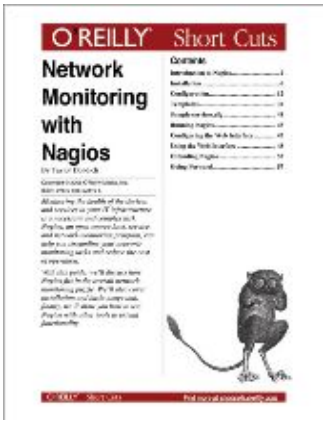
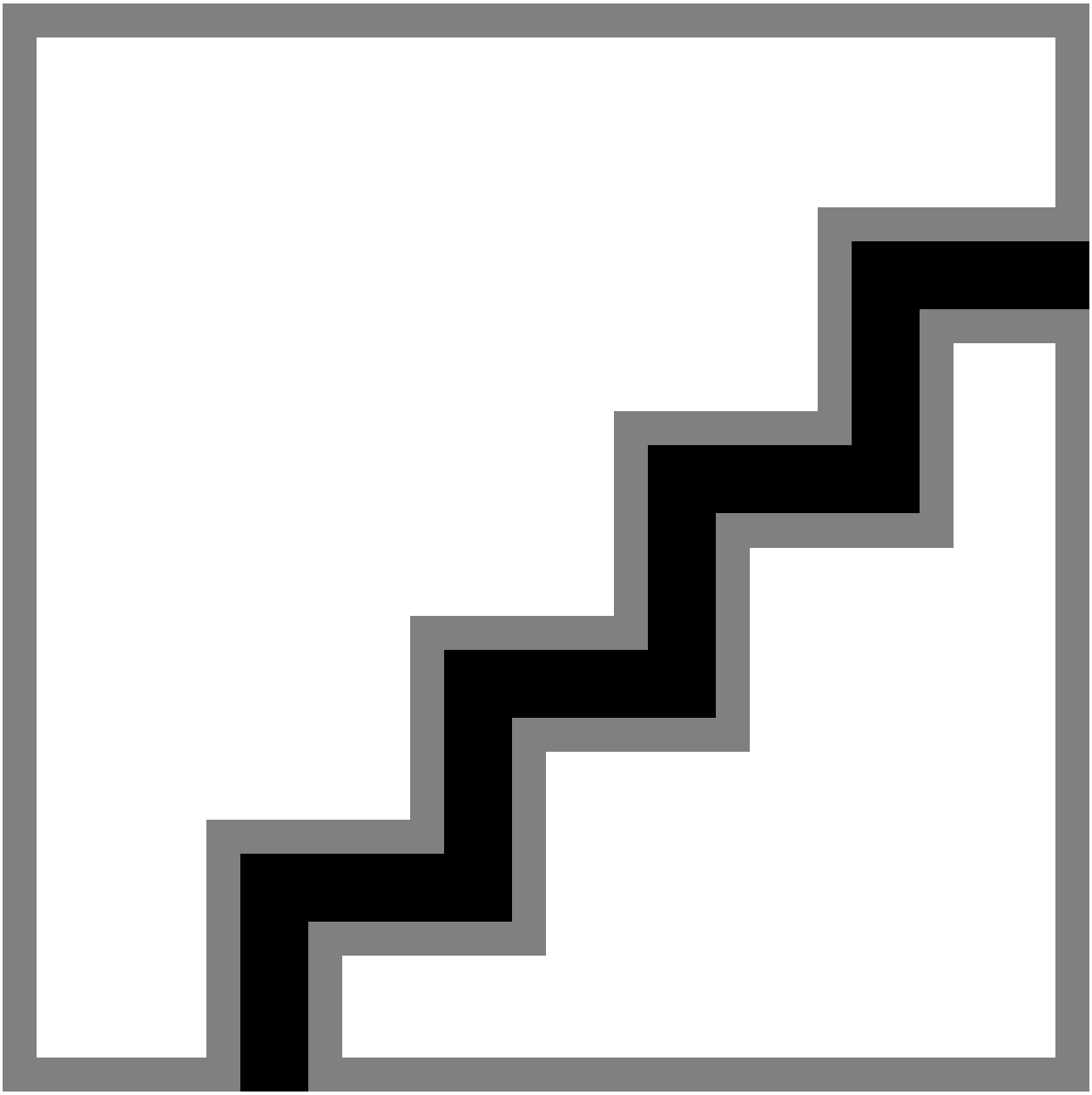

O'REILLY Short Cuts
Network Monitoring with Nagios
Contents:

Table of Contents

Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01



Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

Network Monitoring with Nagios

Monitoring your IT infrastructure has always been a difficult task. There are some common, commercial solutions; however, these offerings can get quite pricey. Furthermore, many organizations may only use a small subset of what those products offer. Many IT organizations have strong heterogeneous environments, with multiple hardware and software vendors running under the same roof so it's important to have a solution that is flexible enough to be able to monitor these ever-changing platforms. In addition to the addition, removal and modification of devices and services, a good monitoring system needs to adapt to process change. The personnel process that follows in reaction to a web server not responding may change over time. Therefore, the monitoring system needs to change to reflect that.

When using open source tools, the preferred methodology is to pick the best tools for specific jobs. The same is true for finding an open source monitoring solution. There are numerous open source IT management tools out there that excel in the tasks for which they were developed. For example, *rrdtool* is used to gather performance data from devices that support SNMP; *Cacti* is used to visually represent gathered performance data. When monitoring the *state* of devices and the functions they perform, *Nagios* stands as the forerunner in open source offerings.

Introduction to Nagios

State monitoring is the task of keeping track of the current status of network devices. A networking device's state could be in different phases such as being available or unreachable.

In order to understand how Nagios achieves state monitoring, it's important to look at key terms used with Nagios and to comprehend how Nagios uses them.

Plug-ins

Nagios by itself is actually quite small. Nagios's use of plug-ins provides the breadth of its functionality. A *plug-in* is an external application that Nagios calls to perform a specific task, such as checking the status of a host or service. Because there's a plethora of hardware and software out there, Nagios cannot possibly be written to support each type, so it relies on plug-ins to provide that support. Nagios has a standard plug-in library, which is maintained by the Nagios Plug-in Project. (**http://nagiosplug.sourceforge.net**) You can obtain other plug-ins written by other people as well. A good source for additional plug-ins is Nagios Exchange (**www.nagiosexchange.org**).

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

Checks

Nagios constantly needs to know the state of a host or service. The process is called a *check*; with Nagios, a check tends to come in two forms:

Active Check

Used when Nagios executes a plug-in to have it check a host or service. Depending on the response from the plug-in, Nagios updates the host or service's status information. Nagios performs active checks at regular intervals defined in the configuration files. When Nagios performs an Active Check, it needs to be able to directly communicate with the target device and potentially a port. Therefore, firewall rules must be in place to allow this communication.

Passive Check

Used when an external source communicates with Nagios to dictate new state information. For example, the NCSA add-on for Nagios gives remote hosts the ability to send updated information for their services to the NCSA daemon, which in turn notifies Nagios. Nagios has a command pipe that applications can use to submit a passive check.

Performance Note

Active checks are more CPU intensive than passive checks. As the number of active checks increase in your Nagios configuration, the more processing time it takes for your Nagios system to perform all the checks, especially if multiple active checks are set to perform at the same time. Passive checks, however, are less intensive and are quicker for the Nagios system to perform. The downside to passive checks is that you will need an agent on the target system to perform the check and then send the result to your Nagios system. Nagios's distributed monitoring setup is designed around this behavior.

Monitoring State

In Nagios, a host or service is always in a monitoring state. Monitoring states differ if you are dealing with a host or service. A host could be in any of the following monitoring states:

Up

It is best for your hosts to be in the Up state. If a host is Up, then Nagios can communicate with it, and the host is deemed okay.

Down

If the host fails to respond to Nagios, then it is placed in the Down state. If a host is Down, then all the services associated with that host are also in a Critical state. Needless to say, this is a bad thing.

