Installing Software and Running Custom Activities

Contents

[Summary 2](#_Toc405540922)

[Installation 2](#_Toc405540923)

[Prerequisites for extending the lab 2](#_Toc405540924)

[PowerShell Remoting 2](#_Toc405540925)

[Invoke-Command 2](#_Toc405540926)

[Introduction 2](#_Toc405540927)

[Persistent Sessions 3](#_Toc405540928)

[Invoke-LabCommand 3](#_Toc405540929)

[Introduction to the cmdlet 3](#_Toc405540930)

[Persistent Sessions 4](#_Toc405540931)

[Double Hop Authentication 5](#_Toc405540932)

[Software Installation 5](#_Toc405540933)

[Installing single packages 5](#_Toc405540934)

[Mass Software Installations 6](#_Toc405540935)

[What’s next? 6](#_Toc405540936)

AL Tutorial Series Part 6 - Installing Software and Running Custom Activities

## Summary

In the previous blog articles you could learn how easy it is to create even complex lab environments with AutomatedLab. This article explains how to customize your existing environment and install software to some or all machines in the lab.

## Installation

If you have a version of AutomatedLab that is earlier than [AutomatedLab](https://gallery.technet.microsoft.com/AutomatedLab-026d81f9), please uninstall it and install the latest version. You can find what you need on Microsoft TechNet: AutomatedLab.

The installation process for AutomatedLab is explained in [AutomatedLab Tutorial Part 2: Create a Simple Lab](http://blogs.technet.com/b/fieldcoding/archive/2014/07/27/automatedlab-introduction-part-2.aspx).

## Prerequisites for extending the lab

The following examples are based on the on the lab created in [AutomatedLab Tutorial Part 3: Working with Predefined Server Roles](http://blogs.technet.com/b/fieldcoding/archive/2014/11/11/automatedlab-tutorial-part-3-working-with-predefined-server-roles.aspx). The full script is provided at the end of this post.

For your convenience, all blog posts are also part of the AutomatedLab installation file. After you install AutomatedLab, you can find several lab scenarios in the **Documents > AutomatedLab Sample Scripts** folder.

## PowerShell Remoting

AutomatedLab heavily leverages Windows PowerShell remoting. The machines running Hyper-V are created locally, but all other setup and configuration tasks are triggered on the virtual machines by using Windows PowerShell remoting.

Some details and information about how AutomatedLab uses Windows PowerShell remoting are explained in [AutomatedLab Tutorial Part 1: Introduction to AutomatedLab](http://blogs.technet.com/b/fieldcoding/archive/2014/07/11/automatedlab-introduction-part-1.aspx).

***Note:*** *There is an excellent tutorial that covers most aspects of Windows PowerShell remoting. Unfortunately, the series was written for Windows PowerShell 2.0, so features in Windows PowerShell 4.0 and Windows PowerShell 3.0 are missing. For more information, see* [*PowerShell 2.0 remoting guide*](http://www.ravichaganti.com/blog/powershell-2-0-remoting-guide-part-1-the-basics/)*.*

## Invoke-Command

### Introduction

One of the first cmdlets you learn about when using Windows PowerShell remoting is Invoke-Command. This cmdlet takes a script block or a path to a script file and the computer name(s) on which to run the task. Another option is providing an existing session instead of the computer name.

***Note:*** *To connect to a machine that is not part of the workstation’s domain, you need to specify credentials and put the machine into the Trusted Hosts list. (This is explained in Part 1 of this series and in the PowerShell 2.0 remoting guide).*

For example, if you want to retrieve the current time from the domain controller (S1DC1), the command would look like this:

$cred = Get-Credential test1\administrator #the password is "Password1"

$command = { Get-Date }

Invoke-Command -ComputerName S1DC1 -Credential $cred -ScriptBlock $command

If you want to hard code the credential inside the script (not a good idea for production environments, but fine when working in a lab), it gets a bit more complex:

$username = 'test1\Administrator'

$password = 'Password1' | ConvertTo-SecureString -AsPlainText -Force

$cred = New-Object pscredential($username, $password)

$command = { Get-Date }

Invoke-Command -ComputerName S1DC1 -Credential $cred -ScriptBlock $command

By providing the computer name, Windows PowerShell creates a new PSSession to the remote computer (and destroys it when done) each time you use **Invoke-Command**. PSSessions can be created manually and can be reused. This is efficient if you want to execute many scripts or script blocks.

