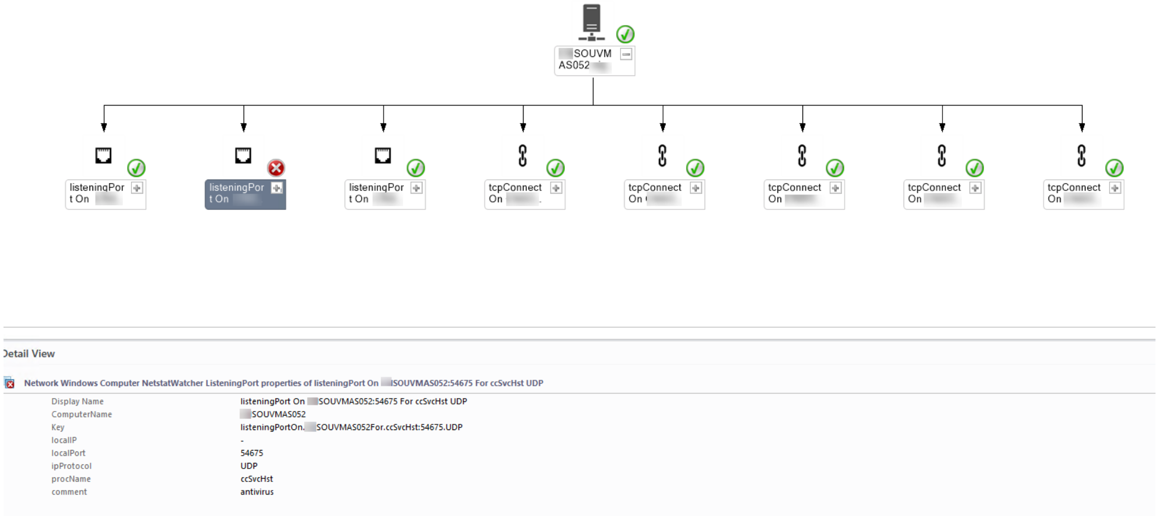
Monitor any network-connection or listening port with SCOM

SCOM offers ‘wizards’ to monitor Windows Services and processes. Most of the cases this is enough to ensure that the application works.
Sometimes however multiple connections are handled by a service or a process.

Ready to use ‘port monitors’ help to identify if a port on a target machine responds.
There are cases however where connection attempts can irritate or even crash an application.

In hybrid scenarios part of the application runs in the cloud. – A windows machine might act as gateway for example. It would be good to know if the connection to endpoint in the cloud is still active.

A custom management pack which uses **netstat** and **powershell** can help.



Detail View

Network Windows Computer NetstatWatcher ListeningPort properties of listeningPort On [redacted]SOUVMAS052:54675 For ccSvcHst UDP

```
Display Name      listeningPort On [redacted]SOUVMAS052:54675 For ccSvcHst UDP
ComputerName      [redacted]SOUVMAS052
Key               listeningPortOn.[redacted]SOUVMAS052For.ccSvcHst:54675.UDP
localIP           -
localPort         54675
ipProtocol        UDP
procName          ccSvcHst
comment           antivirus
```

Diagram view showing monitored listening ports and tcp connections

Following lines explain the briefly the components of the management pack and the logic behind it. – To ensure the code also runs on Windows Server 2008 R2 it’s compatible to PowerShell version 2.

Change History

Date	Build No.	Changes
2017-12-29	1.0.0.183	Initial Upload to GitHub
2018-01-16	1.0.0.304	Updated monitor for tcpConnections to support results based on consequitve samples Changed monitor for listeningPorts to 2 state

Connections or ports that need to be monitored need to be known by SCOM first. The technical

Defining the requirement

Connections or ports that need to be monitored need to be known by SCOM first. The technical term is called ‘discovery’.
To be monitored connections need to be stored in file named ‘monitoredTcpConnects.csv’. The header row must keep and the ‘comment‘ is optional. Either specify remote (host) Name or the remote IP address. E.g.

```
remoteIP,remoteName,remotePort,procName,comment
10.1.11.83,,80,CcmExec,sccm
,linvmas146,5723,HealthService
194.69.46.72,,40936,powershell
```

To be monitored ports need to be stored in file named ‘monitoredListeningPorts.csv’. The header row must keep and the ‘comment‘ is optional. E.g.

```
ipProtocol,localIP,localPort,procName,comment
udp,127.0.0.1,49740,dfsrs
udp,,161,snmp
tcp,,10115,endpoint,perfdata
```

Preparing raw data

Running ‘netstat -ano’ lists all established connections and listening ports including the process identification number (PID) which is using it.

```
C:\>netstat -ano

Active Connections

Proto  Local Address          Foreign Address        State           PID
TCP    0.0.0.0:81             0.0.0.0:0              LISTENING
TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       948
TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
TCP    172.19.18.225:22326    172.19.14.30:8080      ESTABLISHED     7152
TCP    172.19.18.225:22494    172.19.10.55:80        ESTABLISHED     6388
TCP    172.19.18.225:23063    172.19.10.36:17295     ESTABLISHED     2756
UDP    192.168.96.1:5353      *:*                                    1948
UDP    192.168.116.1:137      *:*                                    4
UDP    192.168.116.1:1900     *:*                                    9844
```

