#-------------------------------------------------------------------------------------------------#

# Mangament using PowerShell only #

#-------------------------------------------------------------------------------------------------#

#-----------------------------------------------------------------------------------------------#

# Getting notified of issues

#-----------------------------------------------------------------------------------------------#

function Send-EasyMail ([string]$subject, [string]$body)

{

# Setup some basic info

$From = "arcanecode@gmail.com"

$To = "rcain@pragmaticworks.com"

$SMTPServer = "smtp.gmail.com"

# Create the e-mail object

$SMTPClient = New-Object Net.Mail.SmtpClient($SmtpServer, 587)

# Enable SSL Protocol (Secure Socket Layers) so our e-mail will be sent securely

$SMTPClient.EnableSsl = $true

# Create a credential objec we'll use to authenticate ourselves to the SMTP server

$SMTPClient.Credentials = New-Object System.Net.NetworkCredential("arcanecode", "passwordgoeshere");

# Finally, send the mail

$SMTPClient.Send($From, $To, $Subject, $Body)

}

# Test the function

Send-EasyMail -subject "Test Subject" -body "Testing some body text."

##

#-----------------------------------------------------------------------------------------------#

# Get service status

#-----------------------------------------------------------------------------------------------#

$machines = "PRAGMATICWORKS" # fake an array of machines to process

$serviceStatus = @{} # Initialize or reset our versions hash table

foreach ($machine in $machines)

{

Get-Service -name \*sql\* -ComputerName $machine -ErrorAction SilentlyContinue |

Sort-Object -Property DisplayName |

foreach{

$k = $machine + " - " + $\_.DisplayName # Key

$v = $\_.Status # Value

$serviceStatus[$k] = $v

Get-Service $\_.Name |

Select-Object -ExpandProperty ServicesDependedOn |

foreach{

$kd = $k + " has a dependency on " + $\_.DisplayName

$s = $\_.Status

$serviceStatus[$kd] = $s

} # inner get-service

} # outer get-service

}

# See all the results

$serviceStatus | Format-Table -AutoSize

# Sort by the running status of each service

$serviceStatus.GetEnumerator() |

Sort-Object Value |

Format-Table -AutoSize

# Only show stopped services

$serviceStatus.GetEnumerator() |

Where-Object{$\_.Value -eq "Stopped"} |

Format-Table -AutoSize

# Only show stopped services we care about

$serviceStatus.GetEnumerator() |

Where-Object{$\_.Value -eq "Stopped" `

-and $\_.Key -notlike '\*SQL Server Agent\*'} |

Format-Table -AutoSize

# Notify the DBA of any issues

# Note, anytime you want to get the output into a string you will

# have to use Out-String on the end, otherwise all you get are

# class names.

$body = $serviceStatus.GetEnumerator() |

Where-Object{$\_.Value -eq "Stopped" `

-and $\_.Key -notlike '\*SQL Server Agent\*'} |

Format-Table -AutoSize | Out-String

Send-EasyMail -subject "Stopped services" -body $body

##

#-----------------------------------------------------------------------------------------------#

# Counters

#-----------------------------------------------------------------------------------------------#

# Get a list of all counters

# Use -ComputerName to use with a specific computer

# Note use of single quotes so the $ in our instance name won't try to translate to a variable

Get-Counter -ComputerName $env:COMPUTERNAME -ListSet 'MSSQL$SQL2012\*' |

ForEach-Object {$\_.CounterSetName, $\_.Paths} |

Format-Table -AutoSize

# Counters in the buffer manager

# Omitting the computer assumes local computer

Get-Counter -ListSet 'MSSQL$SQL2012:Buffer Manager' |

ForEach-Object {$\_.CounterSetName, $\_.Paths} |

Format-Table -AutoSize

# Note to see a list of counters, bring up perfmon, then Add Counters.

$counterList = @(

'\MSSQL$SQL2012:Buffer Manager\Buffer cache hit ratio',

'\MSSQL$SQL2012:Buffer Manager\Page reads/sec',

'\MSSQL$SQL2012:Buffer Manager\Page writes/sec'

)

$counterResult = Get-Counter -SampleInterval 5 -MaxSamples 3 -Counter $counterList

foreach($counter in $counterResult)

{

$counterDataTable += $counter.CounterSamples

}

$counterDataTable | Format-Table -AutoSize -Wrap

##

#-----------------------------------------------------------------------------------------------#

# Use WMI to check disk space

#-----------------------------------------------------------------------------------------------#

$unit = "GB" # Valid values are: KB MB GB TB PB

$measure = "1$unit"

$wmiQuery = @"

SELECT SystemName, Name, DriveType, FileSystem, FreeSpace, Capacity, Label

FROM Win32\_Volume

"@

Get-WmiObject -ComputerName "PragmaticWorks" -Query $wmiQuery

Clear-Host

# Get the output and format it nicely

Get-WmiObject -ComputerName "PragmaticWorks" -Query $wmiQuery |

Select-Object SystemName, Name, Label, DriveType, FileSystem ,

@{Label="SizeIn$unit";Expression={"{0:n2}" -f ($\_.Capacity/$measure)}} ,

@{Label="FreeIn$unit";Expression={"{0:n2}" -f ($\_.freespace/$measure)}} ,

@{Label="PercentFree";Expression={"{0:n2}" -f (($\_.freespace/$\_.Capacity)\*100)}} |

