Windows Server

Windows Server 2012 R2

サーバーの管理性および自動化

Microsoft

著作権情報

目次

はじめに 5

Windows Server 2012 R2 での標準ベースの管理アプローチ 6

技術仕様 6

標準 API の向上 6

標準プロトコル 8

標準の管理ツール 10

Windows PowerShell での毎日の反復タスクの簡素化 14

技術仕様 14

堅牢なセッション接続 14

切断されたセッション 14

ジョブ スケジュール 15

Windows PowerShell ISE 16

Windows PowerShell ワークフロー 19

Windows PowerShell Web アクセス 28

更新可能なヘルプ 30

新しいコマンドレット 30

Desired State Configuration 32

構成の定義 34

ノードおよび構成ブロックをネストする場合の規則 36

構成パラメーターの宣言 37

まとめ 38

サーバー マネージャーでのマルチサーバー管理および機能展開 39

技術仕様 39

マルチサーバーのエクスペリエンス 39

リモート サーバーまたはオフライン仮想ハード ディスクへのワークロードの効率的な展開 40
リモート サーバーまたはオフライン仮想ハードディスクへの役割と機能のインストール 41
バッチ展開 42
他の管理ツールとの統合 42
複数のサーバーにわたるサーバーの役割の管理 42
パフォーマンスへの最小の影響 44
リモート サーバー管理ツール 44
まとめ 44
結論 45

はじめに

パブリック クラウドの構築および運用に関するマイクロソフトの実績がサーバー オペレーティング システムに組み込まれている Windows Server 2012 R2 は、パブリック クラウド向けの優れた可用性を備え、コスト効果の高い動的なプラットフォームです。Windows Server 2012 R2 は、社内全体で安全に接続するスケーラブルかつ動的でマルチテナント対応のクラウド インフラストラクチャの基盤をビジネス プロバイダー およびホスト プロバイダー向けに提供し、より迅速かつ効率的にビジネス ニーズに応えることを可能にします。

Windows Server 2012 R2 は、包括的なマルチコンピューターの管理性を備える統合プラットフォームとして、さらに優れた総保有コスト (TCO) を提供します。Windows Server 2012 R2 では、3 つの方法でマルチコンピューター管理が向上します。

- 標準ベースの管理アプローチ: Windows Server 2012 R2 で使用されている業界標準へのフォーカスにより、Windows と Window 以外のデバイスの両方を同時に管理できる高度な管理性が実現します。また、その管理性はネットワーク スイッチや記憶域などのハードウェアにも拡張されます。
- Windows PowerShell 4.0: PowerShell 3.0 の包括的な自動化機能の上に構築された Windows PowerShell 4.0 では、日常的なタスクを異種環境で効率的に自動化することができます。
- サーバー マネージャー: Windows Server 2012 R2 のサーバー マネージャーは、物理サーバーであるか仮想サーバーであるかに関係なく、ローカル サーバーとリモート サーバーの役割と機能を展開および管理するのに役立ちます。

以降のセクションでは、Windows Server 2012 R2 の豊富な機能の詳細について説明します。

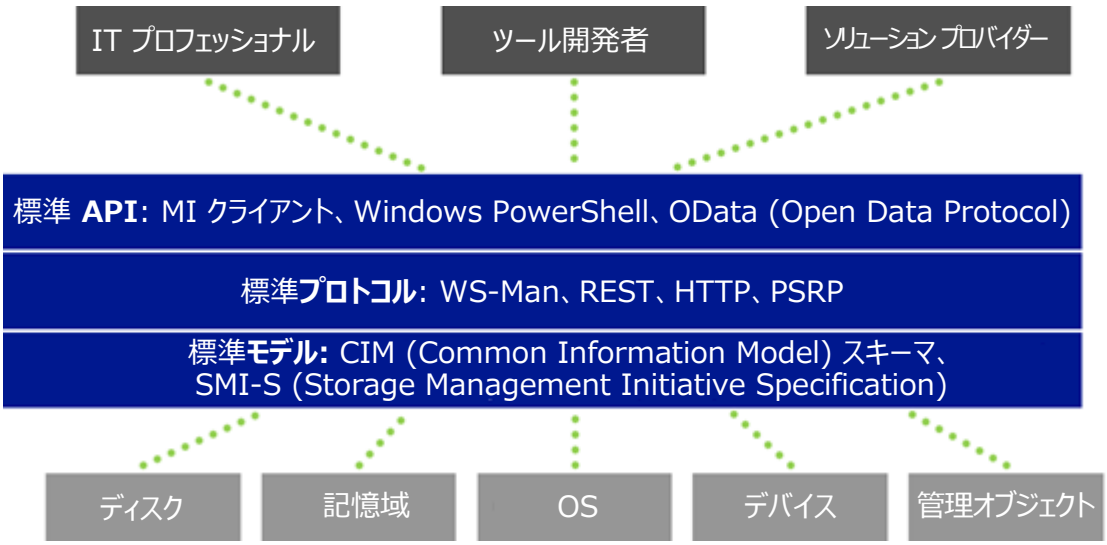
Windows Server 2012 R2 での標準ベースの管理アプローチ

技術仕様

Windows Server 2012 R2 は、標準ベースの強化された管理フレームワークを通して、データセンターおよびクラウド環境内の管理エクスペリエンスを向上させます。Microsoft Windows は、標準ベースの管理をサポートし、DMTF (Distributed Management Task Force) などの組織に参加しています。このような努力は、Windows Management Instrumentation (WMI) および Windows リモート管理 (WinRM) として実を結びました。WMI は Common Information Model (CIM) オブジェクト マネージャーの Windows 実装で、WinRM は Web サービス マネジメント (WS-Man) プロトコルの実装です。CIM と WS-Man は共に DMTF によってリリースされている標準です。

Windows Server 2012 R2 は、標準ベースのインフラストラクチャの大幅な向上を通してデータセンターの管理性を強化します。これは、開発者と IT プロフェッショナルにとって使いやすいアプリケーション プログラミング インターフェイス (API) を提供することによって実現されます。これらの API は、最新の標準のサポートを提供して、データタセンター内の複数のサーバーとデバイスへの接続および管理を簡素化し、コスト効果を向上させる新しい種類の Windows PowerShell コマンドレットを追加します。

図 1: Windows の標準ベースの管理コンポーネントの向上



標準 API の向上

WMI のリリース以降、WMI プロバイダーを使用する多くの管理製品およびツールが登場しましたが、プロバイダーの比例成長に一致するものではありませんでした。WMI プロバイダーを記述するには、コンポーネント オブジェクト モデル (COM) コーディングの深い知識が必要でした。そのため、開発者にとっては、プロバイダーの記述は時間がかかるだけで利点の少ない作業でした。

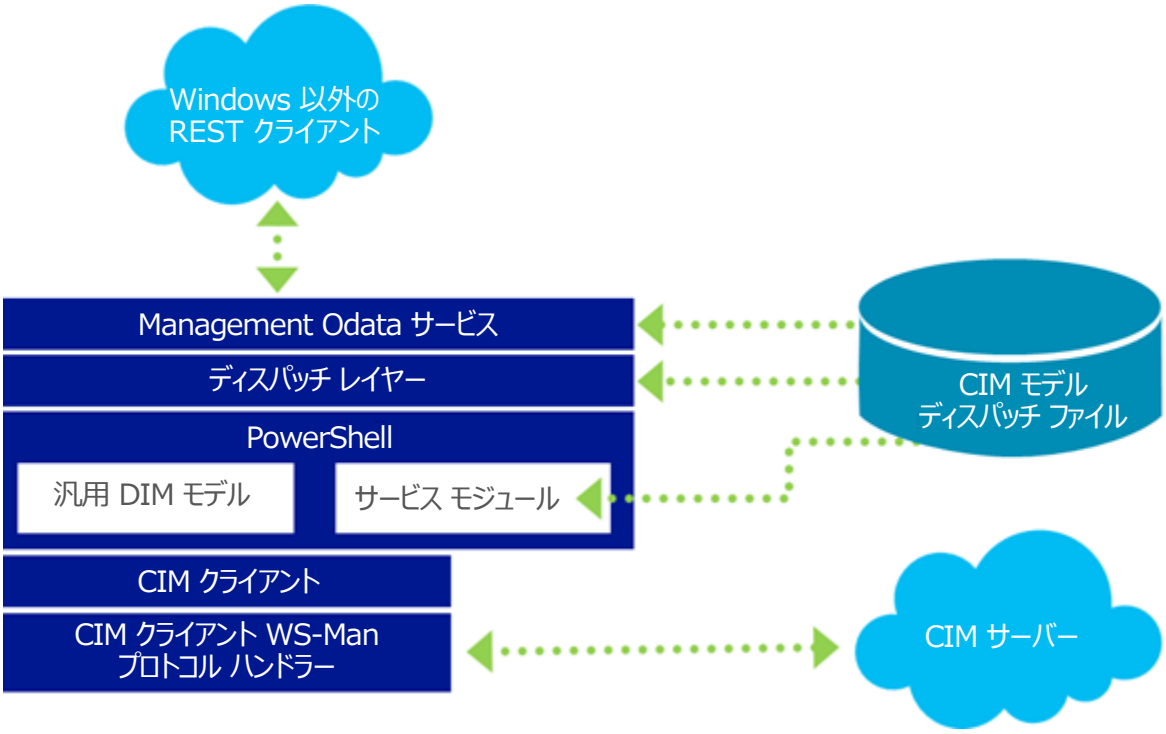
Windows Server 2012 R2 では、Windows 用の Management Infrastructure API (MI API) が導入され、新しいプロバイダーおよびクライアントアプリケーションの開発が飛躍的に簡素化されました。これらの新しい MI API は、クライアント開発においてネイティブ コード (C/C++) と管理コード (.NET) の両方で使用できます。プロバイダー開発では、ネイティブ コードで使用できます。API を使用することによって COM コーディングは不要になります。API には、MOF ファイルに記述されているクラス定義からコード スケルトンとスキーマを生成するツールが付属しています。そのため、プロバイダー開発は非常に容易で、開発者は、ビジネス ロジックの開発に多くの時間を割くことができます。新しい MI API を使用して記述されたプロバイダーは、WMI の以前のリリースからだけでなく、Windows Server 2012 R2 システムへの接続に最新の DMTF WS-Man 標準を使用する Windows 以外のクライアント アプリケーションからも呼び出すことができます。

さらに、新しい MI API は、旧式の WMI API よりも密に CIM インフラストラクチャ標準に準拠するよう更新されていて、既定で、サーバー間の通信に標準 WS-Man プロトコルを使用します。これらの標準への準拠は、その プロトコルを使用します。これらの標準への準拠は、その他の Windows サーバー以外にも最新の DMTF CIM およびプロトコル標準をサポートするあらゆるサーバーやデバイスを管理できる MI API を使用してアプリケーションを記述できるということを意味します。

Windows 以外のプラットフォームから Windows を管理する必要のある Web 開発者のために、Windows Server 2012 R2 には、Management OData IIS 機能拡張が含まれています。これには、REST API (OData サービス エンドポイント) の構築を簡素化するツールおよびコンポーネントが含まれます。

OData は、REST API を構築するための URI
規則、ツール、コンポーネント、およびライブラリのセットです。OData サービスの特長は、そのデータ コンテンツおよび動作を定義する明示的なドメイン モデルをベースとしていることです。そのため、リッチ クライアント ライブラリ (Windows/IoS/Android 端末、ブラウザー、Python、Java など) を自動的に生成でき、広範なデバイスおよびプラットフォーム上でのソリューションの開発が簡素化されます。

図 2: CIM モデルおよび Odata を使用する標準 API の向上



標準プロトコル

標準ベースの管理のもう 1 つの課題は、標準管理プロトコルの定義および可用性です。複数のベンダーが複数のプラットフォームで複数の管理ツールおよびインターフェイスを作成する中で、これらの環境の管理にお関連する複雑さは増大を続けています。

