# Nagios Web Service Checker

## Table of content

# 1. Introduction

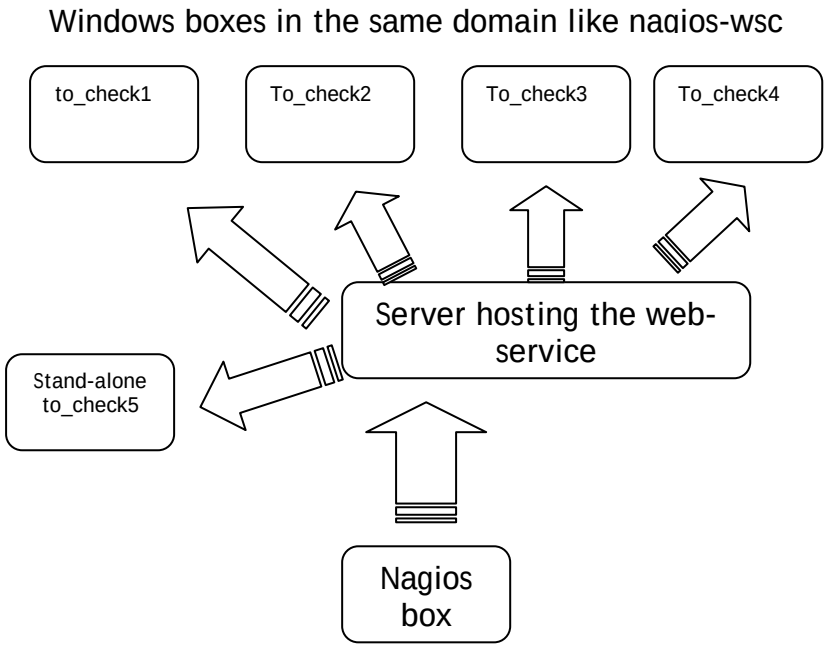
## *1.1. What is Nagios?*

Nagios is a open source, host, network and service monitor that runs under UNIX or Linux. For more information, please check *http://www.nagios.org/*

## *1.2. What is Nagios Web Service Checker?*

Nagios Web Service Checker (or nagios-wsc) is a utility I've written made up of two parts.

- The first part is a .NET based web service written in VB.NET that is capable of querying other windows machines on behalf of Nagios, using a Microsofts technology *Windows Management Instrumentation* (or WMI)
- The second part is a perl plug-in that allow Nagios to 'talk to' the Web Service and get the results back.




**Picture 1**

## *1.3. Why should I use it?*

To my knowledge, you would previously have had to install a piece of software or service on each Windows box you want to query. If you have lots of windows servers, then this can be pain to install on each of these.

Using nagios-wsc, you only need to install one piece of software on an IIS web server with the .Net framework installed - if you've got lots of windows servers, one of them somewhere will be able to run this software.

## *1.4. What do I need?*

To make best use of this software, you'll need:

- Nagios running on a UNIX/Linux box
- A windows machine running (at least) IIS 5 and.NET 1.1 framework installed.
- A username/password that will be allowed access to query the WMI information on each server.

# 2. Installing Nagios Web Service Checker

## *2.1. Installing the Webservice*

Go to *http://www.sourceforge.net/projects/nagios-wsc* and download the latest versions of nagios-wsc and nagios-wsc-plugin.

Firstly, we'll set up the web service:

- Extract the contents of the nagios-wsc.vn.n.zip file to a local directory.
- Create a virtual directory on the IIS server that points to the nagios folder just extracted - ensure that it is created as an application.
- Consider very carefully restricting access to this virtual directory because nagios-wsc can return lots of information, which will be very useful in the hands of the wrong person. Try to limit the access to this directory by denying access to anyone except the internal IP address of your querying host(s). Do this on the "Directory security" tab and enter appropriate details in the "IP Address and Domain Name restrictions"
- In order to use WMI to retrieve information from other servers, you'll need a valid username/password. You need to enter this in the web.config file (near the bottom) in the nagios folder.

