Microsoft

# AZ-040

# Automate Administration with PowerShell

# Learning Path 8: Administer remote computers with Windows PowerShell

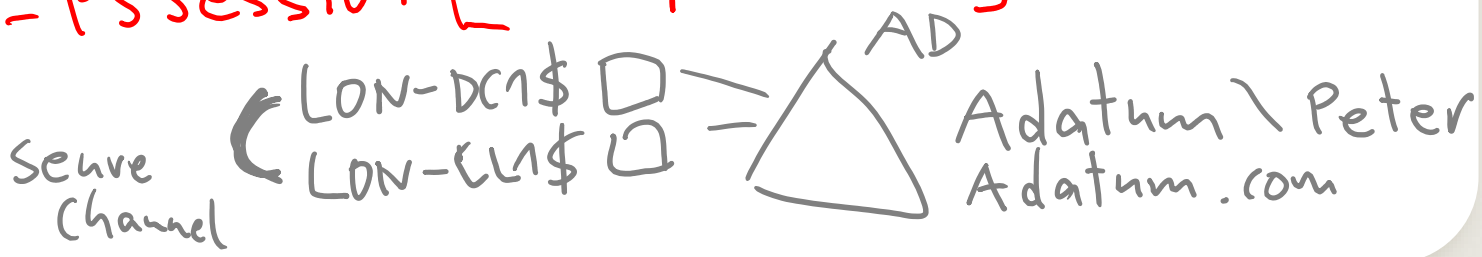
## Learning objectives

- Manage single and multiple computers by using Windows PowerShell remoting
- Use advanced Windows PowerShell remoting techniques
- Manage persistent connections to remote computers by using Windows PowerShell sessions

$DC1 = New-PSSession [-ComputerName] LON-DC1

:80
:443

:5985
:5986
WinRM




Enable-PSRemoting ⊆ Start WinRM
WinRM auto
5985 FW

# Agenda

LP 1 Get started with Windows PowerShell

LP 2 Maintain system administration tasks in Windows PowerShell

LP 3 Work with the Windows PowerShell pipeline

LP 4 Work with PowerShell providers and PowerShell drives in Windows PowerShell

LP 5 Querying management information by using CIM and WMI

LP 6 Use variables, arrays, and hash tables in Windows PowerShell scripts

LP 7 Create and modify scripts by using Windows PowerShell

LP 8 Administer remote computers by using Windows PowerShell

LP 9 Manage cloud resources by using Windows PowerShell

LP 10 Manage Microsoft 365 services by using Windows PowerShell

LP 11 Create and manage background jobs and scheduled jobs in Windows PowerShell

# Overview

Windows PowerShell remoting is a technology that enables you to connect to one or more remote computers, and then have them run commands on your behalf.
It has become the foundation for remote administration in Windows operating system-based environments. Windows PowerShell 5.0 is installed as part of Windows Management Framework 5.0, which offers both Windows PowerShell functionality and remoting capabilities. For example, the graphical Server Manager console in Windows Server relies on remoting to communicate with servers, even to communicate with the local computer on which the console is running. Past Windows operating system versions have included remoting, but its use was largely optional. Now, the technology is quickly becoming a mandatory component for every environment.

Modules:

- Manage single and multiple computers by using Windows PowerShell remoting
- Use advanced Windows PowerShell remoting techniques
- Manage persistent connections to remote computers by using Windows PowerShell sessions