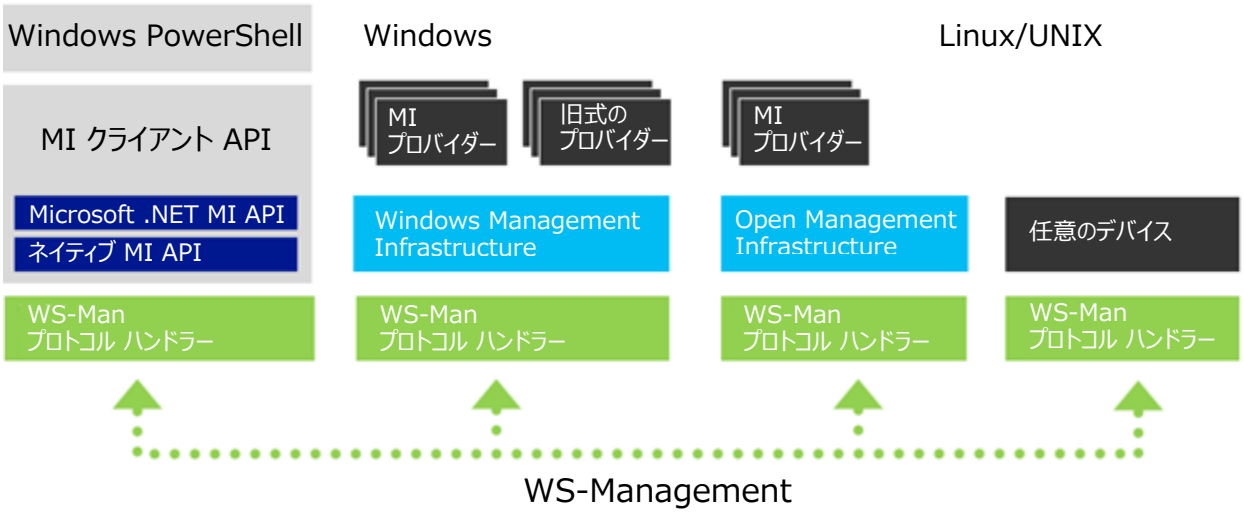
WMI は、多くの標準クラスのプロバイダーをホストする標準の CIM オブジェクト マネージャー (CIMOM) です。しかし、初期においては、これは相互運用が可能な管理プロトコルではなかったので、WMI では分散コンポーネント オブジェクト モデル (DCOM) が使用されていました。その結果、これは Windows を管理する Windows の "管理の孤島" となっていました。

この状況は、DMTF による Web サービスの管理 (WS-Man) の定義および承認によって変化しました。MS-Man は SOAP ベースのファイアウォール フレンドリーなプロトコルで、任意のオペレーティング システム上のクライアントが任意のプラットフォーム上で実行する標準対応の CIMOM に対する操作を起動することを可能にします。マイクロソフトは、WS-Man の最初の部分的実装を Windows Server 2003 に組み込み、Windows リモート管理 (WinRM) という名称を付けました。

Windows Server 2012 では、WinRM は管理用の既定のプロトコルになりました。WinRM は、Openwsman (Perl、Python、Java、および Ruby Binding)、Wiseman、および OpenPegasus を始めとするその他のプラットフォームで使用可能な多くの CIMOM および WS-Man スタックとの相互運用性を提供します。

Windows Server 2012 R2 には、Open Management Infrastructure (OMI) という新しいオープン ソース Windows Management Infrastructure サーバーが実装されています。これには、拡張された Windows PowerShell セマンティクスを含む新しい WMI プロバイダー API (MI) が搭載されています。OMI は、Windows および Linux/UNIX 上での WS-Man の完全な実装です。また、統合された Windows PowerShell レイヤーを含む新しい MI クライアントも提供します。OMI は、Windows および Linux システム用の単一の管理インフラストラクチャを提供します。

図 3. Open Management Infrastructure



OMI は、OpenPegasus などのその他の管理プラットフォームよりも軽量で高速の管理機能を提供します。

図 4: Open Management Infrastructure と OpenPegasus の比較

	OpenPegasus	Open Management Infrastructure	比較
1 秒あたりの要求数 (バイナリ プロトコル)	260	20,000	76 倍
イメージ オブジェクト サイズ	8,000 KB	150 KB	53 倍
簡易プロバイダー サイズ	30 KB	3 KB	10 倍
仮想メモリ サイズ	54 KB	1.5 MB	36 倍
常駐セット サイズ	8,500 KB	500 KB	17 倍

標準の管理ツール

WS-Man が標準プロトコルとして実装されたことに、管理性と相互運用性を容易かつ効率的にするために標準 API を使用する基盤が確立されました。

Windows Server 2012 R2 の目標の 1 つに Windows PowerShell を使用して可能な限り多くのプラットフォームとデバイス、および最新バージョンの Windows と Windows 以外のオペレーティング システムを含む複雑なデータセンター環境を管理できるようにすることがあります。

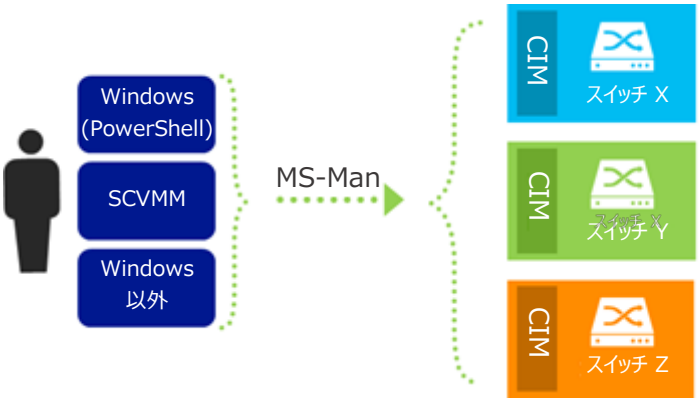
このホワイト ペーパーで説明する主要な管理コンポーネント (WMI、MI API、WinRM、および PowerShell) は、Windows Management Framework 4.0 に含まれています。これには、Windows PowerShell、Windows PowerShell ISE、Windows PowerShell Web サービス (Management OData IIS 機能拡張、Windows リモート管理 (WinRM)、Windows Management Instrumentation (WMI)、サーバー マネージャー WMI プロバイダーの更新、および 4.0 の新機能 Windows PowerShell Desired State Configuration (DSC) が含まれています。このダウンロード可能なパッケージは、Windows Server 2008、Windows Server 2008 R2、Windows 7、および Windows Server 2012 システムにインストールして、Windows Server 2012 R2 に搭載されている更新されたすべての標準準拠機能を活用できます。

Windows Server 2012 R2 は、汎用 CIM 操作に直接対応し、MI クライアント .NET API 上に構築された新しい PowerShell モジュール (CIM コマンドレット) を提供します。このモジュールのコマンドレットを使用して、Windows、最新の WS-Man および CIM をサポートする Windows 以外のデバイスを管理できます。Windows Server 2012 R2 では、CIM ベースのコマンドレットと呼ばれる新しいタイプのコマンドレットもサポートされているので、開発者とスクリプト作成者は、WS-Man を介して、Window でバイスの既存の WMI

プロバイダーと Windows システム以外の CIM プロバイダーの両方を始めとする任意の CIM または WMI プロバイダーと対話することができます。

1 つの例として、Windows Server 2012 R2 では、管理者はネットワーク スイッチを制御できます。そのため、標準のアプローチでネットワーク スイッチの容易な展開と操作を行うことができます。

図 5. ネットワーク スイッチを展開および管理する PowerShell スクリプトの例

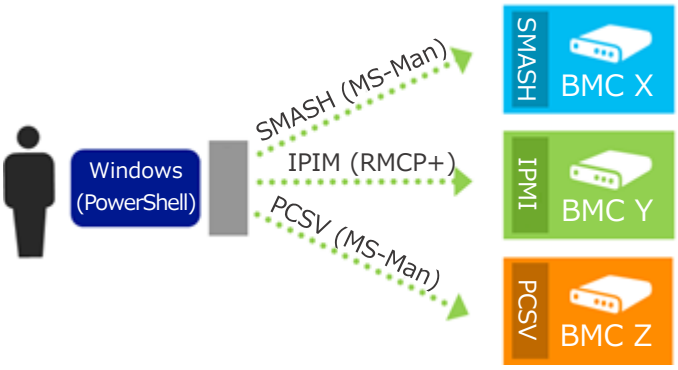


例:

```
$tors  =  "192.168.0.1", "192.168.0.2", "192.168.0.3"
$so    = New-CimSessionOption -UseSsl
$s     = New-CimSession -CN $tors -port 7779 -Auth Basic -Credential $cred -
Sessionoption $so

# SSH 機能を有効にします。
$ssh   = Get-CimInstance -CimSession  $s  MSFT_Feature  |  ? FeatureName -eq 2
         Set-CimInstance -CimSession  $s  $ssh  -Property @{ IsEnabled = $true }

# すべてのポートを有効にします。
$ports = Get-CimInstance  -CimSession $s  CIM_EthernetPort
$ports | Invoke-CimMethod -Method RequestStateChange -Parameter @{ RequestedState = 2}

# ポートをトランク モードに設定して、VLAN の一覧を指定します。
$lanep    =  Get-CimAssociatedInstance  $ports[5] -ResultClassName  Cim_LanEndpoint
$vlanep   =  Get-CimAssociatedInstance  $lanep    -ResultClassName  Cim_VLanEndpoint
$vlanepsd =  Get-CimAssociatedInstance  $vlanep   -ResultClassName
Cim_VLanEndpointSettingData
                      # ポート モードをトランクに設定します。
                      Set-CimInstance  $vlanep  -Property @{DesiredEndpointMode = 5}
                      # トランク VLAN の一覧を設定します。
                      Set-CimInstance  $vlanepsd -Property @{TrunkVlanList= “2,3,4”}
```

今日、さまざまなベンダーが多様なプロトコルおよびメカニズムをサポートしていますが、PowerShell はあまりカバーされていません。そのような状況で、ハードウェアの自動化は容易ではありません。Windows Server 2012 R2 を使用すると、帯域外のハードウェアを標準ベースのプロトコルで管理できます。

図 6: 帯域外のハードウェアを管理する PowerShell スクリプトの例



例:

```
$a =  Get-PCSVDevice  10.20.30.111  -Credential admin -Auth Default -Protocol IPMI
$b =  Get-PCSVDevice  10.20.30.112  -Credential admin -Auth Digest  -Protocol SMASH
$c =  Get-PCSVDevice  10.20.30.113  -Credential admin -Auth Digest  -Protocol PCSV
$hosts = $a,$b,$c

$hosts  | Select Manufacturer, Model, FirmwareVersion
# 製造元、モデル、およびファームウェア バージョンを表示します。
$hosts  | Select PowerState
# 現在の電源状態を表示します。
$hosts  | Set-PcsvDevice -NextBoot "Network“
# ネットワークから PXE ブートを設定します。
$hosts  | Restart-PcsvDevice
# マシンを再起動します。これは "影響の大きい操作" なので、

# ユーザーに確認メッセージが表示されます。
$hosts  | Restart-PcsvDevice –Force
# メッセージを非表示にしてマシンを再起動します。
```

Windows Server 2012 R2 の標準ベースの管理アプローチでは、データセンターを管理する IT プロフェッショナルは、Windows PowerShell および Windows の標準準拠の新しい機能を使用して、最新の CIM、WS-Man、および OData 標準をサポートするあらゆるデバイスを管理できます。

Windows PowerShell での毎日の反復タスクの簡素化

Windows PowerShell は、ほとんどのサーバーの役割およびデータセンター機能を管理するのに役立つ包括的なプラットフォームを提供します。この最新バージョンの Windows PowerShell では、リモート サーバーへのセッションには弾性があり、さまざまな種類の中断に耐えることができます。さらに、Windows PowerShell の習得は、コマンドレットの検索機能の向上および簡素化、すべてのコマンドレットを通じた一貫性のある構文により、以前にも増してより簡単になりました。

技術仕様

以下のセクションでは、Windows PowerShell の主な機能について説明します。

堅牢なセッション接続

Service Pack の展開やデータベースのバックアップなど、時間のかかるタスクは、その操作を開始したクライアントがダウンした場合やネットワーク接続から切断された場合でも継続する必要があります。

堅牢なセッション接続性により、リモート セッションは、クライアントが応答しなくなった場合やアクセス不能になった場合でも接続状態を最長 4 分間維持することができ、管理ノード上のタスクは自動的に継続して実行できるので、エンド ツー エンドのシステムの信頼性がさらに向上します。4 分以内に接続が回復しない場合は、管理ノード上の実行処理はデータ損失なしで保留され、リモート セッションは自動的に切断状態になるので、ネットワーク接続が回復した後に再接続されます。予期しないクライアントの切断によって発生するタスク実行の不完全な終了によるアプリケーションおよびシステム状態の破損は、実質的に排除されます。

