Microsoft

PowerShell Remoting Basics

Learnings covered in this Unit

Enabling PowerShell remoting

Understanding PowerShell remoting

Using PowerShell remoting

Enable PowerShell Remoting

Understanding PowerShell Remoting

PowerShell **Remoting** cmdlets allow for **code** to be executed on one or more **remote** machines

Modern remoting cmdlets use **CIM**, and **WS-MAN** with a dedicated **port**

Legacy remoting cmdlets use **DCOM** and **RPC**, with very **little** functionality

Requirements

Windows PowerShell **2.0** or later, on local **and** remote computers

Remoting must be **enabled** on client machines and is enabled by **default** on Windows Server 2012 and later server versions

Local Administrators or **Remote Management Users** are allowed access by default

Enable PowerShell remoting interactively

Enable-PSRemoting performs the following **actions**:

Starts the **Windows Remote Management (WinRM)** service and sets it to **automatic** startup
Creates an **HTTP listener** to accept remote requests on any IP address for TCP port **5985**
Enables a **firewall exception** for WS-Management
Several **other changes** occur as well, which can be found in the help documentation

Enable PowerShell Remoting using Group Policy

Set **WinRM** Service to **Automatic** Startup

Computer Configuration | Policies | Windows Settings | Security Settings | System Services | Windows Remote Management (WS-Management)

Set Windows **Firewall** Inbound **rule** for Windows Remote Management

Computer Configuration | Policies | Windows Settings | Security Settings | Windows Firewall with Advanced Security | Inbound Rules | Windows Remote Management

Allow remote server management (create **listeners**)

Computer Configuration | Administrative Templates | Windows Components | Windows Remote Management (WinRM) | WinRM Service | Allow remote server management through WinRM

Demonstration: Enable PowerShell Remoting

Using PowerShell Remoting

Types of Remoting

Native OS Remoting (Legacy)

- Built-In cmdlets that take a **ComputerName** parameter but do **not** contain a **session** parameter.
- Does **not** need PowerShell remoting **enabled**
- **Limited** functionality: uses **built-in** Windows services
- Transports over **DCOM** and **RPC**

```
PS> Get-Command -ParameterName ComputerName –Module Microsoft.PowerShell.Management
```

Remoting (Modern)

- **Requires** PowerShell remoting to be **enabled**
- **Sessions** are **created** on a single machine or group of machines
- Transports over **WS-MAN** on a dedicated **port**

PowerShell Remoting

Operates using WS-Man (**WinRM**) using **a** dedicated **port**

Flexible and **powerful**, allowing any **command** on any **machine**

Can execute commands on **multiple machines** at once

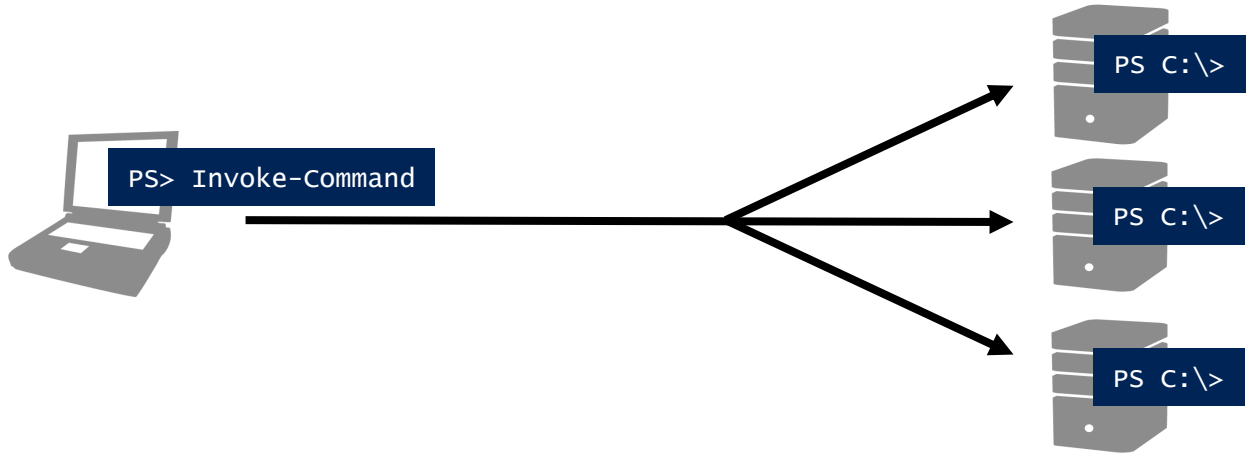
Supports **interactive** sessions, similar to **SSH** sessions

Invoke-Command

Execute any **script block** on any number of machines

Returns the results with a **PSComputerName** property

Execution happens in **parallel**



Remote Computers

Script block to be executed

```
PS> Invoke-Command -ComputerName MS,DC,Win10 -ScriptBlock {Get-Culture}

LCID        Name        DisplayName                    PSComputerName
----        ----        -----------                    --------------
1033        en-US       English (United States)        MS
1033        en-US       English (United States)        DC
1033        en-US       English (United States)        WIN10
```

