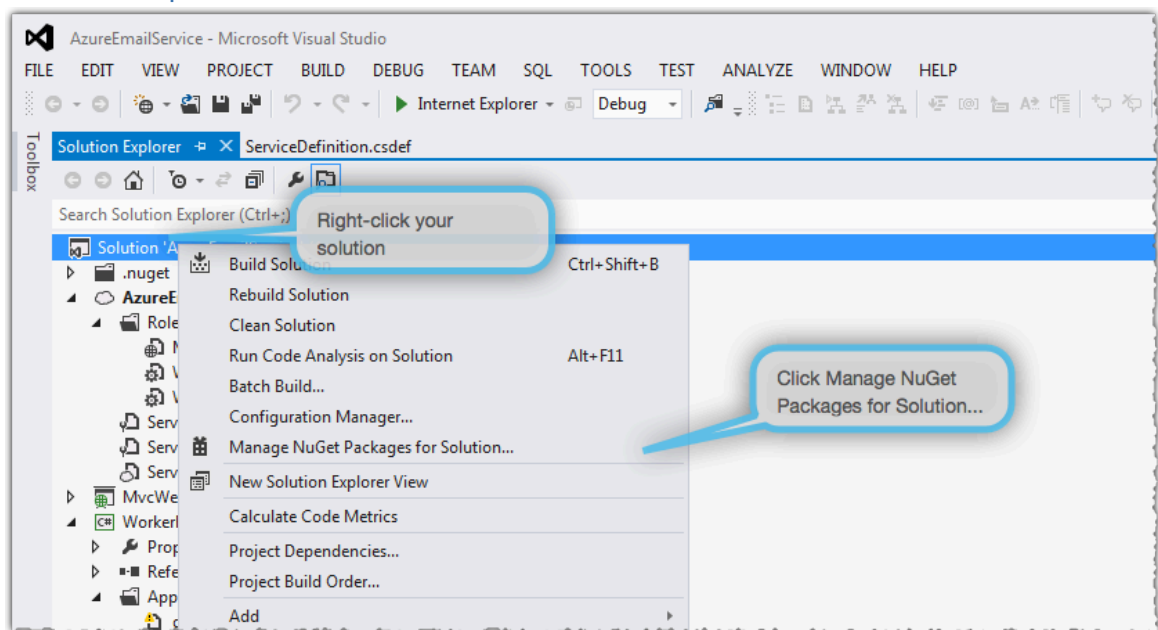
Once you've registered, we will send you a Welcome email with your URL and credentials to connect to the AppDynamics Controller. If you already have AppDynamics credentials from another product, you can sign in using them.

### Add the .NET Agent to Your Azure Solution

Use the NuGet package manager to add the .NET Agent to your Azure solution in Visual Studio.

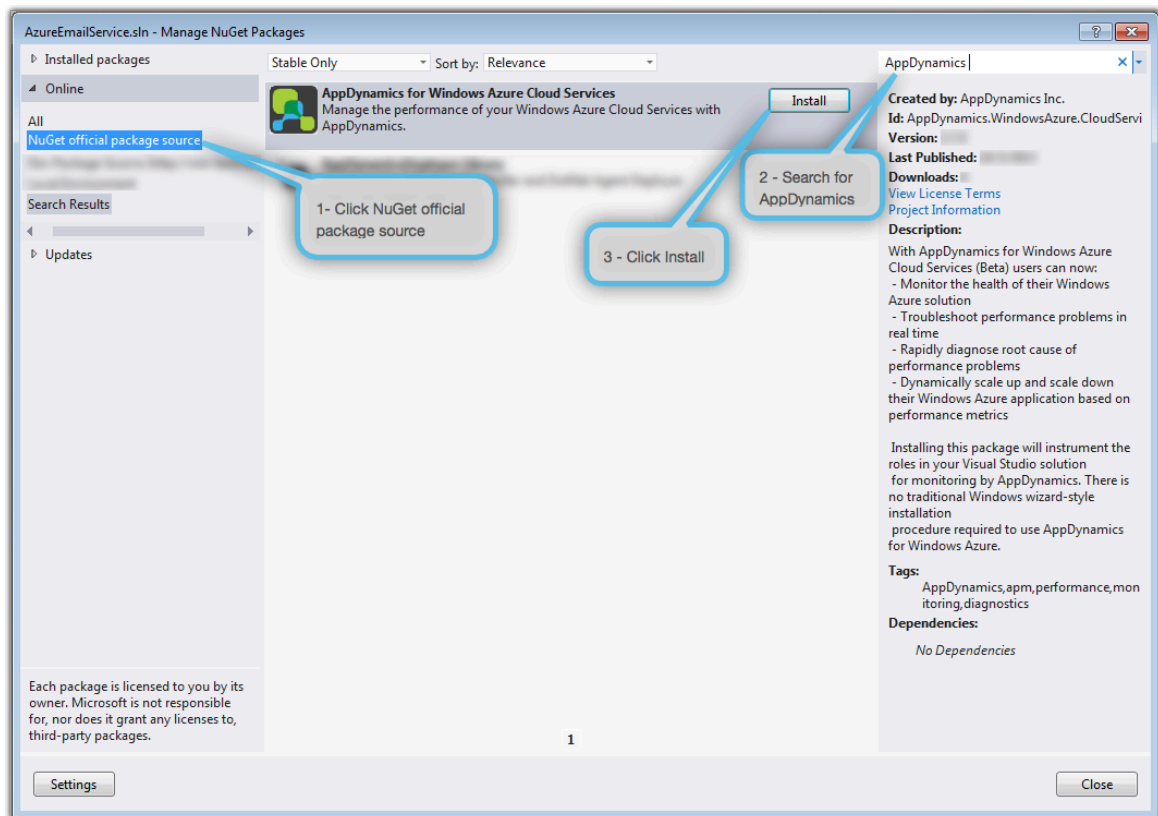
1. In Visual Studio, right-click your solution name and chose **Manage NuGet Packages for Solution**.

▼ [Show this step in Visual Studio](#)

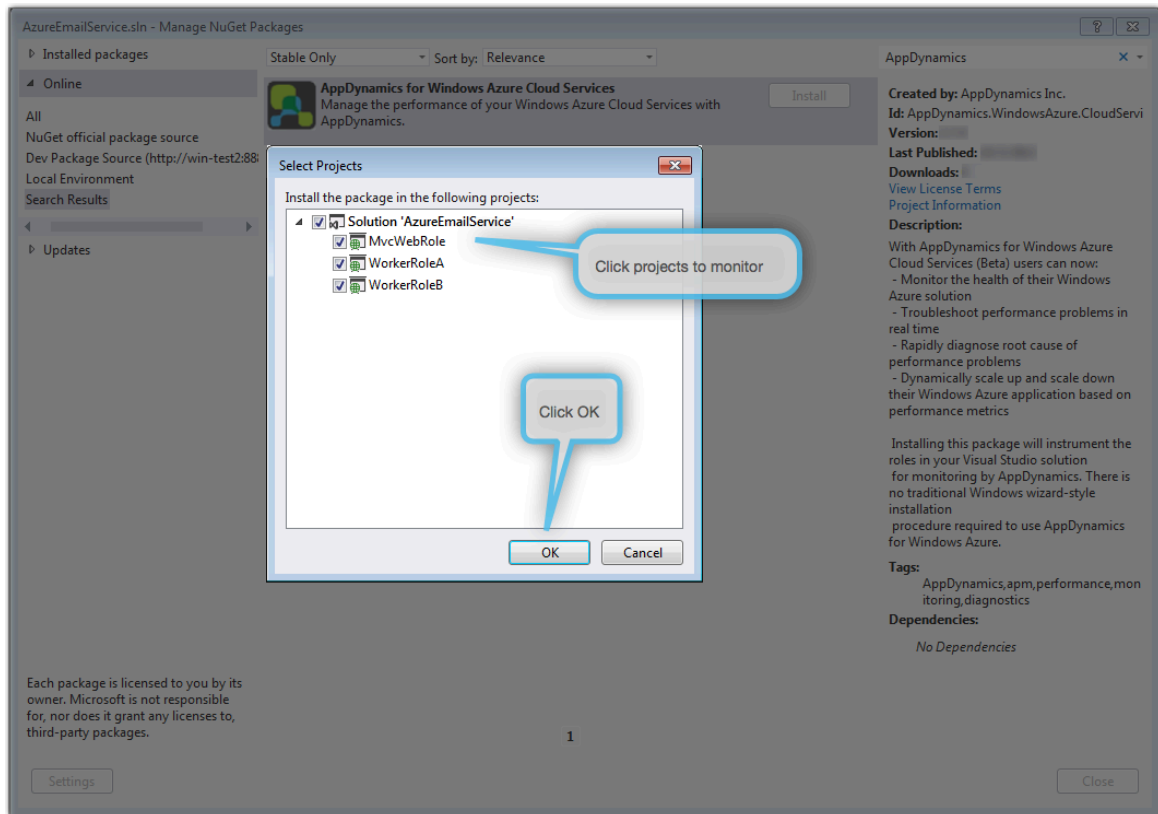


2. Click the **NuGet official package source** in the left menu and type "AppDynamics" in the search bar in the upper right.
3. Select **AppDynamics .NET Agent for Azure** and click **Install**.

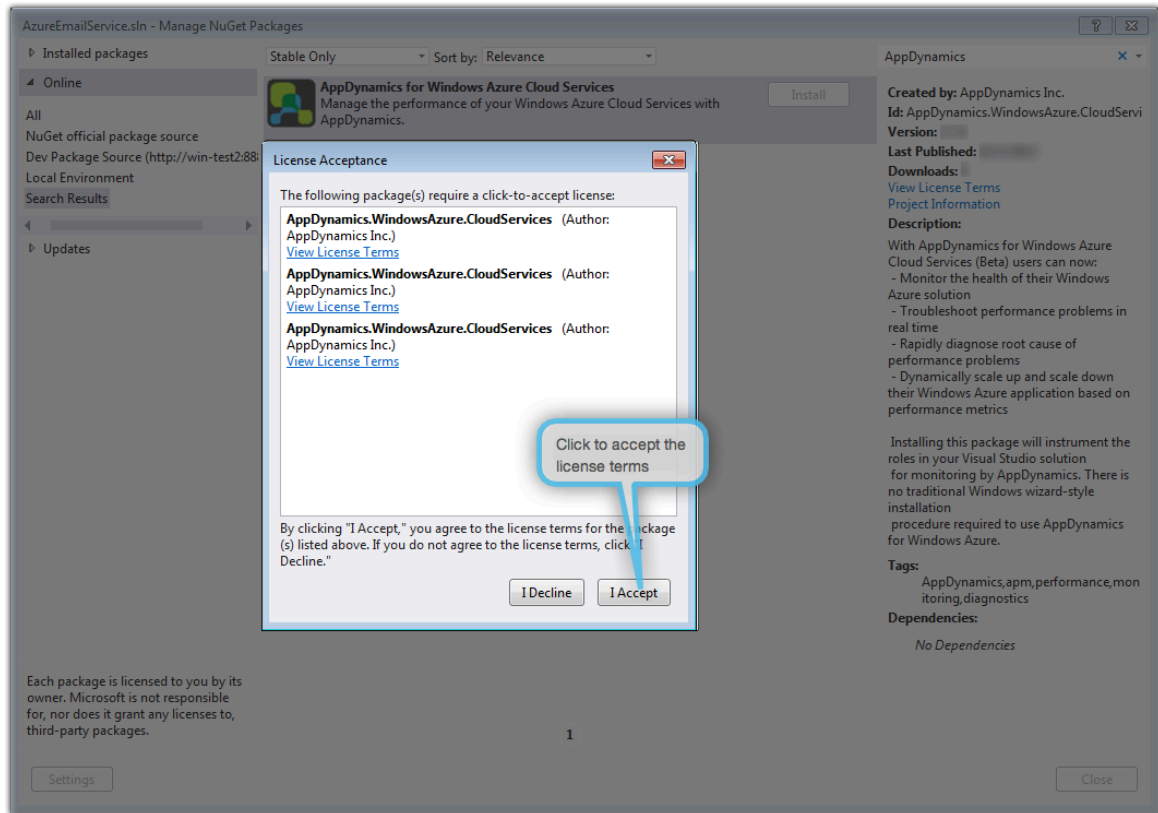
▼ [Show this step in Visual Studio](#)



4. Click the projects you want to monitor, then click **OK**.
  - ✓ [Show this step in Visual Studio](#)



5. Click **I Accept** to agree to the license terms.
  - ▼ [Show this step in Visual Studio](#)

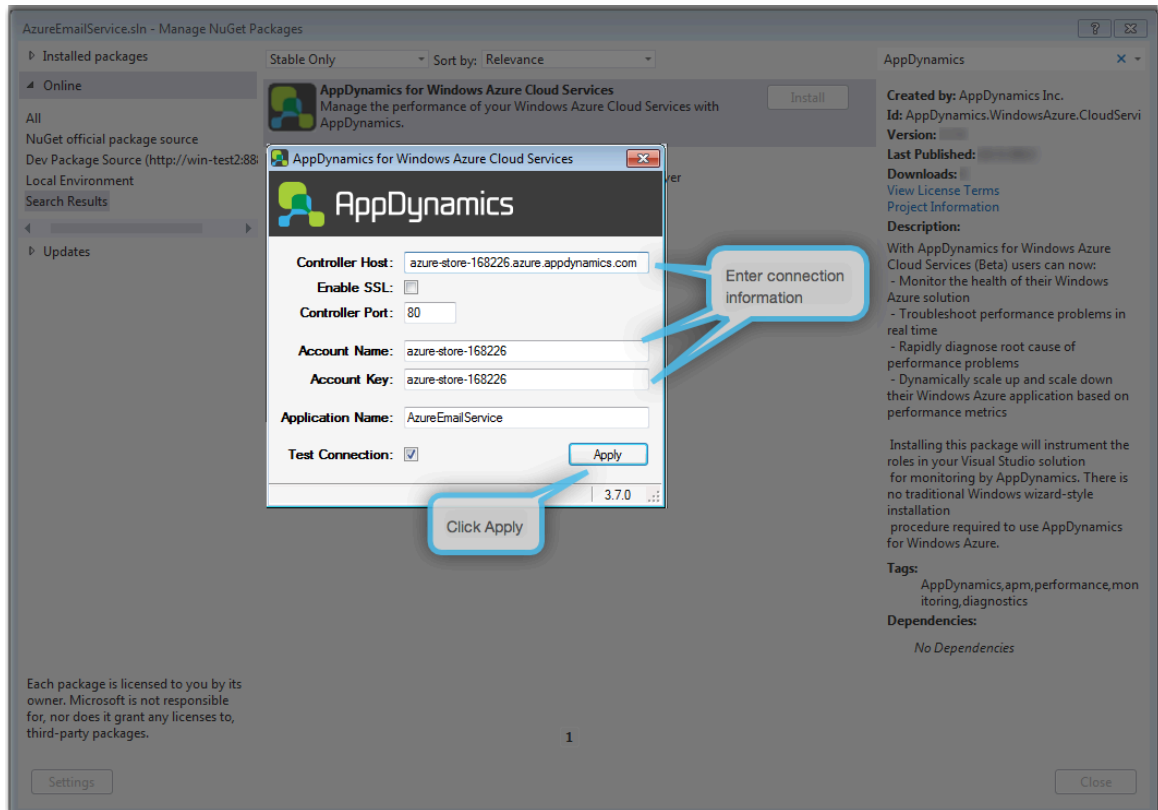


## 6. Enter the connection information from your Welcome email:

Controller Host  
Controller Port  
Account Name  
Account Key

AppDynamics automatically populates **Application Name** with the name of your Windows Azure solution.

