Dotnet Agent Installation with Remote Management - Powershell Extension – Advanced Options

Advanced Agent Configuration

The agent management module allows you to add instrumentation for existing IIS Applications, Windows services, and standalone applications to your existing configuration.

Common Command Options

Use the following syntax for tier assignment for Windows services and standalone applications:

```
@{ Name="<Windows service, or executable name>"; Tier="<tier name>" }
```

Use the following syntax for tier assignment for IIS web applications:

```
Site="<site name>"; Path="<virtual path>"; Tier="<tier name>"
```

Or for multiple tiers:

```
@({ Name="<IIS Application"; Path="<virtual path>"; Tier="<tier
name>" }, { Name="<Windows service, or executable name>"; Tier="<tier
name>" } )
```

Override: Remove the existing configuration before adding the new configuration.

Instrument Standalone Applications

The `Add-StandaloneAppMontoring` cmdlet adds configuration for standalone applications to an existing configuration file. After it modifies the agent configuration, Add-StandaloneAppMonitoring restarts the AppDynamics.Agent.Coordinator service automatically.

Syntax

```
Add-StandaloneAppMontoring @{ Name="<executable name>"; Tier="<tier
name>" }
```

Examples

```
#Instrument a standalone executable locally. Remove existing
configurations. Add-StandaloneAppMontoring @{ Name="MyApp.exe";
Tier="My App" } –Override


 #Instrument a standalone executable on a remote machine. Add-
StandaloneAppMontoring @( @{ Name="MyApp1.exe"; Tier="My App1" }, @{
Name="MyApp2.exe"; Tier="My App2" } ) -ComputerName
machine1.example.com


 #Instrument multiple standalone executables locally. Restart IIS and
a Windows service. Remove existing configurations. Add-
StandaloneAppMontoring @( @{ Name="MyApp1.exe"; Tier="My App1" }, @{
Name="MyApp2.exe"; Tier="My App2" } ) -RestartIIS -
RestartWindowsServices svc1 -Override
```

Why would you want to restart IIS or a Windows service when adding standalone application for monitoring?
The most common use case is when those standalone application(s) are started from IIS or a windows service. In this case we need to restart parent service to start monitoring of the standalone app.You can use the Add-StandaloneAppMonitoring command programmatically. For example first, run a script that lists the executables to monitor and adds them to a data structure to pass to the Add-StandaloneAppMapping cmdlet.

Instrument Windows Services

The `Add-WindowsServiceMontoring` cmdlet adds configuration for Windows services to an existing configuration file. After it modifies the agent configuration, Add-WindowsServiceMonitoring restarts the AppDynamics.Agent.Coordinator service automatically.

Syntax

```
Add-WindowsServiceMontoring @{ Name="<executable name>"; Tier="<tier name>" }
```

Examples

```
#Instrument a Windows service locally. Remove existing configurations. Add-WindowsServiceMonitoring @{ Name="MySvc"; Tier="My Svc" } –Override


#Instrument two Windows Services remotely. Add-WindowsServiceMonitoring @( @{ Name="MySvc1"; Tier="My Svc1" }, @{ Name="MySvc2"; Tier="My Svc2" } ) -ComputerName machine1.example.com


#Instrument two Windows services locally. Restart IIS. Restart the Windows services. Add-WindowsServiceMonitoring @( @{ Name="MySvc1"; Tier="My Svc1" }, @{ Name="MySvc2"; Tier="My Svc2" } ) -RestartIIS -RestartWindowsServices MySvc1, MySvc2
```

The following example automation script discovers local windows services which have names starting with "MyCompany" and adds those for monitoring.

```
$services = @()

foreach($i in (Get-Service -Name "MyCompany.*" }))

{

$svc = @{ Name=$i.Name ; Tier=$i.Name }

$services += $svc

}

Add-WindowsServiceMonitoring $services -RestartWindowsServices ( $services | { Select $_.Name} )
```

Instrument IIS web applications

The `Add-IISApplicationMontoring` cmdlet adds configuration for IIS applications to an existing configuration file. After it modifies the agent configuration, Add-WindowsServiceMonitoring restarts the AppDynamics.Agent.Coordinator service automatically.