The function below runs ‘netstat’, stores the result in a file and converts it then into a list of objects for further processing. A parameter decides whether the list shall contain ‘listening ports objects’ or ‘established connection objects’.

```
#retrieving computer anme and ip addresses for later use
$localComputerName    = $env:COMPUTERNAME
$localIPAddresses     = ([System.Net.Dns]::GetHostAddresses($localComputerName)) | Where-Object {
$_.AddressFamily -eq 'interNetwork' } | `
                                                                                   Select-
Object -ExpandProperty IPAddressToString

Function Format-NetstatData {

       param(
              [Parameter(Mandatory=$true)][object]$netstatInPut,
              [Parameter(Mandatory=$true)][string]$qryType,
              [Parameter(Mandatory=$true)][ref]$nestatIPData
       )

       #retrieving all processes to map PID in netstat to executable name
       $allProcesses     = Get-Process | Select-Object -Property Name, id

       $netStatConnects = New-Object -TypeName System.Collections.Generic.List[object]
       $netStatArr      =  $netstatInPut -split "`r`n"

       $netStatArr | ForEach-Object {

              $netStatItm = $_

              if ($netStatItm -match "\d") {

                     #split the line by using 'more than 2 white spaces' as delimitation
                     $netStatItmParts = [Regex]::Split($netStatItm,"\s{2,}")


                     if ($qryType -eq 'tcpConnection') {

                            $proto           = $netStatItmParts[1]

                            $localIP         = ($netStatItmParts[2] -split ':')[0]
```

```
                                $localPort       = ($netStatItmParts[2] -split ':')[1]

                                $remoteIP        = ($netStatItmParts[3] -split ':')[0]
                                $remotePort      = ($netStatItmParts[3] -split ':')[1]
                                $connectState    = $netStatItmParts[4]
                                $procId          = $netStatItmParts[5]

                                $procInfo        = $allProcesses | Where-Object { $_.id -eq
$procId }
                                $procName        = $procInfo.Name

                                if ($localIPAddresses -contains $localIP) {
                                        $localName = $localComputerName
                                }

                                #filtering records to only contain connections to remote systems
                                if (($localIp -match $regIpPat -and $remoteIp -match $regIpPat) -
and ($remoteIP -notmatch '0.0.0.0|127.0.0.1') ) {
                                        $myNetHsh = @{'proto' = $proto}
                                        $myNetHsh.Add('localIP', $localIP)
                                        $myNetHsh.Add('localName', $localName)

                                        $myNetHsh.Add('remoteIP', $remoteIP)
                                        $myNetHsh.Add('remotePort', $remotePort)
                                        $myNetHsh.Add('connectState', $connectState)
                                        $myNetHsh.Add('procId', $procId)
                                        $myNetHsh.Add('procName', $procName)


                                        $myNetObj = New-Object -TypeName PSObject -Property
$myNetHsh

                                        $null     = $netStatConnects.Add($myNetObj)
                                }

                        } else {

                                $proto               = $netStatItmParts[1]

                                if ($proto -ieq 'TCP') {
                                        $localIP         = ($netStatItmParts[2] -split ':')[0]
                                        $localPort       = ($netStatItmParts[2] -split ':')[1]

                                        $remoteIP        = ($netStatItmParts[3] -split ':')[0]
                                        $remotePort      = ($netStatItmParts[3] -split ':')[1]
                                        $connectState    = $netStatItmParts[4]
                                        $procId          = $netStatItmParts[5]

                                } else {
                                        $localIP         = ($netStatItmParts[2] -split ':')[0]
                                        $localPort       = ($netStatItmParts[2] -split ':')[1]

                                        $remoteIP        = ($netStatItmParts[3] -split ':')[0]
                                        $remotePort      = ($netStatItmParts[3] -split ':')[1]
                                        $connectState    = '-'
                                        $procId          = $netStatItmParts[4]

                                }

                                $procInfo = $allProcesses | Where-Object { $_.id -eq $procId }
                                $procName = $procInfo.Name

                                if ($localIPAddresses -contains $localIP) {
                                        $localName = $localComputerName
                                }

                                if (($localIp -match $regIpPat) -and ($remoteIP -match
'\*|0.0.0.0|127.0.0.1') ) {
                                        $myNetHsh = @{'proto' = $proto}
                                        $myNetHsh.Add('localIP', $localIP)
                                        $myNetHsh.Add('localName', $localName)

                                        $myNetHsh.Add('localPort', $localPort)

                                        $myNetHsh.Add('connectState', $connectState)
                                        $myNetHsh.Add('procId', $procId)
                                        $myNetHsh.Add('procName', $procName)


                                        $myNetObj = New-Object -TypeName PSObject -Property
$myNetHsh

                                        $null     = $netStatConnects.Add($myNetObj)
                                }

                        } # END if ($qryType -eq 'tcpConnect')


                } #END if ($netStatItm -match "\d")

        } #END $netStatIpArr | ForEach-Object {}

        If ($netStatConnects.count -gt 0) {
                $rtn = $true
                $nestatIPData.Value = $netStatConnects
        } else {
                $rtn = $false
        }

        $rtn

} #END Funciton Format-NetstatIPData

#running netsat -ano and piping it into a file which then is read. - Tests reveal that it's
quicker than
#directly storing the result in a variable.

Invoke-Expression "C:\Windows\System32\netstat.exe -ano" | Out-File -FilePath $netStatIpFile
$netStatIp = Get-Content -Path $netStatIpFile | Out-String

$netStatIPConnects = New-Object -TypeName System.Collections.Generic.List[object]
Format-NetstatData -netstatInPut $netStatIp -qryType $discoveryItem -nestatIPData
([ref]$netStatIPConnects)
```