切断されたセッション

Windows PowerShell 3.0 以上では、状態の損失なしでセッションの切断および再接続を行うことができます。切断されたセッションでは、リモート コンピューター上でのセッション作成、コマンドまたはジョブの開始、セッションからの切断、そしてコンピューターのシャットダウンを行った後で、別のコンピューターからセッションに再接続してジョブのステータスを確認することや結果を取得することができます。監理者がセッションを切断した場合でも、コマンドおよびジョブは継続して実行できます。

以下のコマンドレットは、Windows PowerShell 3.0 以降の切断されたセッションの機能を示します。

- Disconnect-PSSession: リモート コンピューターからセッション接続を切断します。

- Connect-PSSession: リモート コンピューターとのセッション接続を再確立します。
- Receive-PSSession: 既定では、リモート セッション上のコマンド実行を再開し、セッションの出力を取得します。暗示的にセッションに再接続します (Connect-PSSession コマンドなし)。

例:

```
# リモート セッションを開始し、セッションから切断します。PowerShell を終了します。
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
# 別のコンピューターで Windows PowerShell を起動します。
PS C:\> $s = Get-PSSession -ComputerName srv1 -Name LongSession
PS C:\> Receive-PSSession $s
```

ジョブ スケジュール

Windows PowerShell 3.0 以降では、ジョブの実行時間をスケジュールすることや、特定のスケジュールに従ってジョブを実行させることができます。スケジュールされたジョブを作成するには、最初にジョブ定義を作成してジョブに名前を付けてコマンドを指定し、次にジョブ トリガーを作成してジョブのスケジュールを指定します。ジョブをスケジュールおよび開始するには、Windows タスク スケジューラーを使用します。ジョブの出力は、後で Windows PowerShell セッションで使用できるように、ユーザーごとのジョブ リポジトリを使用します。

PSScheduledJob モジュールでは、スケジュールされたジョブの操作に役立つ以下のコマンドレットを使用できます。

- 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

ジョブは、以下のジョブ トリガーに基づいて実行するようスケジュールできます。

- 1 回のみ

- 毎日
- 毎週
- 起動時
- ログオン時

例:

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

スケジュールされたジョブは手動で開始することもできます。

例:

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

スケジュールされたジョブは手動で開始することもできます。

例:

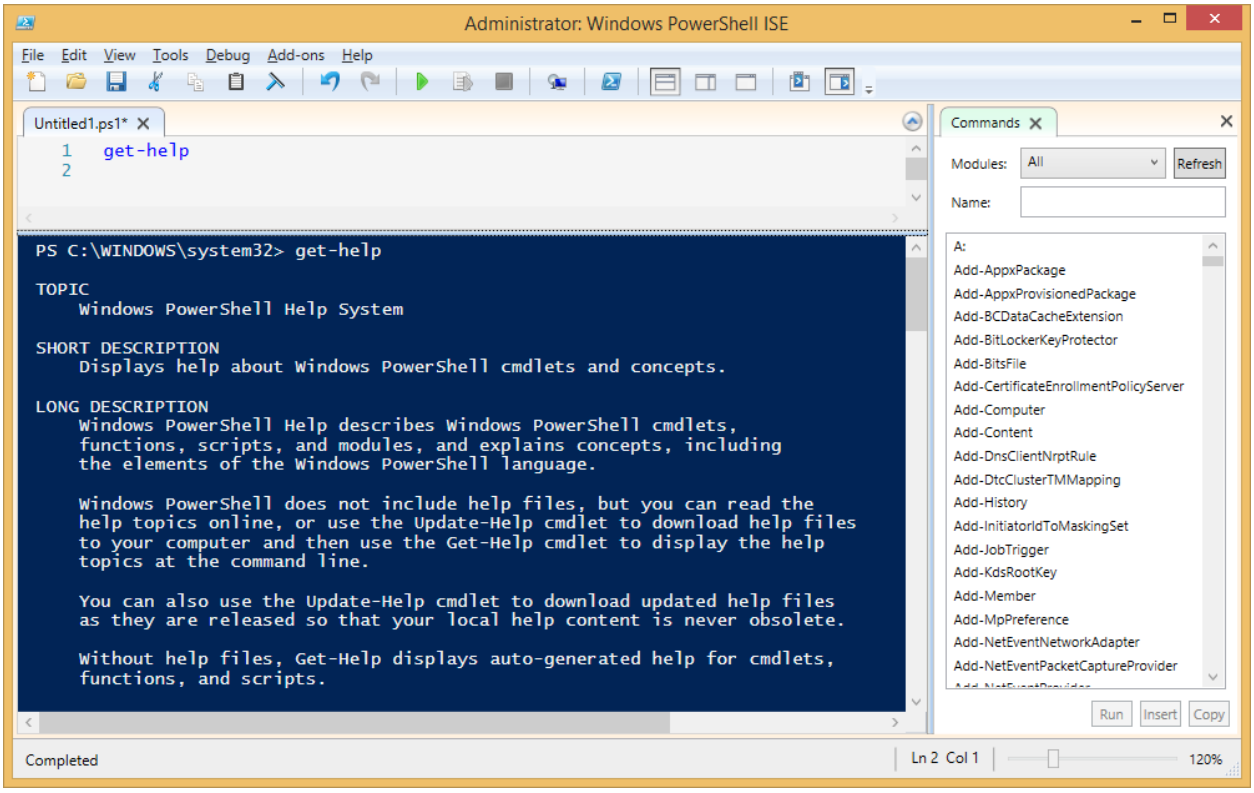
```
Import-Module PSScheduledJob
$j = Get-Job -Name MyScheduledJob
Receive-Job $j
```

Windows PowerShell ISE

Windows PowerShell 3.0 Integrated Scripting Environment (ISE) には、Windows PowerShell の経験が浅いユーザーをサポートする多くの新機能に加えて、スクリプト作成者向けの高度な編集サポートが含まれています。新機能の一例を以下に示します。

- コマンド表示ウィンドウ: ダイアログ ボックスでコマンドレットを検索および実行できます。

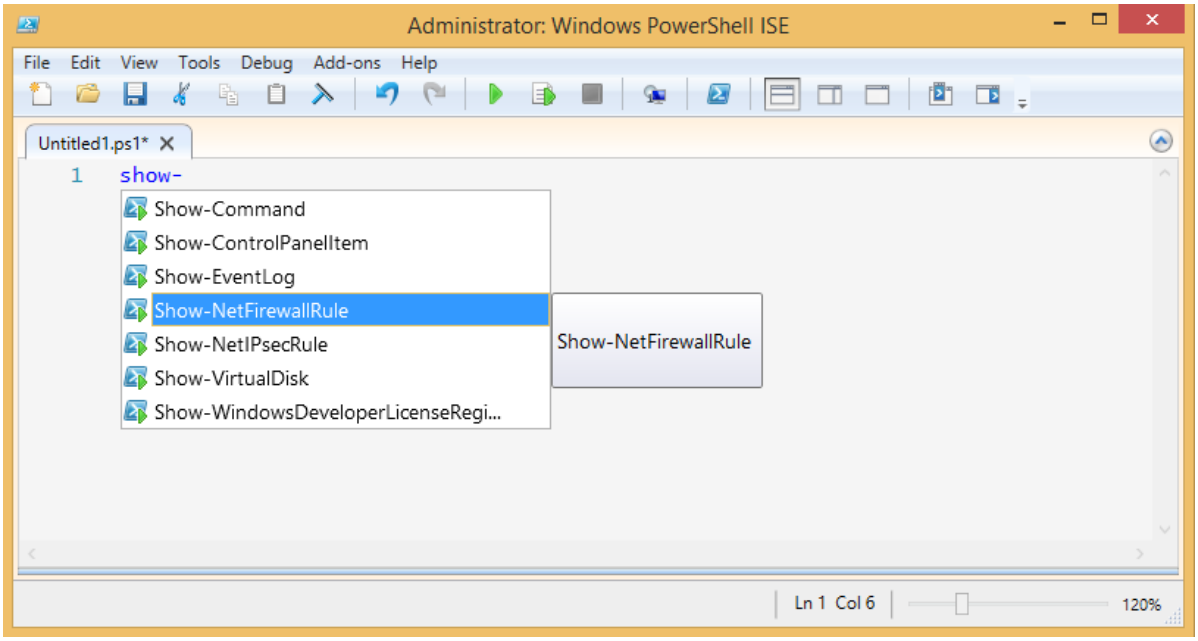
図 7: Windows PowerShell ISE のコマンドレットの操作

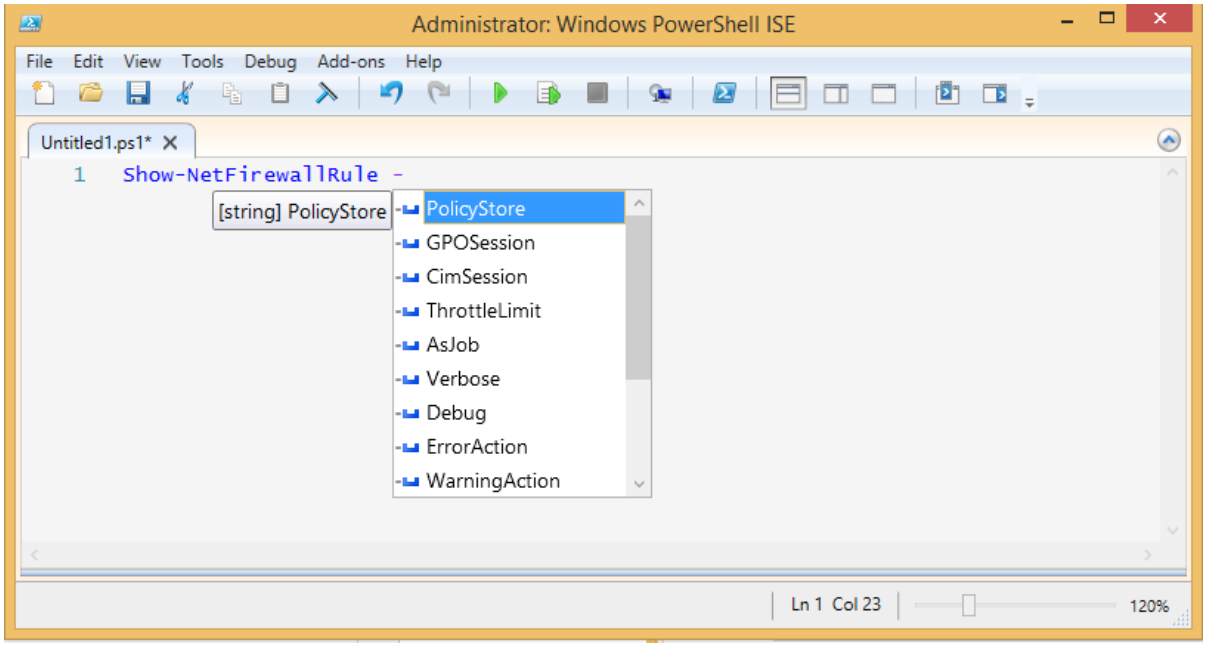


- IntelliSense:

  コマンドレットとスクリプト名、パラメーター名と列挙された値、およびプロパティ名とメソッド名に基づいた状況依存のコマンド完了を提供します。IntelliSense は、パス、型、および変数もサポートします。