Unreachable
: Unreachable is different from Down in that a host is Unreachable if the path from Nagios to the target Host is disrupted by another host in Down state. When a device is unreachable, a network outage could be the reason.

Pending
: A host is Pending if the scheduled first check for that host has not yet executed. Therefore, Nagios has no information regarding the status of this host.

Services have a little more granular detail in regards to their monitoring state. A service can be any of the following monitoring states:

Ok
: If a check of the service sees that there are no current and impending problems, then Nagios dictates that the service is OK. It is best for your services to be in Ok state.

Warning
: A service check can determine whether a service will fail if an action is not taken soon. For example, the disk usage of a server could be in Warning state if the disk usage is currently at 92 percent. It's still functional; however, if more space does not become available soon, it may become full. The threshold in which a service crosses to enter Warning state is defined in the configuration files.

Critical
: If the service fails, Nagios marks the service as being in Critical state. Immediate attention needs to be given to critical services to restore infrastructure functionality.

Unknown
: If the service check determines that it cannot communicate with the service and that it's not the service's fault, then Nagios dictates that the status is unknown. For example, if the service check was improperly configured, then the plug-in will bail out, which causes Nagios to interpret s the status as unknown.

Pending
: Like a host Pending state, a service is Pending if the first check scheduled for that service has not yet run.

Type of State

When Nagios receives a change of monitoring status from a check, it initially declares the state as *Soft*. Nagios does not alert on a Soft state because the check could have returned a failure, but the service could have recovered quickly, so it could have simply been a fluke. After the service has been in the same soft state after a configured amount of checks, Nagios will change the monitoring state of

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

the host or service into a *Hard* State. Nagios performs any reactive processes in Hard states.

Dependencies

Nagios has two types of dependencies:

Host dependency
: One host is dependent one another host in order to be reached by Nagios. If a host goes down that another host is dependent on, Nagios will see this event and will stop checking the dependent host until the broken host recovers.

Service dependency
: This operates in much the same way as the host dependency. Furthermore, when a service or host on which other hosts or services are dependent goes down, no notifications are sent in regards to the dependent hosts or services. Instead, the down or critical host or service only is used in notifications.

Notifications

Depending on your configuration, you may have Nagios notify you when various hosts or services go down or recover. Nagios can be configured to send notifications based on certain state criteria of hosts or services. Also, notifications can be sent out in various methods. A notification is actually handled by an external application, similar to how a plug-in operates. Notification handlers have been written to handle sending notifications by email, sms, VoIP, and other technologies. I'm still waiting for one to support carrier pigeon.

Acknowledgment

When a problem occurs with a host or service, a responsible party is hopefully notified via a notification attempt. Once notified, it's up to the responsible party to recognize the problem via Nagios's acknowledgment process. *Acknowledgment* usually occurs via Nagios's web interface. When an acknowledgment is paid, no further notifications are sent out until the problem is resolved. Acknowledgment is a great way to track progress of problems.

Comment

A *comment* can be attached to a host or service to describe additional information for that host or service. Comments are added through the Nagios web interface. When acknowledging a problem, it's common to also add a comment to describe the work that is being performed.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

Downtime

Sometimes, you simply have to take a host or service down for maintenance or other administrative purposes. During these times, you'd rather Nagios not send notifications of the host or service going down. Scheduling downtime for a host or service will tell Nagios to not send out any notifications of any state changes for that host or service.

Escalation

An *escalation* occurs when a problem with a host or service is not acknowledged after a configured number of notification attempts. For example, if a mail server goes down and Nagios notifies the responsible administrator, Nagios continues to notify the administrator until either the administrator acknowledges the problem or Nagios sends the maximum number of notifications. After that, the problem escalates. When this occurs, Nagios send notifications to the escalated contact. In this case, it's the administrator's manager who isn't too thrilled. The administrator will then be sacked. It's wise to not have your issues escalate; however, if your team requires escalation procedures, Nagios provides the functionality.

Installation

To get Nagios up and running, you initially need to download two packages. One is the Nagios core package, which includes the Nagios daemon and its web interface. The second is the Official Nagios Plug-ins. At the time of this writing, the core Nagios package is at version 2.5, and the Official Nagios Plug-ins package is at version 1.4.3. You can find links to both packages at Nagios's download page at **http://www.nagios.org/download/**. As an alternative to downloading and compiling manually, you can also install a package for your operating system, depending on your distribution, how many distributions you have, which distributions carry Nagios, and which do not carry the 2.x branch and are still using 1.x. This document assumes you are using Nagios 2.x. The rest of our installation steps will deal with downloading and compiling manually. We will start with the Nagios core package.

It is a wise idea to have Nagios operate as its own user. Furthermore, in order to use the web interface, Apache will also need to run with permissions to access some of Nagios's files. Therefore, the first step is to create our required users and set up our groups:

```
localhost:~ # groupadd nagios
localhost:~ # useradd -g nagios -d /usr/local/nagios -c "Nagios User" nagios
localhost:~ # usermod -G nagios www-data
```

In the preceding steps, the first thing we did was to add a new group to our system. The reason we want a new group is so we can also add our web server's user to the

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

group so it can read Nagios's various files for the web interface. Next, we create our ***nagios*** user, making it a member of our ***nagios*** group. Second, we add *www-data* to our *nagios* group. In our case, **www-data** is the user that Apache runs under; however, it may be different on your server.

After you've downloaded the source package, it's time to extract it:

```
localhost~# tar -zxvf nagios-2.5.tar.gz
```

Go into the directory in which the contents were extracted. In this case, it is the *nagios-2.5* directory. The first step is to configure the package by using the provided configure script. To see all available options, use the ***–help*** flag. Nagios provides enough customization to greatly change the dynamics of how it operates.

Common Nagios *configure* parameters

--enable-DEBUG(X)

Debugging information can be enabled, which presents more verbose information in the main ***nagios*** log file. The amount of verbosity goes up if you choose a higher number for X. The valid range is 0–5. If you use ***–enable-DEBUGALL***, Nagios will show all debugging information. Be warned, however. Enabling additional debugging information can slow down the nagios daemon greatly. When using Nagios in a production environment, it's best to not enable any debugging.

--enable-prefix=<path>

The prefix determines where Nagios is to be installed. Your operating system distribution may have a specific layout to which you may want to adhere. The default is */usr/local*, which means it will install everything into */usr/local/nagios*.

--enable-event-broker

Perhaps one of the biggest additions to the Nagios 2 codeline is that of the the Event Broker; however, it is still optional by default. Enabling the Event Broker will give you the capability to plug-in third-party libraries into the Nagios core daemon and extend it's functionality. The event broker is an advanced topic and not needed for basic functionality.

--with-nagios-user=<user>

Nagios should run as a different user other than root. The value for this parameter will specify what user the nagios daemon should run as. This should match the user you created in the previous steps.

--with-nagios-group=<group>

Nagios should run in a different group other than root as well. The value for this parameter specifies what group the nagios daemon should run under. This should match the group you created in the previous steps.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

As stated before, there are additional customizations you can make to the Nagios system. The majority of the options can be figured out by the configure script's help parameter. For our basic installation, we'll accept most of the defaults.

```
localhost:/root/nagios-2.5 # ./configure -with-nagios-user=nagios \
                                         -with-nagios-group=nagios
```

The configure script determines the environment and prepares the process for building. Once completed, you'll be able to use the *make* command to build and deploy Nagios's various components. Nagios has a few *make* targets to deploy only the components you want; however, in our case, we'll want to deploy everything.

```
localhost:/root/nagios-2.5 # make all
localhost:/root/nagios-2.5 # make install
localhost:/root/nagios-2.5 # make install-config
localhost:/root/nagios-2.5 # make install-commandmode
localhost:/root/nagios-2.5 # make install-init
```

The *all* target will compile all the binaries for Nagios. Afterwards, we use the *install* target to deploy the binaries and documentation to the */usr/local/nagios* directory, and then we use the *install-config* target to deploy sample configuration files for Nagios into the */usr/local/nagios/etc* directory. (We'll learn more on how to customize our configuration later.) Next, we use *install-commandmode* to create and configure permissions on the directory that will hold the command file for Nagios to process commands sent from the web interface and from other sources. Finally, we use *install-init* to install the init script into our startup so we can easily start and stop Nagios.