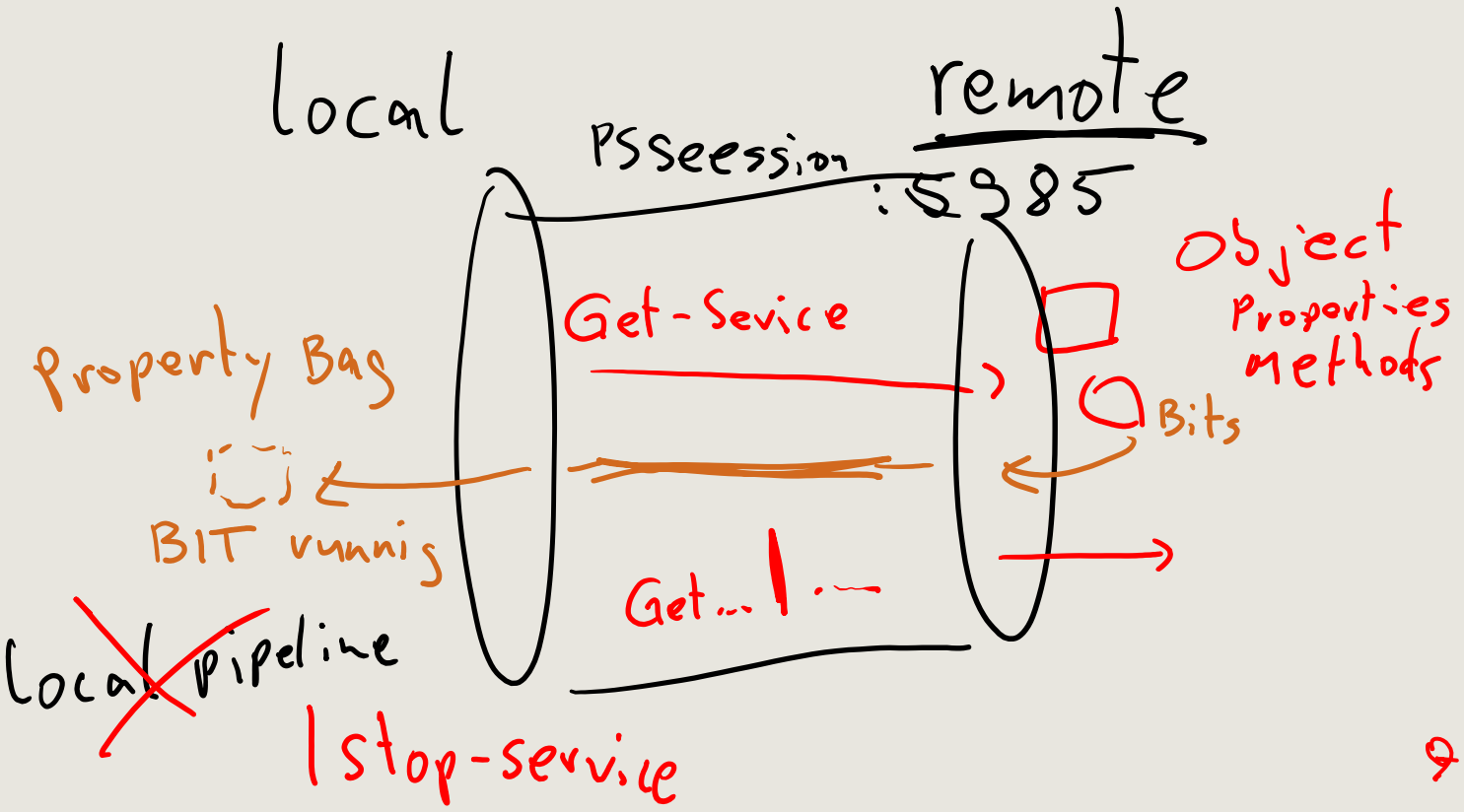
# Manage single and multiple computers by using Windows PowerShell remoting

# Module overview

Although remoting is a complex technology, working with Windows PowerShell remoting is fairly straightforward when you understand the underlying concepts. In this Module, you learn how to use remoting to perform administration on remote computers.

Units:

- Remoting overview and architecture
- Remoting versus remote connectivity
- Remoting security
- Enabling remoting
- Using one-to-one remoting
- Using one-to-many remoting
- Demonstration: Enabling and using remoting
- Remoting output versus local output