Syntax

```
Add-IISApplicationMontoring @{ Site="<site name>"; Path="<virtual path>"; Tier="<tier name>" }
```

Examples

```
#Instrument the the Default Web Site locally. Add-IISApplicationMonitoring @{ Site="Default Web Site"; Path="/"; Tier="Web" }  #Instrument the Default Web Site locally. Remove existing configurations. Add-IISApplicationMonitoring @{ Site="Default Web Site"; Path="/"; Tier="Web" } -Override

#Instrument the Default Web Site locally. Restart IIS Add-IISApplicationMonitoring @{ Site="Default Web Site"; Path="/"; Tier="Web" } -RestartIIS  #Instrument the Default Web Site remotely. Restart IIS Add-IISApplicationMonitoring @{ Site="Default Web Site"; Path="/"; Tier="Web" } -ComputerName machine1.example.com -RestartIIS
```

The following automation script example scan local IIS web sites and instruments the sites ending with ".com" for monitoring:

```
$apps= @()

foreach($i in (Get-WebSite -Name "*.com" }))

{

$app= @{ Site=$i.Name; Path="/"; Tier=$i.Name }

$apps += $app

}

Add-IISApplicationMonitoring $apps -RestartIIS
```

Scripting Agent Configuration Using PowerShell Script Block

The `Update-ConfigirationFromScript` cmdlet is a sophisticated automation example. It lets you use a script block to define monitoring configurations individually for each server.This script block uses smart logic that determines which applications to monitor and configures them accordingly.

Syntax

```
Update-ConfigurationFromScript { <discoveryfunction> }
```

discoveryfunction is a user-defined method. It can execute locally or remotely.

Examples

```
#Update configurations locally. Remove existing configurations. Update-ConfigurationFromScript { MyDiscoveryFunction } –Override


#Update configurations locally. Restart IIS and Windows services. Update-ConfigurationFromScript { MyDiscoveryFunction } -RestartIIS -RestartWindowsServices Service1, Service2


#Update configurations remotely. Restart IIS and Windows services. Remove existing configurations Update-ConfigurationFromScript {
```

```
MyDiscoveryFunction } -RestartIIS -RestartWindowsServices svc1, svc2
-ComputerName ( Get-Content .\servers.txt ) -Override
```

This example function should return a hashtable with optional properties specifying which standalone application(s), windows service(s) and/or IIS application(s) to monitor.

Example Discovery Function 1

```
function MyDiscoveryFunction()

{

# Package return into a hashtable

[hashtable]$configuration = @{}

 # IIS array allows to monitor IIS applications
$configuration.IIS = @()

$configuration.IIS += @{ Site="Default Web Site"; Path="/";
Tier="Web" }

# Standalone allow to monitor standalone applications
$configuration.Standalone = @()

$configuration.Standalone += @{ Name="MyApp.exe"; Tier="My App" }

# WindowsService is an array with the list of windows service to be
monitored

$configuration.WindowsService = @()

$configuration.WindowsService += @{ Name="MySvc"; Tier="My Svc" }

return $configuration

}
```

This scenario enables dynamic discovery that can run on each remote server and return precise configuration.
Below is another example script that checks if the IIS web server is installed and then enables for monitoring only sites ending with ".com".
Additionally, it discovers local windows services with the "MyCompany" in the name.

Example Discovery Function 2

```
function MyDiscoveryFunction()

{

# Package return into a hashtable

[hashtable]$configuration = @{}

# Check if IIS is installed

$w3svc = Get-Service W3SVC -ErrorAction SilentlyContinue

if($w3svc -ne $null)

 {

  $websites = Get-WebSite -Name "*.com"

 }
```

```
if($websites.Count -gt 0)
 {
  $configuration.IIS = @()
  foreach($website in $websites)
   {
     $configuration.IIS += @{ Site = $website.Name; Path = "/"; Tier
= $website.Name }
   }
 }
}
# Check for windows services with naming starting "MyCompany."
if($services.Count -gt 0)
{
  $services = Get-Service -Name “MyCompany.*"
}
$configuration.WindowsService = @()
foreach($service in $services)
 {
  $configuration.WindowsService += @{ Name=$service.Name ;
Tier=$service.Name }
 }
}
# Return resulted configuration
return $configuration
}
```