# Watching Windows: An Open Source approach using PowerShell

**Conference Paper** · September 2015

CITATIONS
0

READS
1,667

**2 authors:**

Ralf C. Staudemeyer
University of Applied Sciences Schmalkalden
**34** PUBLICATIONS **414** CITATIONS
SEE PROFILE

Karl Van der Schyff
Abertay University
**23** PUBLICATIONS **161** CITATIONS
SEE PROFILE

# Watching Windows: An Open Source approach using PowerShell

Karl van der Schyff
Department of Computer Science
Rhodes University
Grahamstown, South Africa
e-mail: k.vanderschyff@ru.ac.za

Ralf C. Staudemeyer
School of Computing
University of South Africa
Johannesburg, South Africa
e-mail: staudrc@unisa.ac.za

***Abstract*—Monitoring servers running Microsoft Windows can be a costly exercise, mainly related to software recommended by vendors. Academic environments with financial constrains have thus to either cease their monitoring endeavours or look towards open source solutions. In this paper we demonstrate how to monitor aspects of a Windows server in a simple and flexible way using open source tools. To accomplish this we make use of the script language PowerShell and the Nagios Remote Plugin Executor (NRPE) to gather information on a scheduled task within Windows Server 2012 R2.**

**This paper addresses a knowledge gap that inhibits open source monitoring of Windows servers. Currently there is a wide range of plugins and scripts available at online repositories, but these do not always address specific aspects of Windows servers. Frequently those plugins deemed suitable are only available in binary form or written in a scripting language not pre-installed on a Windows server. This complicates information gathering. The suggested approach mitigates this by providing an example script, which can easily be customised to monitor almost any scheduled task[1][2].**

***Index Terms*—Network monitoring, Windows environment, PowerShell, NRPE, SNMP, WMI, Icinga, Nagios.**

## I. Introduction

Infrastructure monitoring is an integral part of system administration [1]. It provides mechanisms to monitor computer networks or system components for failures and anomalies. In such an event the system can notify the person responsible for the concerned devices. Unfortunately monitoring is often neglected due to insufficient funds to purchase vendor recommended turnkey solutions.

Administrators can however make use of a number of open source monitoring platforms [1], [2]. For the most part these platforms are derivatives of the well-known Nagios[3] infrastructure monitoring system, with most of the differences related to user interaction [1]. In this paper we make use of the Icinga[4] monitoring system, which is compatible with Nagios. In terms of configuration this is also true for the wide range of plugins and scripts available at repositories such as Monitoring Plugins [5], Monitoring Exchange [6] and NSClient++ [7]. In this paper we demonstrate how to monitor Microsoft Windows Server backups by using the Nagios Remote Plugin Executor (NRPE) and client-side scripts written in PowerShell.

Our approach is relevant since there are very few academic publications addressing the challenge of monitoring Windows systems. Currently, Windows administrators can monitor their network infrastructures with the built-in Windows Performance Monitor [3], which makes use of internal performance counters [4]. However, this approach requires detailed custom configuration, due to the large number of system-specific counters. Furthermore, after locating a relevant counter it has to be queried either with the performance console using the perfmon tool, or remotely with Windows Management Instrumentation (WMI) [5]; the latter an alternative to the Simple Network Management Protocol (SNMP)[6]. In addition to the aforementioned technologies administrators may also use costly third-party tools.

Open source approaches are not only cost effective, well documented and extensible, but also supported by a large developer and user community. Thus, monitoring Windows servers using an open source approach is relevant and addresses aspects of a persistent problem.

The paper is structured as follows: At first the reader is presented with a literature review and discussion on the general use of Icinga. This is followed by a more focused discussion on documented interfaces suitable to monitor a Microsoft Windows server. The configuration of the test environment is followed by a discussion of the example script which is then used to monitor server backups. This script is intended as a template for monitoring other scheduled tasks and can be easily adapted to specific needs. The paper concludes by briefly highlighting areas of future research.

## II. Literature review on infrastructure monitoring

According to [2] there are two primary types of monitoring – real-time monitoring, which is used to gather information on the current state of network infrastructure and services, and historical monitoring, which is used to view performance data. For example, real-time monitoring is especially useful to pro-actively monitor mission-critical systems. A case in point is the monitoring of radiological information systems as discussed in [7]. Such real-time monitoring aids to the

[1]Neither the entire paper nor any part of its content has been published or has been accepted for publication elsewhere.