図 8: IntelliSense での状況依存のコマンド完了

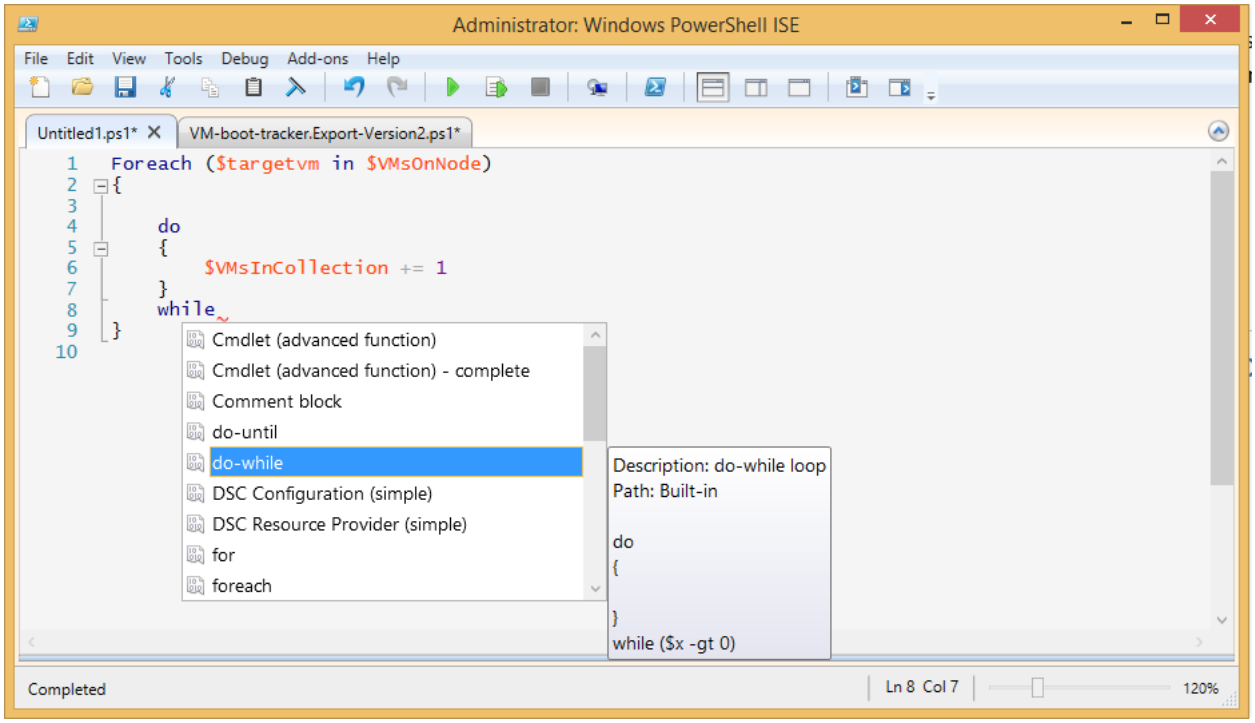




- スニペット:

  再使用可能なテキストを挿入できるコード例。組み込みのスニペットには、関数、ワークフロー、および共通言語パターンのテンプレートが含まれるので、構文を覚える必要はありません。

図 9: PowerShell ISE のスニペット



- スクリプトおよび XML ファイルの折りたたみ可能な領域: 長いスニペットのナビゲーションが容易になります。

Windows PowerShell ワークフロー

IT プロフェッショナルは、時間のかかるタスクやワークフローのシーケンスを管理対象の複数のコンピューターやデバイスに対して同時に実行して、マルチコンピューター環境の管理を自動化することがあります。Windows PowerShell ワークフローを使用すると、ワークフローの利点を Windows PowerShell の自動化機能で活用できます。

ワークフローは、自動化されたステップまたはアクティビティのシーケンスで、1 つまたは複数の管理対象ノード (コンピューターまたはデバイス) に対してタスクの実行やデータの取得を行うことができます。これらのアクティビティには、個々のコマンドやスクリプトを含めることができます。IT プロフェッショナルや開発者は、Windows PowerShell ワークフローを使用して、(一般的に長時間実行、反復可能、高頻繁、並列実行可能、割り込み可能、停止可能、または再開可能という特性がある) マルチマシン管理アクティビティのシーケンスをワークフローとして作成できます。設計により、ワークフローは、意図的または偶発的な中断や割り込み (ネットワーク、再起動、停電など) の後に再開できます。

Windows PowerShell ワークフローの利点

Windows PowerShell ワークフローを使用すると、マルチコンピューター タスクの分散、シーケンス処理、および完了を管理できるので、ユーザーおよび監理者は重要性の高いタスクに集中することができます。次の一覧では、Windows PowerShell ワークフローのさまざまな利点について説明します。

- **PowerShell スクリプト構文の使用。**IT プロフェッショナルは、拡張された PowerShell 言語を使用することで、既存の PowerShell スクリプト スキルを再利用して、スクリプト ベースのワークフローを作成できます。記述が容易なことに加えて、PowerShell スクリプト ベースのワークフローは、電子メールに貼り付けることやオンラインで公開することによって容易に共有することが可能です。
- **マルチコンピューター管理。**数百もの管理対象ノード上で、実行に時間がかかる複数のタスクをワークフローとして同時に実行します。Windows PowerShell ワークフローには、ワークフローの共通管理パラメーターの組み込みライブラリが含まれています。このライブラリにより、PSComputerName や PSConfigurationName などのマルチコンピューター管理シナリオが可能となります。
- **単一のタスクとしての複数の複雑なプロセスの実行。**エンド ツー エンドのシナリオ全体で機能する複数の関連スクリプトまたはコマンドを 1 つのワークフローにまとめることができます。ワークフローのアクティビティの状態と進捗状況はいつでも見ることができます。
- **堅牢さ: 自動化された障害回復。**Windows PowerShell ワークフローは、計画的および計画外の両方の中断 (コンピューターの再起動やネットワーク異常) を乗り切ることができます。ワークフローの実行を中断した後、最後のチェックポイント (通常は中断されたポイント) からそのワークフローを再開できます。
- **永続化。**ワークフローの状態とデータは、その作成者が定義した特定のポイントで保存 (チェックポイント設定) されるので、ワークフローを最初から再実行するのではなく、最後に永続化されたタスク (チェックポイント) から再開できます。
- **接続とアクションの再試行。**ワークフロー共通のパラメーターを使用すると、ネットワーク接続で障害が発生した場合、ワークフロー ユーザーは管理対象ノードへの接続を再試行できます。さらに、ワークフローの作成者は、1 つ以上の管理対象ノードで障害が発生した場合に (コンピューターの 1 つがアクティビティの実行中にダウンした場合など) 再度実行する特定のアクティビティを指定できます。
- **接続と切断の機能。**ユーザーは、ワークフローを実行しているコンピューターに接続したり、ワークフローを実行しているコンピューターから切断したりできますが、その間ワークフローは実行を続けます。たとえば、ワークフローを中断することなく、ワークフロー コンピューターに接続しているコンピューターからログオフすることやコンピューターを再起動することができます。ワークフローの実行は、別のコンピューター (自宅のコンピューターなど) から監視することができます。これは、ワークフロー エンジン コンピューターとは別のコンピューターとは別のコンピューター上でクライアントが実行している場合に可能です。

- **スケジュール機能。**ワークフロー タスクは、Windows PowerShell のコマンドレットやスクリプトと同様にスケジュールできます。
- **ワークフローと接続の調整機能。**ワークフローの実行とノードへの接続は調整することができるので、スケーラビリティや高可用性のシナリオが可能になります。

コマンドレット/スクリプトではなく Windows PowerShell ワークフローを使用する場合

一般的に、次のいずれかの要件を満たす必要がある場合、コマンドレット/スクリプトではなくワークフローの使用を検討してください。

- 時間のかかる複数のステップのシーケンス タスクを実行する必要がある。
- 複数のコンピューター上で 1 つのタスクを実行する。
- チェックポイント設定または永続化が必要なタスクを実行する必要がある。
- 非同期、再起動可能、並列化可能、または割り込み可能という特徴があり時間のかかるタスクを実行する必要がある。
- 調整や接続プールを必要とし、タスクを大規模なスケールまたは高可用性環境で実行する必要がある。

Windows PowerShell でのワークフローの記述と実行の例

通常、ワークフローはクライアント コンピューターから起動され、複数のターゲット コンピューターで時間のかかるタスクを実行するのに適しています。ワークフローは他の Windows PowerShell コマンドレットと似ていて、ワークフローの検出には Get-Command コマンドレット、ワークフローの使用方法を確認するには Get-Help コマンドレットを使用できます

ワークフローは、コマンド ラインで定義する、スクリプト内で定義してドット ソース形式で読み込む、または Import-Module コマンドレットを使用して Windows PowerShell スクリプト ワークフローまたは XAML ベースのワークフローを含むモジュールをインポートするという方法で Windows PowerShell セッションに追加できます。インポートした後は、ワークフローはその他のセッションで他の PowerShell コマンドと同様に動作します。

ワークフロー内の各ステップまたはコマンドは、"アクティビティ" と呼ばれます。各アクティビティはワークフローのプロパティを継承します。これには、前述の強力なワークフローの共通パラメーターも含まれます。

ワークフローを記述するには、通常の PowerShell コンソールまたは Windows PowerShell ISE を使用できます。たとえば、Windows PowerShell ISE のコマンド ウィンドウに以下のワークフローを入力できます。

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

新しい "Workflow" というキーワードに注目してください。これは、コマンドが Windows PowerShell ワークフローであることを示しています。このキーワードにより、20 を超える新しい共通パラメーターがワークフローに追加され、ユーザーは次のような項目を指定できるようになります。

- ワークフローのターゲット コンピューターの一覧 (-PSComputerName)
- ワークフローの実行に使用する資格情報 (-PSCredential)
- 作業の尺度としてワークフローを管理するためのクォータ (-PSRunningTimeoutSec など)
- 接続に問題がある場合に、ワークフロー全体または特定のアクティビティを再試行する機能 (PSConnectionRetryCount など)
- ワークフローのアクティビティを永続化 (チェックポイント設定) する機能。この機能により、ワークフローのメタデータ、出力、およびエラーをディスクに保存し、実行時に指定したポイントでワークフローを再開することが可能になります (-PSPersist)

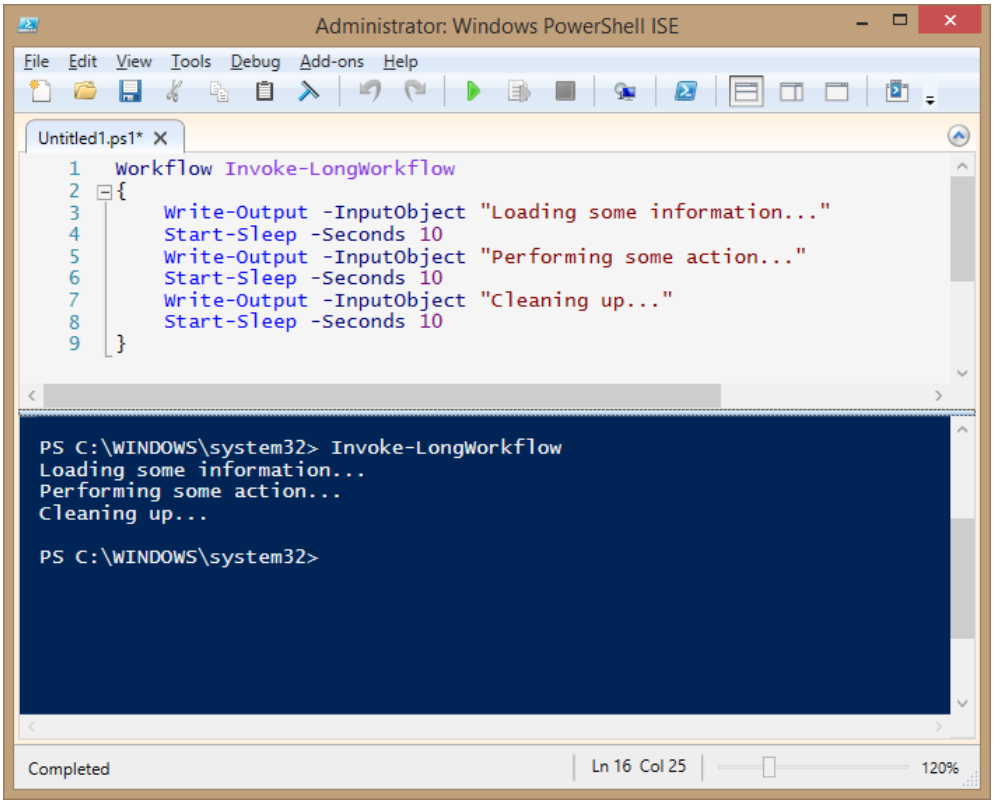
ワークフローを実行するには、他の Windows PowerShell コマンドの実行時と同様に、ワークフロー名を入力します。たとえば、上記の例で作成した新しいワークフローを実行するには、Windows PowerShell ISE プロンプトに「Verb-Noun」と入力します。

以下に "LongWorkflow" というもう 1 つのワークフローの例を示します。これは実行に約 30 秒かかります。

```
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
}
```

この結果、ユーザー定義の LongWorkflow が呼び出されます。

図 10. ISE からのワークフローの実行



このワークフローは時間のかかるタスクを定義しているので、バックグラウンド ジョブとして実行するのに適している場合があります。そのように実行するには、AsJob パラメーターを使用します。また、JobName パラメーターを使用して、ジョブに "LongWF" といった名前を割り当てます。

