

How to use PowerShell Invoke-Command

Do you want to run a PowerShell command on a remote computer or server? Then you don't need to open a remote desktop connection, because you can just use the PowerShell Invoke-Command for that.

Invoke-Command allows you to run PowerShell commands and scripts on one or more remote computers. while redirecting the results to your own console. This makes it a great cmdlet to manage remote devices quickly.


In this article, we are going to take a look at what is required to use Invoke-Command and how to use the cmdlet.

Prerequisites Invoke-Command

Before we can use the Invoke-Command on a remote server or computer, we will need to make sure that the Windows Remote Management service is running. Also if you have the Windows firewall enabled in your network, then we need to check if the firewall is not blocking the connection.

To check if a remote computer accepts Remote Management, we can use a simple PowerShell command, **Test-WSMan**

Test-WSMan -Computername la-srv-dc01



```
Administrator: Windows PowerShell
PS C:\test> Test-WSMan -ComputerName la-srv-dc01

wsmanid      : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor  : Microsoft Corporation
ProductVersion : OS: 0.0.0 SP: 0.0 Stack: 3.0

PS C:\test> |
```

If you get an error that you cannot connect to the destination, then the service Windows Remote Management is not running. To enable it we can use the PowerShell cmdlet **Enable-PSRemoting** on the remote computer:

Enable-PSRemoting

The advantage of this command is that it not only starts the Remote Management services but also sets it to start automatically and creates the required exception rule in the firewall.

But even a better option is to create a new Group Policy Object (GPO). This way you can easily enable the Remote Management service on multiple computers or servers.

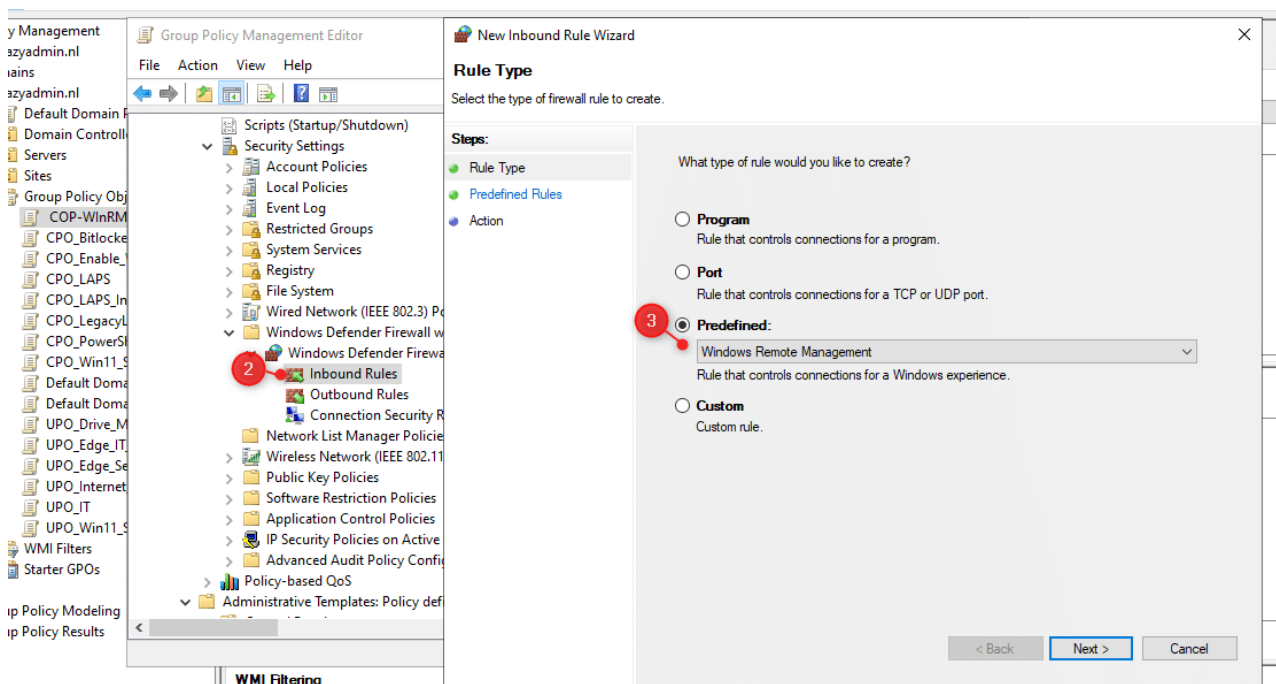
You can find the policy setting in **Computer Configuration > Windows Settings > Security Settings > System Services**.