Once those steps are done, Nagios is installed. However, we have a second part we need to install: the Nagios Plug-in Library. First, we must extract the archive, just like we did for the Nagios core package:

```
localhost:~ # tar -zxvf nagios-plugins-1.4.3.tar.gz
```

Once extracted, we enter the directory *nagios-plugins-1.4.3* and run the configure script. Just like the Nagios package, the configure script has plenty of customization parameters, which will try to force various plug-ins to be built. However, the configure script will, by default, attempt to find any libraries that a plug-in needs and, if any are found, will build that plug-in. As a result, if a plug-in that you expected to be built was not, it's possible the configure script was unable to find a dependent library, and you will need to specify the location manually. For our purposes, we will let the configure script try to determine which plug-ins to build. Later, we will look at some of the most commonly used plug-ins, their purpose, and how to use them.

```
localhost:/root/nagios-plugins-1.4.3 # ./configure
localhost:/root/nagios-plugins-1.4.3 # make
localhost:/root/nagios-plugins-1.4.3 # make install
```

We accepted the defaults by just using the configure script with no parameters The first *make* command will build all the plug-ins that the configure script determined

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

could be built. The *install make* target will deploy the plug-ins to */usr/local/nagios/libexec*. Our plug-ins are now installed and ready to go, but I will first describe some of them in order for you to understand what they do.

Introducing The Standard Nagios Plug-ins

If Nagios itself had the ability to monitor any device in existence, it would become seriously bloated. Since Nagios cannot predict how to monitor any device that you might have, it leaves that responsibility to its available plug-ins. A *plug-in* is a small script or program that takes a small set of parameters and then checks a remote device according to the instructions in the parameters. Based on what type of response it gets from the device, it will return a response code to its caller. Nagios uses its set of plug-ins to achieve the ability to monitor a wide variety of devices. Future device support does not have to be built into Nagios. Only a plug-in that has support for that device is required. Plug-ins can easily be written in any language, as long as it follows the plug-in guidelines from the Nagios Plug-in Project (available at **http://nagiosplug.sourceforge.net**).

The plug-ins that we built and installed earlier were installed in */usr/local/nagios/libexec*. In this directory, you will see a list of all the available plug-ins you can use. You will also learn that some plug-ins are actually symbolic links to other plug-ins, such as *check_tcp*, which can get quite confusing. Furthermore, in order to use these plug-ins in your Nagios configuration, you will need to know the syntax to invoke the plug-in. However, these plug-ins follow some standards, which helps figure out their purpose and their usage. For example, if we want to know what the plug-in *check_by_ssh* does and how to use it, we would invoke the *–h* flag.

```
localhost:/usr/local/nagios/libexec # ./check_by_ssh -h
check_by_ssh (nagios-plugins 1.4.3) 1.37
Copyright (c) 1999 Karl DeBisschop <kdebisschop@users.sourceforge.net>
Copyright (c) 2000-2004 Nagios Plugin Development Team
        <nagiosplug-devel@lists.sourceforge.net>

This plugin uses SSH to execute commands on a remote host


Usage: check_by_ssh [-f46] [-t timeout] [-i identity] [-l user] -H <host> -C <command>
                  [-n name] [-s servicelist] [-O outputfile] [-p port]

Options:
 -h, --help
    Print detailed help screen
<<output truncated>>
```

By using the *–h* flag, we get to see what the version is and also how to use it. It shows us the basic syntax and what each flag means. Every plug-in in the standard Nagios plug-in library follows the same basic syntax, so you will see a good deal

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

of overlap. You can use the *–h* flag on each of the plug-ins to see what it does and how it functions. As for the most common plug-ins, let's take a look at them now.

Common Nagios plug-ins

check_icmp
: The *check_icmp* plug-in can perform ICMP network operations against a target host. The most common use of the *check_icmp* plug-in is to use it to PING a remote host to check for availability.

check_smtp
: Mail servers use SMTP to process incoming messages. The *check_smtp* plug-in checks to see whether SMTP is properly responding to requests by performing a simple SMTP handshake and then expecting a proper response.

check_pop, check_imap
: POP and IMAP protocols are used for mail servers when retrieving mail. The *check_pop* and *check_imap* plug-ins attempt to perform handshakes with these protocols against a target host.

check_ftp
: The *check_ftp* plug-in attempts to open an FTP connection to a target host. By default, it expects a 220 response from the ftp server.

check_http
: The *check_http* plug-in can check both regular and secure web servers. The plug-in can be customized to check not only the status of the web server, but also the content that is returned by the web server. In this situation, it can be used to perform simple web application checking.

check_dns, check_dig
: DNS can be checked using two methods. The *check_dns* plug-in uses the *nslookup* command to perform lookups on hostnames. The *check_dig* plug-in can query for any type of record.

check_ssh
: Remote access to servers is now commonly provided by ssh. The *check_ssh* plug-in queries the remote server to verify that the ssh software is responding. It can optionally check to verify what version of ssh is running as well.

check_mysql, check_pgsql, check_oracle
: Databases tend to power a great deal of infrastructure; therefore, monitoring the performance of these databases is important. The database-related plug-ins (check_mysql, check_pgsql, and check_oracle) can check a local or remote database by attempting to log into it.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

check_ldap
: Many directory services speak LDAP, including Novell E-Directory and Microsoft Active Directory. The *check_ldap* plug-in can check the connection to the LDAP server and also look for specific domain names.

check_dhcp
: The *check_dhcp* plug-in can check a DHCP server. It broadcasts a DHCP-DISCOVER request to the target server and waits for the DHCPOFFER reply.

check_tcp
: Refrigerators now come in models that can speak TCP. Being able to support TCP means to also be able to be monitored by Nagios. The *check_tcp* plug-in is a generic plug-in that communicates over TCP to a device and performs various TCP communications. If a plug-in does not exist for a type of device but speaks TCP, *check_tcp* can try and communicate with it.

check_udp
: For those devices that do not speak TCP but do speak UDP, the *check_udp* plug-in is available. As with *check_tcp*, it has an abundance of configuration parameters to determine behavior.

check_disk
: Disk space is an important commodity on severs that provide database, file, and web services. The *check_disk* plug-in can determine whether a disk is running out of space or is in risk of doing so. By itself, it can only check the disks on the local Nagios server; however, used with the *check_by_ssh* plug-in, it can be used to check remote servers. The *check_by_ssh* plug-in is described below.

check_swap
: The swap space is important on a server because if it fills up, there truly is no more memory for applications to use. The *check_swap* can detect when swap space is reaching critical levels. This plug-in can also be used with the *check_by_ssh* plug-in to check remote hosts.

check_load
: The system load of a host can tell you whether irregular behavior is suddenly eating up CPU time. It determines this result by using the *uptime* system command, which contains load averages over a period of time.

check_users
: The *check_users* plug-in can check to see how many users are logged into a host. You can set thresholds to determine whether there are too many people logged in.

check_ntp
: The *check_ntp* plug-in can determine whether a time server is properly responding to NTP requests, and whether the time it is providing is correct, based on the time of the local Nagios server.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

check_by_ssh

Some of the plug-ins described in this list can only, by themselves, check the resources on the local server, such as *check_load*. However, if the *check_by_ssh* plug-in is used, the *check_load* plug-in can be operated remotely. c*heck_by_ssh* can operate plug-ins remotely. One caveat is that the plug-in must be installed on the remote box. In addition, because *check_by_ssh* runs over SSH, there is additional security. For more secure environments, I suggest installing the required plug-ins on remote hosts and always using *check_by_ssh*.