✓ [Show this step in Visual Studio](#)



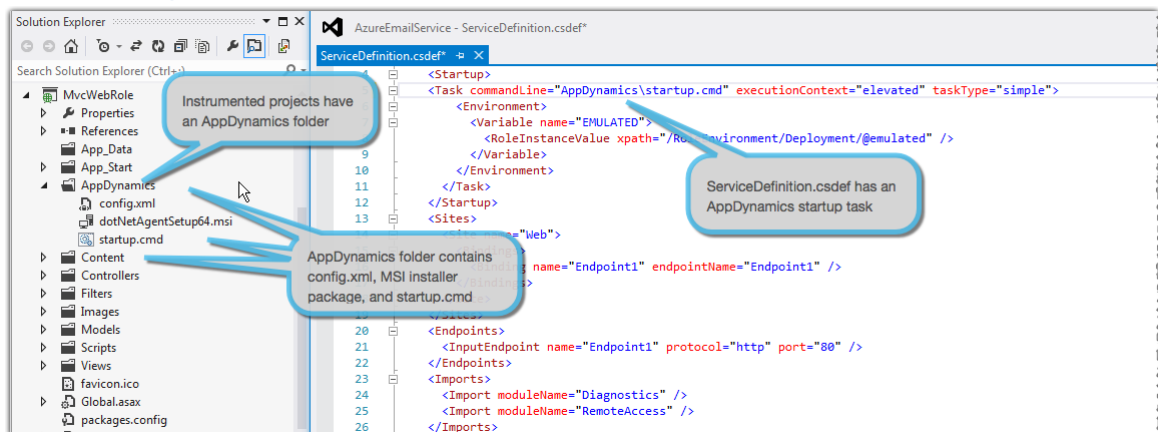
✓ You can also find your Account name and key in the AppDynamics Controller under **Settings -> License**.

7. **Close** the Manage NuGet Packages window.

8. Verify the following:

- Instrumented projects have an AppDynamics folder.
- The AppDynamics folders contain an MSI installer package, a config.xml, and a setup.cmd file.
- The ServiceDefinition.csdef has an AppDynamics startup task.

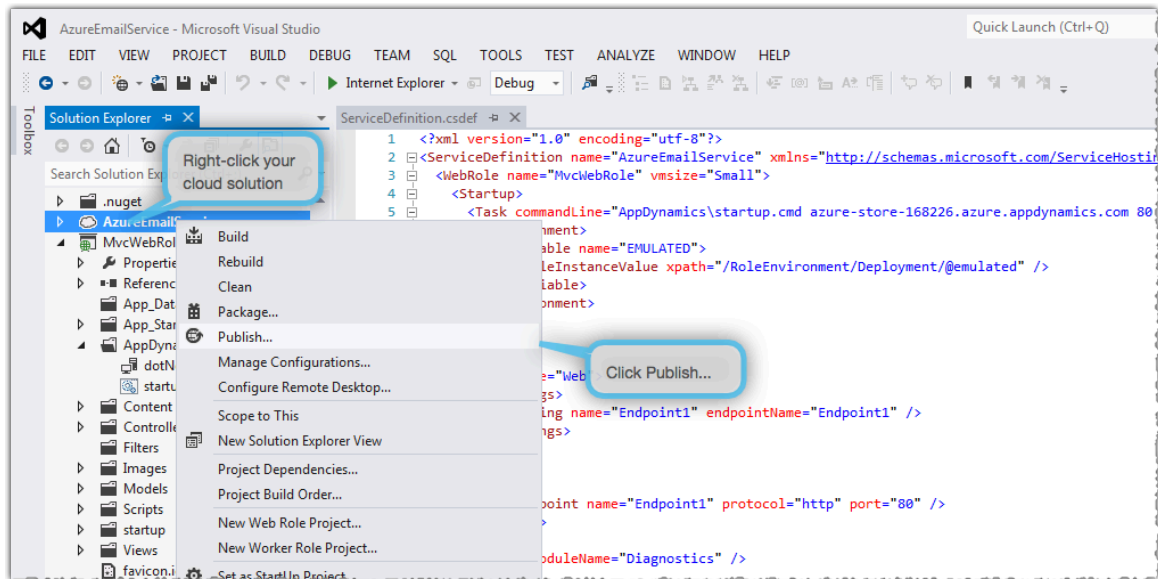
✓ Show this step in Visual Studio



**i** By Default the .NET Agent names Azure tiers for the role name. For example "AzureEmailService". If you want to customize the tier name, follow the steps under [Customize Tier Names for Azure Roles](#) before you publish the Windows Azure cloud solution.

9. Right-click the Windows Azure solution and click **Publish**.

▼ [Show this step in Visual Studio](#)



Once you've successfully published your project, you're ready to log on to the Controller and begin monitoring your solution!

### Update Agent Configuration

*New in 4.0.1*, if you have already published your solution, you can update the .NET Agent configuration for the currently installed version without upgrading the agent.

1. Edit the config.xml file to make configuration changes.
2. Edit the startup.cmd file.
3. Set the CONFIGUPDATE variable to "true".

```
SET CONFIGUPDATE=true
```

4. Publish the Windows Azure cloud solution

### Customize Tier Names for Azure Roles

By default the .NET Agent names tiers for Windows Azure role names. For example: MvcWebRole. If you want to customize the tier name for a role, edit the config.xml to specify the tier name using a regular expression for the IIS site.

1. Edit the config.xml in Visual Studio.
2. Replace the IIS element with the following code block:

```

<IIS>
  </applications>
  <!-- Configure IIS tier names with a regular expresson. -->
  <application path="/" site="" site-regex="true">
    <tier name="" />
  </application>
</applications>
</IIS>

```

3. Enter a regular expression for the Application site attribute. For example for a role named "MvcWebRole":

```
<application path="/" site="MvcWebRole.*" site-regex="true">
```

4. Specify a tier name. For example:

```
<tier name="My Azure Tier" />
```

5. Save changes to the config.xml and repeat the process for other roles.
6. Publish the Windows Azure cloud solution.

**i** If you previously published your solution using automatic tier naming, you may need to delete your deployment for the .NET Agent to use the new tier naming scheme.

#### Sample Azure IIS Application Configuration

```

<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <controller host="nativdemocon1.cloudapp.net" port="8090" ssl="false">
    <application name="AzureEmailService" />
    <account name="customer1" password="SJ5b2m7d1$354" />
  </controller>
  <machine-agent />
  <app-agents azure="true" azure-role-name="" azure-role-instance-id="">
    <IIS>
      </applications>
      <!-- Configure IIS tier names with a regular expresson. -->
      <application path="/" site="MvcWebRole.*" site-regex="true">
        <tier name="My Azure Tier" />
      </application>
    </applications>
  </IIS>
  <standalone-applications>
    <standalone-application executable="WaWorkerHost.exe">
      <tier name="" />
    </standalone-application>
  </standalone-applications>
</app-agents>
</appdynamics-agent>

```

## Register for AppDynamics for Windows Azure

### On this page:

- [What is AppDynamics?](#)
- [Register for an AppDynamics for Windows Azure Account](#)
- [Register for an AppDynamics for Windows Azure Account From Windows Azure Marketplace](#)
- [Next Steps](#)

### Related pages:

- [Install AppDynamics for Windows Azure with NuGet](#)
- [Manually Install the .NET Agent on Windows Azure](#)
- [AppDynamics Essentials](#)
- [APM for .NET](#)

Monitor your Azure cloud solutions with the AppDynamics for Windows Azure NuGet package. Sign up for the AppDynamics add-on in the Windows Azure portal, then enable the AppDynamics agent in your Visual Studio solution.

### What is AppDynamics?

AppDynamics is an application performance monitoring solution that helps you:

- Identify problems, such as slow and stalled user requests and errors, in a production environment
- Troubleshoot and isolate the root cause of such problems

There are two components in AppDynamics for Windows Azure:

**App Agent:** The .NET Agent collects data from your servers. You run a separate agent on every role instance that you want to monitor. You install the agent as part of the NuGet package.

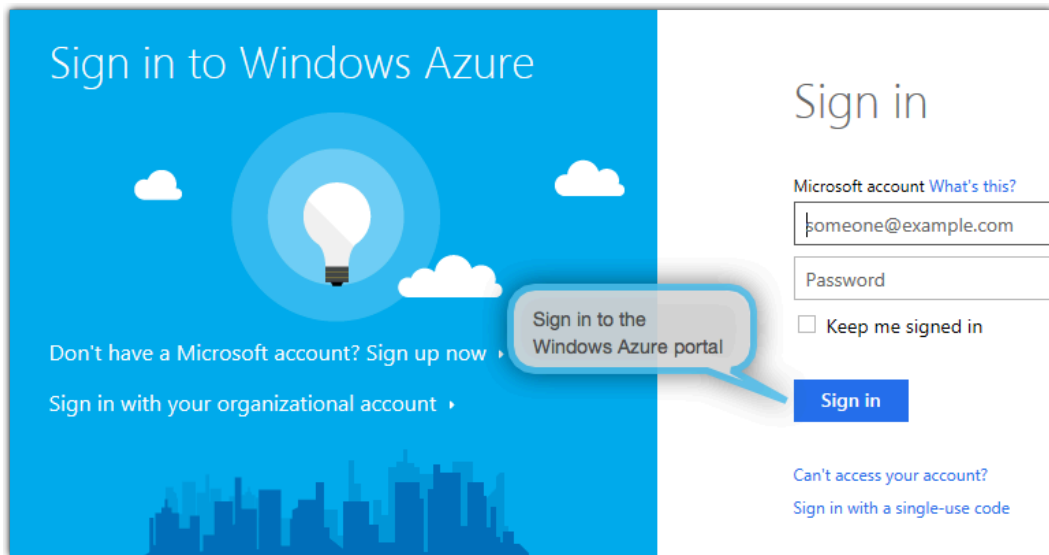
**AppDynamics Controller:** The agent sends information to an AppDynamics Controller hosted service. Using a browser-based console, you log on to the Controller to monitor, analyze and troubleshoot your application.

### Register for an AppDynamics for Windows Azure Account

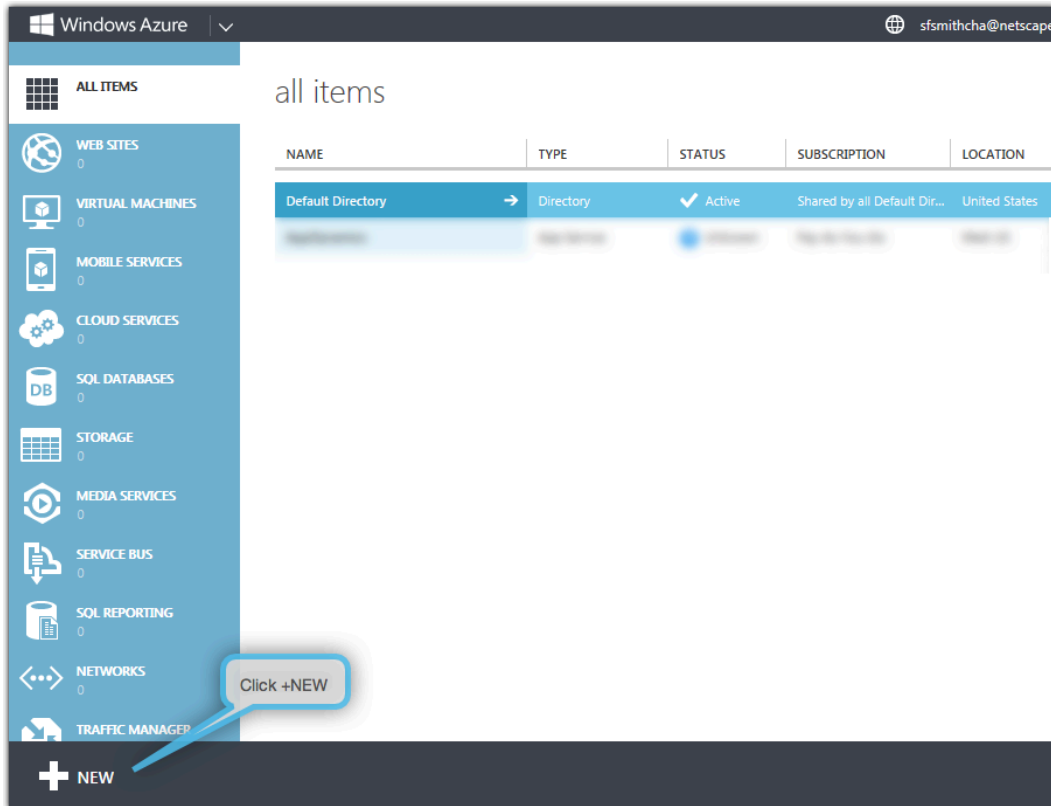
Use the Windows Azure portal to sign up for AppDynamics for Windows Azure.

