Microsoft

Server Manageability and Automation

Windows Server 2012

Table of contents

Copyright information

Introduction

With Windows Server 2012, Microsoft brings its experience in building and operating public clouds to the server operating system, helping to make it a dynamic, highly available, and cost-effective platform for private clouds. Windows Server 2012 offers businesses and hosting providers a basis for a scalable, dynamic, and multitenant-aware cloud infrastructure that more securely connects across premises and helps IT to respond to business needs faster and more efficiently.

Windows Server 2012 offers excellent total cost of ownership (TCO) as an integrated platform with comprehensive, multicomputer manageability. Three ways in which Windows Server 2012 improves multicomputer management are:

- **Standards-based management approach**: The focus on industry standards used in Windows Server 2012 enables greater manageability across both Windows and non-Window devices alike.
- **Windows PowerShell 3.0**: The Windows PowerShell 3.0 command-line interface provides comprehensive automation capabilities.
- **Server Manager**: Server Manager in Windows Server 2012 helps you deploy and manage roles and features on the local server and remote servers, whether physical or virtual.

The following sections provide more detail about these three features of Windows Server 2012.

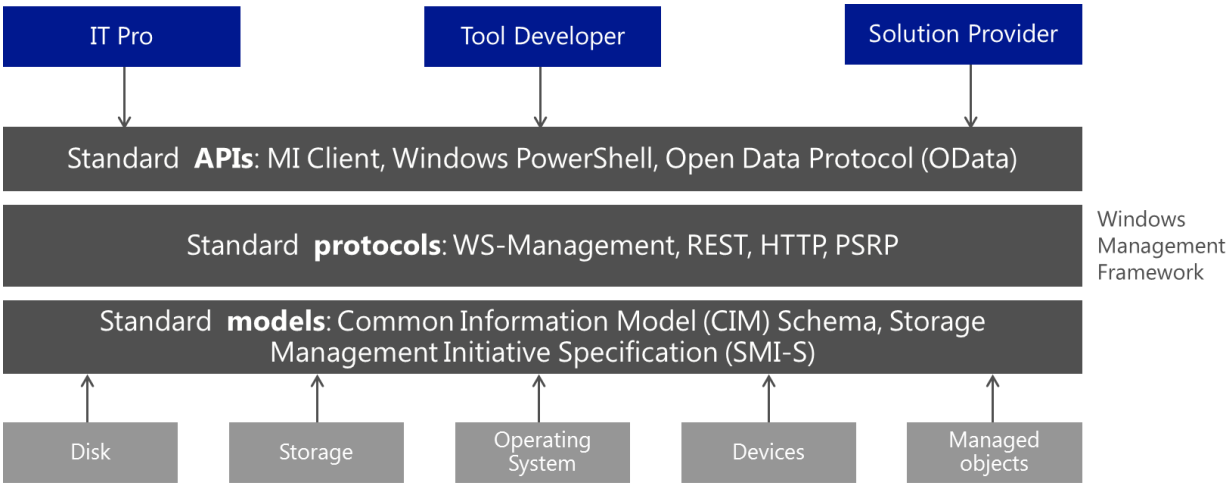
Standards-based approach to management in Windows Server 2012

Technical description

Windows Server 2012 improves the management experience within the datacenter and cloud environment through its enhanced application of standards-based management frameworks. Microsoft Windows has long supported standards-based management and has participated in organizations such as the Distributed Management Task Force (DMTF). These efforts have led to the development of Windows Management Instrumentation (WMI) and Windows Remote Management (WinRM). WMI is the Windows implementation of a Common Information Model (CIM) object manager, and WinRM is the implementation of the Web Services Management (WS-Man) protocol. Both CIM and WS-Man are standards released by the DMTF.

Windows Server 2012 enhances the manageability of datacenters through significant improvements in the standards-based infrastructure. It does this by delivering application programming interfaces (APIs) that are easier for developers and IT Pros to use. These APIs provide support for recent standards and add new kinds of Windows PowerShell cmdlets that make it simpler and more cost-effective to connect to and manage multiple servers and devices in the datacenter.

Figure 1: Improvements in Windows standards-based management components



Standard API improvements

Since the release of WMI, the number of management products and tools that consume its providers has steadily grown, but this traditionally has not been matched by a proportional increase in providers. The challenge for developers was that writing a WMI provider required extensive knowledge of Component Object Model (COM) coding. This made writing providers time consuming, and provided little benefit to the developer.

In Windows Server 2012, Microsoft introduces the Management Infrastructure APIs for Windows (MI APIs), which greatly simplify the development of new providers and client applications. These new MI APIs are available in both native (C/C++) code and managed (.NET) code for client development, and native code for provider development. The APIs remove the need to do COM coding, and they come with tools that generate code skeletons and schema from the class definition described in a MOF file. This makes provider development much easier and allows developers to spend their time on developing the business logic. A provider written using the new MI APIs can be called from the previous release of WMI, as well as from any non-Windows client application that uses the current DMTF WS-Man standard to connect to the Windows Server 2012 system.

In addition, the new MI APIs are updated to adhere to the CIM Infrastructure standard more closely than the classic WMI APIs, and by default, they use the standard WS-Man protocol for communicating across servers. Compliance with these standards means that developers can write applications using the MI APIs, which can manage other Windows servers and any server or device that supports the current DMTF CIM and protocol standards.

For web developers who want to manage Windows from non-Windows platforms, Windows Server 2012 includes the Management OData IIS Extension. This contains tools and components that simplify building REST APIs (OData Service endpoints).

OData is a set of URI conventions, tools, components, and libraries for building REST APIs. What makes the OData services stand out is that they are based on explicit domain models, which define their data content and behavior. This allows rich client libraries (such as Windows/IoS/Android phones, browsers, Python, and Java) to be generated automatically, thus simplifying the development of solutions on a wide range of devices and platforms.

Standard protocols

Another challenge in standards-based management is the definition and availability of a standard management protocol. With multiple vendors creating multiple management tools and interfaces on multiple platforms, the complexity of managing these environments continues to grow.

WMI is a standard Common Information Model Object Manager (CIMOM) that hosts many standard class providers; however, early on, there was not an interoperable management protocol, so WMI used the Distributed Component Object Model (DCOM). This made it an "island of management" for Windows managing Windows.

This situation changed with the DMTF's definition and approval of WS-Man, a SOAP-based, firewall-friendly protocol that allows a client on any operating system to invoke operations on a standards-compliant CIMOM running on any platform. Microsoft shipped the first partial implementation of WS-Man in Windows Server 2003 and named it Windows Remote Management (WinRM).

In Windows Server 2012, WinRM has become the default protocol for management. This provides interoperability with a number of CIMOM and WS-Man stacks available on other platforms, including Openwsman (Perl, Python, Java, and Ruby Bindings), Wiseman, and OpenPegasus.

Standard management tools

The implementation of WS-Man as a standard protocol further established a foundation on which standard APIs could be used to make manageability and interoperability easier and more efficient.

One of the goals for Windows Server 2012 has been to help IT Pros manage as many platforms and devices as possible using Windows PowerShell, as complex datacenter environments include a wide range of systems running recent versions of Windows and non-Windows operating systems.

The core management components described in this paper—WMI, MI APIs, WinRM, and PowerShell—are included in the Windows Management Framework 3.0. This downloadable package can be installed on Windows Server 2008, Windows Server 2008 R2, and Windows 7 systems, and provides them with all of the updated, standards-compliant functionality in Windows Server 2012.