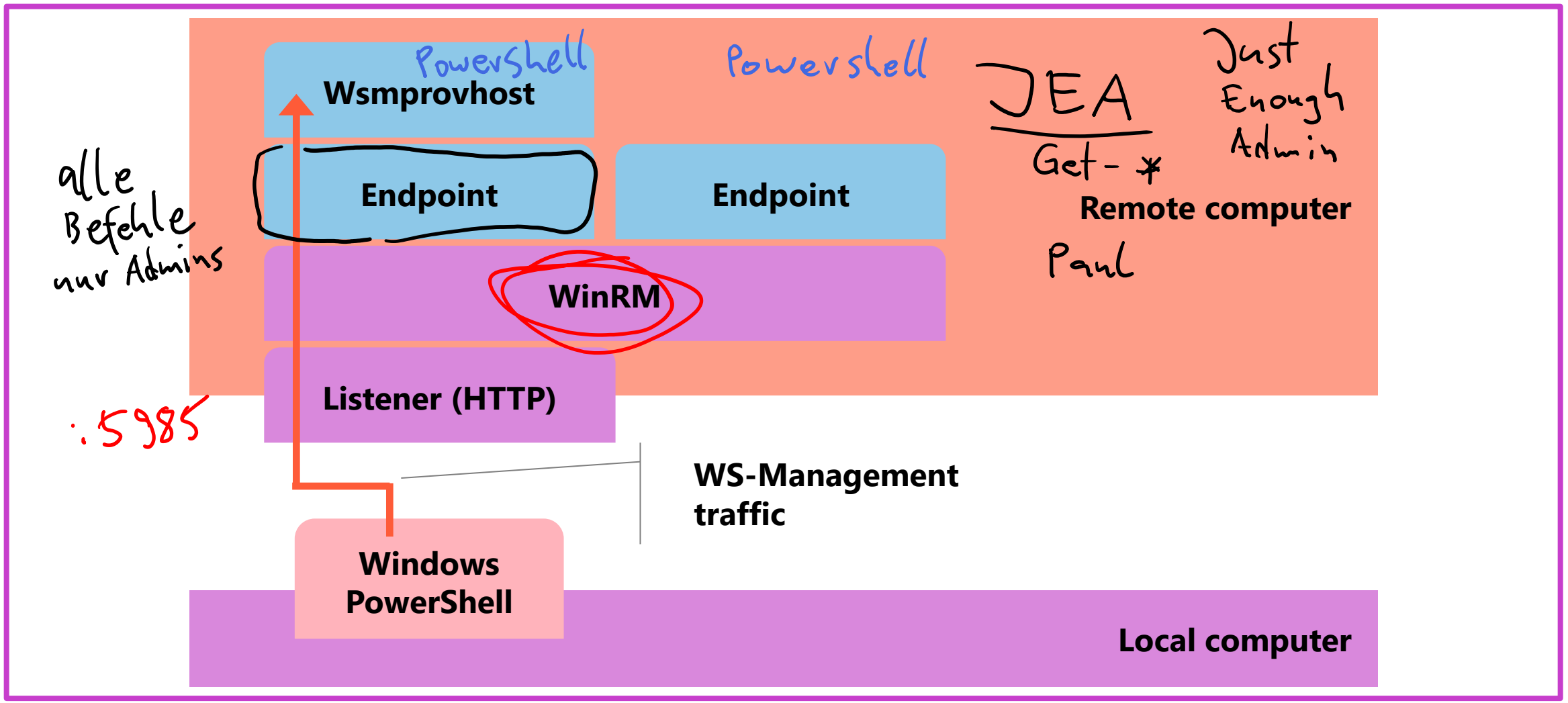


# Remoting overview and architecture

- Remoting:
  - Uses WS-MAN protocol, using HTTP (by default) or HTTPS.
  - Is implemented by WinRM service.
  - Is enabled by default on Windows Server 2019.
  - Is available on any computer running Windows PowerShell 2.0 or newer.
  - Is not enabled on any client operating system.
  - Must be enabled on any computer that will receive incoming connections.
  - Can be established by using SSH for Linux platforms.
- Many Windows PowerShell cmdlets have the -*ComputerName* parameter that enables you to collect data and change settings on one or more remote computers.

SSHd !

# Remoting overview and architecture (Slide 2)



# Remoting versus remote connectivity

- *Remoting* is the name of a specific feature that utilizes a specific service and protocol.
- When used in Windows PowerShell, use the term *Windows PowerShell remoting*.
- Remoting applies to a relatively small subset of commands that can communicate with remote computers.
- A command with a –*ComputerName* parameter doesn't necessarily mean it uses remoting.
- Nonremoting commands use their own protocols:
  - RPCs, which include WMI
  - Remote Registry Service (for example, **Get-Process**)

# Remoting security

- Remoting is security transparent; you can perform only those tasks that your credentials allow.
- Mutual authentication helps prevent delegation of credentials to spoofed or impersonated computers:
  - It works in domain environments by default.
  - It can use SSL in lieu of domain credentials.
  - It can be disabled through the **TrustedHosts** list.

# Remoting security (Slide 2)

- Remoting privacy:
  - Channel-level encryption is provided only with HTTPS connections.
  - Application-level encryption is provided with all connections.
  - Credentials are transmitted in clear text only, with the Basic authentication protocol when HTTPS is not in use (for example, to a nondomain computer on TrustedHosts list).
  - You cannot use Basic authentication unless you enable unencrypted traffic in the client configuration.

