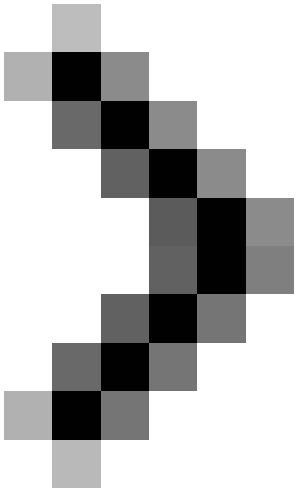
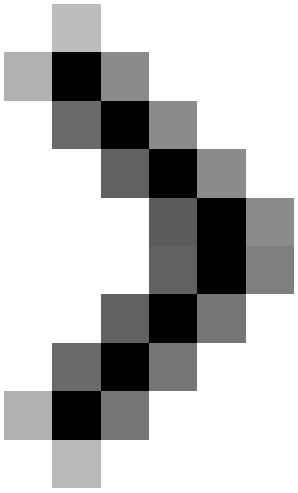
Test SQL Server with Windows PowerShell - Part 3

In Part 3, I will show you how to find some hardware and operating system information from the host machine.



Test SQL Server with Windows PowerShell - Part 1



Test SQL Server with Windows PowerShell - Part 2
Muthusamy

Network Administration - *Part 1 of this series showed you the first check on SQL Server — how to ping a host. In Part 2, I will show you how to check all Windows services related to SQL Server. In Part 3, I will show you how to find some hardware and operating system information from the host machine.*

Step 1

Type or Copy and paste the code below into the C: CheckSQLServerCheckhardware.ps1 file.

```
#Function ?? ki?m tra thông tin c?ng t? m?t máy ph?c v?
Function checkhardware ([string] $ Hostname)
{
$ computer = get-wmiobject -class win32_computersystem -computername
$ hostname -errorvariable errorvar
$ errorvar.size
if (-not $ errorvar)
{
$ message = "Host =" + $ Hostname
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "Description =" + $ computer.Description
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "NumberOfLogicalProcessors =" +
$ computer.NumberOfLogicalProcessors
```

```
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "NumberOfProcessors =" + $ computer.NumberOfProcessors
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "TotalPhysicalMemory =" + $ computer.TotalPhysicalMemory
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "Model =" + $ computer.Model
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "Manufacturer =" + $ computer.Manufacturer
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "PartOfDomain =" + $ computer.PartOfDomain
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "CurrentTimeZone =" + $ computer.CurrentTimeZone
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "DaylightInEffect =" + $ computer.DaylightInEffect
write-host $ message -background "GREEN" -foreground "BLACk"
}
}
```

Step 2

Type or Copy and paste the code below into file C: CheckSQLServerCheckOS.ps1.

```
#Function ?? ki?m tra các thông tin thông tin v? máy host
CheckOS function ([string] $ Hostname)
{
$ os = get-wmiobject -class win32_operatingsystem -computername $ hostname -errorvariable errorvar
if (-not $ errorvar)
{
$ message = "OSArchitecture =" + $ os.OSArchitecture
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "OSLanguage =" + $ os.OSLanguage
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "OSProductSuite =" + $ os.OSProductSuite
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "OSType =" + $ os.OSType
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "BuildNumber =" + $ os.BuildNumber
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "BuildType =" + $ os.BuildType
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "Version =" + $ os.Version
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "WindowsDirectory =" + $ os.WindowsDirectory
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "PlusVersionNumber =" + $ os.PlusVersionNumber
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "FreePhysicalMemory =" + $ os.FreePhysicalMemory
write-host $ message -background "GREEN" -foreground "BLACk"
```

```
$ message = "FreeSpaceInPagingFiles =" + $ os.FreeSpaceInPagingFiles
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "FreeVirtualMemory =" + $ os.FreeVirtualMemory
write-host $ message -background "GREEN" -foreground "BLACk"
$ message = "PAEEnabled =" + $ os.PAEEnabled
write-host $ message -background "GREEN" -foreground "BLACk"
}
}
```

Step 3

Append to file C: CheckSQLServerCheckSQL_Lib.ps1 code below.

```
. ./checkhardware.ps1
. ./checkOS.ps1
```

Now the file C: CheckSQLServerCheckSQL_Lib.ps1 will have pinghost, checkservices, checkhardware and checkOS as shown below.

```
#Source all the relate to functions CheckSQL
. ./PingHost.ps1
. ./checkservices.ps1
. ./checkhardware.ps1
. ./checkOS.ps1
```

Note: This CheckSQL_Lib.ps1 file will be updated from new scripts such as checkhardware.ps1 and checkOS.ps1