Plug-ins are used in the Nagios configuration file as *command* definitions. However, there is a great deal more to Nagios configuration than just specifying commands.

Configuration

Configuration of Nagios is done through flat-text files, so adding devices or changing the configuration of existing parameters can become a tedious task. On top of the time it takes, if you make one mistake, Nagios could more than likely fail to start altogether. Therefore, it's very important that you are careful when modifying your configuration files and that you make backups after each successful change. Believe me; you don't want to rush through it.

The configuration of the Nagios daemon is separated into three major parts. There is the main configuration file (usually named *nagios.cfg*), which dictates how Nagios should operate overall. Next is the Resource File(s), which is referred to inside the main configuration file that contains user-defined macros. Macros will be covered later on; however, it's important to know that you should be storing sensitive configuration information (such as database connection parameters) in these files. Lastly, object definition files provide Nagios with a description of what you have in your infrastructure and how to monitor it. These files contain the potential for quite a lot of directives. We will only go over what is needed to get you up and running with minimal setup. Once you are comfortable and understand this configuration, you should refer to Nagios's documentation to see the full list of directives available to you. By using additional directives, you can have more granular control over how Nagios operates.

Before you run away in terror, thinking you have to write these things by hand, know that Nagios provides sample configuration files when you build and install it. Following the directions in the previous section, you should have a directory, */usr/local/nagios/etc/*, which contains these sample files. You can use these files as reference or start writing your own. We'll look at each of the configuration files and what they define. Afterwards, I'll describe a fictional network and have a set of Nagios configuration files that monitor it.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

The Main NagiosConfiguration File Object

The main Nagios configuration file is usually called *nagios.cfg*. It contains parameters that define the basic behavior of the Nagios monitoring system. These parameters include everything from how often things should be performed to what features are enabled or disabled. When we installed Nagios, a sample configuration file was copied to */usr/local/nagios/etc/nagios.cfg-sample*. Use this file either as a sample *nagios.cfg* file or as a reference. Let's go over some of the common parameters in this file.

Important main configuration file parameters

log_file

Nagios constantly writes information regarding events to the *log_file*. This file is important because it's one of the biggest clues to the parse errors in your configuration files. Specifying this directive first makes Nagios start writing output to it immediately. This file can get quite large depending on the options that you've configured with Nagios; therefore, it's wise to also have *log_rotation_method* enabled (described next).

log_rotation_method

Depending on how large your monitoring configuration is, the *log_file* for Nagios can become quite large pretty quickly; therefore, it's a good idea to have a policy to rotate the log file at regular intervals. The *log_rotation_method* determines that policy. The value is a single character that represents the following policies:

n

This means never rotate the log. Unless you have some other process rotating the log, this is never a good choice. Rotate your logs. They can get big in a hurry.

h

This means rotate the log hourly. Something this aggressive should only be used for large installations.

d

This means Rotate the logs daily, which tends to be a safe bet, and I usually suggest this rotation method.

w

This means rotate the log every week.

m

This means rotate the log every month.

cfg_file

The main configuration file should have references to all your object configuration files. These files can be referenced with either *cfg_file* statements,

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

which refer to the direct path of the file, or with *cfg_dir* statements, described next. Nagios parses each of these files in order. You can either have your entire object configuration in one file or separated into multiple files for organization purposes (which is the suggested method).

cfg_dir

This is ,uch like the *cfg_file* directive. *cfg_dir*, however, takes a path to a directory. Nagios recursively goes through the directory and parses each file with a *.cfg* extension as an object configuration file. Using this directive is easier than *cfg_file*, since you can add new configuration files anytime you want without modifying your main configuration file.

nagios_user

As a security measure, Nagios should not run as root. One reason is that when Nagios executes a plug-in to perform a check or notification, it runs the plug-in as the same user. If a security flaw is in any of these plug-ins, it could be trouble. The *nagios_user* value specifies which user Nagios should run under. The value defaults to the value that you passed to Nagios's configure script; however, you may need to change this value in the future.

nagios_group

Following the same reasoning as for the purpose of *nagios_user*, it's important to specify a restricted group that Nagios should run under as well.

check_external_commands

You can use the web interface to perform checks, acknowledge problems, and change runtime configuration information. The *check_external_commands* parameter enables that feature. Set it to 1 if you want it to be active.

command_check_interval

If *check_external_commands* is enabled, Nagios polls the command file at regular intervals to see whether new commands have been sent. The *command_check_interval* specifies that interval. The default is 60 seconds, which is a safe default; however, if you want it to be aggressive, then you should change the value to something shorter. If you set it to -1, Nagios attempts to check the command file as often as possible.

retain_state_information

If *retain_state_information* is enabled with a value of 1, then Nagios retains the state information regarding hosts and services between restarts.

retention_update_interval

If you are retaining state information, Nagios periodically saves the state of hosts and services to an external file. The *retention_update_interval* determines at what interval Nagios performs that operation.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

enable_flap_detection

If *enable_flap_detection* is enabled with a value of 1, Nagios determines whether a host or service is flapping. When a host or service starts flapping, Nagios suppresses notifications during that time. Nagios tries to do a decent job of flap detection, but I don't rely on it. If you choose to, look up the various flap parameters in the Nagios documentation to determine how to fine-tune flap detection.

The preceding parameters are the most important parameters that come to mind; however, there are quite a lot more. By going through the sample *nagios.cfg* file provided, you can see sample values and descriptions of other parameters as well. As you will learn, you have a great deal of control in regards to how Nagios operates.

Nagios Resources

Nagios Resources are macros that are available to command object definitions. You'll normally store values that do not change often, such as path names. Also, for security reasons, you may wish to store security credentials in here instead of in the command object definitions. The web interface can potentially show the command definition. The web interface, however, cannot view the resource file. A macro is in the form of:

```
$USER#$=<value>
```

Where # is a number, 1 – 32, and the <*value*> is the value of the macro. Due to limitations in the code, you can have a maximum of 32 user macros. In the sample resource file, you'll see one macro is already defined, *$USER1$*. This macro points to the path that the plug-ins reside in. The use of this value as *$USER1$* is a good example of a macro because you may choose to move the plug-ins elsewhere in the filesystem. When doing so, you have only to update the macro's value in the resource file, instead of modifying each command definition. Usage of macros is further demonstrated when describing command objects, which is just one of the types of objects that Nagios supports. Refer to the command object definition in the following section.

Nagios Objects

In Nagios, every monitoring concept can be described as an object. When you are writing the configuration files, you're describing a multitude of objects and their relationship to each other. Each type of object has a different set of parameters that describes not only how Nagios can communicate to it, but also determines the behavior Nagios should take when dealing with that object.

The object definitions are placed in text files referenced by the *cfg_dir* or *cfg_file* parameters in your main configuration file. We'll look at samples of each of these

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

object definitions and describe the parameters required for each object. There may be numerous other parameters available for these objects to provide even more functionality. To know more about every parameter that Nagios provides, refer to the Nagios documentation.

An object definition usually follows this pattern:

```
Define object-type {
       parameter     value
       parameter     value
       parameter     value
       ….
}
```

Let's cover the object types that Nagios supports next.

The timeperiod Object

Nagios uses *timeperiods* to determine what time it should perform certain events, including checking hosts and sending out notifications. In order to do so, Nagios must know what times during each day of the week to perform these tasks. Time periods are somewhat limiting in that you can only define the weekly schedule; therefore, special cases such as holidays cannot be defined in this fashion.