1. Log on to the Windows Azure portal at <https://manage.windowsazure.com>.

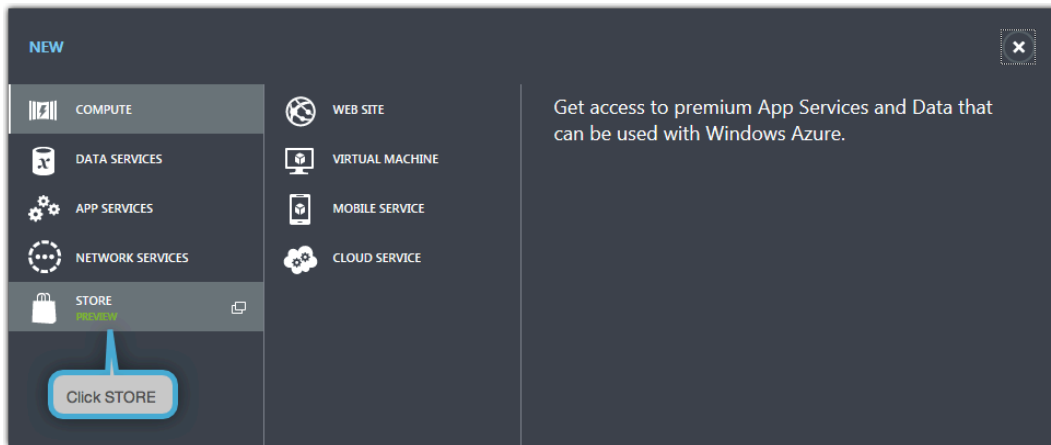




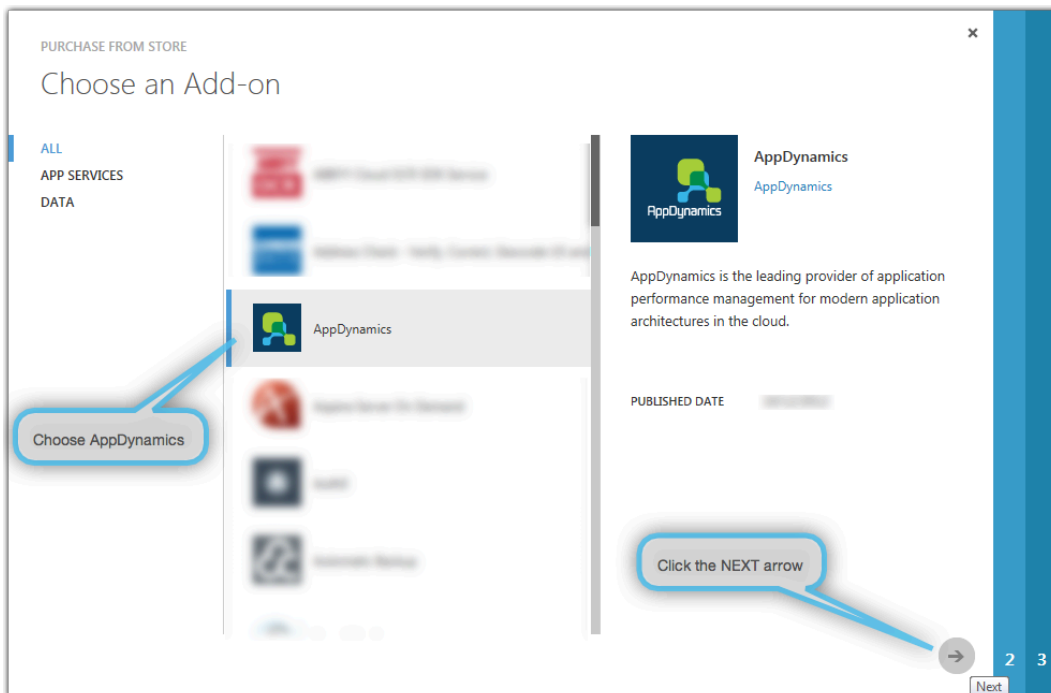
2. Click **+NEW** at the bottom left corner of the portal.



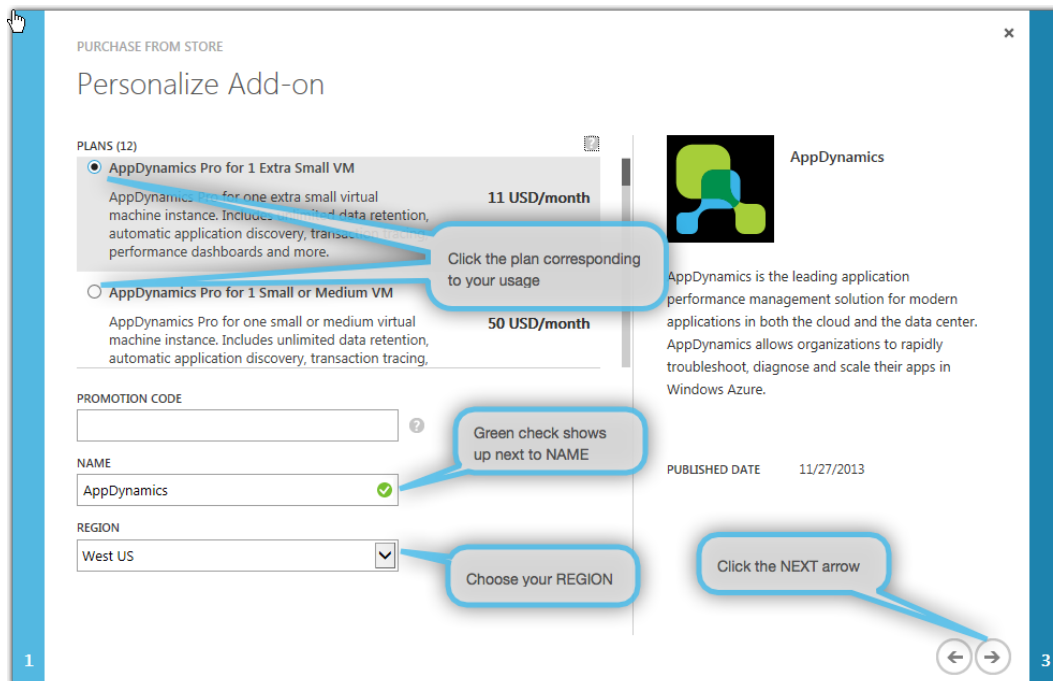
3. In the left menu, click **STORE**.



4. Click **AppDynamics** in the list of services and click the **NEXT** arrow.



5. Click the plan corresponding to the size (Extra Small, Small, Medium, etc.) and number of virtual machines where you will install AppDynamics monitoring agents.
6. Select your **REGION** and click the **NEXT** arrow.
7. After the form validates the **NAME** field and displays a green check, click the **NEXT** arrow.



8. Review your purchase and click the **CHECK** to purchase.

You've successfully signed up for AppDynamics for Windows Azure!

#### Register for an AppDynamics for Windows Azure Account From Windows Azure Marketplace

1. Click **Try Free** or **Sign Up** for AppDynamics on the Windows Azure Marketplace at <https://dynamarket.azure.com/application/f9949031-b8b9-4da5-b500-c615f3f2a7cd>.

If you choose **Sign Up**, you receive a free version of AppDynamics Pro for Windows Azure with full functionality, which downgrades after 30 days to a free version of AppDynamics Lite for Windows Azure with limited functionality. You do not need to provide a credit card for this option. You can upgrade to AppDynamics Pro for Windows Azure at any time.

If you choose **Try Free**, you receive a free version of AppDynamics Pro for Windows Azure with full functionality. You need to provide a credit card for this option. After 30 days your credit account will be charged for continued use of AppDynamics Pro for Windows Azure, unless you cancel your subscription.

You need one agent license for each role instance that you wish to monitor. For example, a site running 2 Web role instances and 2 Worker role instances requires 4 agent licenses.

2. On the registration page, provide your user information, a password, email address, company name, and the name of the application you are monitoring as you will publish it with Windows Azure.
3. Click **Register Now**.

#### Next Steps

AppDynamics will send you a Welcome email with your URL and credentials to connect to the AppDynamics Controller. Then you can use NuGet to add the .NET Agent to your Windows Azure solution using NuGet. See [Install AppDynamics for Windows Azure with NuGet](#).

## Administer the .NET Agent

### On this page:

- [Where to Configure Agent Properties](#)
- [Customize .NET Agent Behavior in config.xml](#)
- [Sample Minimal controller-info.xml](#)
- [Agent Log Files](#)

### Related pages:

- [.NET Agent Configuration Properties](#)

The .NET Agent uses a single configuration file to control agent behavior and .NET Machine Agent behavior. The configuration file specifies Controller connectivity, .NET Machine Agent operations, and app agent functionality for IIS applications, Windows services, and standalone applications. The single configuration file lets you:

- Maintain agent configurations separately from web.config files.
- Enable instrumentation of Windows services and standalone applications without environment variables.
- Control agent behavior for specific applications with hierarchical configuration.

## Where to Configure Agent Properties

Configure the agent properties in the config.xml file in the agent Config directory. To edit the config.xml, you must launch the editor as an administrator. When you run the .NET Agent Configuration Utility, it writes the config.xml to one of the following locations:

### Windows Server 2008 and later

`%ProgramData%\AppDynamics\DotNetAgent\Config\config.xml`

### Windows Server 2003


`%AllUsersProfile%\Application  
Data\AppDataDynamics\DotNetAgent\Config\config.xml`

### Windows Azure

For Windows Azure deployments, the .NET Agent NuGet package contains the config.xml file. See [Install AppDynamics for Windows Azure with NuGet](#).

Sample config.xml files install to the following location:

`%ProgramFiles%\AppDynamics\AppDataDynamics .NET Agent\SampleConfigurations`

 After you edit the config.xml, you must restart the AppDynamics.Agent.Coordinator service. Then restart IIS, your Windows service, or standalone application for your instrumentation

changes to take effect.

## Customize .NET Agent Behavior in config.xml

Some .NET Agent configurations require that you edit the config.xml:

1. Shut down the AppDynamics.Agent.Coordinator service.
2. Edit the config.xml file as an administrator.
3. Modify the xml according to the configuration instructions.
4. Save the config.xml.
5. Start the AppDynamics.Agent.Coordinator service.
6. In some cases you may need to restart IIS, instrumented Windows services, or instrumented Standalone applications. See individual .NET Agent administration topics.

## Sample Minimal controller-info.xml

The most basic configuration demonstrates the required sections for agent configuration. This sample instruments all IIS applications using the automatic element (<automatic />). No Windows services or standalone applications are instrumented.

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090" ssl=false">
    <application name="MyDotNetApplication" />
  </controller>
  <machine-agent />
  <app-agents>
    <IIS>
      <automatic />
    </IIS>
  </app-agents>
</appdynamics-agent>
```

## Agent Log Files

The configuration file that controls log files for the .NET Agent is located at:

```
%ProgramFiles%\AppDynamics\AppDynamics .NET Agent\AppDynamicsAgentLog.config
```

The configuration file uses [NLog configuration format](#).

## .NET Agent Directory Structure

**On this page:**

- [Executables](#)
- [Log files](#)
- [Agent Configuration Files](#)
- [Coordinator Service Configuration Files](#)
- [About Windows Server System Directory Variables](#)

**Related pages:**

- [Install the .NET Agent](#)
- [Administer the .NET Agent](#)

## Executables

The .NET Agent executables and supporting files install to the **AppDynamics .NET Agent** directory:

### Windows Server 2003 and later

```
%ProgramFiles%\AppDynamics\AppDynamics .NET Agent
```

## Log files

The **Logs** directory defaults to the following location:

### Windows Server 2008 and later

```
%ProgramData%\AppDynamics\DotNetAgent\Logs
```

### Windows Server 2003

```
%AllUsersProfile%\Application Data\AppDynamics\DotNetAgent\Logs
```

## Agent Configuration Files

The unified configuration file config.xml installs to the **Config** directory:

### Windows Server 2008 and later

```
%ProgramData%\AppDynamics\DotNetAgent\Config
```

### Windows Server 2003

```
%AllUsersProfile%\Application Data\AppDynamics\DotNetAgent\Config
```

## Coordinator Service Configuration Files

The AppDynamics Agent Coordinator service writes configuration files from the Controller to the **Data** directory:

### Windows Server 2008 and later