[2]It has not been submitted to any other journal.

[3]www.nagios.org

[4]www.icinga.org

[5]www.monitoring-plugins.org

[6]www.monitoringexchange.org

[7]www.nsclient.org

gathering of valuable statistical information. From the aforementioned example, it is clear that monitoring computing infrastructures has its advantages. In fact, [8] deem it to be essential when administering infrastructures with a number of networked systems and services.

It is also not uncommon for organizations to monitor their network infrastructure. For example, [9] describe how Nagios is used to monitor the Enabling Grid for E-sciencE (EGEE). Another example is provided by [2] where the author describes how Nagios is used to monitor specific library services. Even supercomputing clusters are monitored, with [10] describing the migration from Nagios to Icinga within the Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider (LHC).

Some authors have also reported success in monitoring the performance of Windows clusters. In an article by [11] the authors discuss a tool called WatchTower, which they use to simplify the collection of Windows performance data by providing easy access to raw performance counters.

Over and above the use of Nagios and Icinga many other monitoring platforms, tools and extensions exist like Cacti[8], WhatsUpGold[9], or check_mk[10]. Similar tools are discussed in [12], where the authors describe a wide range of local and distributed UNIX and Linux monitoring tools.

In contrast with UNIX and Linux, the monitoring of Windows systems is often done with either Microsoft System Center [13], Windows Performance Monitor [3] or other commercial monitoring solutions like SolarWinds[11]. In the case of System Center a proprietary client needs to be installed on each machine. This makes such a solution only viable in a Windows-only environment.

On the other hand, an open source approach circumvents these limitations. Such approaches are not only cost effective, but also are flexible in terms of the operating systems and network services which can be monitored. As such, an open source approach has much to offer, but is not without its own limitations. One such limitation is the steep learning curve and profound technical knowledge required. Much of what is automatically setup by commercial solutions is controlled via plain text configuration files. Administrators may also be required to write custom plugins. Additionally, unless the monitoring traffic is routed out-of-band, it is often required that the monitoring system's security be reduced. This in turn increases the attack surface.

In the following section we briefly discuss related monitoring technologies and protocols.

### A. Simple Network Management Protocol (SNMP)

The Simple Network Management Protocol (SNMP) [6], [14], [15] describes a generic process to manage the release of data. It was designed to minimize the number and complexity of monitoring agents. It is specified to be extensible and platform independent. This makes SNMP a widely accepted mechanism to monitor a diverse set of hosts and services. Most operating systems and network devices are compatible with SNMP, or can be extended accordingly.

To gather information, SNMP queries Object IDentifiers OIDs, which define the information contained within a Management Information Base (MIB) [16], [17]. To extend the capabilities provided by the standard MIBs, vendors often define custom ones. These are then documented and distributed with the corresponding network device. In general SNMP monitoring information gets pulled from the network device, but can also be pushed from a device. This is referred to as a trap. These information flows take place on UDP-port 161 and 162 respectively.

A distinction is made between SNMP Versions 1, 2c, and 3. Version 2c introduces a larger domain in comparison to Version 1, which is essential when accessing counters in recent devices[14]. SNMPv3 extends the options of 2c improving user management. In particular SNMPv3 introduces encryption of authentication information and transport data [15].

However, within a Windows environment the use of SNMP is limited in that it only supports SNMPv1 and v2c. Moreover, support for SNMP on Windows Server 2012 has been deprecated [18].

### B. Windows Management Instrumentation (WMI)

Windows Management Instrumentation (WMI) [5], [19] was introduced in Windows 2000. It is Microsoft's implementation of the Common Information Model (CIM) [20], which is part of the Web-Based Enterprise Management (WBEM) internet standards suite [21]. As with SNMP, WMI provides remote read and write access to local resources. This includes extended event notification and command execution via a scriptable application programming interface (API) [22]. Amongst other languages this API supports PowerShell, VBScript and Perl.