Example timeperiod

```
define timeperiod {
       timeperiod_name     24x7
       alias               24 Hours A Day, 7 Days A Week
       sunday              00:00-24:00
       monday              00:00-24:00
       tuesday             00:00-24:00
       wednesday           00:00-24:00
       thursday            00:00-24:00
       friday              00:00-24:00
       saturday            00:00-24:00
}
```

Common timeperiod parameters

timeperiod_name
: When other objects need to refer to a timeperiod, it uses the timeperiod's *timeperiod_name*. The name should contain no spaces and be unique from all other timeperiod definitions.

alias
: Wherever this timeperiod is referenced in the web interface, the *alias* will be used as a description.

sunday,monday,tuesday,wednesday,thursday,friday,saturday
: Each day is defined in the timeperiod, with each day's values as the time ranges in which this timeperiod is effective. In the previous example, we made the

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

entire day active for this timeperiod. If you want multiple ranges in a day, a comma-separated list is used. For example:

```
monday        09:00-12:00,13:00-17:00
```

This timeperiod's day value states that on Monday, this *timeperiod* will be active from 9:00 a.m. to noon. The timeperiod is not active during the time between noon and 1:00 p.m.; afterwards, the timeperiod is active from 1:00 p.m. to 5:00 p.m. The time ranges use military time, so be careful. Saying 01:00-04:00 means 1:00 a.m. to 4:00 a.m., and I'm fairly sure your boss would rather not be woken up at that time.

The command Object

Nagios uses external programs to perform the actual checking of hosts and services; furthermore, Nagios also uses external programs to perform notifications. It's because of this functionality that Nagios is a very powerful tool. However, each external program may have different ways of invoking it. Therefore, *command* objects define what commands are available for Nagios to use for checks and notifications. The path to the command and the parameters to pass are provided.

Example command

```
define command {
       command_name  check_ping
       command_line  $USER1$/check_ping –H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$ -p 5
}
```

Common command parameters

command_name

The *command_name* is the short name used by other Nagios object definitions to refer to this command. It's much easier to use a command's short name than to have to type the entire command line of the external program for each object definition.

command_line

The *command_line* is the path to the external program, along with the parameters to pass to the program. You'll notice the use of macros here. The *$USER*$* macros are defined in your resource configuration file. The *$HOSTADDRESS$* and *$ARG*$* macros are provided to this object, depending on the context it's used. For example, if this command is referenced in a host object, then the *$HOSTADDRESS$* macro would be the hosts network address. Other macros are available based in the context that this command is used.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

More Information About Macros

Macros are used in command definitions to pass context-based information to the command line. If a command is used when checking a host, various macros become available to use in the command line. If used when performing a service check, other macros become available. The list of available macros is lengthy; however, the official Nagios documentation has a great table matrix that describes what macros are available in what context. This matrix can be found online at **http://nagios.sourceforge.net/docs/2_0/macros.html**.

The contact Object

Nagios considers anyone it needs to get in touch with as a contact, whether it's human or yet another machine. Every contact definition has directions for when it should be used as a contact and by what process it should it to notify the contact.

```
define contact {
       contact_name        nagios-admin
       alias               Nagios Admin
       service_notification_period      24x7
       host_notification_period         24x7
       service_notification_options     w,u,c,r
       host_notification_options        d,r
       service_notification_commands    notify-by-email
       host_notification_commands       host-notify-by-email
       email                            nagios-admin@localhost
}
```

Common contact Parameters

contact_name
: When other objects need to refer to a contact, it uses the contact's *contact_name*. It should contain no spaces and be unique from all other contact definitions.

alias
: Wherever this contact is referenced in the web interface, the *alias* is used as a description.

service_notification_period
: Whenever a notification needs to be sent out regarding a service, it first checks whether the event occurred inside the *service_notification*_period. The *timeperiod_name* parameter of the timeperiod you wish to use goes here.

host_notification_period
: As with *service_notification_period*, this parameter determines when this contact can be used for notifications; however, this is for host-related events.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

service_notification_options
: The notification options for services determine what type of events will trigger a notification for this contact. A notification will only be sent out to a contact when one of these events occurs on a service for which this contact is listening. . Character letters determine which events:

  w
  : Notify when the service enters a Warning state

  u
  : Notify when the service enters an Unknown state

  c
  : Notify when the service enters a Critical state

  r
  : Notify when the service enters Recovery state

  f
  : Notify when the service starts and stops flapping

host_notification_options
: Similar to *service_notification_options* but related to host events. A comma-separated list determines which events will trigger a notification:

  d
  : Notify when the host enters a Down state

  u
  : Notify when the host enters an Unreachable state

  r
  : Notify when the host enters an Up state (recovery)

  f
  : Notify when the host starts and stops flapping

service_notification_commands
: If a contact is listening in regards to a specific service during the *service_notification_period*, and the service enters a state determined by *service_notification_options*, then Nagios executes each of the commands listed in *service_notification_commands*. Each command is separated by a comma.

host_notification_commands
: Similar to *service_notification_commands*; however, it is used during host events.

email
: Email tends to still be the most popular way for monitoring systems to notify. The *email* parameter determines what email address to send notifications to in

regards to this contact. The *email* parameter will be available to the command via the *$CONTACTEMAIL$* macro.

The contactgroup Object

Contacts can be bundled together to form organizational groups. Contact groups can send notifications to a group of responsible people instead of maintaining lists for each contact individually.

```
define contactgroup {
       contactgroup_name   admins
       alias               Nagios Administrators
       members             nagios-admin
}
```

Common contactgroup parameters

contactgroup_name

When other objects need to refer to a contact group, it uses the contactgroup's *contactgroup_name*. This name should contain no spaces and be unique from all other contactgroup definitions.

alias

The alias is the human-readable name of this contactgroup. This name is shown in the web interface.

members

A comma-separated list of contact's short names is used to identify who belongs to this contact group.

The host Object

Each monitored device is a host to Nagios. A host could be a server, switch, or any other device in which Nagios has a plug-in to support.

```
define host {
       host_name                localhost
       alias                    localhost
       address                  127.0.0.1
       parents                  router
       hostgroups               servers
       check_command            check-host-alive
       max_check_attempts       3
       check_period             24x7
       contact_groups           admins
       notification_interval    120
       notification_period      24x7
       notification_options     d,u,r,f
}
```


Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

Common host parameters

host_name

The hostname is shown in the web interface and is also used in other object definitions where a host is expected. The *host_name* tends to be the defined hostname for the host or, in some cases, the hostname along with the fully qualified domain name, such as mailsrv1.example.com.

alias

The *alias* is the human readable name of the host. The alias may be a more descriptive name for the host, such as *Mail Server for California Office*.

address

The *address* is used by plug-ins to contact the host. The address may be an IPv4 address, or it could be an IPX address. The value of *address* becomes the *$HOSTADDRESS$* macro when used with commands for this host.

parents

A comma-separated list of hosts that are located between Nagios and the host. These hosts are usually network devices such as switches and routers, which are needed in order for Nagios to communicate with this host. If all the parents are not reachable via Nagios, then this host will be in an Unreachable state.

hostgroups

The *hostgroups* parameter is a comma-separated list of hostgroups to which this host belongs. Hostgroups are described later on.

check_command

The *check_command* is the command used to check the status of this host. (A list of typical plug-ins and whether they're suitable for hosts, services, or notifications is shown later.) If you do not want to check this host (and by doing so, it will always remain in an UP state), leave the *check_command* blank.

max_check_attempts

If Nagios checks the host, and it returns something other than an UP state, Nagios continues to check the number of *max_check_attempts*. If the host does recover within the number of *max_check_attempts*, Nagios sends an alert for this host.

check_period

The *check_period* is the timeperiod in which Nagios actively checks this host. However, a passive check can still be given to Nagios to change the status of this host outside the *check_period*.

contact_groups

A comma-separated list of contact groups is used to define which group of contacts will be notified when an alert is sent out in regards to this host.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

notification_interval

The *notification_interval* is used to determine how often Nagios will continue sending out notifications when a problem with a host has not been resolved. It is used in conjunction with the *interval_length* parameter in your main Nagios configuration file. For example, if the *interval_length* has the default value of 60, and the *notification_interval* is 5, then Nagios will send out new notifications every 5 minutes (5 * 60 seconds).

notification_period

The *notification_period* is the timeperiod in which it is acceptable for Nagios to send out notifications in regards to this host. If a problem occurs,, or a host recovers, and this happens outside the scope of the time period specified here, a notification is not sent out.

notification_options

This list is used to determine which events about a host will trigger notifications to be sent out. It is a comma-separated list of characters that represent each of the states. If any of the following states occur and it's in this list, then a notification will be sent out.

d

Send notifications when the host enters a Down state.

u

Send notifications when the host enters an Unreachable state.

r

Send notifications when the host recovers, going back into an OK state.

f

Send notifications when the host begins flapping.

n

No notifications will be sent out, regardless of state.