```
%ProgramData%\AppDynamics\DotNetAgent\Data
```

### Windows Server 2003

```
%AllUsersProfile%\Application Data\AppData\DotNetAgent\Data
```

### About Windows Server System Directory Variables

**%ProgramFiles%** is located at **<system drive>\Program Files** for Windows Server 2003 and later.

**%ProgramData%** is located at **<system drive>\Program Data** for Windows Server 2008 and later.

**%AllUsersProfile%** is located at **<system drive>\Documents and Settings\All Users** for Windows 2003.

## .NET Agent Configuration Properties

### On this page:

- [AppDynamics Agent Element](#)
- [Controller Element](#)
- [Machine Agent Element](#)
- [App Agents Element](#)
- [App Agents - IIS Element](#)
- [App Agents - Standalone Applications Element](#)

### Related pages:

- [Administer the .NET Agent](#)
- [Configure the .NET Agent](#)
- [Name .NET Tiers](#)
- [Instrument Windows Services and Standalone Applications](#)

This topic is a reference for the configuration properties for the .Net Agent config.xml file. For information about how to edit the file and apply your changes, see [Administer the .NET Agent](#).

## AppDynamics Agent Element

The Appdyanmics Agent element is the root container element for configurations in the config.xml.

**Required Element:** <appdynamics-agent

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

## Controller Element

The Controller element is a child of the AppDynamics Agent element. It specifies the connection information for the AppDynamics Pro Controller.

**i** The .NET Agent configuration utility only supports configuration of one Controller per server.

**Required Element:** `<controller host="mycontroller.mycompany.com" port="8090" ssl="false" enable_tls12="false" high_availability="false">`

### *Controller host attribute*

The Controller host attribute indicates the host name or the IP address of the AppDynamics Controller. For an on-premise Controller, use the value for Application Server Host Name you provided when you installed the Controller. If you use the AppDynamics SaaS Controller, see the Welcome email from AppDynamics.

**Type:** String

**Default:** None

**Required:** Yes

### *Controller port attribute*

The Controller port attribute specifies the HTTP(S) port of the AppDynamics Controller. If the Controller ssl attribute is set to true, specify the HTTPS port of the Controller; otherwise specify the HTTP port.

**Type:** Positive Integer

**Default:** 8090

For On-premise installations, the defaults are port 8090 for HTTP and port 8181 for HTTPS.

For the SaaS Controller, use port 80 for HTTP or port 443 for HTTPS.

**Required:** Yes

### *Controller ssl attribute*

To enable encryption over SSL between the agent and the Controller, set the Controller ssl attribute to "true".

**Type:** Boolean

**Default:** false

**Required:** No

### *Controller enable TLS 1.2 attribute*

**4.0 to 4.0.3:** When you enable SSL, the agent encrypts communications with the Controller using TLS 1.2 by default. If you are unwilling or unable to use TLS 1.2, set the Controller enable TLS 1.2 attribute to "false".

**New in 4.0.4:** When you enable SSL, the agent secures communication to the Controller using the



protocols set for `ServicePointManager.SecurityProtocol` in your application. Set the Controller enable TLS 1.2 attribute to "true" to add TLS 1.2 as the first option in the list of protocols. This affects all secure communications from your application, not just requests to the AppDynamics Controller.

**Type:** Boolean

**Default:** 4.0 to 4.0.3: true; 4.0.4 and later: false

**Required:** No

#### ***Controller high availability attribute***

If you have your controller set up for high availability, set the Controller high availability attribute to "true".

**Type:** Boolean


**Default:** false

**Required:** No

### **Controller Application Element**

The Controller Application element is a child of the Controller element. It indicates the name of the logical business application you see in the Controller interface.

The .NET Agent configuration utility only supports one business application per server.

- Use tiers to organize different applications you instrument on a single server.
- Or manually configure support for multiple business applications, see [Configure Multiple Business Application Support for .NET](#).
  -  Multiple application support has requires different configuration elements, including the `<applications>` container element.

**Required Element:** `<application name="MyDotNetApplication"/>`

#### ***Application name attribute***

Set the application name attribute to the business application you use in the controller. If the application name does not exist, the Controller will create it when the agent registers. All instrumented applications in the config.xml register with the same business application in the Controller. See [AppDynamics Concepts](#).

**Type:** String

**Default:** None

**Required:** Yes

 You specify a Tier for individual applications in the [App Agents Element](#).

### **Account Element**

The Account element is a child of the Controller element. If the AppDynamics Controller runs in multi-tenant mode or if you use the AppDynamics SaaS Controller, specify the account name and account access key for the agent to authenticate with the Controller. If you are using the AppDynamics SaaS Controller, the account name is provided in the Welcome email from AppDynamics.

**Optional Element:** `<account name="mycontroller.saas.appdynamics.com" password="myaccesskey" />`

***Account name attribute***

The account name attribute indicates the account name for the SaaS or multi-tenant Controller.

**Type:** String

**Default:** None

**Required:** Only for SaaS or multi-tenant Controllers

***Account password attribute***

The account password attribute indicates the access key for the SaaS or multi-tenant Controller.

**Type:** String

**Default:** None

**Required:** Only for SaaS or multi-tenant Controllers

**Proxy Element**

The Proxy element is a child of the Controller element. Use it to configure connection to the Controller through a proxy server with no authentication.

**Optional Element:** `<proxy host="proxy-name" port="3128" enabled="true" />`

***Proxy host attribute***

The proxy host attribute indicates the proxy server host name or IP address.

**Type:** String

**Default:** None

**Required:** host is required for the proxy element

***Proxy port attribute***

The proxy port attribute indicates the proxy server port.

**Type:** Positive Integer

**Default:** None

**Required:** port is required for the proxy element

***Proxy enabled attribute***

To enable Controller access through a proxy server, set the proxy enabled attribute to "true".

**Type:** Boolean

**Default:** false

**Required:** No

**Machine Agent Element**

The Machine Agent element is a child of the AppDynamics Agent element. An empty Machine Agent element enables the default instrumentation for the .NET machine agent (See [Monitor CLR](#)s and [Monitor IIS](#)). Enable optional additional Microsoft Performance Counters or .NET

Agent instrumentors as children of the Machine Agent element.

**Required Element:** <machine-agent/>

### **Machine Snapshot Element**

The Machine Snapshot element is a child of the Machine Agent element. Use it to tune the settings for machine snapshots in your environment. If you don't specify an attribute, the agent uses the default values for that attribute. See [Machine Snapshots for .NET](#) and [Configure Machine Snapshots for .NET](#).

**Optional Element:** <machine-snapshot enabled="true" window-size="600" samples-per-window="60" violations-per-window="6" max-percent-cpu="80" max-percent-memory="80" max-queue-item-age="100" periodic-collection="600" />

#### ***Machine Snapshot enabled attribute***

Set the Machine Snapshot enabled attribute to "false" to disable machine snapshots.

**Type:** Boolean

**Default:** true

**Required:** No

#### ***Machine Snapshot window size attribute***

Specify the window size time range in seconds for the .NET Machine Agent to take samples. During a window, the agent takes samples and checks them for breached thresholds: max percent CPU, the max percent memory, and max queue item age.

**Type:** Integer

**Default:** 600

**Required:** No

#### ***Machine Snapshot samples per window attribute***

Specify the number of samples the .NET Machine Agent takes during the specified window. For example, if the window size is 600 and the samples per window is 60, the agent takes a sample once every 10 seconds.

**Type:** Integer

**Default:** 60

**Required:** No

#### ***Machine Snapshot violations per window attribute***

When the .NET Machine Agent detects the number of violations per window for one threshold, it takes a snapshot. The agent only takes one snapshot per window for thresholds exceeded. For example, if the violations per window is set to 6, and six samples show memory usage at 80% or greater, the agent takes a snapshot. The counters for each threshold are separate.

**Type:** Integer

**Default:** 6

**Required:** No

***Machine Snapshot max percent memory attribute***

When the .NET Machine agent detects CPU usage on the machine equals or exceeds the max percent cpu value, it flags the sample as a violation. The minimum value is 20; the maximum value is 100.

**Type:** Integer

**Default:** 80

**Required:** No

***Machine Snapshot max percent cpu attribute***

When the .NET Machine agent detects memory usage on the machine equals or exceeds the max percent memory value, it flags the sample as a violation. The minimum value is 20; the maximum value is 100.

**Type:** Integer

**Default:** 80

**Required:** No

***Machine Snapshot max queue item age attribute***

When the .NET Machine agent detects the oldest item in the IIS queue equals or exceeds the max queue item age value in milliseconds, it flags the sample as a violation.

**Type:** Integer

**Default:** 100

**Required:** No

***Machine Snapshot periodic collection attribute***

The .NET Machine agent takes one snapshot per periodic collection time range. Specify the value in seconds. The minimum is 60 seconds.

**Type:** Integer

**Default:** 600

**Required:** No

**CLR Crash Reporting Element**

The CLR Crash Reporting element is a child element of the Machine Agent element. Use the CLR Crash Reporting element to control whether or not the .NET Machine Agent reports CLR crash events to the Controller. See [Monitor CLR Crashes](#).

**Optional Element:** `<clrcrash-reporting enabled="true"/>`

***CLR Crash Reporting enabled attribute***

Set the enabled attribute to "false" to stop reporting CLR Crash events to the controller.

**Type:** Boolean

**Default:** true

**Required:** No

**Process Monitor Element**

The Process Monitor element is a child element of the Machine Agent element. By default, the agent enables process monitoring for all IIS processes.

**Optional Element:** `<process-monitor report-all-iis-processes="true"/>`

***Process Monitor report all IIS processes attribute***

*New in 4.0.1*, set the report-all-iis-prcoesses attribute to "false" to enable process monitoring only for instrumented IIS processes.

**Type:** Boolean

**Default:** true

**Required:** No

### Metrics Element

The Metrics element is a child element of the Machine Agent element. By default, the machine agent registers a maximum of 200 metrics. See [Metrics Limits](#). Use the Metrics element to increase the number of metrics the .NET Machine Agent can register.

Use caution when increasing the metric registration limits. Increasing the limit can increase the resource overhead for agents and Controller.

**Optional Element:** `<metrics max-metrics="200"/>`

***Metrics max-metrics attribute***

Specify the maximum number of metrics the .NET Machine Agent can register.

**Type:** Integer

**Default:** 200

**Required:** No

### Performance Counters Element

The Performance Counters element is a child of the Machine Agent element. It is a container for all performance counters.

**Optional Element:** `<perf-counters>`

***Performance Counter element***

The Performance Counter element is a child of the Performance Counters element. For a list of performance counters to enable, see [Performance Counters in the .NET Framework](#).

**Optional Element:** `<perf-counter cat="category" name="name" instance="instance"/>`

***Performance Counter cat attribute***

The performance counter cat attribute indicates the performance counter category.

**Type:** String

**Default:** None

**Required:** Category is required for the Performance Counter element.

**Performance Counter name attribute**

The performance counter name attribute indicates the performance counter name.

**Type:** String

**Default:** None

**Required:** Name is required for the Performance Counter element.

**Performance Counter instance attribute**

The performance counter instance attribute is the performance counter instance value.

**Type:** String

**Default:** None

**Required:** Instance is required for the Performance Counter element.

**Sample Machine Agent Configuration with Performance Counters**

```
<machine-agent>
  <!-- Additional machine level Performance Counters -->
  <perf-counters>
    <perf-counter cat="Network Interface" name="Bytes Sent" instance="Local
Area Connection"/>
  </perf-counters>
</machine-agent>
```

**Instrumentation Element**

The Instrumentation element is a child of the Machine Agent element. It allows you to enable additional .NET Agent instrumentors such as [thread correlation](#) or [correlation for .NET remoting](#).

**Optional Element:** `<instrumentation>`

**Instrumentor element**

The **Instrumentor** element is a child of the Instrumentation element. The instrumentor element specifies the .NET Agent instrumentor to implement.

**Optional Element:** `<instrumentor name="instrumentor name" enabled="true"/>/>`

**Instrumentor name attribute**

The instrumentor name attribute indicates the instrumentor name.

**Type:** String

**Default:** None

**Required:** Name is required for the Instrumentor element.

**Instrumentor enabled attribute**

Set the instrumentor enabled attribute to "true" to enable instrumentation.

**Type:** Boolean

**Default:** false

**Required:** No.

**i** The current configuration syntax is **enabled="true"**. Versions prior to 3.7.8 used **disabled="false"**.

### Sample Machine Agent Configuration with Thread Correlation Instrumentors

```
<machine-agent>
  <!--Enable thread correlation-->
  <instrumentation>
    <instrumentor name="ThreadCorrelationThreadPoolCLR2Instrumentor"
enabled="true"/>
    <instrumentor name="ThreadStartCLR2Instrumentor" enabled="true"/>
    <instrumentor name="ThreadStartCLR4Instrumentor" enabled="true"/>
  </instrumentation>
</machine-agent>
```

#### Additional .NET Agent Instrumentors

- See [Enable Thread Correlation for .NET](#).
- See [Enable Correlation for .NET Remoting](#).
- See [Enable Instrumentation for WCF Data Services](#).

## App Agents Element

The App Agents element is a child of the AppDynamics Agent element. It is a container for app agent configurations for IIS applications, Windows services, and standalone applications.

**Required Element:** `<app-agents enabled="true">`

#### *App agents enabled attribute*

To disable application monitoring on the server, set the app agents enabled attribute to "false".

**Type:** Boolean

**Default:** true

**Required:** No

#### Default Profiler Element

The default Profiler element is a child of the App Agents element. It defines customizations to the default profiler behavior for all instrumented .NET applications on the machine: IIS applications, application pools, Windows services, and standalone applications.

**Optional Element:** `<profiler>`

**i** To override the default profiler, you can specify a profiler element as a child of the following elements

#### Profiler - Disabled Features Element

The Disabled Features element is a child of the Profiler element. Use this property to disable data

collection mechanisms at the agent level for security or privacy reasons. This configuration overrides any configuration set by the Controller.

**Optional Element:** <disabled-features value="NONE"/>

*Disabled features value attribute*

The disabled features value is a comma separated list of features to disable. The available values are as follows:

- LOG\_PAYLOAD: Override the log-request-payload node property to suppress logging http request payloads
- RAW\_SQL: Override the capture-raw-sql node property to suppress logging raw sql output
- CUSTOM\_EXIT\_SNAP\_DATA: Suppress snapshot data from custom exits points
- METHOD\_INV\_DATA\_COLLECTOR: Suppress method invocation data collector user data
- HTTP\_DATA\_COLLECTOR: Suppress HTTP request data collector user data
- INFO\_POINT: Suppress information point metrics
- ALL: Disable all the available features
- NONE: Don't disable features

**Type:** String

**Default:** NONE

**Required:** No

#### Profiler - Successful Exit Code Element

The Successful Exit Code element is a child of the Profiler element. The default successful exit code determines whether or not the agent flags a CLR restart event as graceful or not for Windows services or standalone applications. This configuration doesn't apply to IIS.

**Optional Element:** <successful-exit-code value="0"/>

*Successful exit code value attribute*

**Type:** Integer

**Default:** 0

**Required:** No

#### Sample Default Profiler Configuration