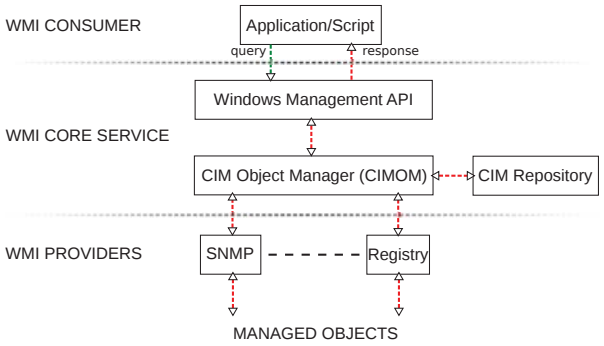


Figure 1. Architecture of the Windows Management Instrumentation (WMI)

Figure 1 shows how WMI integrates with other key Windows components. There are four main components that constitute the core architecture: management applications, the WMI core service, the CIM repository, and WMI providers. The providers handle objects that provide access to different parts of the operating system. To query these providers the WMI core service provides an API for management applications and scripts. This includes scripts, .NET classes (system.management.class) and the Windows Management Instrumentation Command-line (WMIC) tool.

[8] www.cacti.net
[9] www.whatsupgold.com
[10] mathias-kettner.de/check_mk.html
[11] www.solarwinds.com

Similar to SNMP MIB, WMI makes use of a CIM repository providing hierarchically structured namespaces. Several predefined namespaces exist, with each namespace being mapped to the objects handled by their providers. These objects can relate to actions, hard- or software, and events, but also to further instances.

However, WMI is a technology used solely by Windows and it has its limitations [19]. These include the inability to find the same objects on different systems and the limited availability of providers.

## III. The Icinga monitoring platform

Icinga is a monitoring platform that builds on the foundations laid down by Nagios. Improvements include a user-friendly web interface, an open and transparent development architecture as well as support from a large online community who share their experiences and knowledge [10].

Although Icinga and Nagios both have a steep learning curve [2], their implementation reduces reactive infrastructure management and promotes a more proactive approach. Such an approach makes use of a typical Icinga installation, which consists of a hierarchical structure of plain text configuration files [23].

For the most part these configuration files contain detailed information on the monitored devices and services: Host definitions include details such as the hostname, IP address, check periods and check commands of the monitored device. Service definitions typically reference the monitored host and contain more details about the executed commands. Furthermore command definitions are linked to plugins and define the command line to be executed by the plugin.

Icinga uses plugins in order to check defined hosts and services. These are either executed locally or remotely. Local checks primarily monitor local system parameters such as CPU load and memory usage. Remote checks monitor network services. For more information on the configuration of Icinga see [24].

Alternatively administrators could run checks locally on remote hosts by making use of SNMP [6], SSH [25] or NRPE [26]. All these technologies require a daemon running on the target system. The daemon executes the local check on the host and returns the output to the monitoring system. Some well-known default plugins to facilitate remote execution include *check_snmp*, *check_by_ssh*, and *check_nrpe*.

NRPE, although not designed for security like SSH and SNMPv3, is favoured mainly because of its low overhead and the availability of a proven NRPE agent for Windows (such as NSClient++[12]). Here the two required components are the NRPE Daemon running on the remote host and the *check_nrpe* plugin residing on the monitoring server itself.

Icinga also provides a plugin-API for users who wish to develop their own plugins, which is documented in the plugin development guidelines [27]. These guidelines describe how custom written plugins should behave in terms of input and output. In short, every plugin on an Icinga server has to fulfil two requirements in order to be recognized as a plugin. Firstly, it should exit with one of four possible return values and secondly it should return a single line of text to STDOUT. Each of these return values is matched to a corresponding service or host state, as outlined in Table I.

[12]www.nsclient.org

Table I
Icinga plugin-API return values and states

| Return Value | Service State | Host State |
|---|---|---|
| 0 | OK | UP |
| 1 | WARNING | DOWN |
| 2 | CRITICAL | UNREACHABLE |
| 3 | UNKNOWN | n/a |

Optionally plugins may also return multiple lines of text as well as performance data. For example to return performance data a plugin needs to separate the actual performance values from the returned text by using a pipe symbol, as follows:

```
icinga@moni:~$ ./check_disk -w 10% -c 5% -p /
DISK OK - free space: / 4981 MB (68% inode=88%);|
                      /=2311MB;6914;7298;0;7683
```

In addition to this separation, there is also a standard format to which the returned data has to conform. This entails specifying the levels for warning (6914), critical (7298), minimum (0), and maximum (7683). The returned data also includes the used space (2311MB) and a label. Here the label refers to the root directory (/).

Once a plugin has executed, one of these values, together with its corresponding textual description (and optional performance data), is returned to the Icinga server. For example, if the disk usage in the above example takes on a value between the thresholds set for warning and critical, the plugin will return a value of 1. Icinga will then flag this service as being in a WARNING state, as outlined in Table I. This warning state is then displayed on the user interface. The following section will provide a more detailed look at how remote plugins (and their corresponding handlers or scripts) are executed.