The service Object

The monitored aspects of a host are defined as a service to Nagios. A *service* could represent an actual service for other hosts, such as file sharing, mail, and web services. It can also represent various aspects of the host, such as free disk space, memory usage, and the performance of its network interfaces. Again, as long as there's a plug-in that can monitor what you want on a particular host, it can be represented as a service.

```
define service {
       host_name                 localhost
       service_description       PING
       check_command             check_ping!100.0,20%!500.0,60%
       max_check_attempts        3
```

```
        normal_check_interval       5
        retry_check_interval        1
        check_period                24x7
        notification_interval       120
        notification_period         24x7
        notification_options        w,u,c,r,f
        contact_groups              localadmin
}
```

Common service parameters

host_name

The *host_name* is the *shortname* of the host in which this service will reside. It must match that of a host defined elsewhere in your Nagios configuration.

service_description

The *service_description* is a human-readable description of the service. Unlike other places, spaces are allowed here. The description can be as short as you'd like, such as *PING*, or as descriptive, such as *Host availability*. This description will be shown in the web interface when referring to this service.

check_command

The *check_command* specifies which command will be used to check on this service. In the earlier sample definition, you'll notice that the command not only has the command name of a command you defined, but also additional arguments, separated by the ! character. These arguments are available as the *$ARG1$*, *ARG2$* (and so on) macros in your command definitions. For more information on macros and how they are used in command definitions, refer to the command object described previously.

max_check_attempts

If Nagios checks the service and it returns something other than an OK state, Nagios continues to check the number of *max_check_attempts*. If the service does recover within the number of *max_check_attempts*, Nagios will send an alert for this service.

normal_check_interval

The *normal_check_interval* is the interval in timeperiods that is used when Nagios regularly checks the service. It uses this interval whenever the service is in an OK state, or if there is a problem with the service and it has expended the *max_check_attempts* value. The interval works in conjunction with the *interval_length* value that you defined in your main Nagios configuration file.

retry_check_interval

When a service is taken out of an OK state, it attempts to check the service *max_check_attempts* number of times. Each of those attempts is performed at the interval defined by *retry_check_interval*. This action is helpful because when a problem occurs, you will probably want to check the service more aggressively until it recovers.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

check_period
: If Nagios is configured to actively check this service, the *check_period* defines when the active checks should be performed. However, a passive check can still be submitted outside this period.

notification_interval
: The *notification_interval* is used to determine how often Nagios sends out notifications when a problem with this service has not been resolved. The *notification_interval* is used in conjunction with the *interval_length* parameter in your main Nagios configuration file. For example, if the *interval_length* has the default value of 60 and the *notification_interval* is 5, then Nagios will send out new notifications every 5 minutes (5 * 60 seconds).

notification_period
: The *notification_period* is the timeperiod in which it is acceptable for Nagios to send out notifications in regards to this service. If a problem occurs, or this service recovers and it happens outside the scope of the timeperiod specified here, a notification is not sent out.

notification_options
: This list is used to determine for which events this service will trigger notifications to be sent out. It is a comma-separated list of characters that represent each of the states. If any of the following states occur and is in this list, then a notification is sent out.

  w
  : Send a notification when this service enters a Warning state.

  u
  : Send a notification when this service enters an Unknown state.

  c
  : Send a notification when this service enters a Critical state.

  r
  : Send a notification when this service recovers from a problem back into an OK state.

  f
  : Send a notification when this service begins flapping.

  n
  : Never send notifications in regards to this service.

contact_groups
: A comma-separated list of contact groups is used to define which group of contacts will be notified when an alert is sent out in regards to this service.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

The hostgroup Object

You can group together multiple hosts to form organization to match your organization. A *hostgroup* could cluster together multiple hosts based on region, location, or network topology. In the web interface, hostgroups are used to provide a higher view level of the overall health of these groupings.

```
define hostgroup {
       hostgroup_name      servers
       alias               My Servers
       members             localhost
}
```

hostgroup Parameters

hostgroup_name

When other objects need to refer to a host group, it uses the host group's *hostgroup_name*. It should contain no spaces and be unique from all other hostgroup definitions.

alias

The *alias* is used in the Web Interface to describe this hostgroup.

members

The collection of hosts which belong to this hostgroup is defined in the *members* parameter. It is a comma-seperated list with each member being the *shortname* of the host.

The *servicegroup* Object

New in Nagios 2.x, you can now group services together, like hostgroups. So services which are on multiple hosts but all provide mail services, for example, could be grouped together into a working group. The service groups are shown in the web interface to view overall statistics regarding the collection of services.

```
define servicegroup {
       servicegroup_name   mailservices
       alias               Mail Services
       members             localhost,smtp,localhost,pop3
}
```

Common servicegroup parameters

servicegroup_name

When other objects need to refer to a service group, it uses the service group's *servicegroup_name*. It should contain no spaces and be unique from all other servicegroup definitions.

alias

The *alias* is used in the web interface to describe this servicegroup.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

members

The collection of services that belong to this servicegroup is defined in the *members* parameter. The list of members is grouped with the host on which the service resides and with the service description. Groups are separated by semi-colons. The host and service descriptions are separated by semi-colons as well, so make sure you have both the host and the service description in each grouping.

The hostdependency Object

In your network topology, it may be common for a switch to be in between your Nagios server and a monitored device. In this scenario, you could say that Nagios is dependent on that switch to be operational in order to monitor the remote device. Depending on the role of that remote device, you may want to suppress future checks until the switch recovers. If the device is mission-critical, however, you may want to continue sending notifications regarding that device being unreachable. Another example could be when you are monitoring a remote office. If the remote router at the office fails, it is usually best to suppress notifications and checks on any remote monitored devices at that office. A *hostdependency* object sets up these types of relationships.

```
define hostdependency {
       host_name                       localhost
       dependent_host_name             router
       notification_failure_criteria   d
       execution_failure_criteria      d
}
```

Common hostdependency Parameters

host_name

The *host_name* identifies the host that has a dependency on another host. In the preceding example, it would be the host behind the switch.

dependent_host_name

The *dependent_host_name* identifies the host in which the *host_name* has a dependency on. In our example, the *dependent_host_name* would refer to the switch.

notification_failure_criteria

If the host identified by *dependent_host_name* goes into any of the specific statuses described in this list, notifications will be suppressed in regards to the host identified by host. This list is comma-separated:

o

Suppress notifications when the dependent host is in an Up state.

d

Suppress notifications when the dependent host is in a Down state.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

u
: Suppress notifications when the dependent host is in an Unreachable state.

p
: Suppress notifications when the dependent host is in a Prending state.

n
: Never suppress notifications, despite the dependent host's state.

execution_failure_criteria
: If the host identified by *dependent_host_name* goes into any of the specific statuses described in this list, active checks will be suppressed in regards to the host identified by host. This list is comma-separated:

o
: Do not actively check the host when the dependent host is in an Up state.

d
: Do not actively check the host when the dependent host is in a Down state.

u
: Do not actively check the host when the dependent host is in an Unreachable state.

p
: Do not actively check the host when the dependent host is in a Pending state.

n
: Always actively check the host, despite the dependent host's state.