Windows Server 2012 provides a new PowerShell module called *CIM cmdlets* that directly corresponds to the generic CIM operations and is built on top of the MI client .NET APIs. This enables the cmdlets in this module to manage Windows and non-Windows devices that support the current WS-Man and CIM standards. Windows Server 2012 also adds support for a new type of cmdlet, known as *CIM-Based cmdlets,* which provides developers and scripters the ability to interact with any CIM or WMI provider over WS-Man—including both existing WMI providers on Windows devices and CIM providers on non-Windows systems.

In short, the standards-based management approach in Windows Server 2012 enables IT Pros to use Windows PowerShell and the new standards-compliant features of Windows to manage any device in their datacenter that supports the current CIM, WS-Man, and OData standards.

Comprehensive, resilient, and simple automation with Windows PowerShell 3.0

Windows PowerShell 3.0 provides a comprehensive platform to help you manage most server roles and aspects of the datacenter. In this newest version of Windows PowerShell, sessions to remote servers are resilient and can withstand various types of interruptions. In addition, learning Windows PowerShell is now easier than ever through improved cmdlet discovery and simplified, consistent syntax across all cmdlets.

Technical description

The following sections describe the major features of Windows PowerShell.

Robust Session Connectivity

Long-running tasks, such as deploying a service pack or backing up a database, need to continue even if the client computer that initiated the requested operation goes down or disconnects.

With Robust Session Connectivity, remote sessions can remain in a connected state for up to four minutes—even if the client stops responding or becomes inaccessible—and tasks on the managed nodes continue to run on their own, making the end-to-end system more reliable. If connectivity cannot be restored within four minutes, execution on the managed nodes is suspended with no loss of data, and remote sessions automatically transition to a disconnected state, allowing them to be reconnected after network connectivity is restored. Corruption of application and system state from premature termination of running tasks due to unexpected client disconnection is virtually eliminated.

Disconnected Sessions

Windows PowerShell 3.0 lets you disconnect and then reconnect to a session without losing state. With Disconnected Sessions, you can create a session on a remote computer, start a command or job, disconnect from the session, shut down your computer, and then reconnect to the session from a different computer at a later time to check the job status or get the results. When administrators are disconnected from the session, commands and jobs can continue to run.

The functionality of the following cmdlets demonstrates the Disconnected Sessions capability in Windows PowerShell 3.0:

- **Disconnect-PSSession**. Disconnects a session connection from a remote computer.
- **Connect-PSSession**. Reestablishes a session connection with a remote computer.
- **Receive-PSSession**. By default, resumes execution of a command on a remote session and retrieves the session output. Implicitly reconnects to session (without Connect-PSSession command).

Example:

```
# Start a remote session, disconnect from the session, and exit PowerShell.
PS C:\> $s = New-PSSession -ComputerName srv1 -Name LongSession
PS C:\> $job = Invoke-Command $s { 1..10| % {echo "Long running job - part $_";
sleep 5} } -AsJob
PS C:\> Disconnect-PSSession $s
exit

# Start Windows PowerShell on a different computer.
PS C:\> $s = Get-PSSession -ComputerName srv1 -Name LongSession
PS C:\> Receive-PSSession $s
```

Job scheduling

Windows PowerShell 3.0 allows administrators to schedule jobs to be run at a later time, or according to a particular schedule. To create a scheduled job, you first create a job definition, which names the job and specifies the commands that it runs, and then a job trigger, which specifies the job schedule. The Windows Task Scheduler is used to schedule and start the job and a per-user job repository is used to store job output so that it is available later in a Windows PowerShell session on the computer.

The following cmdlets are available in the PSScheduledJob module to help you work with scheduled jobs:

- Add-JobTrigger
- Disable-JobTrigger
- Get-JobTrigger
- Enable-JobTrigger
- New-JobTrigger
- Remove-JobTrigger
- Set-JobTrigger
- Disable-ScheduledJob
- Enable-ScheduledJob
- Get-ScheduledJob
- Register-ScheduledJob
- Set-ScheduledJob
- Unregister-ScheduledJob
- Get-ScheduledJobOption
- New-ScheduledJobOption
- Set-ScheduledJobOption

Jobs can be scheduled to execute based on the following job triggers:

- Once
- Daily
- Weekly
- At startup
- At logon

Example:

```
$trigger = New-JobTrigger -Daily -At 4am
Register-ScheduledJob -Name MyScheduledJob -ScriptBlock { Get-Process } -Trigger
$trigger
Get-ScheduledJob
```

You can start a scheduled job manually.

Example:

```
Start-Job -DefinitionName MyScheduledJob
```

Once the trigger has fired and the job has run, you can work with it the same way you do regular background jobs.

Example:

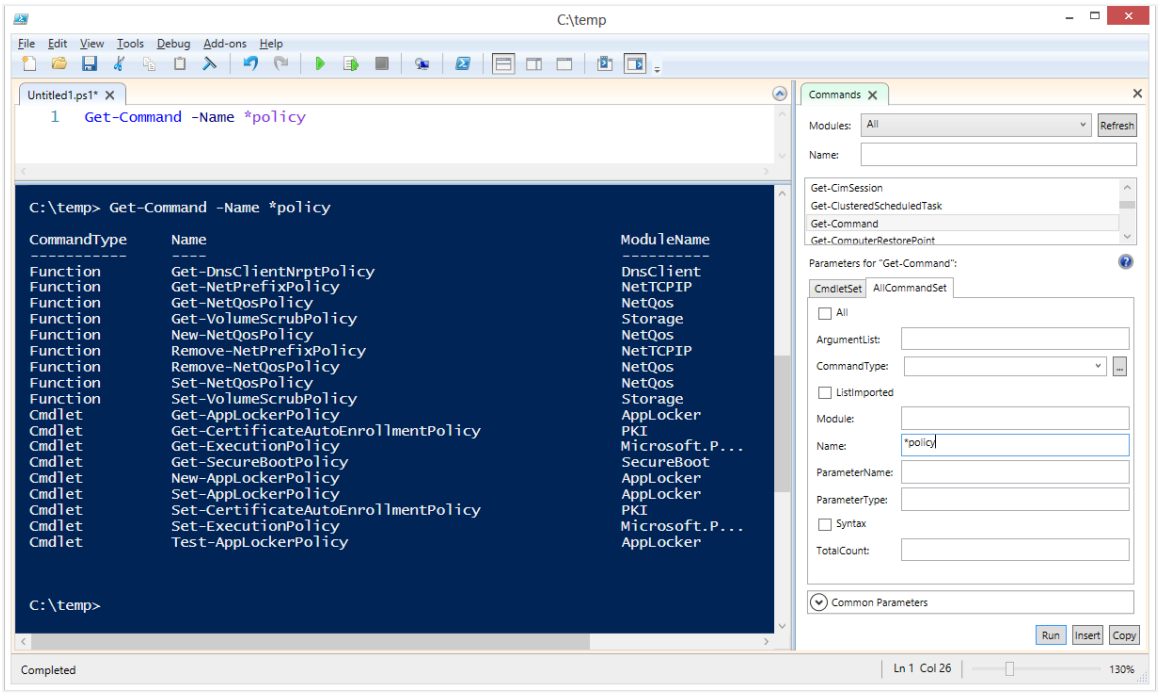
```
Import-Module PSScheduledJob
$j = Get-Job -Name MyScheduledJob
Receive-Job $j
```

New features of Windows PowerShell ISE

The Windows PowerShell 3.0 Integrated Scripting Environment (ISE) includes many new features to ease beginning users into Windows PowerShell and provide advanced editing support for scripters. The following are some of these new features:

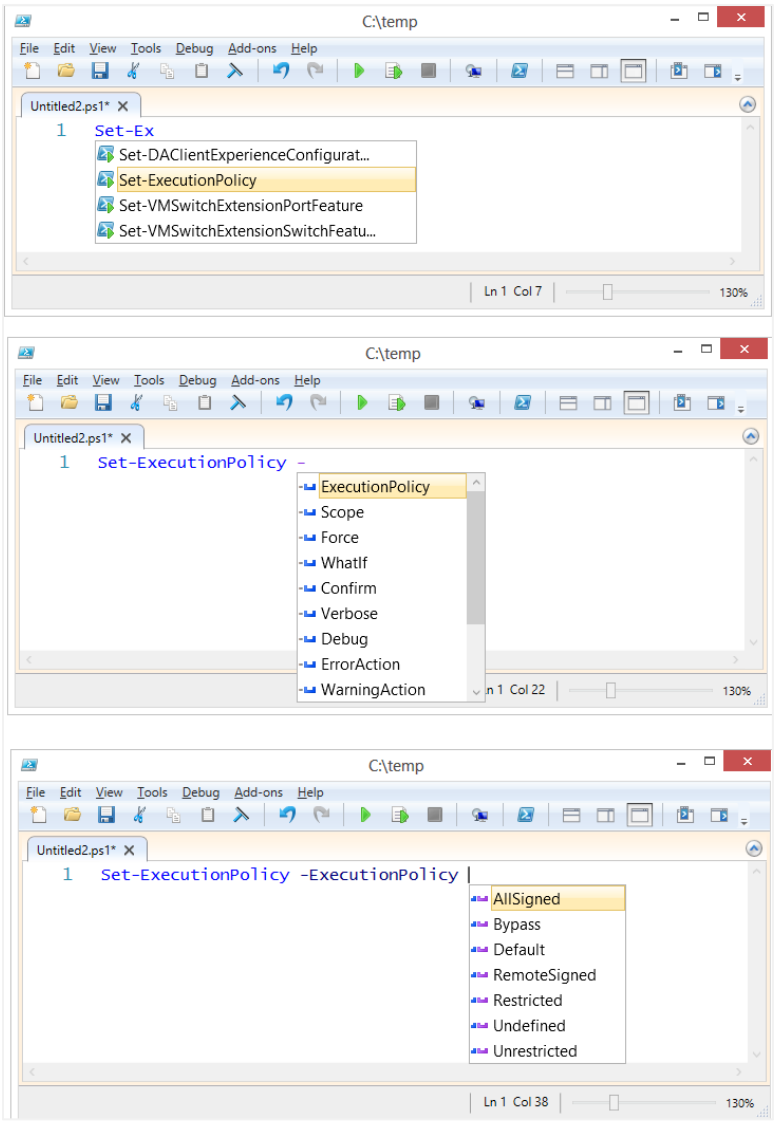
- Show-Command pane lets users find and run cmdlets in a dialog box.

Figure 2: Working with cmdlets in Windows PowerShell ISE



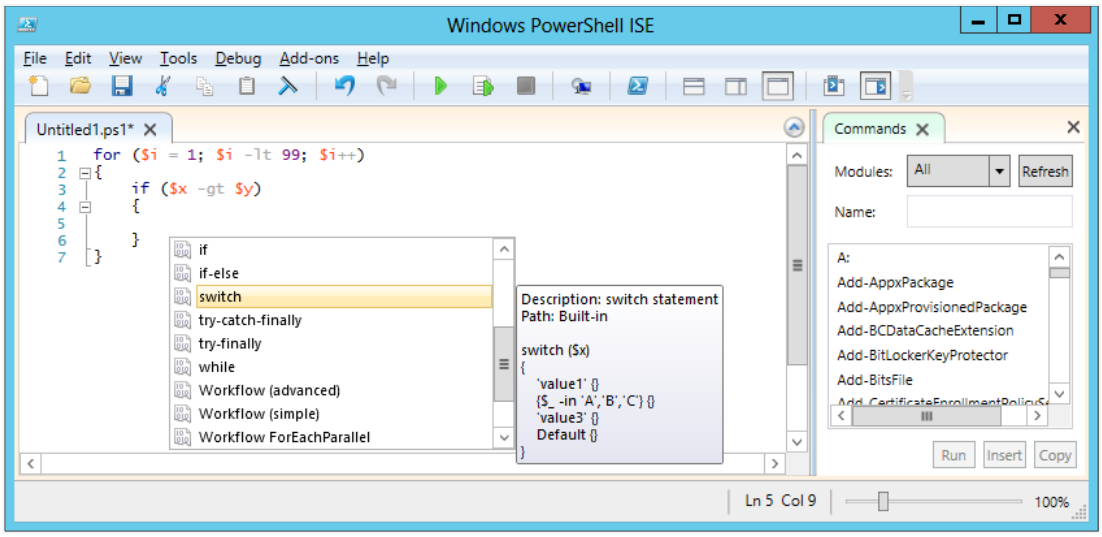
- IntelliSense provides context-sensitive command completion for cmdlet and script names, parameter names and enumerated values, and property and method names. IntelliSense also supports paths, types, and variables.

Figure 3: Context-sensitive command completion with IntelliSense



- Snippets are code examples that allow the user to insert reusable text. The built-in snippets include templates for functions, workflows, and common language patterns so that users do not have to remember the syntax.

Figure 4: Snippets



- Collapsible regions in scripts and XML files make navigation in long scripts easier.

Windows PowerShell Workflow

IT Pros often automate the management of their multicomputer environments by running sequences of long-running tasks or workflows that can affect multiple managed computers or devices at the same time. Windows PowerShell Workflow lets IT Pros and developers apply the benefits of workflows to the automation capabilities of Windows PowerShell.

A workflow is a sequence of automated steps or activities that execute tasks on or retrieve data from one or more managed nodes (computers or devices). These activities can include individual commands or scripts. Windows PowerShell Workflow helps IT Pros and developers to author sequences of multi-machine management activities (which usually are long-running, repeatable, frequent, parallelizable, interruptible, stoppable, or restartable) as workflows. By design, workflows can be resumed from an intentional or accidental suspension or interruption, such as a network outage, reboot, or power loss.

Benefits of Windows PowerShell Workflow

Windows PowerShell Workflow manages the distribution, sequencing, and completion of multicomputer tasks, freeing users and administrators to focus on higher level tasks. The following list describes some of the benefits of Windows PowerShell Workflow:

- **Take advantage of the PowerShell scripting syntax**. IT Pros can reuse their existing PowerShell scripting skills to author script-based workflows using the extended PowerShell language. Apart from being easy to author, PowerShell script-based workflows provide the additional benefit of sharing by simply pasting them into an email or publishing them online.
- **Multi-machine management**. Simultaneously run long-running tasks as workflows on up to hundreds of managed nodes. Windows PowerShell Workflow includes a built-in library of common management parameters for workflows, enabling multi-machine management scenarios such as PSComputerName and PSConfigurationName.