### A. Nagios Remote Plugin Executor (NRPE)

To monitor a Windows system via NRPE [26] a monitoring agent is required. Such an agent facilitates communication between the monitoring server and the Windows host. In this paper, the well known NRPE agent provided by the NSClient++ monitoring daemon was used. However, the default configuration of NSClient++ had to be adjusted to support the execution of external scripts. The process of executing external scripts is illustrated in Figure 2.

Figure 2 further illustrates that NSClient++ is a modular daemon; for this reason only exposes those functions for which the core has loaded modules. In our case the modules NRPEServer and CheckExternalScripts are loaded. It is specifically the CheckExternalScripts module, that provides the ability to run scripts. In Version 0.4.1 of NSClient++, the scripting languages Visual Basic, Python, Perl, and PowerShell are supported, with PowerShell being the only language that is pre-installed on Windows servers. The exact configuration requirements to utilize PowerShell scripts are discussed in the following section.

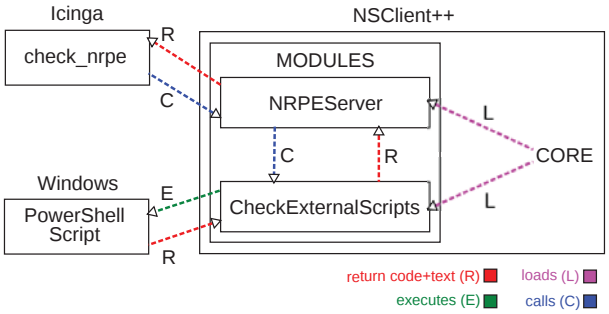


Figure 2. Execution of external scripts with NSClient++ [28]

### B. PowerShell

Windows XP introduced PowerShell to address the shortcomings of the command prompt inherited from MSDOS. PowerShell addresses these shortcomings by allowing administrators to make use of .NET libraries [29]. This enables it to integrate well with other Windows technologies [30], [31], such as WMI [5], [32].

To access .NET libraries and WMI providers Powershell executes specific lightweight commands, called cmd-lets. Each cmd-let addresses a component of the operating system and in most cases also requires a number of arguments. For example, to query the local operating system for all the processes currently running, one can execute the *Get-Process* cmd-let. This will return a list of processes together with a set of corresponding properties.

Using Powershell to leverage technologies like .NET and WMI improves the manageability of Windows systems. This is especially true when combined with NRPE. In the following section we elaborate on how PowerShell can be used to extend the monitoring capabilities of Icinga using NRPE.

## IV. Configuration of the Test Environment

This section contains the specifics of how PowerShell can be used to monitor a scheduled task. In this example the success of a scheduled Windows server backup is monitored.

### A. Windows Server setup

We performed a default installation of Windows Server 2012 R2. As described earlier we prefer to monitor a Windows server via a NRPE agent, which requires the NSClient++ daemon. Version 0.4.1 (x64) of NSClient++ was installed onto the Windows server. This installation proceeded as follows:

After accepting the license agreement, a typical installation was selected and the default configuration settings were accepted. However, on the following screen the IP address of the Icinga server was entered under *Allowed Hosts*. This permits the monitoring server to remotely access the NRPE agent. Then we enabled NRPE by selecting the corresponding check box. A final *Next* finished the installation.

To facilitate the use of PowerShell, the configuration file of NSClient++ was altered as follows: Firstly, the use of external scripts had to be enabled by adding the following lines of text to the nsclient.ini file. This file can be located in the installation directory of NSClient++ (C:\Program Files\NSClient++):

```
[...]
; POWERSHELL WRAPPING -
ps1 = cmd /c echo scripts\\%SCRIPT% %ARGS%; exit
      ($lastexitcode) | powershell.exe -command -
[...]
```

Without this change the *check_nrpe* plugin will fail to run the external PowerShell script. The second configuration change was to add the actual command to run the external script. This required adding the following lines of text to the *nsclient.ini* configuration file:

```
[...]
[/settings/external scripts/scripts]
check_backups = cmd /c echo scripts\check_backups.ps1;
      exit $LastExitCode | powershell.exe -command -
[...]
```

Finally, the *check_backups.ps1* script was copied to the scripts directory (C:\Program Files\NSClient++\scripts) of the NSClient++ installation. The last configuration change ensured that the *User Account Control* security feature of Windows Server did not impede the execution process. To ensure this, it was set to *never notify*. This setting can be verified by opening the *Control Panel*, selecting the option *User Accounts* and clicking on the *Change User Account Control* settings.