Interpreting output and initiate reaction

To check now whether a defined connection is active or a port is listing ‘should and is’ is compared. – As the code for listening ports is very similar, it’s not shown below.

```
if($MonitorItem -eq 'tcpConnection') {

$monitoredTcpConnects = Import-Csv -Path $monitoredTcpConnectsFilePath


                foreach ($tcpConnect in $monitoredTcpConnects) {

                        $remoteIP        = ''
                        $remoteName      = ''
                        $remotePort      = ''
                        $comment         = ''
                        $procName        = ''
                        $connectDetails  = ''
                        $connectionState = ''

                        $remoteIP        = $tcpConnect.remoteIP
                        $remoteName      = $tcpConnect.remoteName
                        $remotePort      = $tcpConnect.remotePort
                        $comment         = $tcpConnect.comment
                        $procName        = $tcpConnect.procName

                        if ($remoteName -and ([String]::IsNullOrEmpty($remoteIP))) {
                                $remoteIP = [system.net.dns]::Resolve($remoteName).AddressList |
Where-Object { $_.AddressFamily -eq 'interNetwork' } | Select-Object -ExpandProperty
IPAddressToString
                        }

                        if ($remotePort -and $remoteIP) {

                                $SampleResults = @()

                                if (-not $WithinSeconds) {
                                        $WithinSeconds = $IntervalSeconds

                                }

                                if ($WithinSeconds -ge $IntervalSeconds) {

                                        $WithinSeconds = $IntervalSeconds

                                }

                                if (-not $MatchCount) {
                                        $MatchCount = 3
                                }

                                if (-not $SampleCount) {
                                        $SampleCount = $MatchCount

                                }

                                if ($MatchCount -gt $SampleCount) {

                                        $MatchCount = $SampleCount

                                }

                                if ($WithinSeconds -gt $TimeoutSeconds) {
```

```
                                         $WithinSeconds = $TimeoutSeconds

                                 }

                                 $timeToWait = $WithinSeconds / $SampleCount
                                 $timeToWait = [Math]::Round($timeToWait)


                                 for ($loopRunner = 1; $loopRunner -le $SampleCount; $loopRunner
++) {

                                         $doForeachFlag = $true

                                         $connectDetails = $netStatIPConnects | Where-Object {
$_.remotePort -eq $remotePort -and $_.remoteIP -eq $remoteIP }


                                         if ([string]::IsNullOrEmpty($connectDetails) -or
[string]::IsNullOrWhiteSpace($connectDetails)) {

                                                  $localIP          = $localIPAddresses

                                                  $displayName      = 'tcpConnect On ' +
$localComputerName + ' To ' + $remoteIP + ':' + $remotePort + ' for ' + $procName
                                                  $Key              =
"tcpConnectOn$($localComputerName)For$($procName)To$($remoteIP):$($remotePort)"

                                                  $testedAt         = "Tested on: $(Get-Date -Format
u) / $(([TimeZoneInfo]::Local).DisplayName)"
                                                  $connectionState = 'No active connection found.'


                                                  $state            = 'Red'
                                                  $localName        = 'NA'
                                                  $localPort        = 'NA'

                                                  $supplement       = "localIP: $($localIP)`t
localPort: $($localPort)`n procName: $($procName)`n ConnecionState: $($connectionState)`n"
                                                  $supplement      += "remoteIP: $($remoteIP)`t
remotePort: $($remotePort)`n"


                                                  $myBagHsh = @{'Key' = $key}
                                                  $myBagHsh.Add('State', $state)
                                                  $myBagHsh.Add('Supplement', $supplement)
                                                  $myBagHsh.Add('TestedAt', $testedAt)
                                                  $myBagObj = New-Object -TypeName PSObject -
Property $myBagHsh

                                                  $SampleResults += $myBagObj

                                                  $doForeachFlag = $false



                                         } #END if ([string]::IsNullOrEmpty($connectDetails) -or
[string]::IsNullOrWhiteSpace($connectDetails))


                                         foreach ($connDetail in $connectDetails) {


                                                  $connectionState = ''
                                                  $supplement       = ''

                                                  $localIP          = $connDetail.localIP
                                                  $localName        = $connDetail.localName


                                                  if ([String]::IsNullOrEmpty($remoteName)) {
                                                          $tmpName = 
[system.net.dns]::Resolve($remoteIP).HostName
                                                          if ($tmpName -ne $remoteIP) {
                                                                  $tmpName    = $tmpName -replace 
$localComputerDomain,''