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

次の例は、さらに複雑なワークフローです。この "Install-VM" というワークフローは、管理対象ノードに仮想マシンを作成し、仮想マシンを起動してドメインに参加させます (仮想マシンの再起動が必要)。

例:

```
<# これは、Hyper-V 対応ホストに VM をインストールする、時間のかかるワークフローです。
このワークフローでは、Window PowerShell 3.0 における新しい
PowerShell ワークフローの一連の機能を紹介します。この例では、管理対象ノードは Hyper-V 対応で、
Hyper-V の役割/モジュールがインストールされている必要があります。#>

# Hyper-V 対応ホストに VM をインストールするワークフロー
```

```powershell
workflow Install-VM
{
    param
    (
    # VM のベース Vhd の完全パス
    [Parameter(Mandatory=$true)]
    [String]$BaseVhdPath,
    # VM 名のプレフィックス
    [String]$VMNamePrefix = "Demo",
    # 作成する VM の数
    [Int]$VMCount = 3,
    # VM をドメインに参加させるために必要なドメイン資格情報
    [Parameter(Mandatory=$true)]
    [System.Management.Automation.PSCredential] $domainCred,
    # ドメインへの参加の前に VM に接続する必要があるローカルの資格情報
    [Parameter(Mandatory=$true)]
    [System.Management.Automation.PSCredential] $localCred
    )

    # VM を並列で作成
    foreach -parallel($i in 1..$VMCount)
    {
        # VM 名の作成
        [string]$VMName = $VMNamePrefix+$i

        # 差分 VHD のフル パス
        [string]$VhdPath = (Split-Path $BaseVhdPath) + "\" + $VMName +".vhd"

        # 差分 VHD の作成
        $DiffVHD = New-VHD -ParentPath $BaseVhdPath -Path $VhdPath

        # 差分 VHD などで新しい VM を作成
        $null = New-VM -MemoryStartupBytes 1GB -Name $VMName `
                    -VHDPath $DiffVHD.Path -SwitchName "InternalSwitch"
    }

    # ワークフローの状態とデータの保存
    Checkpoint-Workflow

    # VM を並列で起動して IP アドレスを収集
```

```
$IPAddresses = foreach -parallel($i in 1..$VMCount)
{
    # VM 名の作成
    [string]$VMName = $VMNamePrefix+$i

    # VM の起動
    Start-VM -Name $VMName

    # IP アドレスが各 VM に割り当てられるのを待機
    # Inlinescript を使用して VM の IP  アドレスを確認
    $VMIP = Inlinescript
        {
            (Get-VM -Name $using:VMName).NetWorkAdapters.IPAddresses
        } -DisplayName "Get-VMIPAddress"

    while($VMIP.count -lt 2)
    {
        # Inlinescript を使用して VM の IP アドレスを確認
        $VMIP = Inlinescript
            {
                (Get-VM -Name $using:VMName).NetWorkAdapters.IPAddresses
            } -DisplayName "Get-VMIPAddress"

        # 進捗ストリームからユーザーに通知
        Write-Progress -Id $i -Activity "Get-VMIPAddress on $VMName" `
                    -Status "Waiting for IP Address ..."

        # 5 秒待って再試行
        Start-Sleep -Seconds 5;
    }
    $VMIP[0]
}

# 収集した ID をワークフロー ユーザーに表示
$IPAddresses

# ワークフローを一時停止する (設定をチェックしてリソースを解放する) 前に
# 上級管理者にワークフローの一時停止された状態を通知する電子メールを送信
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
"@

    # ワークフローの実行の一時停止
    Suspend-Workflow

    # VM をドメインに参加させる Join-Domain ワークフローの呼び出し
    Join-Domain -PSComputerName $IPAddresses -PSCredential $localCred -domainCred
$domainCred

    # 上級管理者にメールを送ってワークフローが完了したことを通知
    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
"@
}

# コンピューターをドメインに参加させるワークフロー
workflow Join-Domain
{
    param
    (
        [string] $domainName="fourthcoffee.com",
        [Parameter(Mandatory=$true)]
        [System.Management.Automation.PSCredential] $domainCred
    )

    # コンピューターが WORKGROUP に参加しているかどうか確認
    Get-CimInstance -ClassName CIM_ComputerSystem

    # コンピューターをドメインに追加して再起動
    Add-Computer -DomainName $domainName -LocalCredential $PSCredential -Credential
$domainCred
    Restart-Computer -Wait -For WinRM -Force -Protocol WSMan
```

```
    # ドメインに参加したことを通知
    Get-CimInstance -ClassName CIM_ComputerSystem
}
```

コマンドレットの検出: Get-Command とモジュールの自動読み込み

Windows Server 2012 R2 には検索と学習が容易なコマンドレットが 3,000 以上含まれています。モジュールは、以前よりも容易に検索、探索、作成、および使用することができ、ユーザーがコマンドレットを使用するために手動でモジュールをインポートする必要もなくなりました。ユーザーがコマンドレットを実行すると、Windows PowerShell が自動でモジュールをインポートします。さらに、Get-Command は、システムにインストールされているすべてのコマンドレットを検索するように更新されています。たとえば、ネットワークに関するすべてのコマンドレットを検索するには、Get-Command *-Net* を実行します。

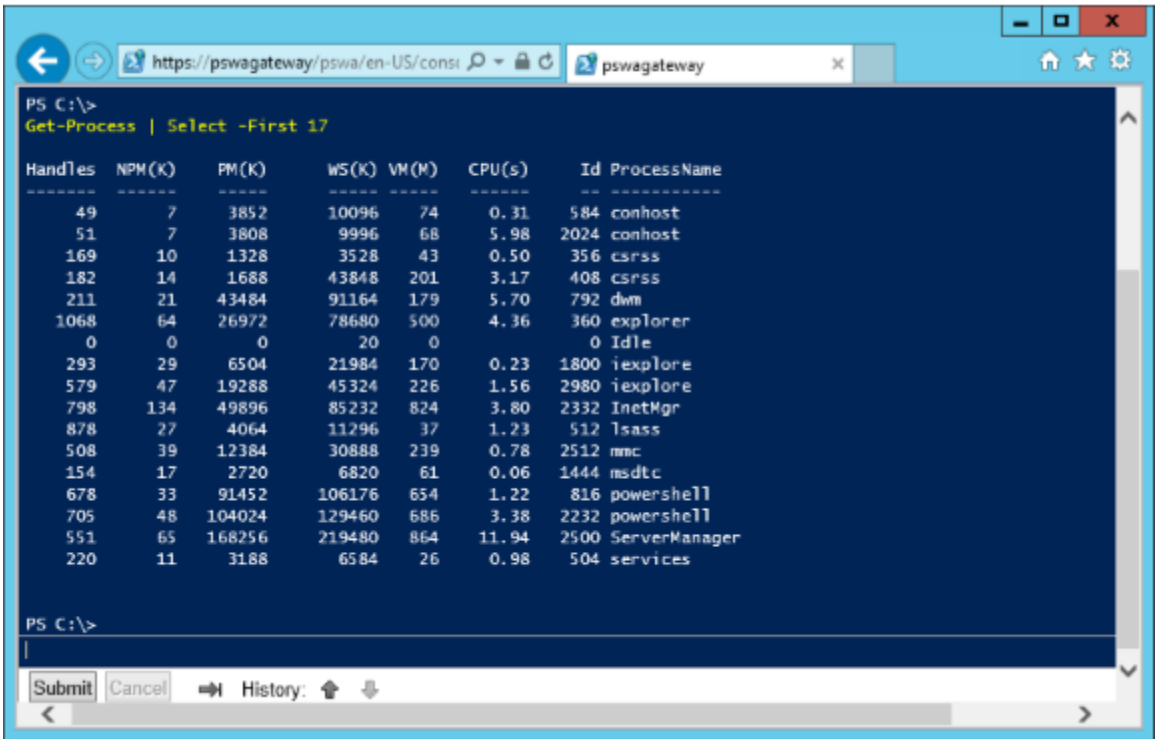
Windows PowerShell ワークフローにおける PowerShell 4.0 の新機能

- System Center Orchestrator で使用される対話型のパイプラインなどのコンテキストにおける新しい PipelineVariable 共通パラメーターのサポートが追加されました。これは、ストリーミングの使用による散在実行とは異なり、左から右にコマンドを実行するパイプラインです。
- パラメーター バインドは、タブ完了シナリオ (現在の実行空間に依存しないコマンドと共に実行するシナリオなど) の外部で機能するよう大幅に強化されました。
- カスタム コンテナー アクティビティのサポートが Windows PowerShell ワークフローに追加されました。アクティビティ パラメーターが Activity, Activity[] タイプ、またはアクティビティの汎用コレクションで、ユーザーがスクリプト ブロックを引数として指定した場合、Windows PowerShell ワークフローは、通常の Windows PowerShell のスクリプトからワークフローへのコンパイルの場合と同様に、スクリプト ブロックを XAML に変換します。
- クラッシュした場合、Windows PowerShell ワークフローは管理ノードに自動的に再接続します。
- ThrottleLimit を使用して Foreach -Parallel アクティビティ ステートメントを調整できるようになりました。
- ErrorAction 共通パラメーターにワークフロー専用の新しい有効な値 Suspend が追加されました。
- アクティブなセッションがない場合、進行中のジョブがない場合、および保留中のジョブがない場合、ワークフロー エンドポイントが自動的に閉じるようになりました。自動終了条件が満たされる場合、この機能によって、ワークフロー サーバーとして機能するコンピューター上のリソースが節約されます。

Windows PowerShell Web アクセス

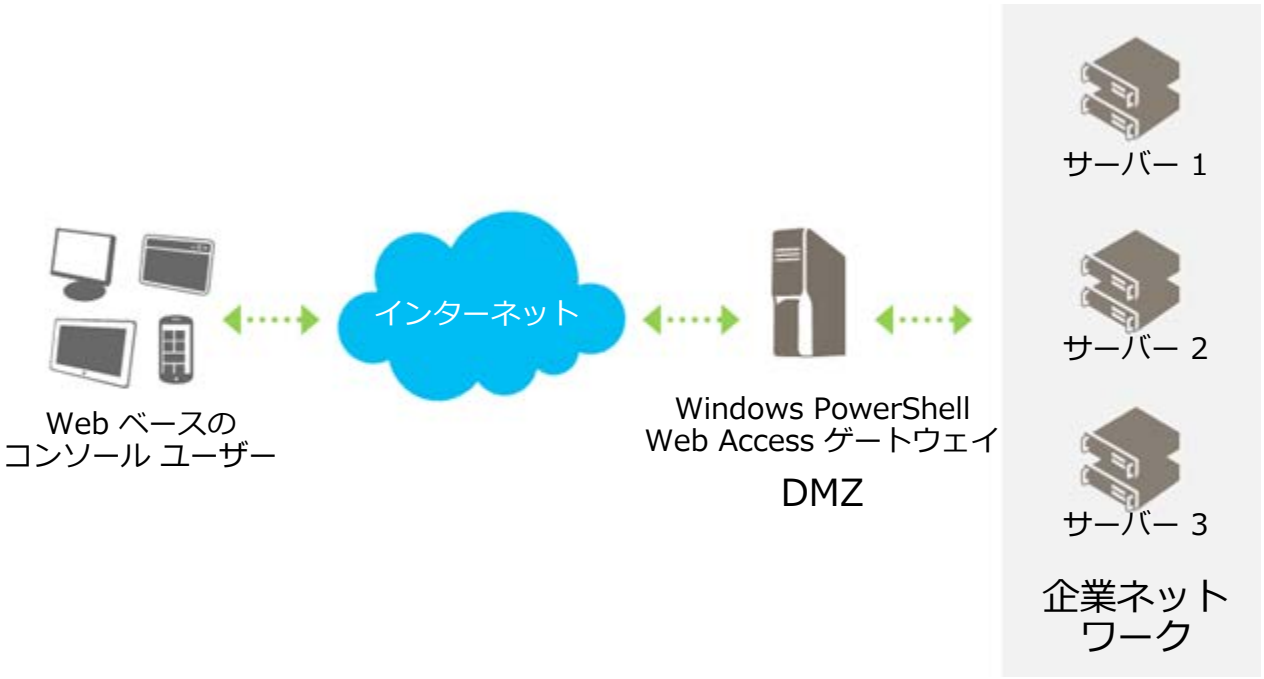
Windows PowerShell Web アクセスは Windows Server 2012 で使用可能になった新しい機能で、Web ブラウザー内で Windows PowerShell を使用して、Windows サーバーを管理することができます。管理対象のターゲット コンピューターでは、Windows PowerShell リモート処理に対応している任意のバージョンの Windows を実行できます。