### B. PowerShell configuration

To enable the execution of local scripts we needed to make security modifications to the PowerShell environment. This was achieved by executing the following command from an administrative PowerShell command prompt:

```
Set-ExecutionPolicy -ExecutionPolicy remotesigned
```

This effectively allows local scripts to be executed, but blocks the execution of remote scripts unless they have been signed by a trusted certificate authority. In this example the script is actually executed locally by NSClient++, so there is no need to sign it.

### C. Icinga configuration

All tests were performed on a default installation of Icinga v1.9.3 on Ubuntu 12.04 LTS Server. To monitor the Windows server a host configuration file was created containing the following details:

```
define host
  {
  use                 template-host-windows
  host_name           backup-server01
  alias               Backup Server
  address             HOSTADDRESS
  }
```

To monitor the scheduled backups of this host a service definition file was created containing the following:

```
define service
  {
  use                 template-service-windows
  service_description Backup status nrpe
  check_command       check_nrpe!check_backups
  host                backup-server01
  }
```

The corresponding command description is as follows:

```
define command
  {
  command_name        check_nrpe
  command_line        $USER1$/check_nrpe -H $HOSTADDRESS$
                        -t 60 -p 5666 -c $ARG1$
  }
```

Note that the argument *check_backups* in the service definition is passed to the command definition. There it is executed by the *check_nrpe* plugin. For more information on the

specifics relating to command definitions the reader is referred to the Icinga documentation [24].

## V. Example of a PowerShell script

In this section we provide a brief discussion of a PowerShell script template, as presented in Figure 3. In this example the script caters for monitoring a scheduled backup task. Such a task can be configured through the graphical Windows Server Backup interface.

```
1  If (-NOT ([Security.Principal.WindowsPrincipal]`
2 [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole`
3 ([Security.Principal.WindowsBuiltInRole] "Administrator"))
4    {
5    $arguments = "& '" + $myinvocation.mycommand.definition + "'"
6    Start-Process powershell -Verb runAs -ArgumentList $arguments
7    Break
8    }
9 $returnOK = 0
10 $returnCrit = 2
11 $returnUnknown = 3
12 $ErrorActionPreference = "Stop"
13 Try
14 {
15 $ST = new-object -com("Schedule.Service")
16 $ST.connect()
17 $fold=$ST.GetFolder("\Microsoft\Windows\Backup")
18 $task=$fold.GetTask("Microsoft-Windows-WindowsBackup")
19 $results=$task.LastTaskResult
20 $sp=(get-date)-($task.LastRunTime)
21 }
22 Catch
23 {
24 $results = 3
25 }
26 Finally
27 {
28 if ($results -eq 3)
29    {
30    write-host "Unknown State: Possible script error"
31    exit $returnUnknown
32    }
33 else
34    {
35    if (($results -ne 0) -or ($sp.Hours -gt 24))
36          {
37          write-host "Backups failed: Check host logs"
38          exit $returnCrit
39          }
40    else
41          {
42          write-host "OK: Backup Successful"
43          exit $returnOK
44          }
45    }
46 }
```

Figure 3. Powershell backup script

The first lines of code ensure that the script runs with administrator privileges. Then lines 9 to 11 declare the return values Icinga expects (see Figure 3), which are the service states 0 (OK), 2 (CRITICAL), and 3 (UNKNOWN). For simplicity's sake this script will never return the value 1 (WARNING), but can be easily adjusted accordingly.

Lines 13 to 23 are part of the error handling structure: In line 15 a new object of the *Schedule.Service* class is instantiated. This is followed by line 16 which establishes a connection to the local host. To locate the correct task the script has to enumerate all the scheduled tasks within the folder specified in line 17. This object is then used to retrieve the status of a specific task by using its name.

For scheduled tasks a return value of 0 indicates that the scheduled task ran successfully. To ensure that the backup task has been executed within the last 24 hours, line 20 ascertains how much time has passed since the last backup. If an error occurred during the execution of line 13 to 21, the *$results* variable is set to the value 3 in line 24.