# Enabling remoting

- To enable Windows PowerShell remoting manually, run **Enable-PSRemoting** as an Administrator.
- To enable Windows PowerShell remoting centrally, configure a GPO.
- There are restrictions on client computers that have a network connection set to Public.
- Windows Server 2012 and newer enable Windows PowerShell remoting by default; no further steps are needed.
- Remoting can be enabled by using Group Policy.

# Using one-to-one remoting

One-to-one Windows PowerShell remoting is similar in concept to SSH, although different in actual operation:

1. Start with **Enter-PSSession –ComputerName *name*.**
   - The Windows PowerShell prompt changes to indicate the connected computer.
2. Exit with **Exit-PSSession.**

Exit

# Using one-to-many remoting

- The **Invoke-Command** can send a command or script to one or more remote computers in parallel.
- Results include a **PSComputerName** property that indicates the computer each result came from.
- Considerations include:
  - Throttling
  - Passing data to remote computers
  - Persistence
  - Ways to specify computer names

```
Invoke-Command -Comp [ ]
               -ScriptBlock { ... }

$DC1 = New-PSSession LON-DC1
Invoke-Command -Session $DC1, $SVR1
               { }
```

# Using one-to-many remoting (Slide 2)

```
Invoke-Command
   -ScriptBlock {
      Param($c,$r)
      New-PSDrive –Name Z
                  -Credential $c
                  -PSProvider FileSystem
                  -Root $r
      }
   -ComputerName SERVER1,SERVER2,SERVER3
   -ArgumentList (Get-Credential),'Path'
```

**This command runs on the remote computer.**

**These computers run the command.**

**This command runs on the local computer.**

# Using one-to-many remoting (Slide 3)

```
Invoke-Command
  -ScriptBlock {
     Param($c,$r)
     New-PSDrive –Name Z
                 -Credential $c
                 -PSProvider FileSystem
                 -Root $r
     }
-ComputerName SERVER1,SERVER2,SERVER3
-ArgumentList (Get-Credential),'Path'
```

**The objects in the argument list are copied into the Param() block variables on each remote computer.**

# Using one-to-many remoting (Slide 4)

```
Invoke-Command
   -ScriptBlock {
      Param($c,$r)
      New-PSDrive –Name Z
                  -Credential $c
                  -PSProvider FileSystem
                  -Root $r
      }
   -ComputerName SERVER1,SERVER2,SERVER3
   -ArgumentList (Get-Credential),'Path'
```

**The parameters are used like variables in the remote command**

# Demonstration: Enabling and using remoting

In this demonstration, you'll learn how to enable remoting on a client computer, and how to use remoting in several basic scenarios.

# Remoting output versus local output

- Results received through remoting are deserialized from XML. As a result, they're not live objects and don't have methods or events.
- As a strategy, try to have as much processing as possible occur on the remote computer, with only the final results coming back to you through remoting.

# Use advanced Windows PowerShell remoting techniques

# Module overview

Windows PowerShell remoting includes several advanced techniques that help achieve specific goals or alleviate specific shortcomings. In the previous Module, you reviewed the shortcomings of some of the basic techniques. In this Module, you'll learn about some useful advanced techniques that will help overcome these challenges.

Units:

- Common remoting options
- Sending parameters to remote computers
- Windows PowerShell scopes
- Demonstration: Sending local variables to a remote computer
- Multi-hop remoting

# Common remoting options

- Common remoting options include:
  - *Port*
  - *UseSSL*
  - *Credential*
  - *ConfigurationName*
  - *Authentication*
- Additional options are available by creating a **PSSessionOption** object, and then passing it to *–SessionOption*.

# Sending parameters to remote computers

- You cannot use local variables in the I**nvoke-Command** script block.
- You can pass data. However, you must use a specific technique.
- Pass local variables to the *–ArgumentList* parameter of **Invoke-Command**; they will map to variables in a **Param()** block inside the script block.

# Windows PowerShell scopes

- Scopes provide access protection to variables, aliases, functions, and Windows PowerShell drives:
  - Limits where they can be changed and read.
  - Ensure you don't inadvertently change an item.
- The scope modifier identifies a local variable in a remote command.
- The syntax of this modifier is **$Using:**

```
$ps = "Windows PowerShell"
Invoke-Command -ComputerName LON-DC1 -ScriptBlock {Get-WinEvent -LogName
$Using:ps}
```

# Windows PowerShell scopes (Slide 2)

```
$quantity = Read-Host "Query how many
log entries?"
```

**The $Using:quantity passes the variable *VALUE* into the remote command.**

```
Invoke-Command -
ComputerName lon-dc1 -
ScriptBlock {Get-EventLog
-LogName Security –
Newest $Using:quantity}
```