The servicedependency Object

A *servicedependency* is much like a hostdependency, except it concerns services. A service on one host can be dependent upon a service provided on another. For example, the web application on one web server could be dependent on the database being functional on another. This relationship is described to Nagios as a servicedependency.

```
define servicedependency {
       host_name                       localhost
       service_description             pop3
       dependent_host_name             nfsserver
       dependent_service_description   nfs
       notification_failure_criteria   w,u,c
       execution_failure_criteria      n
}
```


Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

Common servicedependency parameters

host_name
The *host_name* determines on which host the dependent service resides.

service_description
The *service_description* determines the dependent service that is on the host described by *host*.

dependent_host_name
The *dependent_host_name* is the host on which the service that is being depended on resides.

dependent_service_description
The *dependent_service_description* determines what service on *dependent_host_name* is the service that is being depended on.

notification_failure_criteria
If the dependedservice goes into any of the specific statuses described in this list, notifications will be suppressed in regards to the dependent service:

- *o*
  Suppress notifications when the depended service is in an OK state.
- *w*
  Suppress notifications when the depended service is in a Warning state.
- *u*
  Suppress notifications when the depended service is in an Unknown state.
- *c*
  Suppress notifications when the depended service is in a Critial state.
- *n*
  Never suppress notifications, despite the depended service's state.

execution_failure_criteria
If dependent service goes into any of the specific statuses described in this list, active checks are suppressed in regards to the dependent service. This list is command-separated:

- *o*
  Do not actively check the service when the depended service is in an OK state.
- *w*
  Do not actively check the service when the depended service is in a Warning state.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

u
: Do not actively check the service when the depended service is in an Unknown state.

c
: Do not actively check the service when the depended service is in a Critical state.

n
: Always actively check, despite the depended service's state.

The hostescalation Object

When a problem with a host has not been resolved within a certain amount of time, it's normal for notifications regarding that host to escalate to someone higher. So if the network engineer doesn't fix a switch for which he is the contact after five notifications have been sent out, then the manager of that administrator should be contacted. It could go higher still after another round of notifications. This process is defined as a *hostescalation*.

```
define hostescalation {
       host_name                       localhost
       contact_groups                  managers
       first_notification              3
       last_notification               0
       notification_interval           60
}
```

Common hostescalation parameters

host_name
: The *host_name* defines which host this escalation rule should be applied to.

contact_groups
: This comma-separated list consists of contact groups who should be contacted when this escalation rule is in affect.

first_notification
: The *first_notification* is the first notification in which this escalation rule takes affect. In our preceding sample, this value is 3, which means that after two notifications have been sent out, the escalation rule takes effect and sends notifications to the *contact_groups* in this escalation instead.

last_notification
: The *last_notification* determines the notification number in which this escalation rule will expire. If the value is 0, then the escalation continues for as long as notifications regarding the problem go out. This value does not mean the total number of notifications for this escalation, so be careful. For example, in our earlier sample, *first_notification* is on 3. If *last_notification* is set to 6, it

does not mean a maximum of 6 notifications will be sent out via this escalation; just 3 will be sent out.

notification_interval

The *notification_interval* determines the interval in which notifications should be sent out when this escalation is active.

The serviceescalation Object

Service escalations work much like host escalations; however, they deal with services instead. Other than that, everything else is the same.

```
define serviceescalation {
       host_name                 localhost
       service_description       pop3
       contact_groups            mailmanagers
       first_notification        3
       last_notification         0
       notification_interval     60
}
```

Common hostescalation parameters

host_name

The *host_name* defines which host this escalation rule should be applied to.

service_description

The *service_description* defines what service this escalation should be applied to.

contact_groups

This comma-separated list is of contact groups who should be contacted when this escalation rule is in affect.

first_notification

The *first_notification* is the first notification in which this escalation rule takes affect. In our preceding sample, this value is 3, which means that after two notifications have been sent out, this escalation rule takes effect and sends notifications to the *contact_groups* in this escalation instead.

last_notification

The *last_notification* determines the notification number in which this escalation rule will expire. If the value is 0, then this escalation continues for as long as notifications regarding the problem go out. This value does not mean the total number of notifications for this escalation, so be careful. For example, in our earlier sample, *first_notification* is on 3. If *last_notification* is set to 6, it does not mean a maximum of 6 notifications will be sent out via this escalation; just 3 will be sent out.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

notification_interval

The *notification_interval* determines the interval in which notifications should be sent out when this escalation is active.

The hostextinfo Object

The more descriptive the web interface is in regards to your objects, the better. Visual cues can help identify elements in your network. Extended information in Nagios is meant to help the web interface be more descriptive. A *hostextinfo* object defines the additional properties to use when rendering information regarding the host in the web interface.

```
define hostextinfo {
       host_name           localhost
       notes               This is the server Nagios is running on
       icon_image          server.gif
       2d_coords           50,100
}
```

Common hostextinfo parameters

host_name

The *host_name* defines what host should use these descriptive properties.

notes

When viewing a host in Nagios's interface, it will show any notes you put in the *notes* field. The value could be a longer description of the host's purpose, for example.

icon_image

The *icon_image* is the name of an image that will be displayed when viewing information about this host. This image is also used in the 2-D status map and is stored in the *logos/* subdirectory in the HTML images directory. In the default installation, *icon_image* would point to */usr/local/nagios/share/images/logos*.

2d_coords

The status map can automatically plot hosts by dependency, organization, and other layout methods. You can also manually plot the point of the host by defining the top-left coordinates as the *2d_coords*. Manually plotting coordinates for hosts tends to be difficult for larger installations, however.

The serviceextinfo Object

Extended information can also be provided about services in the web interface.

```
define serviceextinfo {
    host_name           host_name
    service_description service_description
    notes               note_string
    notes_url           url
    action_url          url
```

```
    icon_image          image_file
}
```

Common serviceextinfo parameters

host_name

The *host_name* defines what host the service resides on.

service_description

The *service_description* is the name of the service for which this extended information is for.

notes

When viewing a service in Nagios's interface, it will show any notes you put in the *notes* field. The value could be a longer description of the service's purpose, for example.

notes_url

If a url is provided here, the web interface will create a link labeled "Extra Service Notes," which will point to this destination. A good example could be a link to a Web Administration panel for the service in question.

icon_image

This image will be shown when viewing information regarding this service. This image is stored in the *logos/* subdirectory in the HTML images directory. In the default installation, *icon_image* would point to */usr/local/nagios/share/images/logos*.

Templates

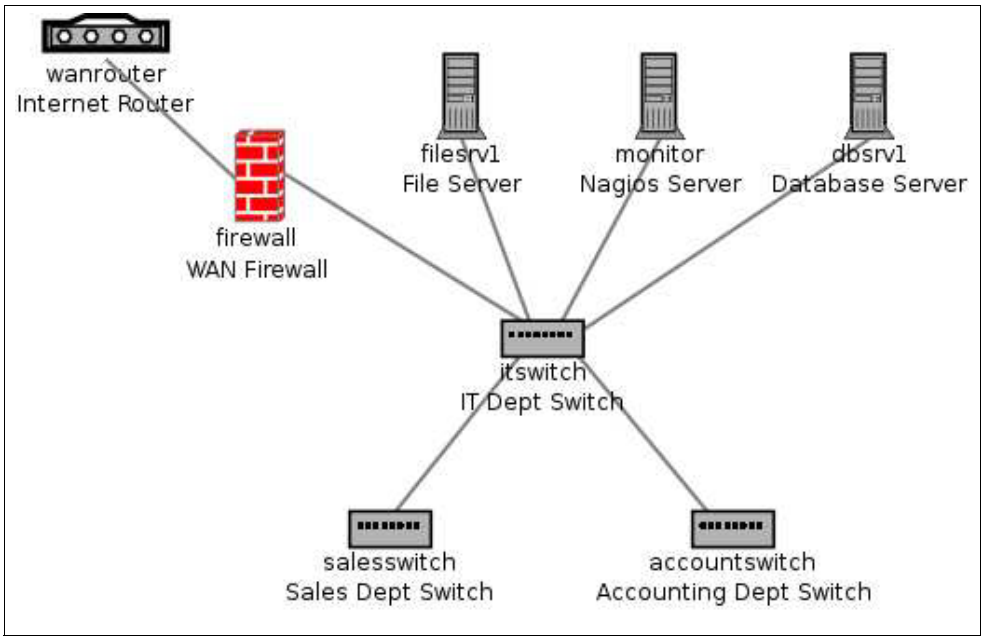
Writing a full object definition for every single service in your infrastructure can be a tiresome and error-prone process. Therefore, Nagios supports the use of templates. A *template* is a partial definition for a type of object. For example, you may have a host template defined for all your switches. By using this template, you can easily change the overall monitoring behavior of a switch by changing the template instead of each object definition for all switches. The switch object definition "pulls" in the parameters defined in the template it uses. It has the option, however, to override any parameters.