### Persistent Sessions

The cmdlet to create a persistent session is **New-PSSession**. You need to provide the computer name and the credentials, for example:

$username = 'test1\Administrator'

$password = 'Password1' | ConvertTo-SecureString -AsPlainText -Force

$cred = New-Object pscredential($username, $password)

$session = New-PSSession -ComputerName S1DC1 -Credential $cred

$command = { Get-Date }

Invoke-Command -Session $session -ScriptBlock $command

After the job is done, you can remove the PSSession manually by calling **Remove-PSSession**:

$session | Remove-PSSession

AutomatedLab makes this easier by providing the cmdlet **Invoke-LabCommand**.

## Invoke-LabCommand

### Introduction to the cmdlet

**Invoke-LabCommand** cares about credentials. This might not be that interesting in a small lab, but if there are multiple forests, domains, or workgroups, it can make things much easier.

You need to import the existing lab first. A validation is not required because the lab already exists.

Import-Lab D:\FirstLab\Lab.xml -NoValidation

Then you can use **Invoke-LabCommand**:

$command = { Get-Date }

Invoke-LabCommand -ComputerName S1DC1 -ScriptBlock $command -PassThru

The **PassThru** switch is required to get back the result. When doing lab installations with AutomatedLab, the result of installations such as SQL Server or Exchange Server would be too large to handle. The result is always stored in a variable and **PassThru** returns the variable content.

VERBOSE: The Output of the task on machine 'S1DC1' will be available in the variable 'fc1725b7-0ff0-47ea-bdc1-1833103a679c'

If you want to get the current time of all lab machines it just a one-liner:

Invoke-LabCommand -ComputerName (Get-LabMachine -All) -ScriptBlock { Get-Date } -PassThru

The same pattern works for scripts—simply use the **FilePath** parameter instead of **ScriptBlock**:

$data = Invoke-LabCommand -ComputerName (Get-LabMachine -All) -FilePath D:\Get-DiagnosticData.ps1 -PassThru

This could be quite a long-running operation. That’s why the **Invoke-LabCommand** cmdlet provides an **AsJob** switch (as does **Invoke-Command**). When **AsJob** is used, the actual data gathered by the script is not returned, but instead, job objects are returned. The cmdlet creates one background job per machine. Everything works in parallel and the runtime is dramatically reduced.

We recommend that you store the jobs in a variable to have a handle on them. The jobs can be piped to **Wait-Job** and then further to **Receive-Job**. In this way, Windows PowerShell waits until all jobs are finished and then retrieves the date from the job objects.

$jobs = Invoke-LabCommand -ComputerName (Get-LabMachine -All) -FilePath D:\Get-DiagnosticData.ps1 -PassThru -AsJob

$data = $jobs | Wait-Job | Receive-Job

***Note:*** *The* ***Get-LabMachine*** *cmdlet has various parameter patterns. It can get machines by name or by role(s), or it can return all lab machines.*

### Persistent Sessions

**Invoke-LabCommand** uses persistent sessions even when using the **ComputerName** parameter. It does not remove the session when the command is complete, but it reuses the session if called at a later time. **Invoke-LabCommand** reuses sessions because AutomatedLab internally tracks all opened sessions. This can be observed when looking at the VERBOSE output, which can be disabled if it is too noisy*:*

$Global:VerbosePreference = 'SilentlyContinue'

This part of the VERBOSE log shows that the internal worker cmdlet, **New-LWPSSession**, looks for existing sessions. In the following example, it found four sessions that are open to the same machine. Three of them are removed and the remaining one will be reused.

VERBOSE: Starting Installation Activity '<unnamed>'

VERBOSE: Credentials prepared for user 'test1.net\administrator'

VERBOSE: Creating session to computer 'S1DC1'

VERBOSE: New-LWPSSession Entering... (ComputerName=S1DC1,Credential=UserName: test1.net\administrator / Password: Password1)

VERBOSE: Found orphaned sessions. Removing 3 sessions: AL\_96f0c611-d31a-4e82-a021-5a09fe8e8318, AL\_3ec60a75-6a4a-4c9f-a34b-9d8caec09d51, AL\_b24ea3d5-c264-4455-b420-cda37fa0a5f4

VERBOSE: Session AL\_63ef90c2-dadc-4785-9c89-0af7e6f10ca7 is available and will be reused

**Invoke-LabCommand** is the ideal tool for making mass changes in your lab because:

* You do not need to care about credentials (if the target machine is part of the lab configuration).
* It reuses sessions for better performance

### Double Hop Authentication

The default authentication protocol in a domain environment is Kerberos. Kerberos protocol does not allow a double-hop authentication, so you cannot connect from a remote machine to another machine. This is sometimes absolutely necessary.