                                                                  $tmpName    = $tmpName -replace 
'\.',''
                                                                  $remoteName = $tmpName
                                                          } else {
                                                                  $remoteName = 'No reverse record 
in DNS.'
                                                          }
                                                  }

                                                  if ($remoteName -match 
'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}') {
                                                          $tmpName = 
[system.net.dns]::Resolve($remoteName).HostName
                                                          if ($tmpName -ne $remoteIP) {
                                                                  $tmpName    = $tmpName -replace 
$localComputerDomain,''
                                                                  $tmpName    = $tmpName -replace 
'\.',''
                                                                  $remoteName = $tmpName
                                                          } else {
                                                                  $remoteName = 'No reverse record 
in DNS.'
                                                          }
                                                  }

                                                  $displayName     = 'tcpConnect On ' + 
$localComputerName + ' To ' + $remoteIP + ':' + $remotePort + ' for ' + $procName
                                                  $Key             = 
"tcpConnectOn$($localComputerName)For$($procName)To$($remoteIP):$($remotePort)"

                                                  $testedAt        = "Tested on: $(Get-Date -Format 
u) / $(([TimeZoneInfo]::Local).DisplayName)"
                                                  $connectionState = $connDetail.connectState

                                                  $supplement      = "localIP: $($localIP)`t  `n 
procName: $($procName)`t `n ConnecionState: $($connectionState)`n"
                                                  $supplement     += "remoteIP: $($remoteIP)`t 
remotePort: $($remotePort)`n"

                                                  if ($connectionState -eq 'ESTABLISHED') {
                                                          $state       = 'Green'

                                                  } elseif ($connectionState -eq 'TIME_WAIT') {

                                                          $state       = 'Yellow'
                                                          $supplement += 'TIME_WAIT = Local 
endpoint (this computer) has closed the connection.'
                                                  } else {
                                                          $state       = 'Red'
                                                          $supplement += 'CLOSE_WAIT = Remote 
endpoint (this computer) has closed the connection.'
                                                  }



                                                  if ($doForeachFlag) {
                                                          $myBagHsh = @{'Key' = $key}
                                                          $myBagHsh.Add('State', $state)
                                                          $myBagHsh.Add('Supplement', $supplement)
                                                          $myBagHsh.Add('TestedAt', $testedAt)
                                                          $myBagObj = New-Object -TypeName 
PSObject -Property $myBagHsh
                                                          $SampleResults += $myBagObj
```

```
                                                }

                                        } #END foreach ($connDetail in $connectDetails)

                                        Start-Sleep -Seconds $timeToWait

                                } #END for ($loopRunner = 1; $loopRunner -le $SampleCount;
$loopRunner++ )

                                $sampleResultsGreen  = ($SampleResults | Where-Object {$_.State -
eq 'Green'}).count
                                $sampleResultsRed    = ($SampleResults | Where-Object {$_.State -
eq 'Red'}).count
                                $sampleResultsYellow = ($SampleResults | Where-Object {$_.State -
eq 'Yellow'}).count

                                if ($sampleResultsGreen -ge $MatchCount) {
                                        $state = 'Green'
                                } else {
                                        $state = ($SampleResults[($SampleResults.count) -
1]).State
                                }

                                $Key        = ($SampleResults[($SampleResults.count) -1]).Key
                                $supplement = ($SampleResults[($SampleResults.count) -
1]).Supplement
                                $testedAt   = ($SampleResults[($SampleResults.count) -
1]).testedAt

                                $bag = $api.CreatePropertybag()

                                $bag.AddValue("Key",$key)
                                $bag.AddValue("State",$state)
                                $bag.AddValue("Supplement",$supplement)
                                $bag.AddValue("TestedAt",$testedAt)
                                $bag


                        } else {

                                $foo = 'No details this time, not sending to inventory.'

                        } # END if ($connectDetails)

                } #END foreach($tcpConnect in $monitoredTcpConnects)

        } else {

                $api.LogScriptEvent('Monitor NetStatWatcher Three
State.ps1',3002,1,"NetStatWatcherMon MonitorItem $($MonitorItem) - File not found in
$($monitoredTcpConnectsFilePath)")

        }

}
```