Step 4

Append to the file C: CheckSQLServerCheckSQLServer.ps1 code below.

```
checkhardware $ Hostname
checkOS $ Hostname
```

Now file C: CheckSQLServerCheckSQLServer.ps1 will have both checkhardware and checkOS as below, We have added a write-host statement to show the whole process.

```
#Objective: To check various status of SQL Server
#Host, instances and databases.
#Author: MAK
#Date Written: June 5, 2008
param (
[string] $ Hostname
)
. ./CheckSQL_Lib.ps1
Write-host "Checking SQL Server ."
Write-host "...."
"Write-host"
```

Write-host "Arguments accepted: $ Hostname"
write-host "...."
Write-host "Pinging the host machine"
write-host "...."
PingHost $ Hostname
Write-host "Check windows services on the host related to SQL Server"
write-host "..... ... "
checkservices $ Hostname
Write-host "Checking hardware Information ."
Write-host "...."
checkhardware $ Hostname
Write-host "Checking OS Information ."
Write-host "...."
checkOS $ Hostname

Note: This CheckSQLServer.ps1 file will be updated with new conditions and parameters in later sections of the series.

The source will load the functions listed in the script file and make it available during the entire PowerShell session. In this case, we source a scenario, which will be sourced from many other scenarios.

Step 5

Now let's execute the script, CheckSQLServer.ps1 by passing 'Powerpc' host as an argument as shown below.

./CheckSQLServer.ps1 PowerServer2

Then the result will be as follows (refer to Figure 1.0)

Checking SQL Server .
....
Arguments accepted: PowerServer2
....
Pinging the host machine
....
PowerServer2 is REACHABLE
Check windows services on the host related to SQL Server
....
Host = PowerServer2 MSSQLSERVER Running OK True .Administrator
Host = PowerServer2 MSSQLServerADHelper100 Stopped OK False NT AUTHORITYNETWORK SERVICE
Host = PowerServer2 MSSQLServerOLAPService Stopped OK False .Administrator
Host = PowerServer2 SQLBrowser Stopped OK False NT AUTHORITYLOCAL SERVICE
Host = PowerServer2 SQLSERVERAGENT Stopped OK False .Administrator
Host = PowerServer2 SQLWriter Stopped OK False LocalSystem
Checking hardware Information .
....
Host = PowerServer2
Description = AT / AT COMPATIBLE
NumberOfLogicalProcessors = 2

NumberOfProcessors = 1
TotalPhysicalMemory = 2145738752
Model = OptiPlex GX270
Manufacturer = Dell Computer Corporation
PartOfDomain = True
CurrentTimeZone = -240
DaylightInEffect = True
Checking OS Information .
....
OSArchitecture = 32-bit
OSLanguage = 1033
OSProductSuite = 274
OSType = 18
BuildNumber = 6001
BuildType = Multiprocessor Free
Version = 6.0.6001
WindowsDirectory = C: Windows
PlusVersionNumber =
FreePhysicalMemory = 1511144
FreeSpaceInPagingFiles = 2402648
FreeVirtualMemory = 3966452
PAEEnabled = False

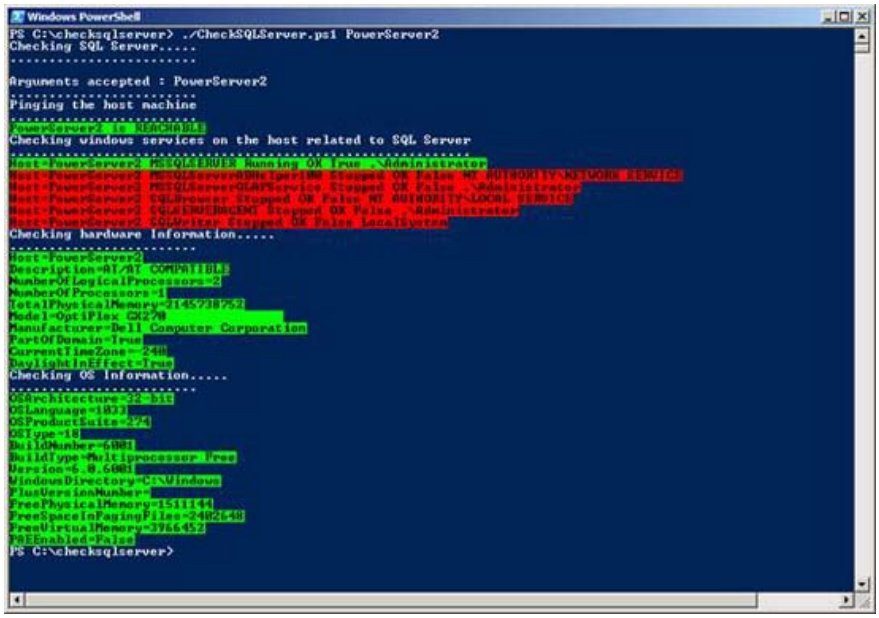


Figure 1.0

From the results, you can see hardware and operating system information.