A remote server does not get the user’s password or ticket-granting ticket (TGT, which can be compared to your passport). Instead, the server that a user connects to receives a Kerberos service ticket with the user’s security identifier (SID) and the SIDs for all groups the user is a member of. The server does not have any information to perform authentication on the user’s behalf because the credentials are not stored on the remote server.

In some situations, when a second (or double-hop authentication) is required, Windows PowerShell offers the CredSSP authentication provider. By using CredSSP, Windows PowerShell forwards the user name and password to the remote computer.

This authentication provider is disabled by default on both the server and the client, but it is enabled on all machines that are installed with AutomatedLab. To make use of it, specify the **UseCredSSP** switch that **Invoke-LabCommand** provides.

The following command results in an error message because requesting data from S1DC1 over S1Sql1 is a double-hop authentication:

Invoke-LabCommand -ComputerName S1Sql1 -ScriptBlock { dir \\S1DC1\c$ } -PassThru

To make this possible, use the **UseCredSSP** switch.

Invoke-LabCommand -ComputerName S1Sql1 -ScriptBlock { dir \\S1DC1\c$ } -PassThru -UseCredSsp

By default, Windows PowerShell can only forward "fresh" credentials. This requires using **Get-Credential** or manually creating a new **PSCredential** object. AutomatedLab also does this in background, so you do not need to provide any credentials.

## Software Installation

A nasty task (especially in larger labs) is software installation, making sure that the same package is installed on all machines. AutomatedLab also helps here. It leverages the infrastructure provided by the commands, and makes it easier to install packages on all lab machines with a one-liner.

### Installing single packages

As an example, popular packages to install on all machines are Notepad++ and Wireshark. AutomatedLab needs to know only two things to install the software on all lab machines (or specific ones):

1. Where is the .exe, .msi, or .msu file?
2. What is the command-line switch for a silent installation?

The following command installs Notepad++ on the lab’s SQL Server:

Install-LabSoftwarePackage -ComputerName S1Sql1 -Path E:\LabSources\SoftwarePackages\Notepad++.exe -CommandLine /S

The file is copied to the machine (also using Windows PowerShell remoting by the PSFileTransfer module). The installation is started as a background job, meaning that you do not have to wait until the installation is finished.

In the following command, Wireshark should be installed on all lab machines and Windows PowerShell should wait until all jobs are finished:

Install-LabSoftwarePackage -ComputerName (Get-LabMachine -All) -Path E:\LabSources\SoftwarePackages\WireShark.exe -CommandLine /S -PassThru | Wait-Job

### Mass Software Installations

If many packages should be installed at the same time, the packages can be defined first and then passed to the **Install-LabSoftwarePackages** cmdlet. **Get-LabSoftwarePackage** lets you define packages and add them to an array.

The following example installs the classic shell, Wireshark, and Notepad++ on all lab machines:

$labSources = 'E:\LabSources'

$packs = @()

$packs += Get-LabSoftwarePackage -Path $labSources\SoftwarePackages\ClassicShell.exe -CommandLine '/quiet ADDLOCAL=ClassicStartMenu'

$packs += Get-LabSoftwarePackage -Path $labSources\SoftwarePackages\Notepad++.exe -CommandLine /S

$packs += Get-LabSoftwarePackage -Path $labSources\SoftwarePackages\winrar.exe -CommandLine /S

Install-LabSoftwarePackages -Machine (Get-LabMachine -All) -SoftwarePackage $packs -PassThru | Wait-Job

So you can see that managing software in your lab can be a trivial and comfortable task if you use AutomatedLab as your lab framework.

## What’s next?

The next article will be a short one about PostInstallationActivities. This is about the same as running scripts or script blocks with *Invoke-LabCommand* but provides a way to set dependencies on ISO images or folders.