図 11. PowerShell Web アクセスを介した別のサーバーへのアクセス



Web ブラウザーで Windows PowerShell を介してリモート サーバーを管理するには、Windows PowerShell Web アクセス機能がインストールされている Windows Server 2012 を実行するサーバーに接続します。このサーバーは、Windows PowerShell のインターフェイスを含む Web ページをリモート クライアントに提供するゲートウェイとして機能します。次の図にこのインフラストラクチャを示します。

図 12. PowerShell Web Access のしくみ



Windows PowerShell Web アクセスにおける PowerShell 4.0 の新機能

- Web ベースの Windows PowerShell Web アクセス コンソールで既存のセッションからの切断および再接続を行うことができます。Web ベースのコンソールの保存ボタンを使用すると、セッションを削除することなくセッションから切断して、別の時間に再接続できます。
- 既定のパラメーターをサインイン ページに表示できます。既定のパラメーターを表示するには、ページに表示できます。既定のパラメーターを表示するには、web.config というファイルでサインイン ページの [オプションの接続設定] 領域に表示されるすべての設定の値を構成します。web.config ファイルを使用して、すべての最適な接続設定を構成できます。
- Windows Server 2012 R2 では、Windows PowerShell Web アクセスの承認規則をリモートで管理できます。Add-PswaAuthorizationRule および Test-PswaAuthorizationRule コマンドレットに Credential パラメーターに含まれるようになりました。管理者は、このパラメーターを使用して、承認規則をリモート コンピューターから、または Windows PowerShell Web Access セッションで管理できます。
- セッションごとに新しいブラウザー タブを使用して、複数の Windows PowerShell Web アクセス セッションを単一のブラウザー セッションで開くことができるようになりました。Windows PowerShell コンソールの新しいセッションに接続するために新しいブラウザー セッションを開く必要はなくなりました。

更新可能なヘルプ

Windows PowerShell 2.0 には、オンラインで頻繁に更新される広範なヘルプ トピックが含まれていました。しかし、ヘルプ ファイルは Windows オペレーティング システムの一部であったため、ユーザーはこれらを更新でず、コマンド ラインに表示されるヘルプ トピックは古いものになってしまう場合がありました。サード パーティ製品では、オンライン ヘルプを XML に変換する必要があり、変換しない場合は情報の古いヘルプ トピックを表示しなければなりませんでした。

Windows PowerShell 3.0 では、Update-Help と Save-Help という新しいコマンドレットが用意されていて、各モジュールの最新のヘルプ ファイルをダウンロードしてインストールできます。これらのコマンドレットは、インターネット上でヘルプ ファイルを探し、それらがローカル ファイルより新しいかどうかを判断して、ファイルをアンパックし、適切な場所にインストールします。更新されたファイルは、Get-Help ですぐに使用できます。Windows PowerShell を再起動する必要はありません。Windows PowerShell 3.0 以降のヘルプ ファイルは "付属品" ではないため、最初に使用する際に最新のものになります。Get-Help では、自動生成されたコマンドのヘルプが表示され、Update-Help コマンドレットを使用してモジュールのヘルプ ファイルをインストールまたは更新するように促すメッセージが表示されます。

大企業などでは、ファイアウォールによってインターネットに直接接続できない場合があります。このような環境では、インターネットからではなく、ローカル共有からヘルプ ファイルを更新できるようにした方が便利です。そのような場合は、Save-Help -DestinationPath <share> を使用して、最新の Windows PowerShell ヘルプ ファイルを格納するローカル共有を作成できます。組織内のユーザーは、その共有を指定して Update-Help –SourcePath <share> を実行することにより、ヘルプ ファイルを更新できます。

PowerShell 4.0 の Save-Help では、リモート コンピューターにインストールされているモジュールのヘルプをを保存できます。Save-Help を使用して、インターネットに接続されているクライアント (ヘルプが必要なすべてのモジュールが必ずしもインストールされているとは限りません) からモジュールのヘルプをダウンロードして、保存したヘルプをリモート共有フォルダー、またはインターネット アクセスのないリモート コンピューターにコピーできます。

更新可能なヘルプは、サードパーティ製モジュールを始めとするすべてのモジュールで使用可能で、複数の言語のサポートが含まれます。

新しいコマンドレット

PowerShell 4.0 では、16 の新しいモジュール、652 の新しいコマンドレット、および 3,603 の新しいパラメーターが追加されています。

16 の新しいモジュール: PowerShell 3.0 との比較

```
AppBackgroundTask               StartScreen
Defender                        SyncShare
DFSR                            TLS
NetEventPacketCapture           WDS
NetNat                          WebApplicationProxy
PcsvDevice                      WindowsSearch
PSDesiredStateConfiguration     WssCmdlets
SoftwareInventoryLogging        WssSetupCmdlets
```

652 の新しいコマンド: PowerShell 3.0 との比較

```
223 WssCmdlets                   13 NetNat                    2 WindowsSearch
 60 ADFS                         13 VpnClient                 2 TrustedPlatformModule
 58 IpamServer                   10 WebApplicationProxy       2 PrintManagement
 42 DFSR                         10 SoftwareInventoryLogging  1 SmbWitness
 36 RemoteAccess                  8 Dism                      1 NetWNV
 33 WDS                           7 SmbShare                  1 AdcsAdministration
 26 NetEventPacketCapture         5 PcsvDevice                1 Deduplication
 18 DhcpServer                    5 WssSetupCmdlets           1 NetSecurity
 18 Storage                       4 TLS                       1 FailoverClusters
 14 SyncShare                     3 StartScreen               1 DnsServer
 14 Hyper-V                       3 NetTCPIP
 13 PSDesiredStateConfiguration   3 IscsiTarget
```

3,603 の新しいパラメーター: PowerShell 3.0 との比較

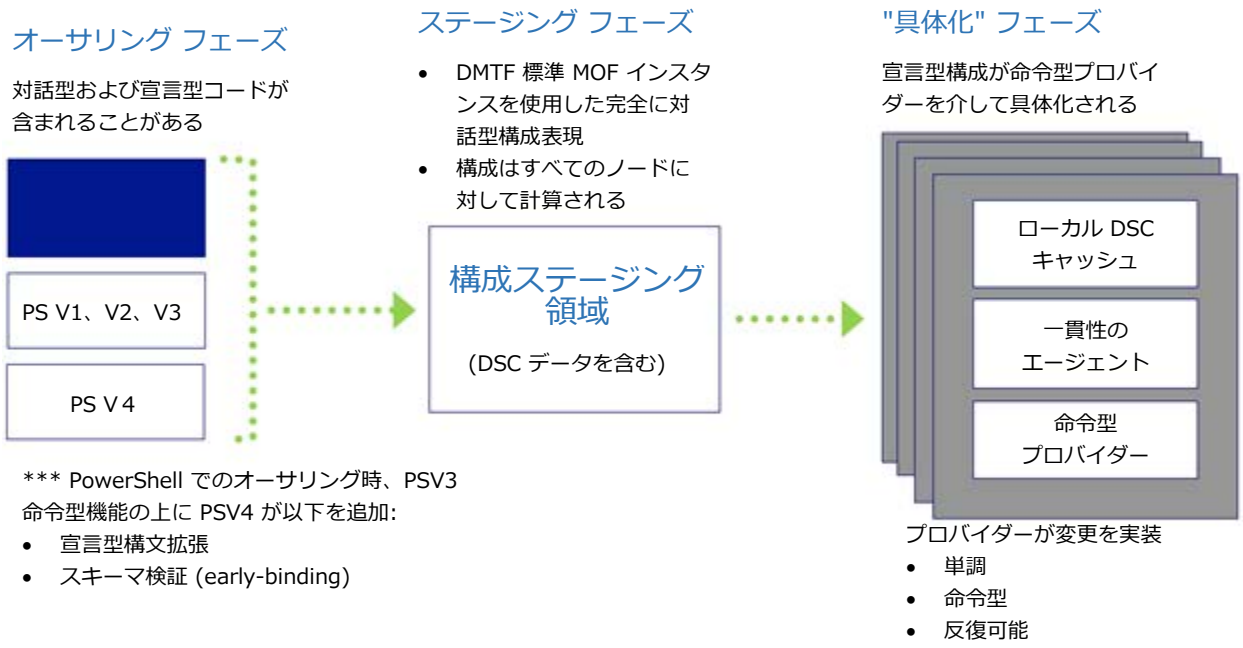
```
619 IpamServer                   67 WebApplicationProxy       6 PSScheduledJob
384 WssCmdlets                   42 SoftwareInventoryLogging  5 AppX
351 RemoteAccess                 40 NetTcpIP                  4 WindowsSearch
315 ADFS                         39 SmbShare                  3 ClusterAwareUpdating
285 WDS                          37 PSDesiredStateConfiguration 2 PowerShellWebAccess
190 DFSR                         19 NetWNV                    2 AdcsAdministration
170 NetEventPacketCapture        19 IscsiTarget               2 BitsTransfer
166 Hyper-V                      15 DnsServer                 2 TrustedPlatformModule
157 Storage                      14 WssSetupCmdlets           2 Microsoft.PowerShell.Management
124 DhcpServer                   13 PrintManagement           2 SmbWitness
117 VpnClient                    12 NetSecurity               1 SMISConfig
 99 SyncShare                    11 FailoverClusters          1 Microsoft.PowerShell.Utility
 82 NetNat                       11 Deduplication             1 NetQos
 79 Dism                         10 TLS                       1 FileServerResourceManager
 74 PcsvDevice                    8 StartScreen
```

Desired State Configuration

Desired State Configuration (DSC) は、Windows PowerShell の管理プラットフォームです。DSC を使用すると、ソフトウェア サービスの構成データの展開と管理、およびそのサービスが実行する環境の管理を行うことができます。

DSC は、ソフトウェア環境の構成を直接指定するために使用できる Windows PowerShell 拡張言語、新しい Windows PowerShell コマンドレット、およびリソースのセットを提供します。また、既存の構成を保守および管理する手段も提供します。

図 13. Desired State Configuration の処理



組み込みの DSC リソースを使用して自動化された方法でコンピューターのセット (ターゲット ノード) を構成および管理できるシナリオの例を以下に示します。

- サーバーの役割および機能の有効化または無効化
- レジストリ設定の管理
- ファイルおよびディレクトリの管理
- プロセスとサービスの開始、停止、および管理
- グループおよびユーザー アカウントの管理
- 新しいソフトウェアの展開
- 環境変数の管理
- Windows PowerShell スクリプトの実行
- 目的の状態から外れた構成の修正
- 特定のノードの実際の構成状態の探索

さらに、カスタム リソースを作成して、アプリケーションまたはシステム設定の状態を構成できます。

構成の定義

DSC では、Configuration という新しいキーワードが導入されています。DSC を使用して環境を構成するには、Configuration キーワードおよびその後に続く識別子を使用して Windows PowerShell スクリプト ブロックを定義し、中かっこ ({}) を使用してブロックを区切ります。

構成ブロック内では、環境兄の各ノード (コンピューター) の目的の構成を指定するノード ブロックを定義できます。ノード ブロックは Node キーワードで開始します。このキーワードの後にターゲット コンピューターの名前を続けます。この場合、変数も使用できます。コンピューター名の後、中かっこ ({}) を使用してノード ブロックを区切ります。

ノード ブロック内で、特定のリソースを構成するリソース ブロックを定義できます。リソース ブロックはリソース名で始まり、そのブロックを指定する ID、そしてブロックを区切る中括弧 ({}) が続きます。

例:

次の例では、DSC を使用して、Web サーバー (IIS) の役割を "Server001" というターゲット コンピューターにインストールし、"Server001" の「wwwroot」フォルダーに特定のセットのファイルを含めます、この処理は、Role リソース (WindowsFeature というフレンドリー タイプ名) および File リソース (File というフレンドリー タイプ名) の 2 つの組み込み DSC リソースを使用して行われます。