Step 6

Please execute the script on the machine that does not exist as shown below.

./CheckSQLServer.ps1 TestMachine

Then the results will be as follows (see Figure 1.1)

Result

Checking SQL Server .
....
Arguments accepted: TestMachine
....
Pinging the host machine
....
TestMachine is NOT reachable
Check windows services on the host related to SQL Server
....
Get-WmiObject: The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)
At C: checksqlservercheckservices.ps1: 5 char: 24
+ $ Services = get-wmiobject -class win32_service -computername $ hostname | where {$ _. name -like ' SQL *'} | select-obj*
ect Name, state, status, Started, Startname, Description
Host = TestMachine
Checking hardware Information .
....
Get-WmiObject: The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)
At C: checksqlservercheckhardware.ps1: 5 char: 24
+ $ computer = get-wmiobject -class win32_computersystem -computername $ hostname -errorvariable errorvar
Checking OS Information .
....
Get-WmiObject: The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)
At C: checksqlservercheckOS.ps1: 5 char: 18
+ $ os = get-wmiobject -class win32_operatingsystem -computername $ hostname -errorvariable errorvar

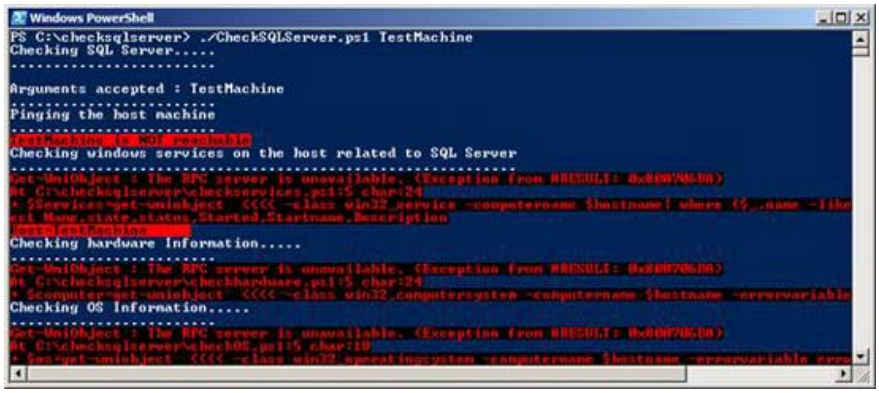


Figure 1.1

Step 7

We do not have to continue with checkservices, checkhardware and checkos if ping fails. So update pinghost.ps1 as shown below.

```
Function Pinghost ([string] $ Hostname)
{
$ status = get-wmiobject win32_pingstatus -Filter "Address = '$ Hostname'" | Select-Object statuscode
if ($ status.statuscode -eq 0)
{
write-host $ Hostname is REACHABLE -background "GREEN" -foreground "BLACk"
}
else
{
$ global: errorvar = "host not reachable"
write-host $ Hostname is NOT reachable -background "RED" -foreground "BLACk"
}
}
```

Update the checksqlserver.ps1 as shown below.

```
#Objective: To check various status of SQL Server
#Host, instances and databases.
#Author: MAK
#Date Written: June 5, 2008
param (
[string] $ Hostname
)
$ global: errorvar = 0
. ./CheckSQL_Lib.ps1
Write-host "Checking SQL Server ."
Write-host "...."
"Write-host"
Write-host "Arguments accepted: $ Hostname"
write-host "...."
Write-host "Pinging the host machine"
write-host "...."
pinghost $ Hostname
if ($ global: errorvar -ne "host not reachable")
{
Write-host "Check windows services on the host related to SQL Server"
write-host "..... ... "
checkservices $ Hostname
Write-host "Checking hardware Information ."
Write-host "...."
checkhardware $ Hostname
Write-host "Checking OS Information ."
Write-host "...."
checkOS $ Hostname
}
```

Now execute the script by passing the name 'testmachine', the name does not really exist as an argument.

./CheckSQLServer.ps1 TestMachine

Result

Checking SQL Server .
....
Arguments accepted: TestMachine
....
Pinging the host machine
....
TestMachine is NOT reachable

Note that you can download the latest code in the third section here.

Conclude

This is the third part of this series. In this third part, I have shown you how to access hardware and operating system information using Windows PowerShell and WMI-Object. In the next section, we will add some other checks and introduce how to capture some hardware and operating system information.