## *2.2. Checking your installation of the web-service*

Use your favorite browser and navigate to the URL where the service is installed.

- Test the installation by using the following link (default URL): *http://name_of_your_server/nagios/service1.asmx*.
  With any luck, you'll get a list of methods (or things you can query) being displayed. (If this doesn't happen - report the problem in the forums on the *http://www.sourceforge.net/projects/nagios-wsc* project page and I'll try and help you out.)
- Choose the "disks" method. On the next screen, you enter "localhost" as server name and in the param field, type "1000,250". Finally, click the Invoke button. All being well you should get some XML that contains the result of the query.

That should conclude the setting up of the nagios-wsc web service. Now we need to set up Nagios to talk to it.

## *2.3. Installing the PERL-Script*

Put the check_wsc.pl file from the nagios-wsc-plugin zip file into the nagios libexec folder with the other plug-ins. You also can keep the perl-script in any other place. If you prefer to do so, you'll have to change the definition in Nagios checkcommand.cfg (see below).

## *2.4. Adding definitions for Nagios*

**Checkcommands.cfg**

Add the following lines in checkcommands.cfg: (command_line should be all on one single line)

```
define command {
      command_name check_wsc
      command_line $USER1$/check_wsc.pl -H $HOSTADDRESS$ -r
      server_hosting_.net_service -p $ARG2$ -t $ARG1$
      }
```

If you installed the perl script in a different location than the standard plug-ins then your definition should look like this:

```
define command {
      command_name check_wsc
      command_line <full_path_to_>/check_wsc.pl -H $HOSTADDRESS$ -r
      server_hosting_.net_service -p $ARG2$ -t $ARG1$
      }
```

**services.cfg**

Assume you have a host definition named "to_check1". Edit the services.cfg to make use this command just defined – example entries are show below:

```
Define_command {
      service_name      disks
      check_command check_wsc!disks!
      host_name   to_check1
      …
      }
```

This above entry will check all fixed, local disks on the machine "to_check1" for free disk space and warn if any have less than 1000MB free, or be critical if there is less than 250MB free. Details about the possible service checks see chapter 3 "Webservice check methods".

# 3. Webservice check methods

## *3.1. cpu*

This will compare the average cpu utilization and return a status based on the values entered. If you are running this check against a multiprocessor-system it will check the avarage load over all cpus.

```
check_wsc!cpu!"<warning_value>,<critical_value>"
```

All values are percent. If the avarage cpu load will rise above the corresponding values, the check will issue a warning or critical state.

## *3.2. disk*

This can be used the check a single disk for the used disk space.

**Form 1**

nagios-wsc will compare the available disk space of the disk specified with the values. If the disk has available disk space less than the warning value, it will return a warning. However, if the disk has available disk space less than the critical value, it will return a critical status.

```
check_wsc!disk!"drive_letter,warning_value,critical_value"
        values accept integer representing Megabytes
```

Note: values are in megabytes

**Form 2**

The second form is used to check the available disk space compared to the total one in percent.

```
check_wsc!disk!"drive_letter,warning_value%,critical_value%"
        values accept integer representing percent of total disk space
```

## *3.3. disks*

nagios-wsc will compare the available disk space of all local fixed disks with the values. If any drive has available disk space less than the warning value, it will return a warning. However, if any drive has available disk space less than the critical value, it will return a critical status.

**Form 1**

Values are integers representing Megabytes

```
check_wsc!disks!"warning_value,critical_value"
        e.g. disks!"1000,250"
```

**Form 2**

Values are integers representing percentage of the available space compared to the total disk space.

```
check_wsc!disks!"warning_value%,critical_value%"
        e.g. disks!"80%,90%"
```

**Form 3**

This form allows to define a list of disks which should be checked:

```
check_wsc!disks!"warning_value,critical_value,drive-letter:[|driveletter:]..."
        e.g. disks!10%,5%,D:|E:|F:|G:
```

This will check disks D,E,F,G. Warning threshold - 10% of a disk size, critical - 5% of a disk size. It can be used with the form for checking MB as.