Management Pack components

Classes

Everything in SCOM that has a Health State is an object. Instead of checking all Windows computers for the existing of those files and changing their health state (green/yellow/red) directly, a dedicated computer class is defined.

```
<ClassType ID="Network.Windows.Computer.NetstatWatcher.Computer" Accessibility="Public"
Abstract="false" Base="Windows!Microsoft.Windows.ComputerRole" Hosted="true" Singleton="false"
Extension="false">
  <Property ID="FilePath" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="256" MinLength="0" Required="false" Scale="0" />
  <Property ID="NodeName" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="256" MinLength="0" Required="false" Scale="0" />
</ClassType>
```

Also, a class for ‘tcp connections’ and ‘listening ports’ is required:

```
<ClassType ID="Network.Windows.Computer.NetstatWatcher.TcpConnection" Accessibility="Public"
Abstract="false" Base="System!System.LogicalEntity" Hosted="false" Singleton="false"
Extension="false">
    <Property ID="ComputerName" Type="string" AutoIncrement="false" Key="false"
CaseSensitive="false" MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="Key" Type="string" AutoIncrement="false" Key="true" CaseSensitive="false"
MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="localIP" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="localName" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="remoteIP" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="remoteName" Type="string" AutoIncrement="false" Key="false"
CaseSensitive="false" MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="remotePort" Type="string" AutoIncrement="false" Key="false"
CaseSensitive="false" MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="procName" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="512" MinLength="0" Required="false" Scale="0" />
    <Property ID="comment" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="1024" MinLength="0" Required="false" Scale="0" />
</ClassType>

<ClassType ID="Network.Windows.Computer.NetstatWatcher.ListeningPort" Accessibility="Public"
Abstract="false" Base="System!System.LogicalEntity" Hosted="false" Singleton="false"
Extension="false">
    <Property ID="ComputerName" Type="string" AutoIncrement="false" Key="false"
CaseSensitive="false" MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="Key" Type="string" AutoIncrement="false" Key="true" CaseSensitive="false"
MaxLength="256" MinLength="0" Required="false" Scale="0" />
```

```
    <Property ID="localIP" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="localPort" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="ipProtocol" Type="string" AutoIncrement="false" Key="false"
CaseSensitive="false" MaxLength="256" MinLength="0" Required="false" Scale="0" />
    <Property ID="procName" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="512" MinLength="0" Required="false" Scale="0" />
    <Property ID="comment" Type="string" AutoIncrement="false" Key="false" CaseSensitive="false"
MaxLength="1024" MinLength="0" Required="false" Scale="0" />
</ClassType>
```

To create a relation between computer and it’s monitored tcp-connections or listening-ports two additional classes are required:

```
<RelationshipType ID="Network.Windows.Computer.NetstatWatcher.ComputerHostsTcpConnection"
Accessibility="Public" Abstract="false" Base="System!System.Containment">
    <Source ID="Source" MinCardinality="0" MaxCardinality="2147483647"
Type="Network.Windows.Computer.NetstatWatcher.Computer" />
    <Target ID="Target" MinCardinality="0" MaxCardinality="2147483647"
Type="Network.Windows.Computer.NetstatWatcher.TcpConnection" />
</RelationshipType>

<RelationshipType ID="Network.Windows.Computer.NetstatWatcher.ComputerHostsListeningPort"
Accessibility="Public" Abstract="false" Base="System!System.Containment">
    <Source ID="Source" MinCardinality="0" MaxCardinality="2147483647"
Type="Network.Windows.Computer.NetstatWatcher.Computer" />
    <Target ID="Target" MinCardinality="0" MaxCardinality="2147483647"
Type="Network.Windows.Computer.NetstatWatcher.ListeningPort" />
</RelationshipType>
```

Discoveries

The mechanism of finding objects that match the definition and storing it in the SCOM database is called discovery. There are different types of discoveries, starting from matching registry values over results of an WMI query to scripts that can cover everything. Targets define on which component the discovery shall run.

First discovery **Discovery.NetstatWatcher.Computer** is used to find computer objects. Targeted are all Windows computers (which are already monitored by SCOM).
The FilteredRegistryDiscoveryProvider' scans the registry and if the key HKLM\SOFTWARE\ABCIT\NetstatWatcher exists, the object will be created. The interval is daily.

Also discovered here is the 'FiltePath' which is used to define the path in the file system where both text files shall be found.

Second discovery **Discovery.NetstatWatcher.listeningPorts** finds listening ports reading out 'monitoredListeningPorts.csv'. Targeted are the previously discovered '…NetstatWatcher.Computer' – computer objects.
The 'TimedPowerShell.DiscoveryProvider' triggers the 'DiscoverNetstatWatcherItems.ps1' – PowerShell script which does the logic (see above: Preparing raw data). Interval is hourly.

Third discovery **Discovery.NetstatWatcher.tcpConnections** finds listening ports reading out 'monitoredTcpConnects.csv'. Targeted are the previously discovered '…NetstatWatcher.Computer' – computer objects.
The 'TimedPowerShell.DiscoveryProvider' triggers the 'DiscoverNetstatWatcherItems.ps1' – PowerShell script which does the logic (see above: Preparing raw data). Interval is hourly.

Fourth and Fifth discovery **Discovery.NetstatWatcher.ComputerHostsTcpConnections / …ComputerHostsListeningPorts** creates the relation between computers and the monitored objects.
The 'TimedPowerShell.DiscoveryProvider' triggers the 'DiscoverNetstatWatcherItemRelations.ps1'. Interval is hourly.