- **Single task execution of complex processes**. You can combine related scripts or commands that act on an entire end-to-end scenario into a single workflow. Status and progress of activities with the workflow are visible at any time.
- **Robustness: Automated failure recovery**. Windows PowerShell Workflow survives both planned and unplanned interruption (such as machine reboots or network flakiness). You can suspend workflow execution and then resume the workflow from the last checkpoint, which is normally the point at which it was suspended.
- **Persistence**. Workflow status and data are saved (or "checkpointed") at specific points defined by its author, so you can resume the workflow from the last persisted task (or checkpoint), instead of restarting the workflow from the beginning.
- **Connection and action retries**. Using workflow common parameters, workflow users can retry the connections to managed nodes in case of network connection failures. Additionally, workflow authors can designate specific activities to run again in case of failure on one or more managed nodes (for example, if one of the computers was down at the time the activity ran).
- **Ability to connect and disconnect**. Users can connect to and disconnect from the machine running the workflow, and the workflow will continue to run. For example, you can log off or restart the computer connected to the workflow machine, and monitor the workflow execution from another computer (such as a home computer)—all without interrupting the workflow. This is possible as long the client is running on a different computer than the workflow engine computer.
- **Scheduling**. Workflow tasks can be scheduled just like any Windows PowerShell cmdlet or script.
- **Workflow and connection throttling**. Workflow execution and connections to nodes can be throttled, enabling scalability and high availability scenarios.

When to use Windows PowerShell Workflow instead of a cmdlet/script

In general, you should consider using a workflow instead of a cmdlet/script when you need to meet any of the following requirements:

- You need to perform a long-running task that combines multiple steps in a sequence.
- You need to perform a task that runs on multiple computers.
- You need to perform a task that requires checkpointing or persistence.
- You need to perform a long-running task that is asynchronous, restartable, parallelizable, or interruptible.
- You need the task to run at scale or in high availability environments, potentially requiring throttling and connection pooling.

Writing and running workflows in Windows PowerShell: Examples

Typically, workflows are started from a client computer and are ideal for executing long-running tasks across multiple target computers. Workflows are like any other Windows PowerShell cmdlet, which means that you can use the **Get-Command** cmdlet to discover them, and the **Get-Help** cmdlet to learn how to use them.

You can add a workflow to a Windows PowerShell session by defining it at the command line, defining it in a script and dot sourcing it, or using the **Import-Module** cmdlet to import a module with a Windows PowerShell script workflow or a XAML-based workflow. Once imported, the workflow then behaves like any other PowerShell command in that session.

Each step or command inside the workflow is called an *activity*. Each activity inherits the properties of the workflow, including the powerful Workflow common parameters mentioned above.

To write a workflow, you can use either the regular PowerShell console or Windows PowerShell ISE. For example, you can type the following workflow into the Windows PowerShell ISE Command pane:

```
Workflow Verb-Noun
{
     Write-Output -InputObject "Hello from Workflow!"
}
```

Notice the new *workflow* keyword, which indicates that the command is a Windows PowerShell Workflow. The keyword adds more than 20 new common parameters to the workflow, allowing users to specify items such as:

- A list of target computers for the workflow (-PSComputerName).
- Credentials to use for running the workflow (-PSCredential).
- Quotas to manage the workflow as the work scales (for example, -PSRunningTimeoutSec).
- Ability to retry the whole workflow or specific activities in case there are connection issues (for example, PSConnectionRetryCount).
- Ability to persist or checkpoint workflow activities, which will save the workflow metadata, output, and errors to disk so you can resume workflow execution at given points during the execution (-PSPersist).

To run a workflow, type the workflow name just like you would to run any other Windows PowerShell command. For example, to run the new workflow we have just created, you can type **Verb-Noun** at the Windows PowerShell ISE prompt.

The following is another example workflow, named "LongWorkflow," that runs for approximately 30 seconds:

```
Workflow Invoke-LongWorkflow
{
     Write-Output -InputObject "Loading some information..."
     Start-Sleep -Seconds 10
     Write-Output -InputObject "Performing some action..."
     Start-Sleep -Seconds 10
     Write-Output -InputObject "Cleaning up..."
     Start-Sleep -Seconds 10
}
```

Because this workflow defines a long task, you might want to run it as a background job. To do so, you can use the **AsJob** parameter, along with the **JobName** parameter to assign the "LongWF" name to the job.

Example:

```
Invoke-LongWorkflow –AsJob –JobName LongWF
```

The following example is a more complex workflow. This workflow, "Install-VM," creates virtual machines on managed nodes, starts the virtual machines, and joins them to a domain (which requires a reboot of the virtual machines).

Example:

```
<# This is a long running workflow that installs VMs on a Hyper-V capable host. This
workflow showcases the new PowerShell Workflow feature set of Window PowerShell 3.0.
In this particular example, the managed nodes must be Hyper-V capable with Hyper-V
role/module installed. #>

# Workflow that installs VMs on a Hyper-V capable host
workflow Install-VM
{
    param
    (
        # Full path to base Vhd for VMs
        [Parameter(Mandatory=$true)]
        [String]$BaseVhdPath,
        # Prefix for VM names
        [String]$VMNamePrefix = "Demo",
        # Number of VMs to create
        [Int]$VMCount = 3,
        # Domain credential required to join the VMs to a domain
        [Parameter(Mandatory=$true)]
        [System.Management.Automation.PSCredential] $domainCred,
        # Local credential required to connect to VMs before being joined to domain
        [Parameter(Mandatory=$true)]
        [System.Management.Automation.PSCredential] $localCred
    )

    # Create VMs in parallel
    foreach -parallel($i in 1..$VMCount)
    {
        # Create the VM name
        [string]$VMName = $VMNamePrefix+$i

        # Full path for the differencing VHDs
        [string]$VhdPath = (Split-Path $BaseVhdPath) + "\" + $VMName +".vhd"

        # Create differencing VHDs
        $DiffVHD = New-VHD -ParentPath $BaseVhdPath -Path $VhdPath

        # Create New VM with the differencing VHD etc
        $null = New-VM -MemoryStartupBytes 1GB -Name $VMName `
                       -VHDPath $DiffVHD.Path -SwitchName "InternalSwitch"
    }

    # Save the workflow state and data
    Checkpoint-Workflow

    # Start VMs in parallel and collect their IP addresses
    $IPAddresses = foreach -parallel($i in 1..$VMCount)
    {
        # Create the VM name
        [string]$VMName = $VMNamePrefix+$i

        # Start the VM
        Start-VM -Name $VMName

        # Wait for IP Address to be assigned to each VM.
        # Use Inlinescript to check for VMs IP address
        $VMIP = Inlinescript
```

```
                {
                    (Get-VM -Name $using:VMName).NetWorkAdapters.IPAddresses
                } -DisplayName "Get-VMIPAddress"

        while($VMIP.count -lt 2)
        {
            # Use Inlinescript to check for VMs IP address
            $VMIP = Inlinescript
                    {
                        (Get-VM -Name $using:VMName).NetWorkAdapters.IPAddresses
                    } -DisplayName "Get-VMIPAddress"

            # Notify user via progress stream
            Write-Progress -Id $i -Activity "Get-VMIPAddress on $VMName" `
                           -Status "Waiting for IP Address ..."

            # Wait for 5 seconds and retry
            Start-Sleep -Seconds 5;
        }
        $VMIP[0]
    }

    # Show the IPs collected to workflow user
    $IPAddresses

    # Before suspending the workflow (say for checking some settings, freeing up
resources),
    # send mail to senior admin notifying the suspended state of workflow
    Send-MailMessage -From "juniorAdmin@contoso.com" -To "seniorAdmin@contoso.com" `
                     -SMTPServer "your SMTP sever" -PSComputerName "" `
                     -Subject "Suspended workflow $jobCommandName requires
attention" `
                     -Body `
                     @"
        A workflow running on $hostname with name $jobCommandName requires your
attention.
        Please use Resume-Job cmdlet to resume the workflow execution
"@

    # Suspend the workflow execution
    Suspend-Workflow

    # Call the Join-Domain workflow to join the VMs to the domain
    Join-Domain -PSComputerName $IPAddresses -PSCredential $localCred -domainCred
$domainCred

    # Send mail to senior admin notifying the completion of workflow
    Send-MailMessage -From "juniorAdmin@contoso.com" -To "seniorAdmin@contoso.com" `
                     -SMTPServer "your SMTP sever" -PSComputerName "" `
                     -Subject "Workflow $parentjobname with
InstanceID:$parenetjobinstanceid has completed" `
                     -Body `                      `
                     @"
        A workflow running on $hostname with name $jobCommandName completed
successfully.
        Please use Receive-Job cmdlet to see the output of workflow execution
"@
}

# Workflow that will join a machine to a domain
workflow Join-Domain
{
    param(
```

```
         [string] $domainName="fourthcoffee.com",
        [Parameter(Mandatory=$true)]
        [System.Management.Automation.PSCredential] $domainCred
    )

    # Check that the machine is joined to WORKGROUP
    Get-CimInstance -ClassName CIM_ComputerSystem

    # Add the machine to domain and restart
    Add-Computer -DomainName $domainName -LocalCredential $PSCredential -Credential
$domainCred
    Restart-Computer -Wait -For WinRM -Force -Protocol WSMan

    # Notice that now it is joined to domain!
    Get-CimInstance -ClassName CIM_ComputerSystem
}
```