## *3.4. generic_WMI*

WMI_Class: Specify a Win32_xxx class to query. For a full list, check out the Microsoft site here:
*http://msdn.microsoft.com/library/default.asp?url=/library/enus/wmisdk/wmi/i*

*nstalled_applications_classes.asp*. Select_Fields: if left blank, all fields are returned; otherwise input a commaseparated list of fields to return.

Where_clause: If left blank, all records are returned; otherwise specify a condition to apply to records nagios-wsc will check the win32 class entered and return the values required in as an xml document.

Example:

WMI_Class: win32_logicaldisk,

Select_fields: filesystem

Where_clause: drivetype=3

These parameters will return the filesystem of all local fixed disks

Note: Please take care with this as this proxy method is very powerful and can return lots of interesting information to anyone that is capable of invoking it. It will be useful to hackers if you have your nagios-wsc directory accessible to the internet.

To reduce this risk, there are two entries in the web.config file.

- "valid_callers" will specify IP addresses of hosts that can invoke this particular method (comma-separated)
- "valid_calls" specifies which WMI Classes can be invoked (again commaseparated )

## *3.5. memory*

This one checks memory and pagefile size. You can miss any parameters you want. For example, if you miss memwarning and memcritical, the method will check only virtual memory size.

```
check_wsc!Memory![“[memwarning], [memcritical], [vmemwarning],
        [vmemcritical])”]
```

- memwarning, memcritical - thresholds for physical memory
- vmemwarning, vmemcritical - thresholds for virtual memory

Values are integers representing MB or integers% representing percentage of total memory.

Example: 50%, 499, 10, 50%.

## *3.6. ntevent*

This is used to check the value which is returned by a special event entry.

```
check_wsc!ntevent!(eventtypes, logfiles, sourcenames, eventcodes,
        searchstring, period, type of returned state if found, type of returned
        state if not found
```

- eventtypes - list of EventTypes (integer) to query, for exampe: "1|2" or just "3".
- logfiles - list of LogFiles to query, for exampe: "Application|System" or just "Security"
- sourcenames - source to query, for example: Service Control Manager|eventlog
- eventcodes - event codes to query, for example: 51|7023
- searchstring - string for searching in event body
- period - period in minutes to query
- type of returned state if found - state returned if some events found with query, i - OK, w - Warning, c- Critical, default is "c"
- type of returned state if not found - state returned if no events found with query, "i"

You can miss any parameters.

Just for example, in my nagios I'm using this method for checking 37 event from Ntfs (User hit their quota limit):

```
check_wsc!ntevent!3,System,Ntfs,37,,5,w,i
```

## *3.7. ntevents*

The method checks the appearance of NT Events for defined period of time.

The methods return status depends on most important returned event. For example if there would be some event with type “Error” than the method will return Critical, for Warning the method return Warning.

```
check_wsc!ntevents!”(logfiles,sourcenames,eventcodes,period) “
```

- logfiles - list of LogFiles to query, for exampe: "Application|System" or just "Security"
- sourcenames - source to query, for example: Service Control Manager|eventlog
- eventcodes - event codes to query, for example: 51|7023
- period - period in minutes for which events will be queried, for example: 10

You can miss any parameters. For example, if you miss logfiles than the method will check all Log Files on remote system.

Sample: Common case, check for important events on Windows Server:

```
check_wsc!ntevents!System,Service Control Manager|NetBT|System
        Error|W32Time|Server|LDM|dmio|disk,,360
```

## *3.8. process*

The method checks the number of processes running.

**Form 1**

The warning or critical state is raised if the count of the specified process is below the configured values.

```
Check_wsc!process!warning,critical
```

**Form 2**

```
Check_wsc!process
```

Using this syntax, the method just checks whether the process exists on the remote system.