```
<app-agents>
  <profiler>
    <disabled-features value="LOG_PAYLOAD,RAW_SQL,CUSTOM_EXIT_SNAP_DATA" />
    <-- Set the successful exit code for Windows services and standalone
    applications to "1." -->
    <successful-exit-code value="1"/>
  </profiler>
  ...
</app-agents>
```

## App Agents - IIS Element



The **IIS** element is a child of the App Agents element. There are three options to configure IIS applications:

- Automatic configuration
- Application pool configuration
- Application configuration

The settings for any application pool apply to all applications within the app pool unless the individual application has a specific configuration.

Explicit child-level configurations override parent-level configurations. Otherwise, children inherit parent configurations.

**Optional Element:** <IIS>

### IIS Automatic Instrumentation Element

The Automatic element is a child of the IIS element. Use the Automatic element to enable or disable automatic instrumentation for all IIS apps. You can configure automatic instrumentation and manual instrumentation both. Manual configurations override automatic ones.

**Optional Element:** <automatic enabled="false" />

#### *Automatic enabled attribute*

Set the automatic enabled attribute to "true" to enable instrumentation for all IIS applications. This is the default setting if you use the .NET Agent Configuration Utility automatic configuration option. To disable automatic instrumentation for all IIS applications, set the value to "false".

**Type:** Boolean

**Default:** true

**Required:** No


### IIS Application Pools Element

The IIS Application Pools element is a child of the IIS element. It is a container element for all the IIS application pools you configure for instrumentation.

**Optional Element:** <application-pools>

### IIS Application Pool Element

The Application Pool element is a child of the Application Pools element.. You may have multiple application pool elements distinguished by the name attribute. Use the application pool element to configure the app agent for all applications within an application pool. For more information on IIS application pools, see [Managing Application Pools in IIS](#).

 Application-specific configurations in the IIS Applications element override application pool configurations.

**Optional Element:** <application-pool name="DefaultAppPool" enabled="false">

#### *Application pool name attribute*

The application pool name attribute indicates the name of the IIS Application Pool.

**Type:** String

**Default:** None

**Required:** Name is required for the Application Pool element.

#### *Application pool enabled attribute*

Set the application pool enabled attribute to "false" to disable instrumentation for all applications in the application pool. Set the value to "true" to instrument all applications in the application pool.

**Type:** Boolean

**Default:** None. Defaults to true if not specified.

**Required:** No

#### Application Pool Tier Element

The Tier element is a child of the Application Pool element. If you enable instrumentation for an Application pool, you must use a Tier element to assign the pool's applications to a tier in the Controller. See [AppDynamics Concepts](#).

**Required Element:** `<tier name="Inventory" />`

#### *Tier name attribute*

Use the tier name attribute to specify the tier.

**Type:** String

**Default:** None

**Required:** Yes

#### IIS Applications Element

The **IIS Applications** element is a child of the IIS element. It is a container element for all the IIS applications you configure for instrumentation.

**Optional Element:** `<applications>`

#### Application Element

The Application element is a child of the Applications element. Use multiple application elements to instrument different sites and applications. To learn more about IIS sites and applications, see [Understanding Sites, Applications, and Virtual Directories on IIS 7 and Above](#).

**Optional Element:** `<application path="/" site="FirstSite" port="8008" site-regex="false">`

#### *Application site attribute*

The application site attribute indicates the root site in IIS for the application. The site name accepts a [regular expression](#) for cases like windows Azure where you may only know a partial site name. If you use a regular expression, set the Application site-regex attribute to true. For an example, see "Customize Tier Names for Azure Roles" on [Install AppDynamics for Windows Azure with NuGet](#).

**Type:** String

**Default:** None

**Required:** Site is required for the Application element.

***Application site-regex attribute***

Set the Application site-regex attribute to true to treat the value of the Application site attribute as a [regular expression](#).

**Type:** Boolean

**Default:** false

**Required:** No

***Application path attribute***

The application path attribute indicates the application's path relative to the root site. Use the forward slash to indicate the root site and instrument all children applications. Use the path to an application to instrument the specific application and any children.

For example: Site1 hosts two applications AppX and AppY. To instrument Site 1, AppY and AppZ, set the path to "/". To instrument AppY, but not AppZ, set the path to "/AppY".

**Type:** String

**Default:** /

**Required:** Path is required for the Application element.

***Application port attribute***

For cases where two or more sites in IIS 6 have the same site name, set the site port attribute to differentiate between the sites.

**Type:** Positive Integer

**Default:** None

**Required:** No

***Application enabled attribute***

In certain cases you may want to enable instrumentation for a parent application, but disable it for a child application. In this case create an Application element for the child application to disable and set the application enabled attribute to "false".

**Type:** Boolean

**Default:** true

**Required:** No

***Application Tier Element***

The Tier element is a child of the Application element. If you enable instrumentation for an application, you must use a Tier element to assign the application to a tier in the Controller. See [AppDynamics Concepts](#).

**Required Element:** `<tier name="Consumer" />`

***Tier name attribute***

The tier name attribute indicates the business application tier.

**Type:** String

**Default:** None

**Required:** Yes

## Sample IIS Application Configuration

```
<IIS>
  <!-- Automatic instruments all IIS applications when enabled. -->
  <automatic enabled="false" />

  <!-- Application Pool agent configurations -->
  <application-pools>
    <!-- Do not instrument applications in DefaultAppPool when "enabled"
    attribute is set to false. -->
    <application-pool name="DefaultAppPool" enabled="false">
      <tier name="Tier Name"/>
    </application-pool>

    <!-- Instrument applications in the OtherAppPool and assign them to the
    Inventory tier. -->
    <application-pool name="OtherAppPool">
      <tier name="Inventory"/>
    </application-pool>
  </application-pools>
  <applications>
    <!-- Instrument all applications in the First Site. -->
    <application path="/" site="FirstSite">
      <tier name="Order"/>
    </application>
    <!-- Instrument the /app application and child apps in the Second Site
    -->
    <!-- but not the root Second Site application. -->
    <application path="/app" site="SecondSite">
      <tier name="Consumer"/>
    </application>
    <!-- Regular expression for site name -->
    <!-- assigns all sites beginning with "MyRole" to the Credit Services
    tier. -->
    <application path="/" site="MyRole_\w+" site-regex="true">
      <tier name="Credit Services"/>
    </application>

  </applications>
</IIS>
```

## App Agents - Standalone Applications Element

The Standalone Applications element is a child of the App Agents element. It is a container element for all the standalone applications you configure for instrumentation.

**i** For instructions to instrument standalone applications, see [Instrument Windows Services and Standalone Applications](#).

**Optional Element:** <standalone-applications>

## Standalone Application Element

The Standalone Application element is a child of the Standalone Applications element. It specifies a Windows service or standalone application to instrument.

**Optional Element:** `<standalone-application  
executable="MyWindowsApplication.exe" command-line="">`

***Standalone Application executable attribute***

The standalone application executable attribute specifies the file name of the Windows application to instrument. Set the Standalone Application element executable attribute to one of the following:

- Executable name. For example, MyStandaloneApp.exe or MyWindowsService.exe. The file extension is optional, so MyStandaloneApp also works.
- Full path to the executable. For example, C:\Program Files\MyApplication\MyStandaloneApp.exe.
- Partial path to the executable. For example, MyApplication\MyStandaloneApp.exe. Use the full or partial path to the executable when you want to assign different tiers to separate instances of the same executable running from different paths.

**Type:** String

**Default:** None

**Required:** Yes

***Standalone Application command-line attribute***

To differentiate between two instances of the same executable, specify any unique portion of the application's command line, such as an argument, in the Standalone Application command-line attribute.

**Type:** String

**Default:** None

**Required:** No

***Standalone application app-domain-name attribute***

For applications with multiple application domains, the app-domain-name attribute enables you to limit instrumentation to specific application domains. See [Configure Application Domain Monitoring](#).

**Type:** String

**Default:** None

**Required:** No

***Standalone Application Tier Element***

The Tier element is a child of the Standalone Application element. If you enable instrumentation for an application, you must use a Tier element to assign the application to a tier in the Controller. See [AppDynamics Concepts](#).

**Required Element:** `<tier name="Consumer" />`

***Tier name attribute***

The tier name attribute indicates the business application tier.

**Type:** String  
**Default:** None  
**Required:** Yes

### Sample Windows service and Standalone Application Configuration

```
<standalone-applications>
  <standalone-application executable="ExampleApplication.exe">
    <tier name="Standalone Application Tier"/>
  </standalone-application>
  <!-- Instrument a Windows service using arguments. -->
  <!-- The following example matches the command "MyWindowsService.exe -d -x
-r". -->
  <standalone-application executable="MyWindowsService.exe" command-line="-x">
    <tier name="Windows Service Tier"/>
  </standalone-application>
</standalone-applications>
```

## Configure Multiple Business Application Support for .NET

### On this page:


- [Prepare to Configure Multiple Business Applications](#)
- [Manually Configure the .NET Agent](#)
- [Sample Configuration](#)
- [Agent Configuration Properties for Multiple Application Support](#)

### Related pages:

- [.NET Agent Configuration Properties](#)

.NET Agent versions prior to 3.9 required that monitoring data for all .NET applications running on a Windows host report to one business application in the Controller. This topic describes how to configure the agent to register different applications on the same Windows host with multiple business applications in the Controller.

If a tier in the first business application makes an exit call to a second tier in another business application, AppDynamics applies [Cross Application Flow](#). To learn more about AppDynamics business applications, see [AppDynamics Concepts](#).

 Multiple business application support in the .NET Agent requires manually editing the config.xml to use a schema that the AppDynamics Agent Configuration utility doesn't currently support.

If you follow these instructions to configure multiple application support, you can't use the configuration utility to make configuration changes afterward. If you launch the configuration utility on a server where you've configured multiple application support, the utility will prompt you to delete the configurations.

### Prepare to Configure Multiple Business Applications

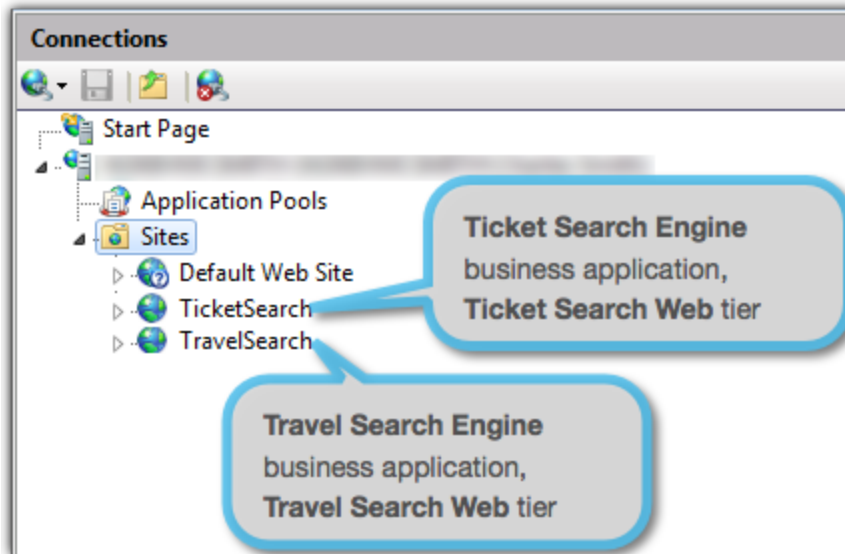
Before you configure the .NET Agent, you must install the agent. Use the AppDynamics Agent Configuration utility to perform basic configuration tasks.

1. Identify how you want to organize your business applications and identify the business application names.

For example, consider a Windows host running IIS. The IIS instance serves two applications for two separate customers: Ticket Search and Travel Search. The applications perform similar functions, but it makes sense to monitor them separately because they function independently. In this case, create two business applications based upon the application name: Ticket Search Engine and Travel Search Engine.

2. Map your IIS applications or application pools, Windows services, and standalone applications to tiers in the different business applications.

For example, map the TicketSearch site to the Ticket Search Web tier in the Ticket Search Engine business application. Map the TravelSearch site to the Travel Search Web tier in the Travel Search Engine business application.



3. If you have not already done so, install the .NET Agent. See [Install the .NET Agent](#).
4. Run the AppDynamics Agent Configuration utility to generate a config.xml and configure the Controller connection. See [Configure the .NET Agent](#)
5. When prompted, choose **Manual** for the method of tier generation and assignment.

**i** Currently the configuration utility only supports mapping one business application per server.

### Manually Configure the .NET Agent

Once you have configured the Controller properties for the .NET Agent, instrument your .NET Applications in the config.xml.

1. Open the config.xml file as administrator and edit the file. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).

- Copy the **Controller Applications** block and paste it as a child element of the **Controller** element. Replace any existing `<applications>` or `<application>` elements:

```
<!--Configure multiple business applications-->
<applications>
  <application name="ApplicationName1" default ="true"/>
  <application name="ApplicationName2"/>
</applications>
```

- Add an **Application** element for each of the business applications in the Controller.
  - Edit the **name** attribute for the **Application** elements to match the business application name in the Controller. If the application doesn't exist yet, the Controller creates it.
  - Set the **default** attribute to "true" for one **Application** element. If the agent can't find a match for the business application name in the IIS application, Windows service, or standalone application configuration, the tier reports to the default business application.

In this example, Ticket Search is the default business application:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090" ssl="false">
    <!--Configure multiple business applications-->
    <applications>
      <application name="Ticket Search" default ="true"/>
      <application name="Travel Search"/>
    </applications>
  </controller>
  ...
```

- Add configuration elements for the IIS applications or application pools, Windows services, or standalone applications to instrument.