Cmdlet discovery: Get-Command and module auto-loading

Windows Server 2012 includes **more than 2,300 cmdlets** that you can easily find and learn. Modules are easier than ever to find, explore, create, and use, and users no longer have to import modules manually to use cmdlets. Users can run a cmdlet, and Windows PowerShell will automatically import the module. In addition, the Get-Command has been updated to find all cmdlets installed on the system. For example, to find all networking cmdlets, you can run Get-Command *-Net*.

Syntax simplification

Windows PowerShell 3.0 includes simplified, consistent syntax across all cmdlets. The ForEach-Object and Where-Object cmdlets have been updated to support an intuitive command structure that more closely models natural language. Users are able to construct commands without script block, braces, current object automatic variable ($_), or dot operators to get properties and methods. In short, the "punctuation" that plagued beginning users is no longer required.

PowerShell 2.0 Where-Object syntax:

```
get-process | where {$_.handles –gt 800}
```

PowerShell 3.0 simplified Where-Object syntax:

```
get-process | where handles –gt 800
```

PowerShell 2.0 ForEach-Object syntax:

```
get-process | foreach {$_.name}
```

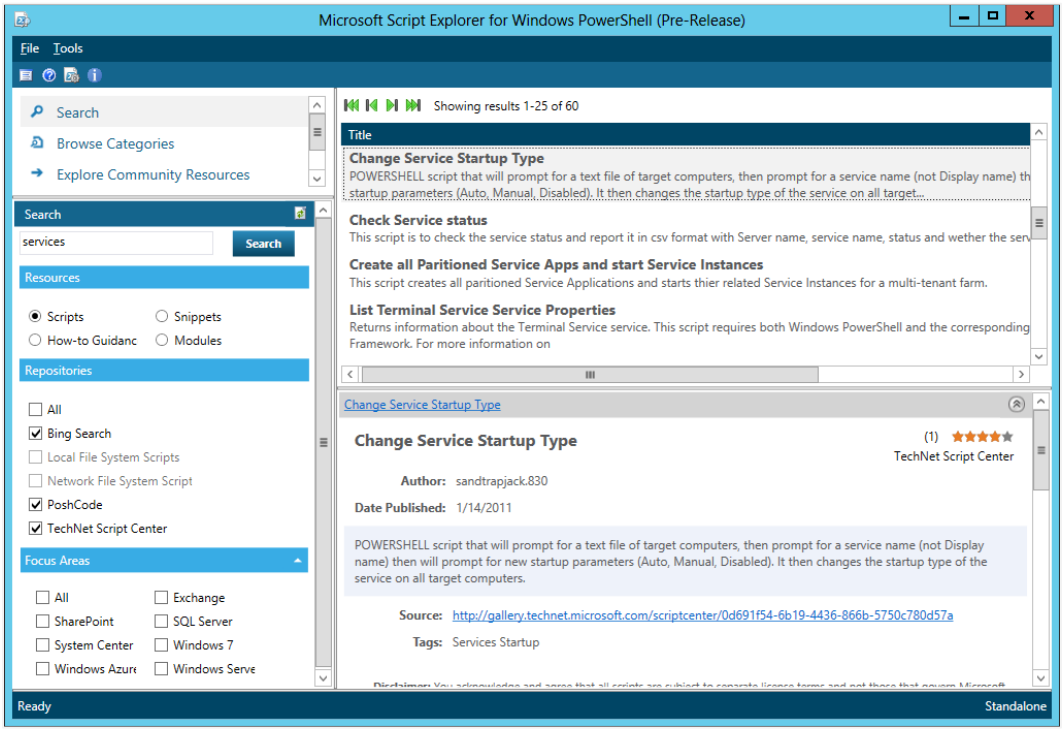
PowerShell 3.0 simplified ForEach-Object syntax:

```
get-process | foreach name
```

Script Explorer

Windows PowerShell 3.0 helps IT Pros by providing access to a community-generated library of Windows PowerShell scripts, modules, and how-to guidance. To access these scripts, the user needs to install Script Explorer for Windows PowerShell from the Microsoft Download Center.

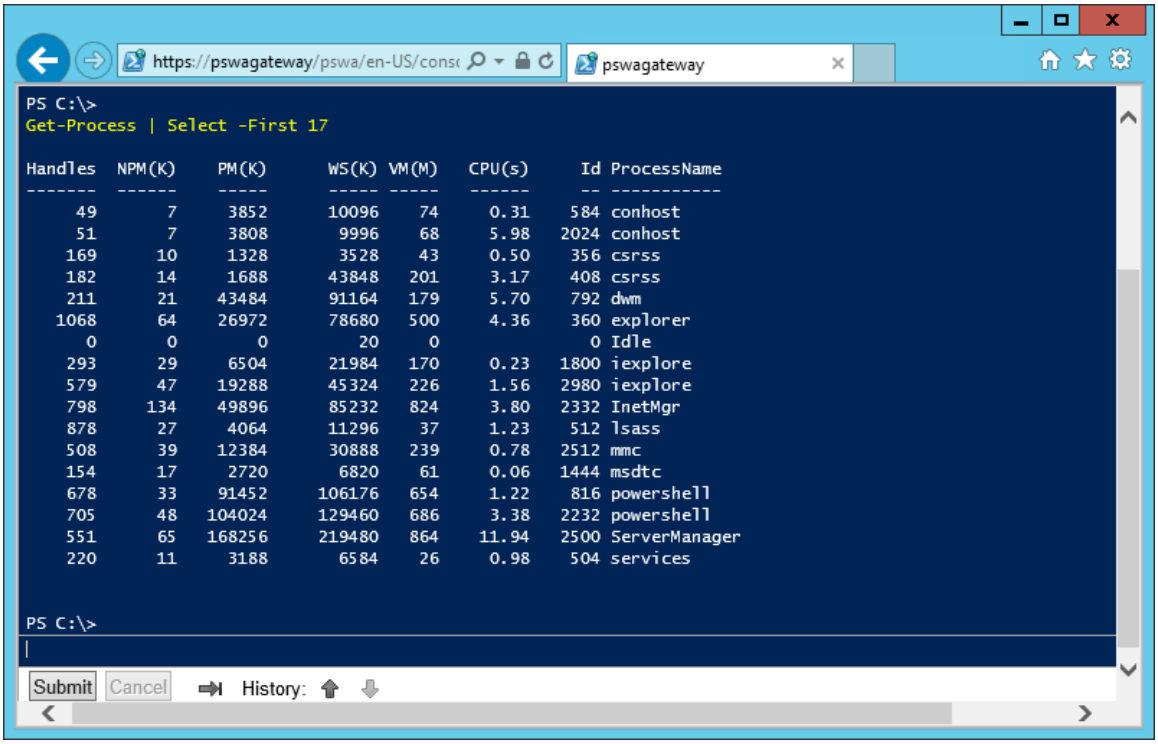
Figure 5: Microsoft Script Explorer for Windows PowerShell