Where-Object {$\_.Name -NotLike '\\?\\*'} |

Sort-Object Name |

Format-Table -AutoSize -Property SystemName, Name, Label, DriveType, FileSystem,

@{Label="Size In $unit";Align="Right";Exp={($\_."SizeIn$unit")}} ,

@{Label="Free In $unit";Align="Right";Exp={($\_."FreeIn$unit")}} ,

@{Label="Percent Free";Align="Right";Exp={($\_.PercentFree)}}

##

#### DON'T SHOW IN VIDS NOT WORKING QUITE RIGHT

<#

$drives = Get-WmiObject -ComputerName "PragmaticWorks" -Query $wmiQuery |

Select-Object SystemName, Name, Label, DriveType, FileSystem ,

@{Label="SizeIn$unit";Expression={"{0:n2}" -f ($\_.Capacity/$measure)}} ,

@{Label="FreeIn$unit";Expression={"{0:n2}" -f ($\_.freespace/$measure)}} ,

@{Label="PercentFree";Expression={"{0:n2}" -f (($\_.freespace/$\_.Capacity)\*100)}} |

Where-Object {$\_.Name -NotLike '\\?\\*'} |

Sort-Object Name

$fg = "White"

$bg = "Black"

foreach($drive in $drives)

{

$row = "{0,15}" -f $drive.SystemName

$row += " {0,5}" -f $drive.Name

$row += " {0,10}" -f $drive.Label

$row += " {0,-2:n0}" -f $drive.DriveType

$row += " {0,10}" -f $drive.FileSystem

$row += " {0,-10:n0}" -f $drive."SizeIn$unit"

$row += " {0,-10:n0}" -f $drive."FreeIn$unit"

$row += " {0,-10:n0}" -f $drive.PercentFree

if(($drive.PercentFree) -lt 15)

{

Write-Host $row -BackgroundColor Black -ForegroundColor Red

}

else

{

Write-Host $row -BackgroundColor Black -ForegroundColor White

}

}

#>

##

#-----------------------------------------------------------------------------------------------#

# Event Logs

#-----------------------------------------------------------------------------------------------#

# Basic Event Logs

Get-EventLog -List

# On vista and later can use WinEvent to get more detail log info

Get-WinEvent -ListLog \* | Format-Table -Autosize

# Get-EventLog gives a bit more info

Get-EventLog System -Newest 20 | Format-List

Get-EventLog System -Newest 20 |

Format-Table -Autosize EntryType, Index, Message, TimeGenerated

# Get all of the errors for the last 24 hours

Get-EventLog System |

Where-Object {$\_.EntryType -eq "Error" `

-and $\_.TimeGenerated -ge ((Get-Date).AddHours(-24))} |

Format-List

# Most SQL Server events go into the application log

Get-EventLog Application -Newest 20 | Format-List

# We can narrow down the list by filtering on the source

Get-EventLog Application |

Where-Object {$\_.Source -like '\*sql\*' `

-and $\_.EntryType -eq "Error" `

-and $\_.TimeGenerated -ge ((Get-Date).AddHours(-96)) `

} |

Format-List

# We can further narrow to a specific instance

# (note having to use single quotes for source since instance has a $ in it)

Get-EventLog Application |

Where-Object {$\_.Source -eq 'MSSQL$SQL2012' `

-and $\_.EntryType -eq "Error" `

-and $\_.TimeGenerated -ge ((Get-Date).AddDays(-15)) `

} |

Format-List

# Discovering how often an error occurs

# (Note the TimeGenerated is expanded to the last month.

# A bigger window will give you a higher level view of errors)

Get-EventLog Application |

Where-Object {$\_.Source -eq 'MSSQL$SQL2012' `

-and $\_.EntryType -eq "Error" `

-and $\_.TimeGenerated -ge ((Get-Date).AddMonths(-1)) `

} |

Group-Object Message |

Sort-Object -Desc Count |

Format-Table -Autosize Count, Name

# You can even add your own messages to the event log

# Important! Script must be run in Admin mode to enable writing to event log

# Before we can write, we must register our "Source" with the log.

New-EventLog -LogName Application -Source MyCoolPowerShellScript

# Now that it's registered, we can write the message

Write-EventLog -LogName Application `

-Source MyCoolPowerShellScript `

-EventId 0001 `

-Message "I have something to say" `

-EntryType Information

# Note there's nothing returned. This is good, as most of the time

# if you are writing to the event log it's doing unattended execution

# Valid values for -EntryType are:

# Error, Warning, Information, SuccessAudit, FailureAudit

# Add an Error

Write-EventLog -LogName Application `

-Source MyCoolPowerShellScript `

-EventId 0002 `

-Message "You're doing it wrong" `

-EntryType Error

# Add a warning

Write-EventLog -LogName Application `

-Source MyCoolPowerShellScript `

-EventId 0003 `

-Message "I'm The Doctor. Basically, run." `

-EntryType Warning

Get-EventLog Application |

Where-Object {$\_.Source -eq 'MyCoolPowerShellScript'} |

Sort-Object TimeGenerated, Index |

Format-List

Remove-EventLog -Source MyCoolPowerShellScript