Monitors

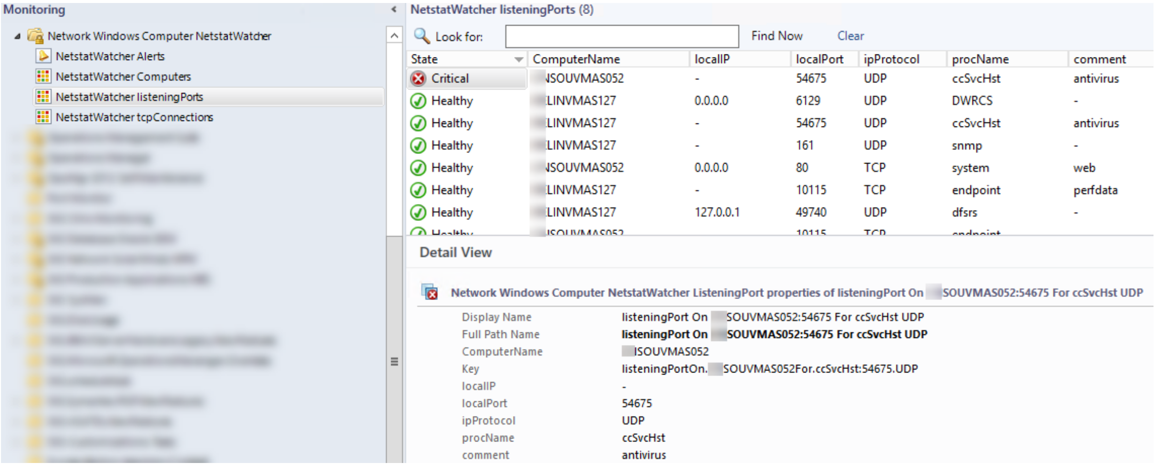
Monitors are for finding out which Health State an object has. – An object

- Monitor.tcpConnection targets all objects of the class Network.Windows.Computer.NetstatWatcher.TcpConnection
- Monitor.listeningPort targets all objects of the class Network.Windows.Computer.NetstatWatcher.ListeningPort

This monitor here uses PowerShell script MonitorNetstatWatcherItems.ps1 to determine the state of object. (See above: Interpreting output and initiate reaction) Interval is every 5 minutes.

Views

To make all discovered objects and their health state visible a state views are used.



Monitoring
Network Windows Computer NetstatWatcher
NetstatWatcher Alerts
NetstatWatcher Computers
NetstatWatcher listeningPorts
NetstatWatcher tcpConnections
Look for:
Find Now
Clear

[redacted]

NetstatWatcher listeningPorts (8)

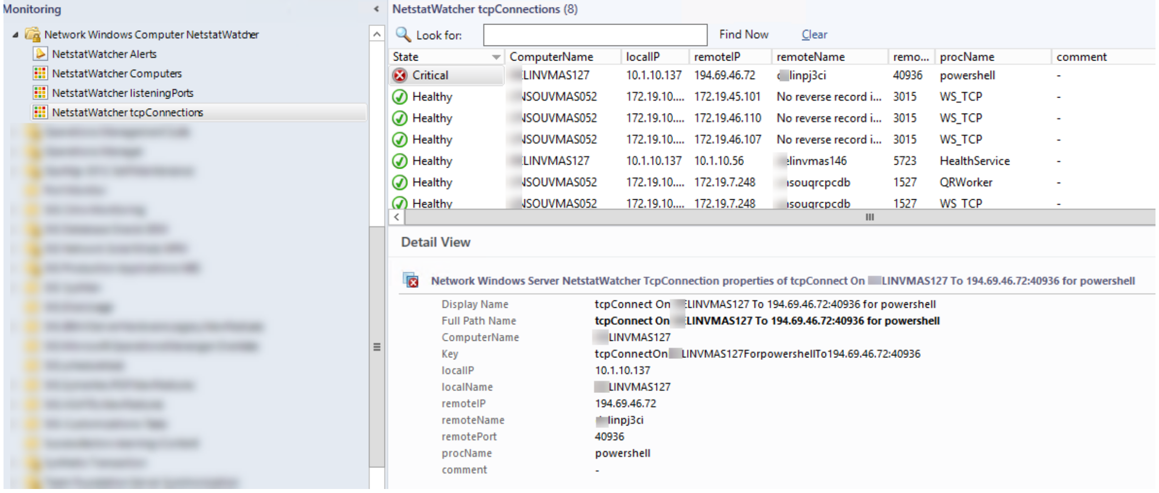
State	ComputerName	localIP	localPort	ipProtocol	procName	comment
Critical	[redacted]SOUVMAS052	-	54675	UDP	ccSvcHst	antivirus
Healthy	[redacted]LINVMAS127	0.0.0.0	6129	UDP	DWRCS	-
Healthy	[redacted]LINVMAS127	-	54675	UDP	ccSvcHst	antivirus
Healthy	[redacted]LINVMAS127	-	161	UDP	snmp	-
Healthy	[redacted]SOUVMAS052	0.0.0.0	80	TCP	system	web
Healthy	[redacted]LINVMAS127	-	10115	TCP	endpoint	perfdata
Healthy	[redacted]LINVMAS127	127.0.0.1	49740	UDP	dfsrs	-
Healthy	[redacted]SOUVMAS052		10115	TCP	endpoint	

Detail View

Network Windows Computer NetstatWatcher ListeningPort properties of listeningPort On [redacted]SOUVMAS052:54675 For ccSvcHst UDP