Windows PowerShell Web Access

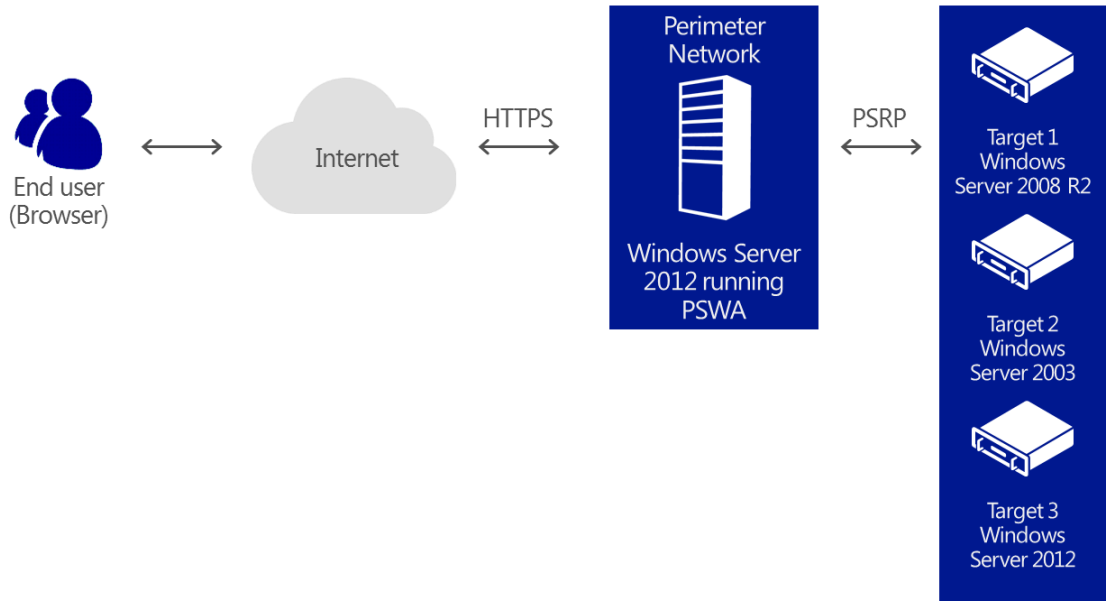
Windows PowerShell Web Access is a new feature in Windows Server 2012 that lets you manage Windows servers by using Windows PowerShell in a web browser. The target computers to be managed can run any version of Windows that is enabled for Windows PowerShell remoting.

Figure 6: Windows PowerShell Web Access



To manage the remote server through Windows PowerShell in a web browser, you connect to a server that is running Windows Server 2012 and has the Windows PowerShell Web Access feature installed. This server acts as a gateway that serves the web pages containing a Windows PowerShell interface to the remote clients. The following illustration shows this infrastructure:

Figure 7: PowerShell Web Access infrastructure



Updatable Help

Windows PowerShell 2.0 included extensive help topics that were frequently updated online. However, because the help files were part of the Windows operating system, users could not update them, and the help topics that were displayed at the command line could soon become outdated. Third-party products had to convert online help to XML or display outdated help topics.

In Windows PowerShell 3.0, new Update-Help and Save-Help cmdlets download and install the newest help files for each module. The cmdlets find the help files on the Internet, determine whether they are newer than local files, unpack them, and then install them in the correct location. The updated files are ready for immediate use in Get-Help; you do not have to restart Windows PowerShell. Help files for Windows PowerShell 3.0 are up to date on first use because they do not ship "in the box." Get-Help displays auto-generated help for commands, and then prompts you to use the Update-Help cmdlet to install or update the help files for your modules.

For some environments, such as large enterprises behind Internet firewalls, it is preferable to be able to update help files from a local share instead of from the Internet. In these cases, you can use Save-Help –DestinationPath <*share*> to create a local share that stores the latest Windows PowerShell help files. Users within the organization can then update their help files by pointing to that share and running Update-Help –SourcePath <*share*>.

Updatable Help is available for all modules, including third-party modules, and includes support for multiple languages.

Session Configuration Files

Windows PowerShell simplifies the process of defining a new session configuration by allowing the administrator to specify the configuration in a declarative manner, using name-value pairs in a PowerShell data file. For most settings, this is much simpler than writing a PowerShell script. It's also easier to understand how a session configuration is defined by inspecting the file.

RunAs capability

When remote administration is delegated, scenarios can result where users lack the credentials required to perform needed tasks. With Windows PowerShell 3.0, administrators can configure sessions so that certain commands are run by default with the credentials of a different user. Credentials are stored securely in the WSMan provider.

For example, to change the credentials under which commands will be executed in the regular PowerShell endpoint, you would execute the following:

```
cd WSMan:\localhost\Plugin\microsoft.powershell
$cred = Get-Credential
Set-Item .\RunAsUser $cred
```

You must restart the Windows Remote Management (WinRM) service for the changes to take effect.

Default Parameter Values

The new $**PSDefaultParameterValues** preference variable in Windows PowerShell 3.0 lets you specify default values for cmdlet parameters. You can set values for a parameter on a particular cmdlet or a set of cmdlets that match a wildcard expression.

The value of $PSDefaultParameterValues is a hash table that consists of a collection of key/value pairs. Each key consists of a command name and a parameter name separated by a colon. The command name and/or the parameter name can be enclosed in quotation marks ("CommandName":"ParameterName").

To override a default parameter value, add an explicit parameter value to the command. To disable all default parameter values, enter the following key/value pair: "**Disable=$true**".

By default, the value of $PSDefaultParameterValues is session-specific. To set it for all Windows PowerShell sessions, add the $PSDefaultParameterValues variable to your Windows PowerShell profile.

Example:

```
$PSDefaultParameterValues=@{Invoke-
Command:ConfigurationName="AdminSession.PowerShell";*-Job:Verbose=$true}
```

New cmdlets

Windows PowerShell 3.0 includes more than 2,300 new cmdlets that expand its power and reach. The following table includes a partial list of new cmdlets included in Windows PowerShell 3.0.