Firewall rule

The Windows Firewall will block all incoming connections, including the Windows Remote Management connection. So before we can use the Invoke-Command, we will need to allow the inbound connection.

We can use the same GPO to create the inbound rule as well.

1. Navigate to **Computer Configuration > Windows Settings > Security Settings > Windows Defender Firewall with Advanced Security**
2. Right-click on **Inbound Rules** and select **New Rule**
3. Select **Predefined** and choose **Windows Remote Management**
4. Select the rule with profile **Domain, Private**
5. Choose **Allow the connection** and click **finish**



Make sure that the new policies are updated on the server/client, by using the [GPOupdate command](#).

Using the Invoke-Command

The Invoke-Command in PowerShell allows you to run a command or script on a remote computer. When you need to run multiple commands on a remote computer, then it's better to use the New-PSSession cmdlet.

The structure of the Invoke-Command is pretty straightforward, but nevertheless, there are still quite a few options (parameters) that we can use:

Parameter	Description
ComputerName	Computername or IP Address to run the command on. For multiple computer, separate the names with a comma ,
Credential	Credentials to connect to remote computer
FilePath	Specifies the path to a local script to run on the remote computer(s)
AsJob	Run the remote command as background job on the local computer
InDisconnectedSession	Used to run the command in a disconnected session on the remote computer
ScriptBlock	Script or command to run
ArgumentList	Used to pass arguments to the cmdlet in the scriptblock

Invoke-Command Parameters

So to simply run a PowerShell command on a remote computer, we only need to specify the name of the remote computer and the command that we want to run inside a scriptblock. For example, to retrieve the computer info from the domain controller we can do:

Invoke-Command -ComputerName la-srv-dc01 {Get-ComputerInfo}

```
Administrator: Windows PowerShell
PS C:\> Invoke-Command -ComputerName la-srv-dc01 {Get-ComputerInfo}

PSComputerName           : la-srv-dc01
RunspaceId                : b207be94-82fb-4f71-962f-a47fab109914
WindowsBuildLabEx         : 20348.1.amd64fre.fe_release.210507-1500
WindowsCurrentVersion     : 6.3
WindowsEditionId          : ServerStandardEval
WindowsInstallationType   : Server
WindowsInstallDateFromRegistry : 4/16/2023 2:40:00 PM
WindowsProductId          : 00454-40000-00001-AA032
WindowsProductName        : Windows Server 2022 Standard Evaluation
WindowsRegisteredOrganization :
WindowsRegisteredOwner    : Windows User
WindowsSystemRoot         : C:\Windows
WindowsVersion            : 2009
OSDisplayVersion          : 21H2
BiosCharacteristics        : {3, 9, 15, 16...}
BiosBIOSVersion           : {VIRTUAL - 1, Hyper-V UEFI Release v4.1, Microsoft - 100032}
BiosBuildNumber           :
```

This will run the command `Get-ComputerInfo` on the domain controller (la-srv-dc01) and return the results to the console.

Running a Script

In the example above we only run a simple PowerShell cmdlet. But we can also run complete scripts on the remote computer. You could just write the script inside the scriptblock parameter, but it's easier to store the script in a variable or use a file.