1. 管理者として Windows PowerShell ISE を起動します。
2. [表示] メニューで [スクリプト ウィンドウの表示] を選択します。
3. スクリプト ウィンドウで $WebsiteFilePath という変数を定義し、 Web サイト ファイルを含むフォルダーのパスに割り当てます。この例の目的では、ファイルのコンテンツや種類は関係ありません。
4. スクリプト ウィンドウで変数の定義の下に以下のスクリプトをコピーします。

```
Configuration MyWebConfig
{
   # 構成ブロックには 0 以上のノード ブロックを指定できます。
   Node "Server001"
   {
      # 次に、リソース ブロックを指定します。


      # WindowsFeature はノード ブロックで使用できる組み込みリソースです。
      # この例は Web サーバー (IIS) の役割がインストールされていることを確認します。
      WindowsFeature MyRoleExample
      {
```

```
            Ensure = "Present" # 役割をアンインストールするには、Ensure を "Absent" に設定します。
            Name = "Web-Server"
        }

        # File はファイルおよびディレクトリを管理するために使用できる組み込みのリソースです。
        # この例はソース ディレクトリのファイルがターゲット ディレクトリに存在することを確認します。
        File MyFileExample
        {
            Ensure = "Present"  # Ensure を "Absent" に設定することもできます。
            Type = "Directory" # 既定は “File” です。
            Recurse = $true
            # これは Web ファイルが存在するパスです。
            SourcePath = $WebsiteFilePath
            # Web ファイルが存在することを確認するパス。
            DestinationPath = "C:\inetpub\wwwroot"
            # これで、このブロックが実行する前に MyRoleExample が正常に完了します。
            Requires = "[WindowsFeature]MyRoleExample"
        }
    }
}
```

5. ノード ブロックの名前を "Server001" から Windows 機能がインストールされていること、および「C:¥inetpub¥wwwroot」フォルダーにソース ディレクトリからの適切なコンテンツが含まれていることを確認するサーバーの名前に変更します。別の方法として、ノードの名前を "localhost" に設定することもできます。この場合、構成をローカルに適用できます。
6. サンプル スクリプトを実行します。スクリプトがコンソール ウィンドウに表示されます。

これで構成例が定義されました。次に、この構成を呼び出して、実行する必要があります。次に例を示します。

```
PS C:\Scripts> MyWebConfig
```

構成を呼び出すと、MOF
ファイルが作成され、構成ブロックと同じ名前の新しいディレクトリに配置されます。新しいディレクトリは、現在のディレクトリの子ディレクトリです (MOF ファイルに別のディレクトリを指定するには、構成を呼び出すときに OutputPath パラメーターを使用します)。新しい MOF ファイルには、ターゲット ノードの構成情報が含まれます。構成を実行するには、次のコマンドを実行します。

```
Start-DscConfiguration -Wait -Verbose -Path .\MyWebConfig
```

このコマンドレットは DSC システムの一部です。Wait パラメーターは、コマンドレットを対話的に実行するオプションです。このパラメーターを使用しない場合、コマンドレットはジョブを作成して返します。構成を呼び出すときに OutputPath パラメーターを使用する場合、Start-DscConfiguration コマンドレットの Path パラメーターを使用して同じパスを指定する必要があります。

ノードおよび構成ブロックをネストする場合の規則

次のネスト規則が適用されます。

- 構成ブロックの内部には、ゼロ以上のノード ブロックを指定できます。
- ターゲット ノードでは、構成ブロック内に複数のブロックを設定できます。つまり、同じノード ブロック識別子が 1 つの構成ブロック内に複数存在することがあります。これは、機能的には 1 つのノード ブロック内のすべてのブロックのコンテンツをグループ化することと同等です。
- 特定の種類のリソースは、同じノード ブロック内にゼロ以上のブロックを持つことができます。しかし、重複したリソース識別子を指定することはできません。1 つのノード ブロック内の各リソース ブロックの識別子は一意である必要があります。

構成パラメーターの宣言

パラメーターを取る構成ブロックを定義できます。以下に例を示します。

```
Configuration MyParametrizedConfiguration
{
    # パラメーターはオプションです。
    param ($MyTargetNodeName, $MyGroupName)

    Node $MyTargetNodeName
    {
        # Group はローカルの Windows グループを管理するために使用できる組み込みのリソースです。
        # この例は $MyGroupName によって指定された名前のグループが存在することを確認します。
        Group MyGroupExample
        {
            Ensure = "Present" # この GroupName の名前のグループが既に存在するかどうかを確認します。
                               # 存在しない場合は作成します。
            Name = $MyGroupName
        }
    }
}
```

上記の構成は次のコマンドで呼び出すことができます。

```
MyParametrizedConfiguration -MyTargetNodeName "Server001" -MyGroupName "TestGroup"
```

まとめ

Windows PowerShell 4.0 の以下の機能を使用すると、包括的で回復性の高いシンプルな方法で Windows サーバーを自動化できます。

- コンピューター上でセッションを開始し、セッションを切断した後に (別のコンピューターから) 再開する機能
- 定義済みのスケジュールに従って、スクリプトおよびワークフローを実行し、後で取得できるよう結果を保存するジョブ スケジュール
- 時間のかかるタスクを複数のコンピューター間にわたって回復性の高い方法で自動化するワークフロー
- 検索および実行が容易な 3,300 以上の新しいコマンドレット
- 構成のずれを防止する Desired State Configuration

サーバー マネージャーでのマルチサーバー管理および機能展開

Windows Server 2012 R2 では、サーバー マネージャーの機能は大きく拡張され、物理サーバーおよび仮想サーバーの両方に対するリモートでの役割および機能の展開、リモートでの役割および機能の管理、カスタム サーバー グループの作成など、マルチサーバー タスクが容易になりました。

Windows Server 2012 R2 のサーバー マネージャーを使用することにより、IT プロフェッショナルは、システムへの物理アクセスや各サーバーへのリモート デスクトップ プロトコル (RDP) 接続のいずれも必要とすることなく、ローカルのデスクトップからサーバーおよびオフライン仮想ハード ディスクのプロビジョニングを行うことができます。管理者はサービス マネージャーを使用して、単一の統合コンソールからサーバー グループを総体として管理できるので、ビジネス クリティカルな問題に迅速かつ俊敏に対応できます。

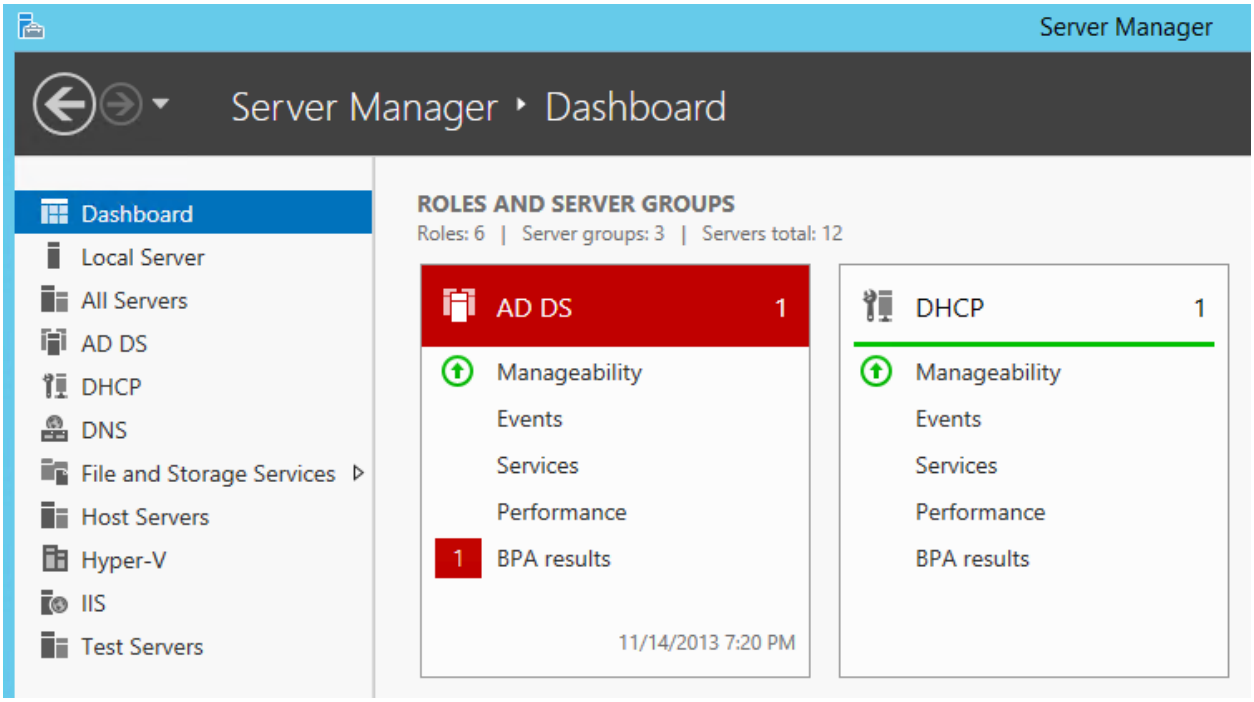
技術仕様

Windows Server 2012 のサーバー マネージャーには、多くのマルチサーバー管理機能が搭載されています。次のセクションでは、これらの新機能について説明します。

マルチサーバーのエクスペリエンス

サーバー マネージャーでは、サーバー プール内の複数のサーバーの管理およびサーバー グループの作成を行うことができます。グループを使用すると、複数のサーバーを論理的なビューに編成することができます (マイ メイン ホスト サーバーやマイ テスト サーバーなど)。既定では、サーバー マネージャーではサーバーは役割ごとにグループ化されます。

図 14. 複数のサーバーを管理できるサーバー マネージャーのダッシュボード

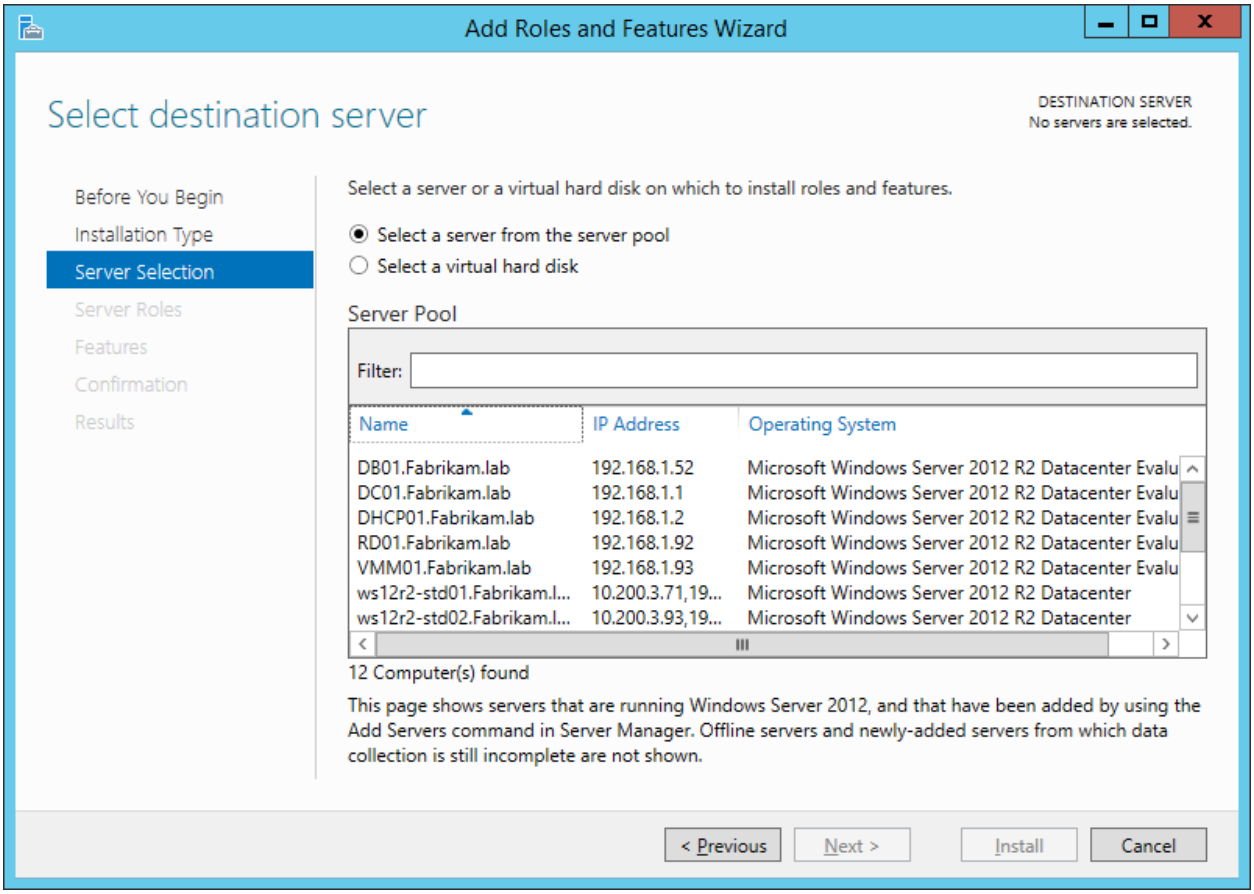


リモート サーバーまたはオフライン仮想ハード ディスクへのワークロードの効率的な展開

Windows Server 2008 では、役割を機能は、ローカル サーバー上で実行するサーバー マネージャーの役割の追加ウィザードまたは機能の追加ウィザードを使用して展開します。したがって、サーバーへの物理アクセスまたは RDP によるリモート デスクトップ アクセスが必要です。リモート サーバー管理ツールをインストールすると、Windows ベースのクライアント コンピューター上でサーバー マネージャーを実行できますが、リモート展開がサポートされないので役割および機能の追加は無効になります。