The *Finally* clause (line 26 to 46) sets the service state and a corresponding line of text. The clause first checks if an error occurred during the execution of the script. If this is the case (*$results*=3) the constant *$returnUnknown* is returned to Icinga. Icinga then flags the service state as UNKNOWN.

If no errors were encountered lines 28 to 32 are skipped. The *Else* statement is executed next to ascertain whether the backup was successful. In this instance there are two conditions which equate to a failed backup. The statement tests whether the scheduled task exited with a value not equal to 0 or if the backup is older than 24 hours. If either one of these conditions are true, the backup is deemed a failure and the constant *$returnCrit* is returned. Icinga then flags the service state as CRITICAL. However, if these conditions are false a valid backup has taken place in the last 24 hours and the script returns the constant *$returnOK*. Icinga then flags the service as OK.

We would like to point out that with a few changes this script can check any scheduled task. This can be accomplished by changing the location (line 17) and name of the scheduled task (line 18). Additionally, this script can be expanded to accept parameters allowing it to check multiple scheduled tasks.

## VI. Conclusion

In this paper we addressed the challenge of monitoring Windows servers. To achieve this the open source tool Icinga and the NRPE-agent (provided by NSClient++) were used to monitor a scheduled task by using PowerShell. First we provided a brief introduction to infrastructure monitoring and the Nagios-derived monitoring platforms. The reader was then presented with background information on the limited number of monitoring technologies available for Windows platforms. This was followed by a discussion on the configuration changes required to enable NRPE and PowerShell to monitor a Windows server. A discussion on the mechanics of the PowerShell script followed, providing the reader with the required background information to understand how it operates.

The development of scripts and plugins to extend the capabilities of monitoring systems is an ever growing area. This is especially true for Windows systems. For this reason it is pertinent that further research and development take place in this field.

One such area is the lack of literature on monitoring Windows systems using open source tools. There is a requirement to provide researchers with a taxonomy of such technologies. This would not only provide researchers with an overall understanding of Windows monitoring, but also point out knowledge gaps. These could then be addressed using approaches similar to what was demonstrated in this paper.

Besides just monitoring hosts, further research could be directed towards methods of self-healing. Especially within Windows environments. For example, it is common to apply a number of Windows updates on a regular basis. Instead of just reporting on these outstanding updates, scripts could be developed to automatically trigger their installation on certain conditions. This would be especially useful in large production environments. The same applies to automatically adjusting Windows firewalls, intrusion detection and log analysis.

## VII. Acknowledgments

This research was partially funded by the fellowship programs of Rhodes University (2013) and the University of South Africa (2014).

## References

[1] J. Reams, "Extensible monitoring with Nagios and messaging middleware," in *Proceedings of the 26th international conference on Large Installation System Administration (LISA' 12)*. USENIX Association, 2012, pp. 153–162.

[2] T. M. M. Silver, "Monitoring network and service availability with open-source software," *Information Technology and Libraries*, vol. 29, no. March, pp. 8–22, 2013.

[3] Microsoft, "Windows Performance Monitor," 2014. [Online]. Available: http://technet.microsoft.com/en-us/library/cc749249.aspx

[4] ——, "Performance Counters," 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/aa373083(v=vs.85).aspx

[5] ——, "Windows Management Instrumentation," 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa394582(v=vs.85).aspx

[6] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple Network Management Protocol (SNMP)," *RFC 1157*, pp. 1–36, 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1157.txt

[7] C. Toland, C. Meenan, M. Warnock, and P. Nagy, "Proactively monitoring departmental clinical IT systems with an open source availability system." *Journal of Digital Imaging*, vol. 20 Suppl 1, no. September, pp. 119–24, Nov. 2007.

[8] M. Matýsek, M. Adámek, M. Kubalčík, and M. Mihok, "Monitoring of Computer Networks and Applications using Nagios," *Advances in Data Networks, Communications, Computers and Materials Monitoring*, pp. 63–67, 2012.

[9] E. Imamagic and D. Dobrenic, "Grid infrastructure monitoring system based on Nagios," in *Proceedings of the 2007 workshop on Grid monitoring (GMW '07)*. New York, New York, USA: ACM Press, 2007, pp. 23–28.