Display Name	listeningPort On [redacted]SOUVMAS052:54675 For ccSvcHst UDP
Full Path Name	**listeningPort On [redacted]SOUVMAS052:54675 For ccSvcHst UDP**
ComputerName	[redacted]SOUVMAS052
Key	listeningPortOn.[redacted]SOUVMAS052For.ccSvcHst:54675.UDP
localIP	-
localPort	54675
ipProtocol	UDP
procName	ccSvcHst
comment	antivirus

stateview showing listeningPorts



Monitoring
Network Windows Computer NetstatWatcher
NetstatWatcher Alerts
NetstatWatcher Computers
NetstatWatcher listeningPorts
NetstatWatcher tcpConnections
Look for:
Find Now
Clear

[redacted]

NetstatWatcher tcpConnections (8)

State	ComputerName	localIP	remoteIP	remoteName	remo...	procName	comment
Critical	[redacted]LINVMAS127	10.1.10.137	194.69.46.72	[redacted]linpj3ci	40936	powershell	-
Healthy	[redacted]SOUVMAS052	172.19.10....	172.19.45.101	No reverse record i...	3015	WS_TCP	-
Healthy	[redacted]SOUVMAS052	172.19.10....	172.19.46.110	No reverse record i...	3015	WS_TCP	-
Healthy	[redacted]SOUVMAS052	172.19.10....	172.19.46.107	No reverse record i...	3015	WS_TCP	-
Healthy	[redacted]LINVMAS127	10.1.10.137	10.1.10.56	[redacted]linvmas146	5723	HealthService	-
Healthy	[redacted]SOUVMAS052	172.19.10....	172.19.7.248	[redacted]souqrcpcdb	1527	QRWorker	-
Healthy	[redacted]SOUVMAS052	172.19.10....	172.19.7.248	[redacted]souqrcpcdb	1527	WS_TCP	-

Detail View

Network Windows Server NetstatWatcher TcpConnection properties of tcpConnect On [redacted]LINVMAS127 To 194.69.46.72:40936 for powershell

Display Name	tcpConnect On [redacted]LINVMAS127 To 194.69.46.72:40936 for powershell
Full Path Name	**tcpConnect On [redacted]LINVMAS127 To 194.69.46.72:40936 for powershell**
ComputerName	[redacted]LINVMAS127
Key	tcpConnectOn[redacted]LINVMAS127ForpowershellTo194.69.46.72:40936
localIP	10.1.10.137
localName	[redacted]LINVMAS127
remoteIP	194.69.46.72
remoteName	[redacted]linpj3ci
remotePort	40936
procName	powershell
comment	-

stateview showing tcpConnections

Alerts are created if a port is not listening or a connection is lost. Those are shown in the 'NetstatWatcher Alerts' view.

Conclusion

You can download the management pack with the extensions .xml or. mpb. I published the software under GNU General Public License. Feel free to use it without costs or obligations. The software is provided "as is" without express or implied warranty.

If you don’t like the naming used, feel free to change the text in the XML file. Make sure that your search with case sensitivity. I used Visual Studio 2015 with Authoring Extensions for this management pack. Feel free to use the sources I published on Github.

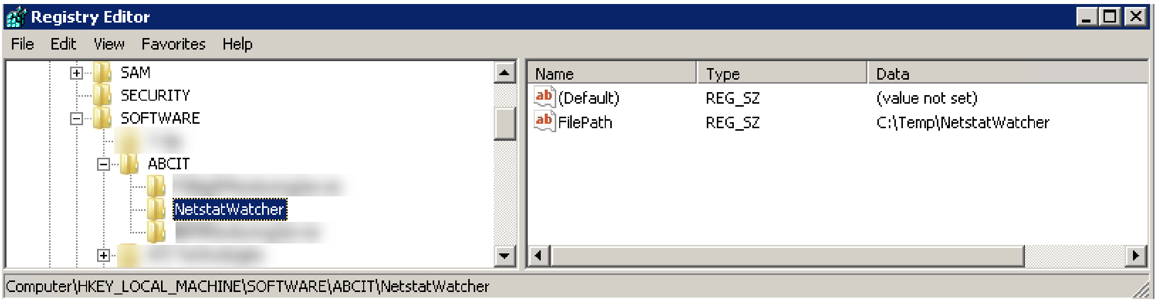
Setup

If you like the to monitor **listening ports** or **tcp connecitons** on a computer, follow these 2 / 3 steps:

1. Open notepad and copy the following text into a text file, rename it as *.reg and import it to the registry via double click:

   ```
   Windows Registry Editor Version 5.00
   [HKEY_LOCAL_MACHINE\SOFTWARE\ABCIT\NetstatWatcher]
   "FilePath"="C:\\Temp\\NetstatWatcher"
   ```

   It will look like this in the registry:

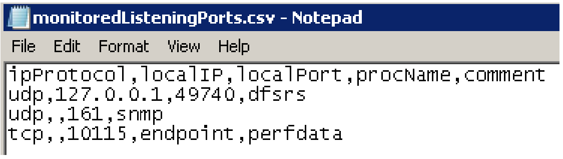
Name	Type	Data
(Default)	REG_SZ	(value not set)
FilePath	REG_SZ	C:\Temp\NetstatWatcher

   Computer\HKEY_LOCAL_MACHINE\SOFTWARE\ABCIT\NetstatWatcher

2. If you like to monitor listening ports, open notepad and create a text file named **monitoredListeningPorts.csv** in the path you have defined in the registry under ‘FilePath’. For example with the following content:

   ```
   ipProtocol,localIP,localPort,procName,comment
   udp,127.0.0.1,49740,dfsrs
   udp,,161,snmp
   tcp,,10115,endpoint,perfdata
   ```

   It will look like this in notepad:

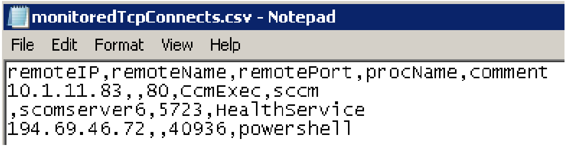
   
   monitoredListeningPorts.csv - Notepad
   File Edit Format View Help

   ```
   ipProtocol,localIP,localPort,procName,comment
   udp,127.0.0.1,49740,dfsrs
   udp,,161,snmp
   tcp,,10115,endpoint,perfdata
   ```

3. If you like to tcp connections, open notepad and create a text file named **monitoredTcpConnects.csv** in the path you have defined in the registry under ‘FilePath’. For example with the following content:

   ```
   remoteIP,remoteName,remotePort,procName,comment
   10.1.11.83,,80,CcmExec,sccm
   ,scomserver6,5723,HealthService
   194.69.46.72,,40936,powershell
   ```

It will look like this in notepad:


monitoredTcpConnects.csv - Notepad

File Edit Format View Help

```
remoteIP,remoteName,remotePort,procName,comment
10.1.11.83,,80,CcmExec,sccm
,scomserver6,5723,HealthService
194.69.46.72,,40936,powershell
```

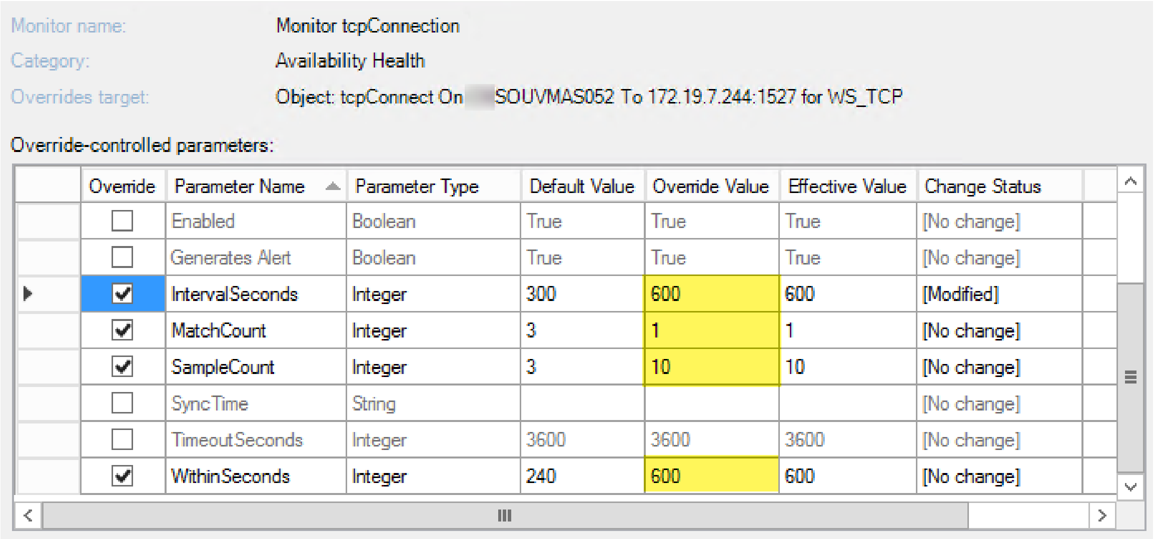
4. Create a management pack to store the overrides. Name it like Network.Windows.Computer.NetstatWatcher.Overrides for example

5. Optional override tcpConnection monitor settings for one or all objects

The default settings are as follows:

Parameter	Value	Meaning
IntervalSeconds	300	How often the script which checks the objects is executed.
SampleCount	3	How many times the connection status is checked
MatchCount	3	How many times the connection must be there
WithinSeconds	240	Period in which the samples are taken
TimeoutSeconds	3600	Maximum time the script can run without been terminated by the SCOM agent.

E.g.


Monitor name: Monitor tcpConnection

Category: Availability Health

Overrides target: Object: tcpConnect On [redacted] SOUVMAS052 To 172.19.7.244:1527 for WS_TCP

Override-controlled parameters:

Override	Parameter Name	Parameter Type	Default Value	Override Value	Effective Value	Change Status
☐	Enabled	Boolean	True	True	True	[No change]
☐	Generates Alert	Boolean	True	True	True	[No change]
☑	IntervalSeconds	Integer	300	600	600	[Modified]
☑	MatchCount	Integer	3	1	1	[No change]
☑	SampleCount	Integer	3	10	10	[No change]
☐	SyncTime	String				[No change]
☐	TimeoutSeconds	Integer	3600	3600	3600	[No change]
☑	WithinSeconds	Integer	240	600	600	[No change]

The script runs every 10 minutes. Every minute the connection state is checked. Only one attempt need to succeed to identify a healthy communication pattern.