Windows Server 2012 R2 では、展開機能が拡張され、役割および機能の堅牢なリモート展開がサポートされます。Windows Server 2012 R2 のサーバー マネージャーを使用することにより、IT プロフェッショナルは、システムへの物理アクセスや各サーバーへの RDP 接続の有効化を必要とすることなく、ローカルのデスクトップからサーバーのプロビジョニングを行うことができます。

図 15. 管理対象サーバーへのリモートでの役割および機能の追加



リモート サーバーまたはオフライン仮想ハードディスクへの役割と機能のインストール

Windows Server 2012 R2 のサーバー マネージャーを使用すると、統合された役割と機能の追加ウィザードを使用して、単一のセッションで役割と機能の両方を展開できます。Windows Server 2012 R2 の役割と機能の追加ウィザードは、インストール プロセスの一部として、展開に選択した 1 つのサーバー上で検証を実行します。サーバーが役割をサポートするように適切に構成されていることを別途、事前に検証する必要はありません。

監理者は、サーバー マネージャーからリモート サーバーおよびオフラインの仮想ハード ディスクに役割および機能を展開できます。役割と機能の追加ウィザードを使用すると、単一のセッションで目的の役割および機能をオフライン仮想ディスクに追加できるので、目的の構成を一貫して繰り返すことができます。

役割と機能の追加ウィザードでは、役割のインストールのプロセスは以前と同様のプロセスです (以前のリリースの Windows Server の役割の追加ウィザードとも一貫しています)。しかし、いくつかの変更が適用されています。オフライン仮想ハード ディスクへのリモート展開およびインストールをサポートするために、いくつかの役割が

初期構成 (以前の役割の追加ウィザードで実行されたタスク) がインストール後の構成ウィザード荷移動されました。一部のオフライン仮想ハード ディスク展開では、仮想マシンの初回起動時にインストールが実行されるようスケジュールされます。

バッチ展開

Windows Server 2012 R2 では、役割と機能の追加ウィザードを使用して、構成オプションを XML ファイルにエクスポートして、後で Windows PowerShell の展開コマンドレットで使用できます。Windows PowerShell のファンアウト機能を使用して、複数のリモート サーバーに役割および機能のバッチ展開を実行し、以前のウィザード ベースの展開で保存された構成設定を適用できます。

他の管理ツールとの統合

サーバー マネージャーは、依然としてサーバー管理ツールの主要なアクセス ポイントであり、開始ポイントです。サーバーマネージャーは、管理しているリモート サーバーの状況に応じて、これらのツールを起動します (サポートされている場合)。新しい役割固有のツール (ファイル記憶域管理、リモート デスクトップサービス、IP アドレス管理など) が、サーバー マネージャーのコンソールに統合されています。

複数のサーバーにわたるサーバーの役割の管理

サーバーの役割の管理は向上し、単一サーバーの単一の役割モデルから、複数のサーバーの役割を単一の管理アプリケーションを使用してリモートで管理できるモデルに移行しました。

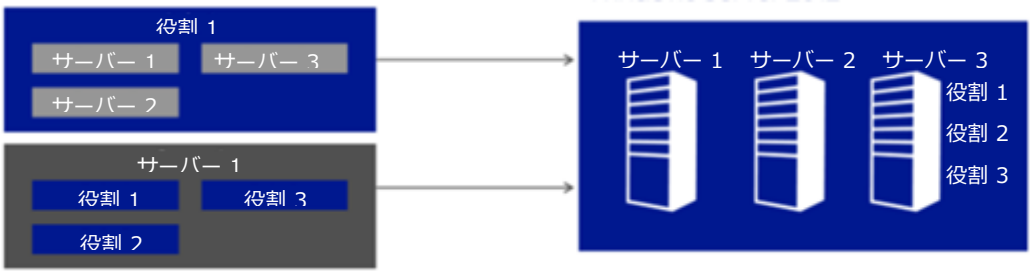


図 16. 1 つの特定の役割の複数サーバーの管理

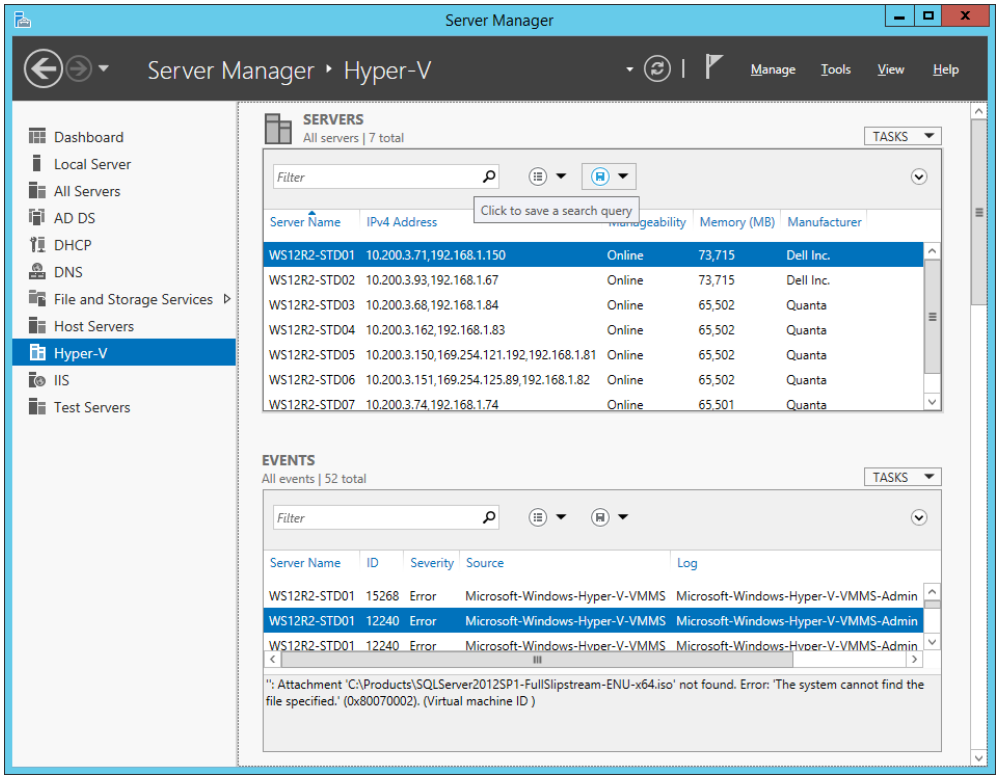
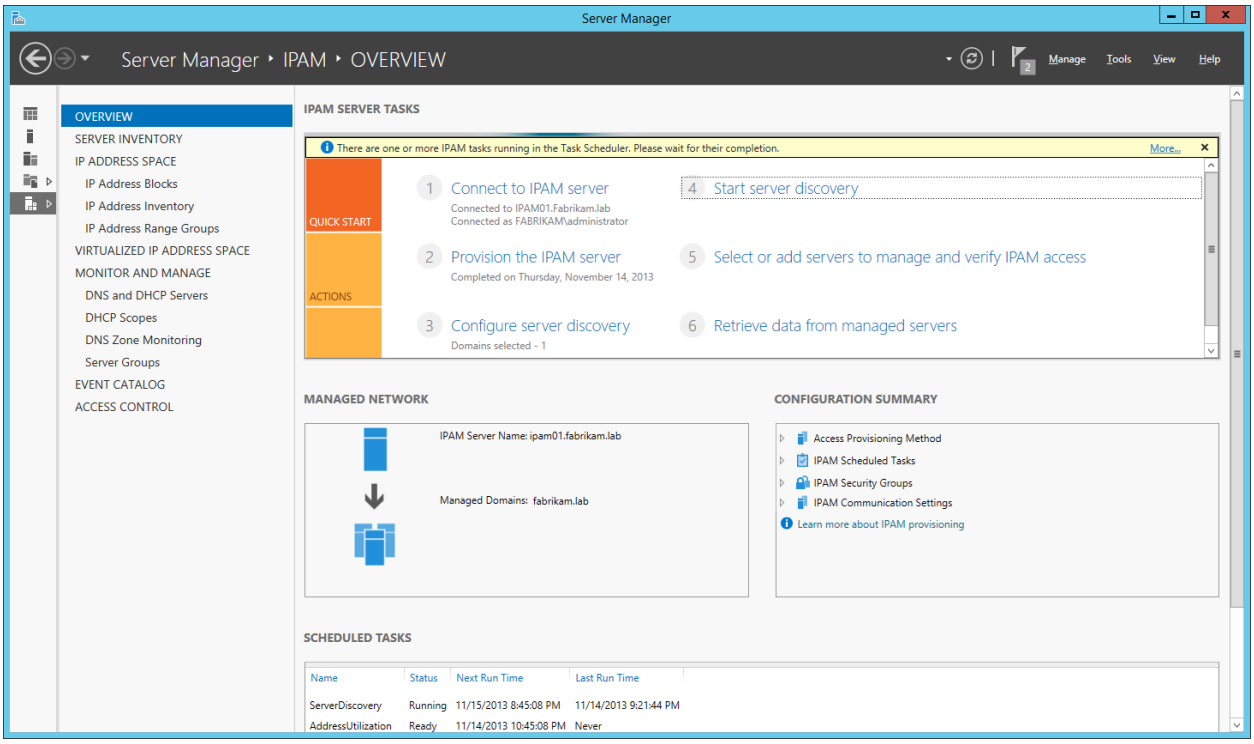


図 17. サーバー マネージャーによる IP アドレス管理



パフォーマンスへの最小の影響

サーバー マネージャー ダッシュボードの既定のポーリング サイクルは 10 分です (ユーザーがコンソールで変更できます)。既定で比較的頻度の低いポーリング サイクルが使用され、各ポーリングでは増分データのみが返されることから、個々のサーバーにおけるパフォーマンスと負荷の影響は最小限に抑えられます。サーバー マネージャーは新しい Windows Management Instrumentation (WMI) プロバイダーおよび Windows PowerShell コマンドレットを使用して、サーバーから更新されたステータス情報を入手します。

リモート サーバー管理ツール

Windows Server 2012 R2 で優先される展開オプションはサーバー コアです。サーバー マネージャーでは、サーバーを最小限のサーバー インターフェイスか GUI 使用サーバーで実行する必要があるので、IT 管理者は、リモート サーバー管理ツール (RSAT) を使用することで、Windows 8.1 を実行するリモート コンピューターから、Windows Server 2012 R2 を実行しているコンピューターにインストールされている役割と機能を管理できます。RSAT は、Microsoft ダウンロード センターから入手できます。

まとめ

Windows Server 2012 R2 のサーバー マネージャーでは、以下のようにデータセンターの管理が強化されています。

- 明確で強力な役割中心のダッシュボードによる複数のサーバーの容易な管理
- 新しいサーバーの構成プロセスの簡素化
- リモート サーバーおよびオフライン仮想ハード ディスクへの役割および機能の展開
- 単一のツールによる複数のサーバーの状態の明確な概要の取得
- RSAT を使用した Windows 8.1 からの Windows Server 2012 R2 の管理

結論

今日、IT プロフェッショナルは、増加するミッション クリティカルなサーバーとサービスを少ないリソースで管理および維持しなければならないという課題に直面しています。Windows Server 2012 では、機能が向上した標準標準モデル、プロトコル、API を採用し、Windows PowerShell およびサーバー マネージャーの新機能と強化された機能を提供することで、この問題に対処しています。同時に、これらの機能強化により、監理者はマルチサーバー環境を効果的にコスト効率良く管理できます。