Table 1: New cmdlets in Windows PowerShell 3.0

cmdlet	function
Get-CimAssociatedInstance	Gets Common Information Model (CIM) instances connected to the given instance via an association.
Get-CimClass	Enables the user to enumerate the list of CIM Classes under a specific namespace.
Register-CimIndicationEvent	Subscribes to indications using the Filter Expression or Query Expression.
Get/New/Remove/Set-CimInstance	Gets, creates, removes, or edits a CIM instance on the server. For Get-CimInstance, the instance contains only the properties specified in the **Property** parameter, **KeyOnly** parameter, or the **Select** clause of the **Query** parameter.
Invoke-CimMethod	Invokes a method on a CIM object.
Get/New/Remove-CimSession	Gets, creates, or removes a CIM session on the client representing a connection with a remote computer.
New-CimSessionOption	Creates an instance of a CimSessionOption, which can be used as an argument to the New-CimSession cmdlet.
Show-Command	Shows a graphical representation of a cmdlet as a Windows form.
Rename-Computer	Renames a computer.
Get/Show-ControlPanelItem	Gets a list of Control Panel applets installed on the local computer. Show-ControlPanelItem is used to launch the Control Panel applet.
Unblock-File	Removes the ZoneTransfer alternate NTFS stream (for example, the “Downloaded From Internet” stream).

cmdlet	function
Save/Update-Help	Save-Help: Exports the currently installed help files to a location on the File System. Update-Help: Downloads help files from the Internet or a file share, and then installs them on the local computer.
Resume/Suspend-Job	Suspends or resumes a job. These cmdlets currently only work with Workflow Jobs.
Add/Disable/Enable/Get/New/ Remove/Set-JobTrigger	Manipulates job triggers that define when a scheduled job will execute.
ConvertFrom/ConvertTo-Json	Converts objects to/from a JSON-formatted string representation.
Connect/Disconnect/Receive-PSSession	Connects/disconnects from a remote session. Receive-PSSession resumes execution of a command in a disconnected session and gets the session output (implicitly reconnecting to the session).
New/Test-PSSessionConfigurationFile	Creates or validates a PSSession Configuration File that can be used to create a constrained endpoint.
New-PSTransportOption	Creates a new PSTransportOption object.
New-PSWorkflowExecutionOption	Creates an object that contains session configuration options for workflow sessions.
Invoke-RestMethod	Makes an HTTP or HTTPS request to a RESTful web service and returns the response.
Disable/Enable/Get/Register/Set/Unregister-ScheduledJob	Manipulates scheduled jobs on the computer.
Get/New/Set-ScheduledJobOption	Gets, creates, or sets an object that can be used to specify advanced configuration for Scheduled Jobs.
Get/Remove-TypeData	Gets or removes TypeData.
Invoke-WebRequest	Makes an HTTP or HTTPS request to a web service and returns the response.
New-WinEvent	Creates an event in the event log.

Summary

The following features of Windows PowerShell 3.0 offer comprehensive, resilient, and simple automation of your Windows Servers:

- More than 2,300 new cmdlets that are easy to find and execute.
- Workflows that automate long running tasks across multiple computers in a resilient way.
- Disconnected Sessions to start execution on a computer and return to it later (possibly from another computer).
- The ability to delegate a set of credentials that will be used when commands are run in certain sessions.
- Job scheduling that lets you run your scripts and workflows according to your defined schedule and stores results for later retrieval.

Multiserver management and feature deployment with Server Manager

In Windows Server 2012, the capabilities of Server Manager have expanded considerably to facilitate multiserver tasks, such as remote role and feature deployment to both physical and virtual servers, remote role and feature management, and custom server group creation.

By using Server Manager in Windows Server 2012, IT Pros now can provision servers and offline virtual hard disks from their desktops without requiring either physical access to the system or Remote Desktop Protocol (RDP) connections to each server. Server Manager also helps administrators manage groups of servers collectively from a single, integrated console, allowing them respond to business-critical problems with greater speed and agility.

Technical description

Server Manager in Windows Server 2012 has evolved to include many new multiserver management features. The following sections describe some of these new capabilities.

Multiserver experience

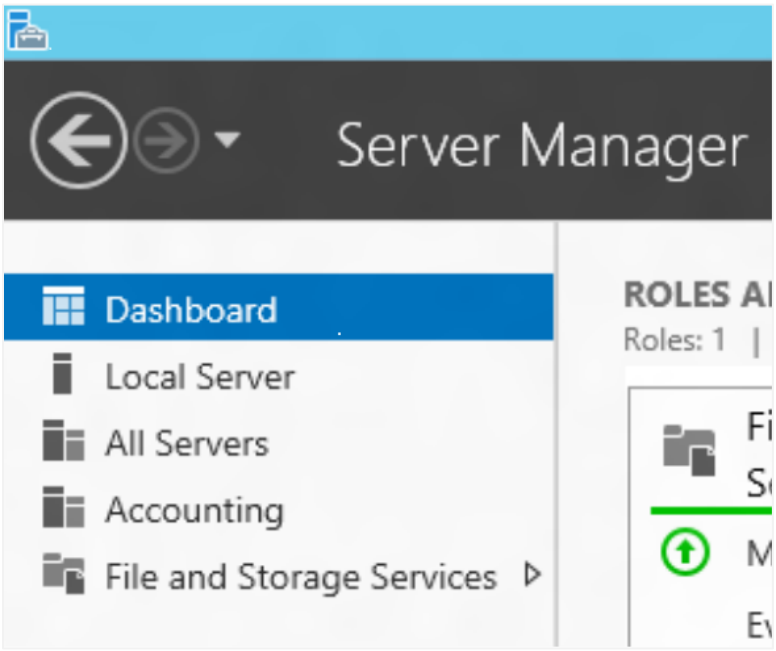
Server Manager can manage multiple servers in a server pool, and create server groups to organize them. Groups let you organize your servers into logical views—for example, My Seattle Servers or My Test Servers. By default, Server Manager groups the servers by role.

Improved management through high availability

Windows Server 2012 introduces new storage and networking features that improve manageability by preventing downtime by enduring various failures while maintaining service availability. For example, Windows Server 2012 introduces Server Message Block (SMB) 3.0, which improves the availability of server applications through features such as SMB Transparent Failover and SMB Multichannel, which make effective, fault-tolerant use of multiple NICs. Another feature, NIC Teaming, supports multichannel traffic and failover for traffic that is not SMB-based.

For more information about availability improvements and SMB 3.0, see the white paper, "Windows Server 2012 Storage." For more information about NIC Teaming, see the white paper, "Windows Server 2012 Networking."

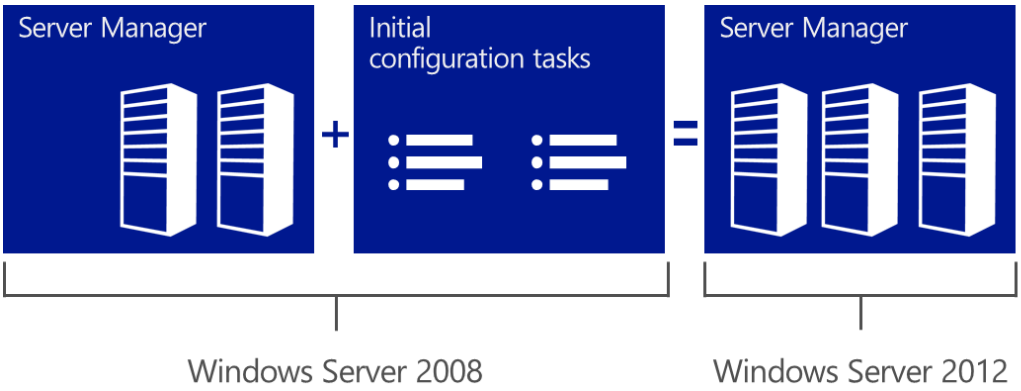
Figure 8: Server group in Server Manager



Streamlined server configuration and deployment

In Windows Server 2012, Server Manager includes configuration functionality previously provided by the Initial Configuration Tasks window. The result is a single surface for managing the configuration of Windows Server and its roles and features.

Figure 9: Combined tool functionality in Windows Server 2012 Server Manager



Efficient deployment of workloads to a remote server or offline virtual hard disk

In Windows Server 2008, roles and features are deployed by using the *Add Roles Wizard* or *Add Features Wizard* in Server Manager running on a local server. This requires either physical access to the server or Remote Desktop access by using RDP. Installing the Remote Server Administration Tool lets you run Server Manager on a Windows-based client computer, but adding roles and features is disabled because remote deployment is not supported.