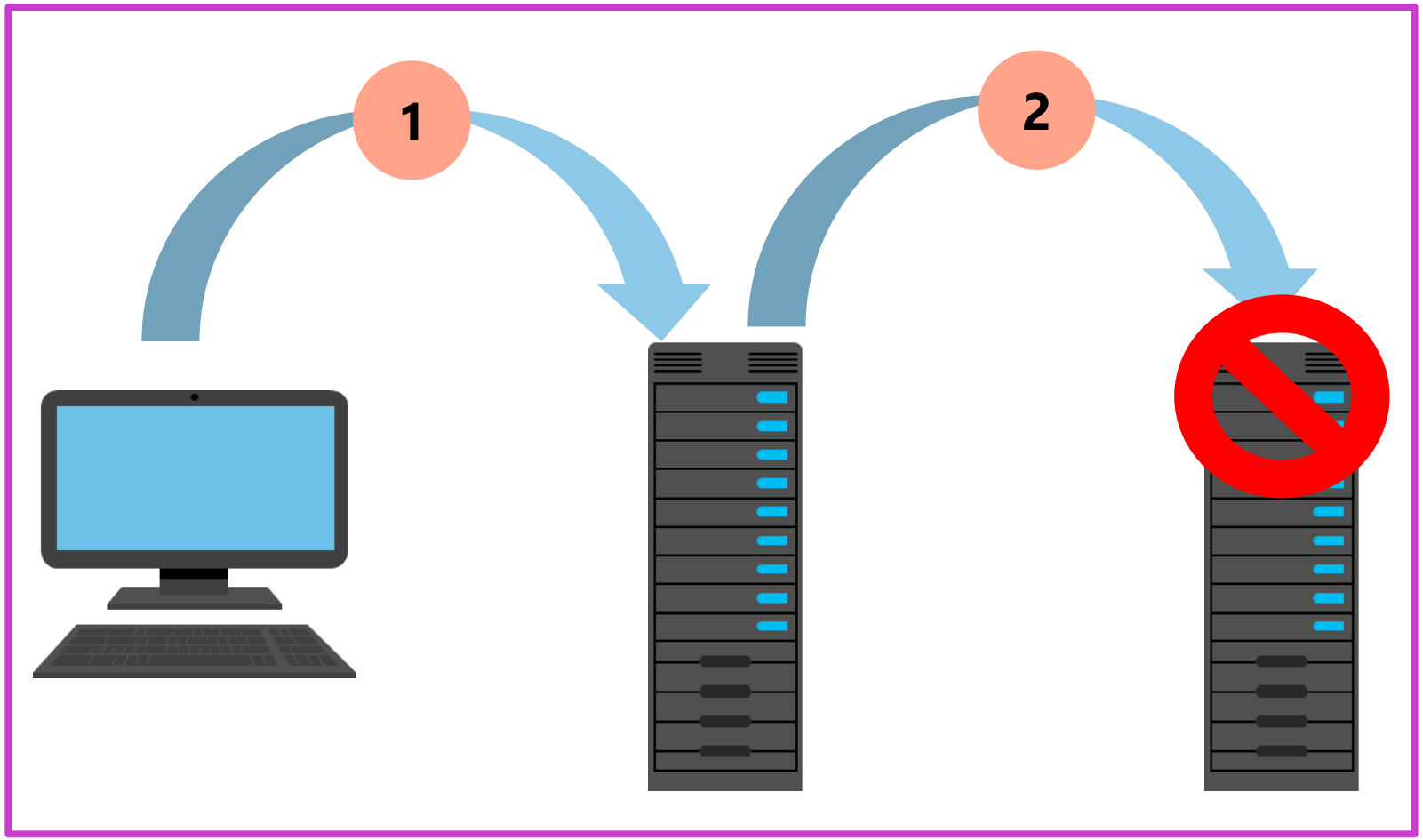
# Demonstration: Sending local variables to a remote computer

In this demonstration, you'll learn two ways to pass local information to a remote computer by using **Invoke-Command.**

# Multi-hop remoting

**You can delegate credentials in multi-hop remoting by using:**

- CredSSP (not recommended)
- Resource-based Kerberos-constrained delegation
- JEA