Using Alternate Credentials

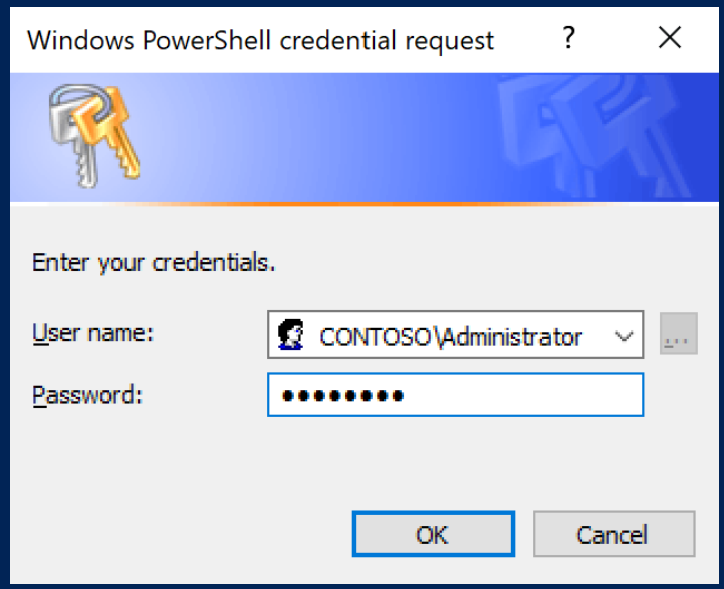
The **-Credential** parameter specifies alternate credentials to **authenticate** to the remote machine

Credentials can be saved to a **variable** with **Get-Credential**

Useful when logged in as a standard **user** account but **administrative permissions** are required on the **remote** machine

```
PS> Invoke-Command -ComputerName MS -ScriptBlock {$env:USERNAME}
    -Credential CONTOSO\Administrator

Administrator
```



Invoke-Command with Script Files

The **-FilePath** parameter sends a **local** script to a **remote** computer

Converts code from file into a **script block**

Use **-ArgumentList** to specify the values of **parameters** for the **script**

```
PS> Invoke-Command -ComputerName MS, DC -FilePath C:\MyScript.ps1

LCID        Name        DisplayName                    PSComputerName
----        ----        -----------                    --------------
1033        en-US       English (United States)        MS
1033        en-US       English (United States)        DC
```

Demonstration: Using Invoke-Command

Enter-PSSession

Starts an **interactive** session with a single remote computer

Code is **executed** on **remote** machine

Results **live** on **remote** machine

End session with **Exit-PSSession**

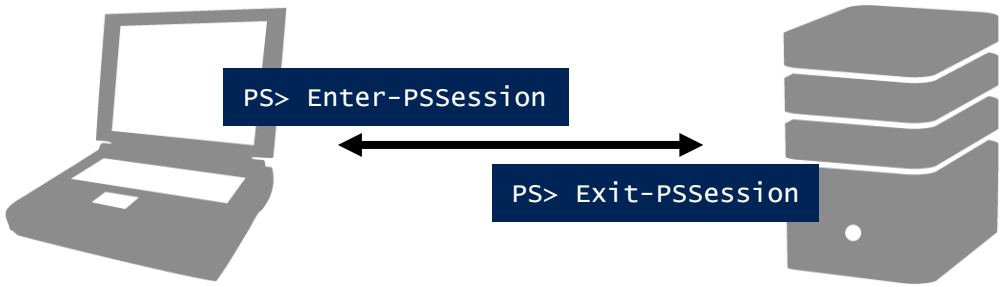
```
PS> $env:COMPUTERNAME
WIN10

PS> Enter-PSSession -ComputerName MS

[MS]: PS> $env:COMPUTERNAME
MS

[MS]: PS> Exit-PSSession

PS> $env:COMPUTERNAME
WIN10
```



Demonstration: Using Enter-PSSession

Temporary vs Persistent Sessions

Temporary sessions

- **Closes** the session when the command is **completed** or when the interactive session **ends**
- Uses the **-ComputerName** parameter
- Executes a **single** script block on a remote machine

Persistent sessions

- A **PSSession** that remains **available** even after a command is **completed** or an interactive session ends
- Uses the **-Session** parameter
- Can **disconnect** and **reconnect**, as needed
- Remains open until it is **deleted** or **times out**

Using Persistent Sessions

New-PSSession creates a persistent connection to a remote computer

Can be used with **Invoke-Command**, ***-PSSession** cmdlets, and other cmdlets

Generally saved into a **variable** for easier reuse

Session **exists** on the **remote** computer and is **available** to **connect** to and use as needed

```
PS C:\> $Session = New-PSSession -ComputerName MS

PS C:\> Invoke-Command -Session $Session -ScriptBlock {$Var = 123}

PS C:\> Invoke-Command -Session $Session -ScriptBlock {$Var}
123
```

Variable still exists on remote machine

Demonstration: Using Persistent Sessions

Lab 11:
Remoting Basic

60 minutes