In Windows Server 2012, the deployment capabilities are extended to support robust remote deployment of roles and features. Using Server Manager in Windows Server 2012, IT Pros can provision servers from their desktops without requiring either physical access to the systems or the need to enable an RDP connection to each server.

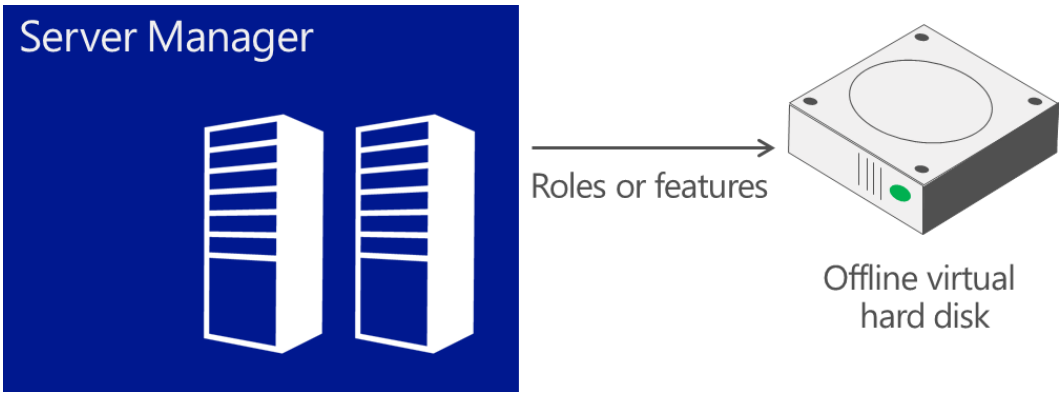
Installing roles and features on a remote server or offline virtual hard disk

Windows Server 2012 with Server Manager can deploy both roles and features in a single session using the unified *Add Roles and Features Wizard*. The Add Roles and Features Wizard in Windows Server 2012 performs validation passes on a server that you select for deployment as part of the installation process. You do not need to separately pre-verify that a server is properly configured to support a role.

Administrators can deploy roles and features to remote servers and offline virtual hard disks from Server Manager. In a single session with the Add Roles and Features Wizard, you can add your desired roles and features to an offline virtual hard disk, allowing for faster and simpler repetition and consistency of desired configurations.

With the Add Roles and Features Wizard, the process of installing roles is familiar (and also consistent with the Add Roles Wizard in earlier Windows Server releases); however, there are some changes. To support remote deployment and installation on offline virtual hard disks, some roles have moved some initial configuration (tasks formerly performed in the Add Roles Wizard) into post-installation configuration wizards. For some offline virtual hard disk deployments, installation tasks are scheduled to run the first time the virtual machine is started.

Figure 10: Deployment of roles or features to an offline virtual hard disk



Batch deployment

In Windows Server 2012, the Add Roles and Features Wizard lets you export configuration options to an XML file for later use with Windows PowerShell deployment cmdlets. By using the fan-out capabilities of Windows PowerShell, you can perform batch deployment of roles and features on multiple remote servers, applying configuration settings that were saved during a previous wizard-based deployment.

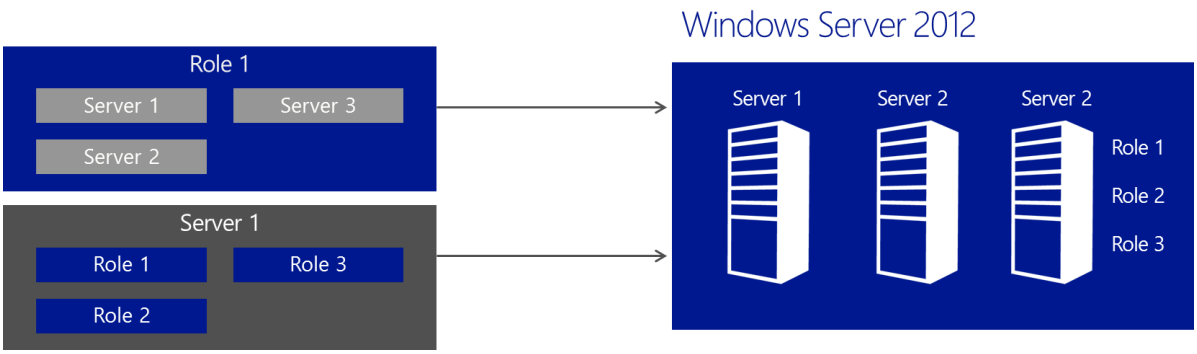
Integration with other management tools

Server Manager remains the key access or starting point for server management tools. Where supported, Server Manager starts these tools in the context of the remote server that you are managing. New, modern, role-specific tools (such as File Storage Management, Remote Desktop Services, and IP Address Management) are integrated into the Server Manager console.

Server role management across multiple servers

Management of server roles is improved by shifting from a single-server, single-role model to one in which multiple server roles can be managed remotely by using a single management application.

Figure 11: Management of server roles (such as File Services) across multiple servers



Remote Desktop Services configuration

Remote Desktop Services provides session virtualization and virtual desktop infrastructure (VDI) technologies that enable users to access session and virtual desktop collections. In Windows Server 2012, new management features of Server Manager simplify how Remote Desktop Services is deployed and managed in a multiserver environment. Scenario-based deployment reduces the complexity of installing different Remote Desktop Services components across multiple servers based on how Remote Desktop Services will be used. New multiserver management tools then simplify how administrators manage different servers that are running Remote Desktop Services role services and virtual desktop infrastructures.

Minimal performance impact

The Server Manager dashboard has a default 10-minute polling cycle that users can modify in the console. By using a relatively infrequent default polling cycle and returning only incremental data with each poll, the performance-load impact on individual servers is minimized. Server Manager uses new WMI providers and Windows PowerShell cmdlets to pull updated status information from servers.

Requirements

Server Manager in Windows Server 2012 requires the following prerequisites to be met:

- Remote deployment is only supported to computers that are running Windows Server 2012. Remote deployment of roles and features to Windows Server 2008 and earlier versions of Windows is not supported.
- To remotely manage a server, the value of the **Remote Management** property must remain **Enabled** on the **Local Server** page in the Server Manager console on that server. (This property is enabled by default in both Server Manager and Windows PowerShell.)

Note: Server Manager will not run on a Server Core Installation.

Remote Server Admin Tools

The preferred deployment option for Windows Server 2012 is Server Core. As Server Manager requires the server to be running either the Minimal Server Interface or Server with a GUI, Remote Server Admin Tools (RSAT) enables IT administrators to manage roles and features installed on computers running Windows Server 2012 from a remote computer running Windows 8. RSAT is available from the Microsoft Download Center.

Summary

Windows Server 2012 Server Manager helps to improve manageability in the datacenter so you can:

- Manage multiple servers easily, with a clear and powerful role-centric dashboard.
- Simplify the processes of configuring new servers.
- Deploy roles and features even to remote servers and offline virtual hard disks.
- Consult a single tool for a clear summary of multiple server states.
- Manage Windows Server 2012 from Windows 8 using RSAT.

Conclusion

IT Pros today face the challenge of managing and maintaining an increasing number of mission-critical servers and services, all with fewer resources. Windows Server 2012 addresses this problem by adopting enhanced standard models, protocols, and APIs, and by offering new and improved features in Windows PowerShell and Server Manager. Together, these enhancements help administrators manage multiserver environments more efficiently and cost effectively.

List of charts, tables, and figures