Example: check_wsc!process!svchost.exe!10,5
If svchost.exe is running at less than 10 times a warning state is raised, below 5 a critical state is issued.

## *3.9. services*

This will check all services listed and return their status. If any services specified are recognized and not running, then a critical state will be returned. Note: specifying a service that does not exist on the server should not cause an error and will not cause a warning or critical state.

Note: When entering names of services to check for, you need to use the “system name”, rather than the descriptive name.

```
check_wsc!services!ListOfServices
```

ListOfServices is a comma-separated list of services to be checked (e.g. “alerter, w32time”). “*” will check all services. This can be used for verifying the name of services.

For display options on the Nagios web-pages see section 4 “Config options in Web.Config”.

### *3.10. uptime*

nagios-wsc will attempt to determine how long the server has been up for and compare to the values entered. The state returned will change if the current value is lower than the configured values.

```
Check_wsc!uptime!“warning_value,critical_value”
```

The values accept integers representing hours. For instance configuring “24,5” will check the uptime of a machine and warn if it’s been up for less than 24 hours or be critical if it’s been up for less than 5 hours.

For display options on the Nagios web-pages see section 4 “Config options in Web.Config”

# 4. Config options in Web.Config

To grant access for nagios-wsc to the servers to be checked, you have to specify a user in the Web.Config file. All keys are located in the <appSettings>-section of </system.web>.

### *4.1. Default user*

Use the following keys to specify the user which will be used to get access to remote machines.

```
<add key="sysusername" value="wsc-test" />
<add key="syspassword" value="pwd" />
```

### *4.2. User for a specific server*

```
<add key="servername" value="user:password" />
        e.g. <add key="server_to_check1" value="UserAtServer:password" />
```

The name of the server must be written in the form which is sent during the request. It corresponds to the entry “server_name” in the hosts.cfg definition of Nagios.

### *4.3. Access for the generic WMI function*

These options define for which servers it is allowed to use the check “Generic WMI” as well as the parts of the WMI, which are allowed to be queried.

```
<add key="valid_callers" value="127.0.0.1" />
<add key="valid_calls" value="Win32_LogicalDisk,Win32_Service" />
```

„valid_callers“ accepts IP-Addresses (seperated by comma), „valid_calls“ a list of classes in the WMI which are permitted to be part of the query.

### *4.4. Behaviour of check “services”*

```
<add key="ServicesOptions" value="warnmissing,smart" />
```

Possible values:

- “warnmissing” results in check warning if you requested a service to be checked, which is not defined on the remote server.

- “smart” does not output running services in the result section of nagios. Only services in a warning or critical state are displayed.

### *4.5. Behaviour of check “uptime”*

```
<add key="UptimeOptions" value="smart" />
```

Possible values:

- “smart” – if state returned is OK only the current uptime is display in Nagios. Otherwise last boot time and local time on the server are displayed as well.

### *4.6. Commandline options of check_wsc.pl*

```
usage check_wsc.pl -H <hostaddress> -r <remote_wsc> -p <params> -t
        <servicetype> [-w <webservice>] [-v]
check_wsc (v1.0.1)
Options:

 --help
   Print detailed help screen
 -H, --hostname=STRING
   Hostaddress to be checked
 -r, --remote_wsc=STRING
   Address of the gateway serving the webservice
 -p, --parameter=<integer|string|list>
   Parameter for a specified service type
 -t, --servicetype=STRING
   Type of check to be executed
 -v, --verbose
   Detailed output of check-script
 -w, --webservice=STRING
   URL where the webservice can be found. Example: "/Service1.asmx".
   Default: "/nagios/service1.asmx".
 --timeout=integer
   Timeout in seconds waiting for response message from webservice
 --useragent=string
   String to be sent in http header as "User Agent"
```

## 5. Other Stuff

NAGIOS-WSC is a registered project at SourceForge.Net.

NAGIOS-WSC is published under GPL (general public license).

NAGIOS is a registered trademark of Ethan Galstad.