For IIS applications, add the **IIS Applications** block as a child of the **IIS** element. Replace any existing `<applications>` element. For more information, see [IIS Applications Element](#).

• [Show IIS Application configuration.](#)



```
<applications>
  <application path="/" site="FirstSite"
    controller-application="Application1">
    <tier name="FirstSite Tier"/>
  </application>
  <application path="/" site="SecondSite"
    controller-application="Application2">
    <tier name="SecondSite Tier"/>
  </application>
</applications>
```

For IIS application pools, add the **IIS Application Pools** block as a child of the **IIS** element. Replace any existing `<application-pools>` element. For more information, see [IIS Application Pools Element](#).

▼ [Show IIS Application Pool configuration.](#)

```
<application-pools>
  <application-pool name="MyAppPool1"
    controller-application="Application1">
    <tier name="App1 AppPool Tier"/>
  </application-pool>
  <application-pool name="MyAppPool2"
    controller-application="Application2">
    <tier name="App2 AppPool Tier"/>
  </application-pool>
</application-pools>
```

For Windows services or standalone applications, add the **Standalone Applications** block as a child of the **App Agents** element. For more information, see [Standalone Applications Element](#).

▼ [Show standalone application configuration.](#)

```
<standalone-applications>
  <standalone-application executable="MyStandaloneApp.exe"
    controller-application="ApplicationName1">
    <tier name="Standalone App Tier"/>
  </standalone-application>
  <standalone-application executable="MyWindowsService.exe"
    command-line="-x" controller-application="ApplicationName2">
    <tier name="Windows Service Tier"/>
  </standalone-application>
</standalone-applications>
```

##### 5. Configure your application elements as follows:

- Add the corresponding element for each IIS application or application pool, Windows service, or standalone application to instrument. For information on specific elements see [.NET Agent Configuration Properties](#).
- For each application element set the **controller-application** attribute to the name of the corresponding business application. If you omit the controller-application attribute,

- the agent adds the application to a tier in the configured default business application.
- Set the **Tier** element **name** attribute to the business application tier name.
6. After you complete configuration, save the changes to config.xml.
  7. Restart the AppDynamics.Agent.Coordinator service.
  8. Restart IIS applications or application pools, Windows services, or standalone applications.
  9. As your applications begin processing traffic, the agent registers them with the Controller. Log on to the Controller to see that your applications have registered with the corresponding business application.

## Sample Configuration

This sample config.xml demonstrates configuration for multiple business applications in the Controller. Because the Windows service TicketService doesn't specify a controller-application attribute, it reports to the default business application Ticket Search Engine. All applications in the TravelAPIPool report to the Travel Search Engine.

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090"
ssl="false">
    <!--Configure multiple business applications-->
    <applications>
      <application name="Ticket Search Engine" default ="true"/>
      <application name="Travel Search Engine"/>
    </applications>
  </controller>
  <machine-agent />
  <app-agents>
    <IIS>
      <automatic enabled="false" />
      <application-pools>
        <application-pool name="TravelAPIPool"
controller-application="Travel Search Engine">
          <tier name="Travel APIs"/>
        </application-pool>
      </application-pools>
      <applications>
        <application path="/" site="TicketSearch"
controller-application="Ticket Search Engine">
          <tier name="Ticket Search Web"/>
        </application>
        <application path="/" site="TravelSearch"
controller-application="Travel Search Engine">
          <tier name="Travel Search Web"/>
        </application>
      </applications>
    </IIS>
    <standalone-applications>
      <standalone-application executable="StandaloneApp.exe"
controller-application="Ticket Search Engine">
        <tier name="Ticket Standalone Tier"/>
      </standalone-application>
      <standalone-application executable="WindowsService.exe"
command-line="-x" controller-application="Travel Search Engine">
        <tier name="Travel Windows Service Tier"/>
      </standalone-application>
    </standalone-applications>
  </app-agents>
</appdynamics-agent>
```

## Agent Configuration Properties for Multiple Application Support

Multiple business application support includes configuration properties for the .NET Agent. These configuration properties supersede the ones documented in [.NET Agent Configuration Properties](#).

### Controller Applications Element

The **Controller Applications** element is a child of the **Controller** element. It is a container element for all controller applications elements that map to business applications in the Controller.

**Required Element:** <applications>

### Controller Application Element

The **Controller Application** element is a child of the **Controller Applications** element. It indicates the name of the logical business application you see in the Controller. When you have more than one Controller Application element, you must set the default attribute to "true" for one of them.

**Required Element:** <application name="MyDotNetApplication" default="true"/>

#### *Application name attribute*

Set the application **name** attribute to the business application name in the Controller. If the application name does not exist, the Controller creates it when the agent registers.

**Type:** String

**Default:** None

**Required:** Yes

#### *Application default attribute*

Set the application **default** attribute to "true" for one **Controller Application** element. Instrumented applications without the **controller-application** attribute register with the default business application in the Controller.

**Type:** Boolean

**Default:** false

**Required:** For one application in multiple application configurations

#### *Controller-Application Attribute*

The **IIS Application**, **IIS Application Pool**, **Windows Service**, and **Standalone Application** elements accept the **controller-application** attribute. Set the value to the **Controller Application** element name. If you don't include a controller-application attribute, the application registers with the default business application.

For example, an IIS application:

```
<application path="/" site="MySite" controller-application="My Business Application">
```

**Type:** String

**Default:** None

**Required:** No

## Disable Instrumentation for an IIS Application Pool

### Related pages:

- [Administer the .NET Agent](#)
- [.NET Agent Configuration Properties](#)

When you install the .NET Agent on a machine and use automatic tier naming, the agent instruments every IIS application by default. If you don't need to monitor all application pools, disable monitoring for selected pools.

1. Open the config.xml file for editing as administrator. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).
2. Add the Application Pools block as a child of the IIS element. See [.NET Agent Configuration Properties](#).

```
<!-- Disable instrumentation for an application pool -->
<application-pools>
  <!-- Do not instrument applications in DefaultAppPool when "enabled"
attribute is set to false -->
  <application-pool name="DefaultAppPool" enabled="false" />
</application-pools>
```

3. Set the Application Pool element name attribute to the application pool name. This example disables instrumentation for the DefaultAppPool. You may add multiple Application Pool elements.
4. Restart the AppDynamics.Agent.Coordinator.
5. Restart IIS.

## Configure Application Domain Monitoring

### On this page:

- [Overview of AppDomains in .NET](#)
- [Configure Monitoring for Multiple Application Domains](#)

### Related pages:

- [Instrument DefaultDomain for Standalone Applications](#)
- [Application Domains](#)
- [.NET Agent Configuration Properties](#)
- [Instrument Windows Services and Standalone Applications](#)

You can configure the .NET Agent to monitor ASP.NET applications with multiple Application Domains (AppDomains). This topic assumes you have a working knowledge of AppDomains and that you are familiar with the AppDomain implementation in your application.

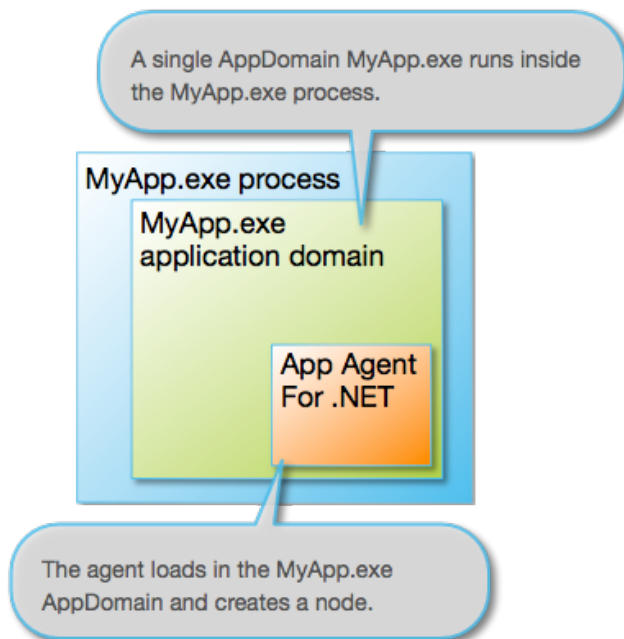
This topic does not cover the System Domain, Shared Domain, or DefaultDomain AppDomains the CLR instantiates before it executes the managed code. If your standalone application runs in the DefaultDomain, see [Instrument the DefaultDomain for Standalone Applications](#).

### Overview of AppDomains in .NET

Windows uses processes to manage security and performance isolation between running applications. Process isolation ensures one application's running code doesn't interfere with another application. However, for applications that share data, making calls between Windows processes can introduce complications and performance issues. AppDomains enable developers to create several applications that run inside a single process but maintain application isolation.

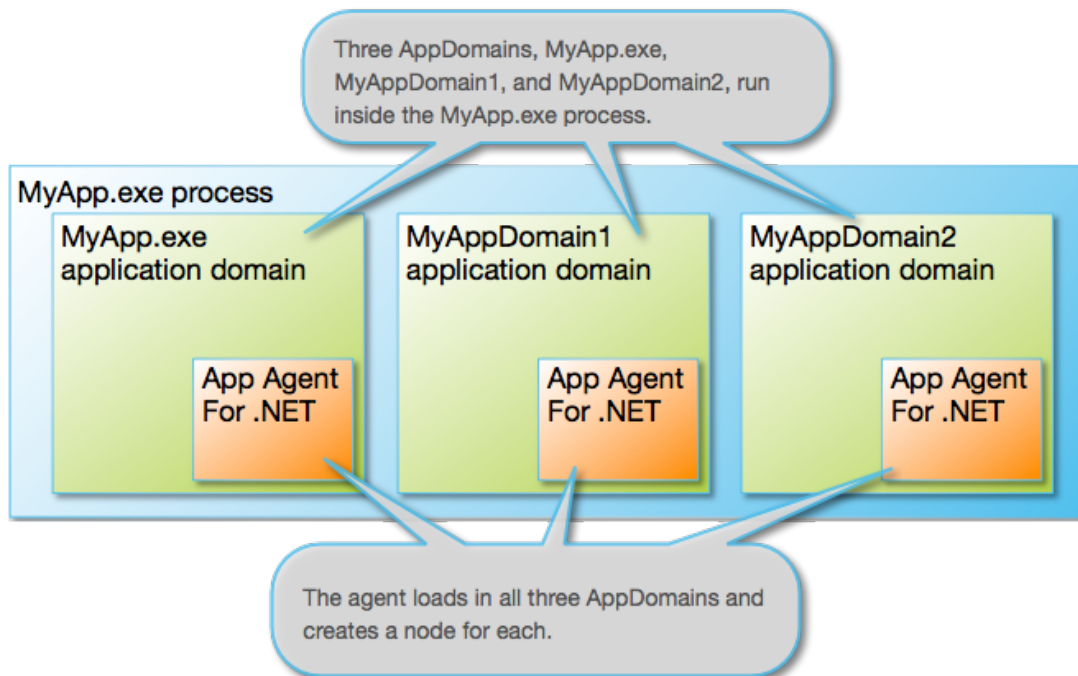
#### Single Application Domain

In the case of a single application running inside its own process, the runtime host typically manages the AppDomain. The application executable and the AppDomain have the same name. The .NET Agent (agent) installs itself inside the single AppDomain and creates a node for the application.



#### Multiple Application Domains

When developers include multiple AppDomains in an application, all the AppDomains run inside a single process. The application executable may have the same name as one AppDomain, but there are other, uniquely named AppDomains. By default, the agent installs itself inside all the AppDomains and creates nodes for them.



### Configure Monitoring for Multiple Application Domains

If the application you monitor contains multiple AppDomains, the App Agent for .NET automatically instruments each AppDomain and creates a node. You can configure the .NET Agent to instrument only the AppDomains you specify. This is useful to exclude AppDomains you don't want to monitor and to limit the number of nodes in a tier.

You can configure application domain monitoring for:

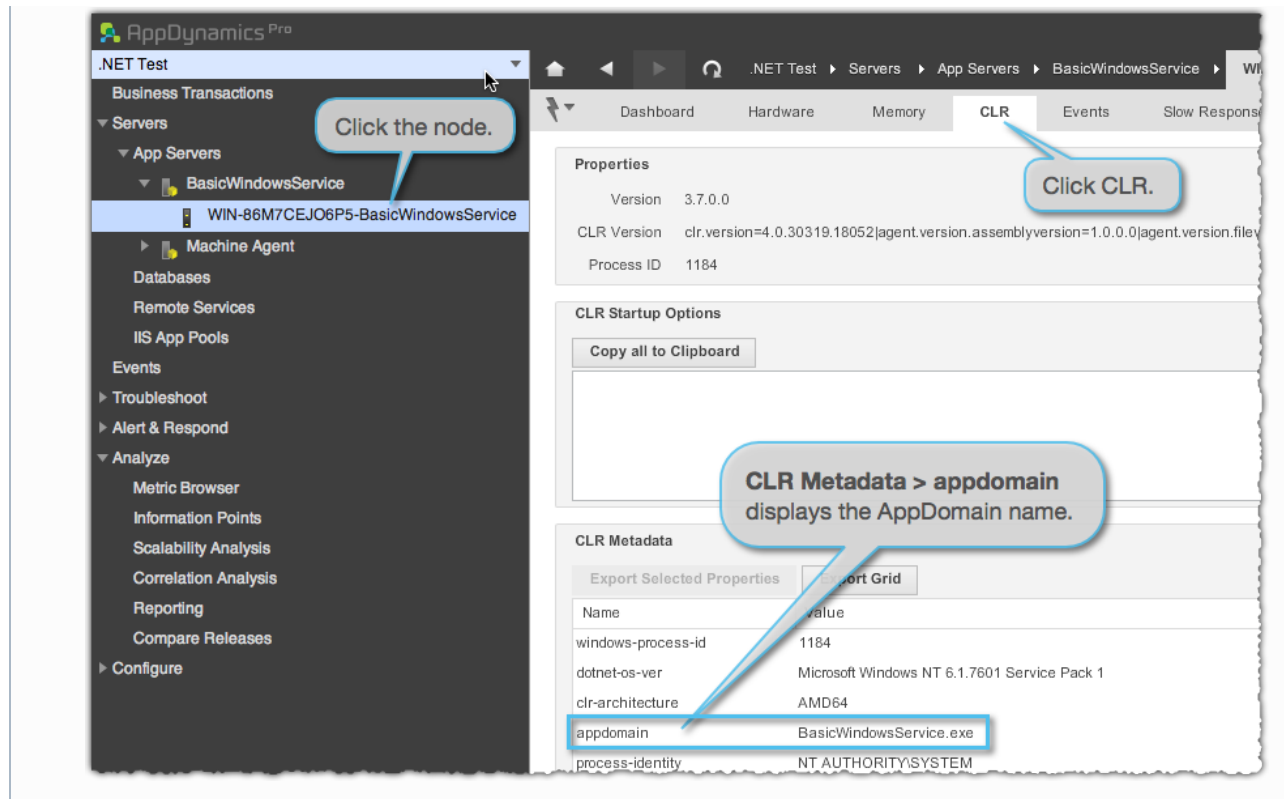
- [Windows Services](#)
- [Standalone Applications](#)

Configure all instrumentation settings for the .NET Agent in the config.xml file. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).

1. Identify the name of the AppDomains you want to instrument.

**i** If you have already instrumented your application, you can see the AppDomain names in the Node Dashboard.

Click the node in the left navigation pane, then click **CLR**.



2. Launch a text editor as administrator.
3. Edit the config.xml file as an administrator. See [Administer the .NET Agent](#).
4. Find the element that corresponds to your application with multiple AppDomains:

**Windows Service** element: `<windows-service name="MyWindowsService">`

**Standalone Application** element: `<standalone-application executable="MyWindowsApplication.exe">`

5. Add the **app-domain-name** attribute to the element.

For example, to instrument the MyApp.exe AppDomain for the MyApp.exe standalone application:

```
<standalone-application executable="MyApp.exe" app-domain-name="MyApp.exe">
  <tier name="StandaloneApplication Tier"/>
</standalone-application>
```

**i** As soon as you instrument one AppDomain in the config.xml, the agent instruments only the AppDomains you specify. Other AppDomains are not instrumented.

6. To instrument additional AppDomains, add an element for each AppDomain as if they were separate applications.

For example, to instrument MyAppDomain1 in MyApp.exe:

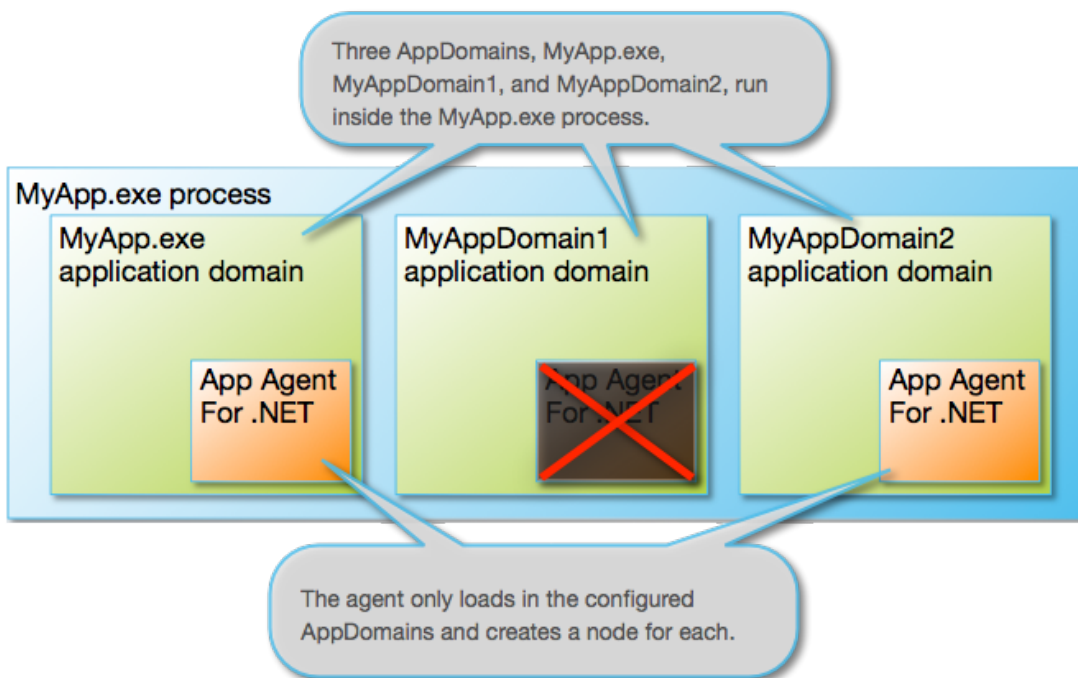


```
<standalone-application executable="MyApp.exe" app-domain-name="MyAppDomain1">
  <tier name="StandaloneApplication Tier"/>
</standalone-application>
```

7. Save the config.xml file.
8. Restart the AppDynamics.Agent.Coordinator service.
9. Restart instrumented applications: Windows services or standalone applications.

## Sample Standalone Application configuration with multiple AppDomains

This sample config.xml shows the configuration for the application MyApp.exe. Instrumentation only applies to the AppDomains specified in the config.xml: MyApp.exe and MyAppDomain2.



```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090" ssl=false">
    <application name="MyDotNetApplication" />
  </controller>
  <machine-agent />
  <app-agents>
    <standalone-applications>
      <standalone-application executable="MyApp.exe"
app-domain-name=" "MyApp.exe">
        <tier name="StandaloneApplication Tier" />
      </standalone-application>
      <standalone-application executable="MyApp.exe"
app-domain-name=" "MyAppDomain2">
        <tier name="StandaloneApplication Tier" />
      </standalone-application>
    </standalone-applications>
  </app-agents>
</appdynamics-agent>
```

## Instrument the DefaultDomain for Standalone Applications

### On this page:

- [Check if Your Application Runs in the DefaultDomain](#)
- [Instrument the DefaultDomain](#)

### Related pages:

- [Instrument Windows Services and Standalone Applications](#)
- [POCO Entry Points](#)
- [.NET Agent Configuration Properties](#)

By default, the .NET Agent does not instrument the .NET **DefaultDomain** AppDomain. Before you instrument the DefaultDomain:

- Follow the instructions to instrument a [standalone application](#).
- Create a [POCO entry point](#) for a class/method in the application.

If you complete those steps and still don't see business transactions in the Controller, check if your managed code runs in the DefaultDomain. If so, you must configure the agent to instrument the DefaultDomain.

### Check if Your Application Runs in the DefaultDomain

If you are unfamiliar with your application's managed code, you can use the agent logs to identify the AppDomain.

1. Open the agent log:

**Windows Server 2008 and later:** %ProgramData%\AppDynamics\DotNetAgent\Logs\AgentLog.txt

**Windows Server 2003:** %AllUsersProfile%\Application Data\AppDynamics\DotNetAgent\Logs\AgentLog.txt

2. Search the agent log for "AppDomain".

Few log entries contain "AppDomain" when the agent starts up. Look for an entry by "dllhost" or your instrumented application similar to the following:

```
2013-12-16 08:23:02.3120 3068 MYPROGRAM 1 1 Info Configuration
appDomainName=DefaultDomain appDomainId=1 iis-app=null site=null port=null
appPoolId=
2013-12-16 08:23:02.6240 3192 dllhost 1 17 Info ConfigurationManager Not
instrumenting DefaultDomain for pid 3068
```

In this example MYPROGRAM is the name of the instrumented standalone application. You can see the name of the AppDomain in the log entry: appDomainName=DefaultDomain.

## Instrument the DefaultDomain

1. Open the config.xml file for editing as administrator. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).
2. Copy the code block below to a child element of the **Standalone Application** element. See [.NET Agent Configuration Properties](#).

```
<profiler>
  <instrument-defaultdomain enabled="true"/>
</profiler>
```

 The Profiler element must follow the Standalone Application Tier element.

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <controller host="mycontroller.mycompany.com" port="8090" ssl="false">
    <application name="My Business Application" />
  </controller>
  <machine-agent />
  <app-agents>
    <IIS>
      <applications />
    </IIS>
    <standalone-applications>
      <standalone-application executable="MyStandaloneApp.exe">
        <tier name="Standalone Tier" />
        <profiler>
          <instrument-defaultdomain enabled="true"/>
        </profiler>
      </standalone-application>
    </standalone-applications>
  </app-agents>
</appdynamics-agent>
```

3. Save the config.xml file.
4. Restart the AppDynamics.Agent.Coordinator service.
5. Restart the standalone application for your changes to take effect.

## Configure the .NET Machine Agent

### On this page:

- [Customize .NET Machine Agent Behavior](#)
- [.NET Machine Agent Configuration Options](#)
- [Configure the .NET Machine Agent Without App Agents](#)

### Related pages:

- [Administer the .NET Agent](#)
- [.NET Agent Configuration Properties](#)

The AppDynamics .NET Agent includes an embedded .NET Machine Agent that runs as part of the AppDynamics.Agent.Coordinator service. Among other things, the Machine Agent regularly gathers system performance data and reports it back to the Controller as metrics.

**i** Do not confuse the .NET Machine Agent with the Standalone Machine Agent, a Java application. The Standalone Machine Agent provides the capability to use extensions (plugins, metric listener, orchestration). See [Standalone Machine Agent](#).

The app agent MSI Installer package and .NET Agent Configuration Utility automatically install and configure the .NET Machine Agent to connect to the Controller. The connection information is the same for the app agent and the .NET Machine agent. See [Configure the .NET Agent](#).

## Customize .NET Machine Agent Behavior

Customize instrumentation settings for the [Machine Agent element](#) in the config.xml file. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).

1. Shut down the AppDynamics.Agent.Coordinator service.
2. Edit the config.xml file as an administrator. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).
3. Modify the Machine Agent element and add any children elements according to the .NET Machine Agent configuration topics.
4. Save the config.xml.
5. Start the AppDynamics.Agent.Coordinator service.
6. In some cases you may need to restart IIS, instrumented Windows services, or instrumented Standalone applications. See individual .NET Machine Agent configuration topics.

## .NET Machine Agent Configuration Options

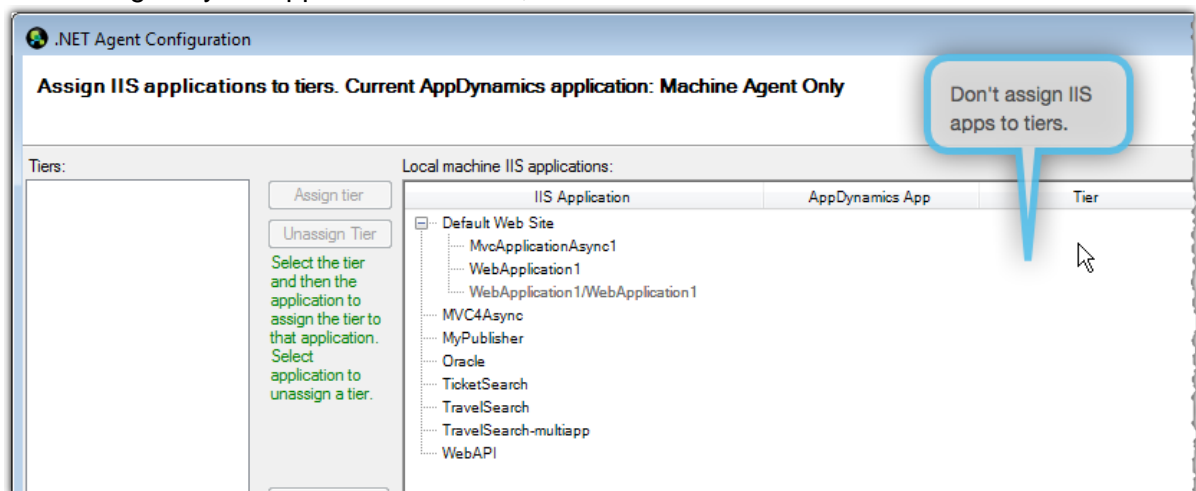
The following topics cover specific customizations for the .NET Machine Agent:

- [Enable Monitoring for Windows Performance Counters](#)
- [Enable Thread Correlation for .NET](#)
- [Enable Correlation for .NET Remoting](#)
- [Enable Instrumentation for WCF Data Services](#)
- [Configure Machine Snapshots for .NET](#)

## Configure the .NET Machine Agent Without App Agents

If you want to monitor the Windows hardware performance data, but do not want to monitor any .NET applications, you can configure the .NET Machine agent to run without the .NET Agent.