# Manage persistent connections to remote computers by using Windows PowerShell sessions

# Module overview

- In this Learning Path, you've learned that each remoting command you used created a connection, used it, and then closed it. However, as previously discussed, this approach doesn't provide the option to persist session data across remote connections. Using a persistent connection allows you to interactively query and manage a remote computer. In this Module, you'll learn how to establish and manage persistent connections to remote computers, known as Windows PowerShell sessions, or *PSSessions*.
- Units:
  - Persistent connections
  - Creating and using a PSSession
  - Demonstration: Using PSSessions
  - Disconnected sessions
  - Demonstration: Working with disconnected sessions
  - Implicit remoting

# Persistent connections

- PSSessions:
  - Represent a persistently running connection on the remote computer.
  - Can execute multiple sequences of commands, be disconnected, reconnected, and closed.
- Numerous configuration parameters in the drive WSMan control idle session time and maximum connections.

# Creating and using a PSSession

- Create sessions by using **New-PSSession:**
  - The command produces a reference to the PSSessions it creates.
- Assign PSSessions to variables to make them easier to refer to later.
- Pass a session object to the –*Session* parameter of **Enter-PSSession** to interactively enter that session.
- Alternatively, pass session object(s) to the –*Session* parameter of I**nvoke-Command** to run a command against those PSSessions.
- The PSSessions remain open and connected after you are finished, leaving them ready for additional use.

# Demonstration: Using PSSessions

In this demonstration, you'll learn how to create and manage PSSessions.

# Disconnected sessions

- **Disconnect-PSSession** disconnects from a PSSession while leaving Windows PowerShell running:
  - This doesn't happen automatically when you close the host application.
- **Get-PSSession –ComputerName** displays a list of your PSSessions on the specified computer:
  - You cannot review other users' PSSessions.
- **Connect-PSSession** reconnects a PSSession, making it available for use.

# Demonstration: Working with disconnected sessions

## In this demonstration, you'll learn how to:

1. Create a PSSession.
2. Disconnect a PSSession.
3. Display PSSessions.
4. Reconnect a PSSession.

# Implicit remoting

- Imports commands from a remote computer to the local one:
  - Imported commands still run on the remote computer through an established remoting session.
- Lets you utilize commands without needing to install them.
- Help is also drawn from the remote computer.

# Lab – Performing remote administration with PowerShell

# Lab: Performing remote administration with PowerShell

- **Exercise 1:** Enabling remoting on the local computer
- **Exercise 2:** Performing one-to-one remoting
- **Exercise 3:** Performing one-to-many remoting
- **Exercise 4:** Using implicit remoting
- **Exercise 5:** Managing multiple computers

## Sign-in information for the exercise(s):

Virtual machines:

- **AZ-040T00A-LON-DC1**
- **AZ-040T00A-LON-CL1**

Username: **Adatum\Administrator**

Password: **Pa55w.rd**

# Lab scenario

You're an administrator for Adatum Corporation and must perform maintenance tasks on a server running Windows Server 2019. You don't have physical access to the server, and instead plan to perform the tasks using Windows PowerShell remoting. You also have some tasks to perform against both a server and another client computer that runs Windows 10. In your environment, communication protocols such as remote procedure call (RPC) are blocked between your local computer and the servers. You plan to use Windows PowerShell remoting and want to use sessions to provide persistence and reduce the setup and cleanup overhead that improvised remoting connections will impose.

# Lab-review questions

- You established a PSSession from **LON-CL1** to **LON-DC1**, and then within that PSSession, you tried to establish a PSSession back to **LON-CL1**. This failed. Why?
- What are some of the benefits of using implicit remoting?

# Lab-review answers

- You established a PSSession from **LON-CL1** to **LON-DC1**, and then within that PSSession, you tried to establish a PSSession back to **LON-CL1**. This failed. Why?

  You receive an error that you cannot use the Enter-PSSession cmdlet to enter another PSSession. By default, you cannot establish a connection through an already-established connection.

- What are some of the benefits of using implicit remoting?

  One benefit is that administrators don't have to install administrative tools such as Windows PowerShell commands on their local computers. Instead, they can connect to a server or another computer that already has the tools, and then use them as if they were installed locally. Another benefit is that administrators can centrally monitor and control access to tools. By keeping Windows PowerShell commands on a smaller number of computers, administrators also can easily update the commands as needed.

# Learning Path Recap

## In this learning path, we:

- Learned how to use remoting to perform administration on remote computers.
- Explored some useful advanced techniques that will help overcome the limitations of basic Windows PowerShell remoting.
- Learned how to establish and manage persistent connections to remote computers, known as Windows PowerShell sessions or PSSessions.

# End of presentation