The example below works fine, but it's a bit harder to read this way:

```
Invoke-Command -ComputerName la-srv-dc01 -ScriptBlock {Get-CimInstance Win32_LogicalDisk | Select-Object DeviceID, MediaType, @{Name="FreeSpace (GB)"; Expression="{0:N2}" -f ($_.FreeSpace / 1GB)}}}
```

A better option would be to store the script inside a variable first. We can then simply pass the variable to the scriptblock:

```
$script = {  
Get-CimInstance Win32_LogicalDisk |  
Select-Object DeviceID, MediaType, @{Name="FreeSpace (GB)"; Expression="{0:N2}" -  
f ($_.FreeSpace / 1GB)}}  
}
```

```
Invoke-Command -ComputerName la-srv-dc01 -ScriptBlock $script
```

The third option is to store the script on your local computer and refer to the script using the `-FilePath` parameter:

```
Invoke-Command -ComputerName la-srv-dc01 -FilePath c:\test\diskInfo.ps1
```

Run on Multiple Computers

One of the big advantages of the Invoke-Command cmdlet in PowerShell is that you can also run it on multiple computers simultaneously. There are a couple of options to do this, we can simply specify the computer names separated by a comma:

```
Invoke-Command -ComputerName la-srv-dc01, la-srv-app01, la-srv-file01 -FilePath c:\test\diskInfo.ps1
```

Another option to write this is to use a hashtable and splatting. Splatting the parameters will make your script easier to read when you need to add a lot of properties.

```
$parameters = @{  
ComputerName = 'la-srv-dc01', 'la-srv-app01', 'la-srv-file01', 'la-win11-lab03'  
ScriptBlock = { Get-ComputerInfo }  
}
```

```
Invoke-Command @parameters
```

When you need to run a command on many computers, let's say a hundred or more, then you want to minimize the load on your local computer. For this, we can use the `InDisconnectedSession` parameter.

If you run the Invoke-Command cmdlet normally, then your local computer will connect to the remote computer, start the script, wait for the results, and disconnect. The waiting parts consume the most time and resources from your local computer, especially when connecting to 100 or more computers.

With the **InDisconnectedSession** parameter, the cmdlet will only connect to the remote computer, start the script, and then disconnect. This reduces the whole process time significantly.

```
$parameters = @{  
  ComputerName = (Get-Content -Path C:\Test\Servers.txt)  
  InDisconnectedSession = $true  
  FilePath = '\\lazyadmin\scripts\InstallWinUpdates.ps1'  
}  
Invoke-Command @parameters
```

Using Credentials

You will need to have permission on the remote computer to run the PowerShell commands. When running the command on a server you probably won't have the correct permission from your local workstation.

To solve this we can use the **-Credentials** parameter. Here you can enter the user account that you want to use to run the command. PowerShell will prompt you for the password when you execute the **Invoke-Command** cmdlet with the **-Credential** parameter:

```
Invoke-Command -ComputerName la-srv-dc01 -Credential lazyadmin\administrator -  
Scriptblock {Get-ComputerInfo}
```

Just like with most PowerShell cmdlets, you can also pass a credential object to the **-Credential** cmdlet. This method is especially handy when you want to run multiple commands or need to run different commands on different remote computers:

```
$cred = Get-Credential  
Invoke-Command -ComputerName la-srv-dc01 -Credential $cred -Scriptblock {Get-  
ComputerInfo}
```

Include Local Variables

Using variables inside the scriptblock goes a bit differently than you might expect. Let's take the following simplified example, we want to get all the files from a given path on the remote computer. In this case, we have stored the path inside a variable:

```
$path = "c:\test"  
Invoke-Command -ComputerName la-srv-dc01 -Scriptblock {Get-Childitem -Path $path}  
The example above isn't going to work. The remote computer, in this case la-srv-dc01, doesn't know what the variable $path is, so it won't return any results.
```

To pass the variable path into the scriptblock, we will have to use the **Using** scope modifier. The Using scope indicates that the variable is created in the local session and not in the remote one. So to pass the **\$path** variable to the remote session we need to do:

```
$path = "c:\test"
```

```
Invoke-Command -ComputerName la-srv-dc01 -Scriptblock {Get-Childitem -Path  
$Using:path}
```

Wrapping Up

The PowerShell Invoke-Command is a great tool to manage remote computers. It allows you to quickly execute a command without the need to open a remote desktop. Especially when you need to run the command on multiple computers at once.

Make sure that you also look at the New-PSSession cmdlet, this is really a better option when you need to execute multiple commands on a remote computer.

I hope you found this article helpful, if you have any questions, just drop a comment below.

Did you **Liked** this **Article**?

Get the latest articles like this **in your mailbox**
or share this article

I hate spam to, so you can unsubscribe at any time.