1. [Install the .NET Agent](#).
2. Launch the AppDynamics Agent Configuration utility and follow the steps until you reach the Assign IIS applications to tiers window.
3. Click **Manual** for the method of tier generation and assignment and click **Next**.
4. Don't assign any IIS applications to tiers, click **Next**.



**i** If you configured any Windows services or standalone applications, manually disable those agents in the config.xml.

5. Continue with the remaining steps and click **Done**.

Monitoring resumes for the .NET Machine Agent only. Metrics appear in the Controller under the Machine Agent tier, see [Monitor Windows Hardware Resources](#).

## Enable Monitoring for Windows Performance Counters

### On this page:

- [Performance Counters and the .NET Machine Agent](#)
- [Configure Additional Performance Counters for .Net](#)
- [Sample .NET Machine Agent Configuration with Performance Counters](#)

### Related pages:

- [Monitor CLR's](#)
- [Monitor IIS](#)

### Performance Counters and the .NET Machine Agent

By default, the .NET Machine Agent uses Microsoft Performance Counters to gather and report .NET metrics. For details on the preconfigured .NET metrics see [Monitor CLR's](#) and [Monitor IIS](#).

You can specify additional performance counters to be reported by the .NET Machine Agent.

### Configure Additional Performance Counters for .Net

1. Shut down the AppDynamics.Agent.Coordinator service.
2. Open the config.xml file for editing as administrator. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).
3. Add the Performance Counters block as a child of the Machine Agent element.

```
<perf-counters>
  <perf-counter cat="" name="" instance="" />
</perf-counters>
```

4. Create a Performance Counter element for each performance counter you want to add. Use any of the performance counters as specified in [Performance Counters in .NET Framework](#).
  - Set the cat attribute to the category of the performance counter.
  - Set the name attribute to the performance counter name.
  - Set the instance attribute to the of the performance counter.

**i** If a particular performance counter has many instances you can specify the following options:

- instance = "\*" OR
- instance = "all" (This will report the sum of all instances)

For example, to add the performance counter for measuring CPU Idle time(%), add the following element in the <perf-counters> block:

```
<perf-counter cat="Processor" name="% Idle Time"
instance="_Total"/>
```

5. Save the config.xml.
6. Start the AppDynamics.Agent.Coordinator service.

### Sample .NET Machine Agent Configuration with Performance Counters

```
<machine-agent>
  <!-- Additional machine level Performance Counters -->
  <perf-counters>
    <perf-counter cat="Processor" name="% Idle Time" instance="_Total"/>
  </perf-counters>
</machine-agent>
```

## Enable Correlation for .NET Remoting

### On this page:

- [Instrument Applications That Use .NET Remoting](#)
- [Specify an Agent Trigger](#)

### Related pages:

- [Configure Backend Detection for .NET](#)
- [Monitor Remote Services](#)

Developers use [.NET remoting](#) to build distributed applications that share objects across processes or across application domains running in the same process. AppDynamics disables correlation for .NET remoting functions by default.

### Instrument Applications That Use .NET Remoting

You can configure the .NET Agent to discover .NET remoting entry and exit points.

1. Open the config.xml file for editing as administrator. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).
2. Copy the code block below to a child element of the Machine Agent element. See [.NET Agent Configuration Properties](#):

```
<instrumentation>
  <instrumentor name="RemotingMscorlibEntryInstrumentor"
enabled="true"/>
  <instrumentor name="RemotingExitInstrumentor" enabled="true"/>
</instrumentation>
```

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...
  <machine-agent>
    <!--Enable correlation for .NET remoting-->
    <instrumentation>
      <instrumentor name="RemotingMscorlibEntryInstrumentor"
enabled="true"/>
      <instrumentor name="RemotingExitInstrumentor" enabled="true"/>
    </instrumentation>
  </machine-agent>
...
</appdynamics-agent>
```

3. Save the config.xml file.
4. Restart the AppDynamics.Agent.Coordinator Service.
5. Restart instrumented applications for your changes to take effect.

If the agent doesn't discover the entry points after configuration, [specify an agent trigger](#).

### Specify an Agent Trigger

.NET remoting entry point functions execute in low-level .NET libraries that may not trigger automatic agent instrumentation. If the agent doesn't discover the .NET remoting entry points after configuration you can specify a function that triggers the agent to begin instrumentation.

1. Identify a function to trigger the agent to begin instrumentation. The function can be any function that executes as part of the application process.

For example, consider the following code for a MovieTicket remoting object. In this case, use the function GetTicketStatus to trigger the agent.



```
using System;
namespace MovieGoer
{
    public class MovieTicket : MarshalByRefObject
    {
        public MovieTicket()
        {
        }
        public string GetTicketStatus(string stringToPrint)
        {
            return String.Format("Enquiry for {0} -- Sending back status:
{1}", stringToPrint, "Ticket Confirmed");
        }
    }
}
```

2. Edit the config.xml file as an administrator. See [Administer the .NET Agent](#).
3. Update the Instrumentation element to include the AgentTriggerInstrumentor. See [.NET Agent Configuration Properties](#):

```
<instrumentation>
  <instrumentor name="AgentTriggerInstrumentor" enabled="true" args="" />
  <instrumentor name="RemotingMscorlibEntryInstrumentor"
enabled="true"/>
  <instrumentor name="RemotingExitInstrumentor" enabled="true"/>
</instrumentation>
```

4. Set the AgentTriggerInstrumentor args value to the name of the trigger function from step 1.

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...
  <machine-agent>
    <!--Enable correlation for .NET remoting-->
    <instrumentation>
      <instrumentor name="AgentTriggerInstrumentor" enabled="true"
args="MovieGoer.MovieTicket.GetTicketStatus" />
      <instrumentor name="RemotingMscorlibEntryInstrumentor"
enabled="true"/>
      <instrumentor name="RemotingExitInstrumentor" enabled="true"/>
    </instrumentation>
  </machine-agent>
...
</appdynamics-agent>
```

5. Save the config.xml file.
6. Restart instrumented applications for your changes to take effect.

## Enable Thread Correlation for .NET

### On this page:

- [Configure Thread Correlation for .NET](#)

### Related pages:

- [Configure Backend Detection for .NET](#)
- [Monitor Remote Services](#)

The AppDynamics .NET Agent supports multi-threaded correlation for the following patterns:

- Thread.Start on the Common Language Runtime (CLR) 2.x and CLR 4.x
- ThreadPool.QueueUserWorkItem on the Common Language Runtime (CLR) 2.x and CLR 4.x

### Configure Thread Correlation for .NET

Configure all instrumentation settings for the .NET Agent in the config.xml file. See [Administer the .NET Agent](#).

1. Open the config.xml file for editing as an administrator. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).
2. Copy the code block below to a child element of the Machine Agent element. (See [Machine Agent Element](#)):

```
<instrumentation>
  <instrumentor name="ThreadCorrelationThreadPoolCLR2Instrumentor"
enabled="true"/>
  <instrumentor name="ThreadStartCLR2Instrumentor" enabled="true"/>
  <instrumentor name="ThreadStartCLR4Instrumentor" enabled="true"/>
</instrumentation>
```

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...
  <machine-agent>
    <!--Enable thread correlation-->
    <instrumentation>
      <instrumentor name="ThreadCorrelationThreadPoolCLR2Instrumentor"
enabled="true"/>
      <instrumentor name="ThreadStartCLR2Instrumentor" enabled="true"/>
      <instrumentor name="ThreadStartCLR4Instrumentor" enabled="true"/>
    </instrumentation>
  </machine-agent>
...
</appdynamics-agent>
```

The configuration syntax is **enabled="true"**.

**i** The agent enables the instrumentor for ThreadPool.QueueUserWorkItem on the CLR 4 by default without changes to the XML.

3. Save the config.xml file.
4. Enable the [thread-correlation](#) node property.
5. Restart the AppDynamics.Agent.Coordinator Service.
6. Restart instrumented applications for your changes to take effect.

## Enable Instrumentation for WCF Data Services

### Related pages:

- [Configure Business Transaction Detection for .NET](#)
- [.NET Agent Configuration Properties](#)

This topic describes how to configure the AppDynamics .NET Machine agent to enable instrumentation for WCF Data Services including WCF RIA Services for Microsoft LightSwitch.

1. Open the config.xml file for editing as administrator. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).
2. Copy the code block below to a child element of the Machine Agent element. See [.NET Agent Configuration Properties](#).

```
<instrumentation>
  <instrumentor name="WCFDSEntryInstrumentor" enabled="true" />
</instrumentation>
```

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...
  <machine-agent>
    <!--Enable instrumentation for WCF Data Services correlation-->
    <instrumentation>
      <instrumentor name="WCFDSEntryInstrumentor" enabled="true" />
    </instrumentation>
  </machine-agent>
...
</appdynamics-agent>
```

The configuration syntax is **enabled="true"**.

3. Save the config.xml file.
4. Restart the AppDynamics.Agent.Coordinator Service.
5. Restart instrumented applications for your changes to take effect.

## Configure Machine Snapshots for .NET

### On this page:

- [Default Settings and Configuration Considerations](#)
- [Configure machine snapshots for the .NET Machine Agent](#)

### Related pages:

- [Machine Snapshots for .NET](#)
- [.NET Agent Configuration Properties](#)

The .NET Machine Agent takes machine snapshots to capture vital system data for Windows and IIS. Use these guidelines to tune the frequency the .NET Machine Agent takes machine snapshots in your environment.

To learn how to analyze machine snapshots in the Controller, see [Machine Snapshots for .NET](#).

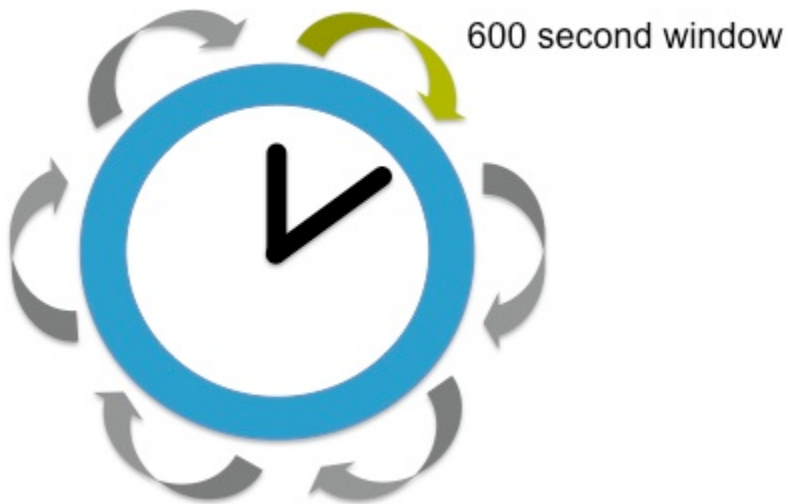
### Default Settings and Configuration Considerations

By default the .NET Machine Agent takes machine snapshots under the following conditions:

- Periodic collection: The agent takes one snapshot every 10 minutes.
- Breached thresholds: The .NET Machine agent takes samples of machine statistics every 10 seconds within a 10 minute window. For each sample, the agent checks the CPU percent usage, the memory percent usage, and oldest item in the IIS application pool queue. The agent flags a sample as a violation when the current usage meets or exceeds one of the following thresholds:
  - CPU at 80% or higher
  - Memory at 80% or higher
  - IIS application pool queue item older than 100 milliseconds

The agent takes a snapshot when it identifies 6 violations of a single type, such as CPU usage, within the window. The agent only takes one snapshot per window for breached thresholds.

With the default window size of 10 minutes and the violations per window of six, the sixth violation of a single type triggers a machine snapshot:



The 6th **violation** in the window triggers a snapshot.

Before you change the machine snapshot settings, decide which configuration options work best for your environment. The following questions and considerations should help you decide how to set your thresholds:

- What percentage CPU or memory usage might flag the beginnings of an issue in your environment?
- How long is too long for items to wait in the IIS queue?
- Do you expect occasional CPU or memory spikes?
- Are periodic collections every ten minutes frequent enough?
- Make sure the value for violations is larger than the number of samples per window.

For example: If you decrease the window size to 60 seconds and take 6 samples per window, the agent takes samples at the same frequency as the default settings, once every 10 seconds. However you are likely to get more snapshots because the agent only takes one snapshot per

window. If you set the violations per window to 5, the agent takes a snapshot any time half the samples in the window violate a specific threshold.

### Configure machine snapshots for the .NET Machine Agent

Configure all instrumentation settings for the .NET Machine Agent in the config.xml file. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).

1. Open the config.xml file for editing as an administrator. See "Where to Configure Agent Properties" on [Administer the .NET Agent](#).
2. Copy the code block below to a child element of the AppDynamics Agent element. See [AppDynamics Agent Element](#).

```
<machine-agent>
  <machine-snapshot enabled="true" window-size="600"
samples-per-window="60" violations-per-window="6" max-percent-cpu="80"
max-percent-memory="80" max-queue-item-age="100" periodic-collection="600"
/>
</machine-agent>
```

**i** If you have already customized the .NET Machine Agent, only copy the Machine Snapshot element and paste it as a child of the Machine Agent element.

3. Edit the values for the Machine Snapshot element attributes:

```
<?xml version="1.0" encoding="utf-8"?>
<appdynamics-agent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...
  <machine-agent>
    <!--Configure machine snapshots-->
    <machine-snapshot enabled="true" window-size="60" samples-per-window="10"
violations-per-window="5" max-percent-cpu="80" max-percent-memory="80"
max-queue-item-age="100" periodic-collection="600"/>
  </machine-agent>
...
</appdynamics-agent>
```

4. Save the config.xml file.
5. Restart the AppDynamics.Agent.Coordinator Service.