[10] G. Bauer, U. Behrens, O. Bouffet, M. Bowen, J. Branson, S. Bukowiec, M. Cigane, S. Cittolin, J. a. Coarasa Perez, C. Deldicque, M. Dobson, A. Dupont, S. Erhan, A. Flossdorf, D. Gigi, F. Glege, R. Gomez-Reino, C. Hartl, J. Hegeman, A. Holzner, Y. L, L. Masetti, F. Meijers, E. Meschi, R. K, V. O'Dell, L. Orsini, C. Paus, A. Petrucci, M. Pieri, G. Polese, A. Racz, O. Raginel, H. Sakulin, M. Sani, C. Schwick, D. Shpakov, M. Simon, A. C, and K. Sumorok, "Health and performance monitoring of the online computer cluster of CMS," *Journal of Physics: Conference Series*, vol. 396, no. 4, p. 7, Dec. 2012.

[11] M. W. Knop, P. A. Dinda, and J. M. Schopf, "Windows Performance Monitoring and Data Reduction Using WatchTower," in *Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing*. Citeseer, 2002, p. 14.

[12] Z. Škiljan and B. Radić, "Monitoring systems: Concepts and tools," in *Proceedings of the 6th CARNet Users Conference*, 2004.

[13] Microsoft, "Microsoft System Center 2012 R2 Evaluation Resources," 2014. [Online]. Available: http://technet.microsoft.com/en-US/evalcenter/dn205296.aspx?wt.mc_id=TEC_149_1_27

[14] W. Stallings, "SNMP and SNMPv2: The infrastructure for network management," *Communications Magazine, IEEE*, no. March, pp. 37–43, 1998.

[15] ——, "SNMPv3: A security enhancement for SNMP," *IEEE Communications Surveys & Tutorials*, vol. 1, no. 1, pp. 2–17, 1998.

[16] K. McCloghrie and F. Kastenholz, "The Interfaces Group MIB," *RFC 2863*, p. 69, 2000. [Online]. Available: https://www.ietf.org/rfc/rfc2863.txt

[17] D. Perkins, "Understanding SNMP MIBs – Revision 1.1.7," in *Proceedings of the Nineteenth Internet Engineering Task Force*, 1993, p. 44.

[18] Microsoft, "Features Removed or Deprecated in Windows Server 2012," 2014. [Online]. Available: http://technet.microsoft.com/en-us/library/hh831568.aspx

[19] ——, "WMI .NET," 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms257340%28v=vs.80%29.aspx

[20] Desktop Management Task Force, "Common Information Model (CIM) Infrastructure Specification Version 2.7.0," Desktop Management Task Force, Inc., Tech. Rep., 2012.

[21] ——, "Web-Based Enterprise Management (WBEM)," 2014. [Online]. Available: http://www.dmtf.org/standards/wbem

[22] Microsoft, "Windows Management Instrumentation and the Common Information Model," 1998. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms811552.aspx

[23] M. A. Pervilä, "Using Nagios to monitor faults in a self-healing environment," in *Seminar on Self-Healing Systems*, 2007, pp. 1–6.

[24] Icinga Development Team and E. Galstad, "Icinga Documentation – Version 1.11," 2014. [Online]. Available: http://docs.icinga.org/latest/en/

[25] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," *RFC 4251*, 2006. [Online]. Available: https://www.ietf.org/rfc/rfc4251.txt

[26] E. Galstad, "NRPE Documentation," Nagios, Tech. Rep., 2007.

[27] Icinga, "Icinga Plugin API," 2014. [Online]. Available: http://docs.icinga.org/latest/en/pluginapi.html

[28] M. Medin, "NSClient++ external scripts howto," 2014. [Online]. Available: http://docs.nsclient.org/howto/external_scripts.html

[29] S. Talaat, *Windows PowerShell 4.0 for NET Developers*. Packt Publishing Ltd, 2014.

[30] J. Andersson, *Microsoft Exchange Server 2013 PowerShell Cookbook*. Packt Publishing Ltd, 2013.

[31] T. Kopczynski, P. Handley, and M. Shaw, *Windows PowerShell Unleashed*. Pearson Education, 2008.

[32] R. Siddaway, *PowerShell and WMI*. Manning, 2012.

**Karl van der Schyff (b. 1979)** received his undergraduate degree in 2004 from the University of South Africa. His research interests include information security, network monitoring and large-scale operating system deployment.

**Ralf C. Staudemeyer (b. 1973)** has a doctorate in computer science and more than 20 years of international experience in research, teaching and application. His areas of expertise include the planning, administration, protection and monitoring of modern networks. He is author of the "Nagios/Icinga Cookbuch". Currently, he is a globetrotter and scientist.