A template can also pull in parameters from another template. It is because of this fact that properly designing your template hierarchy is the best way to decrease the effort needed to maintain your Nagios configuration files. We'll use simple templates for the following sample configuration.

Sample Configuration

After looking at all the configuration files that are involved with Nagios, it's best that we put it all into context. Let's describe a fictional company, Widgets

Incorporated. The company has grown over the years and has gained a decent sized IT infrastructure. Being the company's network administrator, we want to monitor our company's network and have the individuals in our IT staff be notified of any failures of systems they are responsible for. Let's take a look at our network topology.



Network topology for Widgets Incorporated

You'll notice we didn't include any of the systems that are behind the Sales Dept Switch and the Accounting Switch. Our knowledge is that there are only workstations behind these switches. Therefore, all devices in this topology show the mission-critical devices.

Our IT team consists of a handful of individuals who are responsible for various aspects of our network. Let's have a look at them.

Billy Bob – IT Manager

Billy Bob is responsible for the actions of his team. He wants only to know when his team members are not doing their job; however, he doesn't want to be disturbed Sunday while he attends Church, otherwise known as golf.

Tom Thompson – Network Engineer

Tom is responsible for the backbone of the network. This includes the WAN connection to the Internet and the switches in the network.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

Sally Simms – Server Administrator

Sally has the lucky job of maintaining the servers. This includes the monitoring server, the file server, and the database server.

Widgets Incorporated's IT team of has performed a default installation of Nagios, as described in this document. The team currently wants only to check to see whether the devices are reachable. The team has created configuration files to describe its network and its notification and escalation policies to Nagios. Let's take a look at each of the configuration files, starting first with *nagios.cfg*.

Sample nagios.cfg

```
log_file=/usr/local/nagios/var/nagios.log
cfg_file=/usr/local/nagios/etc/checkcommands.cfg
cfg_file=/usr/local/nagios/etc/misccommands.cfg
cfg_file=/usr/local/nagios/etc/contacts.cfg
cfg_file=/usr/local/nagios/etc/contactgroups.cfg
cfg_file=/usr/local/nagios/etc/timeperiods.cfg
cfg_file=/usr/local/nagios/etc/hosts.cfg
cfg_file=/usr/local/nagios/etc/hostgroups.cfg
cfg_file=/usr/local/nagios/etc/services.cfg
cfg_file=/usr/local/nagios/etc/escalations.cfg
resource_file=/usr/local/nagios/etc/resource.cfg
status_file=/usr/local/nagios/var/status.dat
nagios_user=nagios
nagios_group=nagios
check_external_commands=1
command_check_interval=-1
command_file=/usr/local/nagios/var/rw/nagios.cmd
comment_file=/usr/local/nagios/var/comments.dat
downtime_file=/usr/local/nagios/var/downtime.dat
lock_file=/usr/local/nagios/var/nagios.lock
temp_file=/usr/local/nagios/var/nagios.tmp
log_rotation_method=d
log_archive_path=/usr/local/nagios/var/archives
illegal_macro_output_chars=`~$^&"|'<>
```

We've accepted most of the sane defaults that Nagios provides; however, there are a few that we've decided to include. By default, Nagios will never rotate the logs, which is what we want to change right away. Furthermore, we want Nagios to check for external commands via the command file. This is so we can send commands to Nagios via the web interface when responding to issues The rest of our configuration has been separated into individual files, which are referred to by the *cfg_file* statements.

Sample resources.cfg

```
$USER1$=/usr/local/nagios/libexec
```

The *resources.cfg* file is used to define user macros that will be used in our command definitions. We need only one macro, *$USER1$*, to be defined. The *$USER1$* macro contains the path where our Nagios plug-ins are on the filesystem. This macro is used in our command definitions, next.

Sample checkcommands.cfg

```
define command {
 command_name    check_ping
 command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$ -p 5
}


define command {
 command_name    check-host-alive
 command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w 3000.0,80% -c 5000.0,100% -p 1
}
```

Because we are only checking to see whether the devices are reachable, we'll want the commands to support that. The first command, *check_ping*, will be used for a PING service for each of the devices. The command line dictates what Nagios will execute when running this command. For *check_ping*, this means executing the *check_ping* plug-in located in the directory defined by our *$USER1$* macro. The *check_ping* plug-in takes a few parameters. The *–H* parameter dictates which host to send the ping request to; in this case, it's the *$HOSTADDRESS$* macro, which macro will be filled in during runtime, depending on the context. For example, if the *check_ping* command is being run in a service definition for the *filesrv1* host, then the *$HOSTADDRESS$* macro would be filled in with the network address of that host. The *–w* parameter specifies what the warning threshold should be when checking for ping responses. Thresholds for the *check_ping* plug-in follows the format *<rta>,<pl>%*, where *<rta>* is the round-trip average travel time (milliseconds) and *<pl>* is the percentage of packet loss. Whenever these thresholds are met, the service that uses this command will be put into a Warning status. The *–c* parameter specifies the threshold that will put the service that uses this command in a *Critical* state. The *–p* parameter specifies how many ping requests to send. The *check-host-alive* command is similar to the *check_ping* command, but it will be used in the context of host checks.

Sample misccommands.cfg

```
define command{
        command_name    host-notify-by-email
        command_line    /usr/bin/printf "%b" "***** Nagios  *****\n\nNotification
Type: $NOTIFICATIONTYPE$\nHost: $HOSTNAME$\nState: $HOSTSTATE$\nAddress:
```

```
$HOSTADDRESS$\nInfo: $HOSTOUTPUT$\n\nDate/Time: $LONGDATETIME$\n" |  /usr/bin/mail -s
"Host $HOSTSTATE$ alert for $HOSTNAME$!" $CONTACTEMAIL$
        }
define command{
        command_name    notify-by-email
        command_line    /usr/bin/printf "%b" "***** Nagios  *****\n\nNotification
Type: $NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost: $HOSTALIAS$\nAddress:
$HOSTADDRESS$\nState: $SERVICESTATE$\n\nDate/Time: $LONGDATETIME$\n\nAdditional
Info:\n\n$SERVICEOUTPUT$" |  /usr/bin/mail -s "** $NOTIFICATIONTYPE$ alert -
$HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ **" $CONTACTEMAIL$
        }
```

Unfortunately, the command-line definitions for these commands are quite long. These commands are used when Nagios needs to send notifications our in regards to our hosts and services. The *host-notify-by-email* command is used when sending out notifications in regards to host checks. The *notify-by-email* command is used when sending notifications out in regards to service checks. Both commands use the *printf* utility located in */usr/bin/* to print a formatted email, and then sends it to the *mail* command located in */usr/bin/* to send it out with an appropriate subject. The various macros are all filled in during runtime. For example, the *$CONTACTEMAIL$* macro is filled in with the value of the contacts *email* directive.

Sample timeperiods.cfg

```
define timeperiod{
        timeperiod_name 24x7
        alias           24 Hours A Day, 7 Days A Week
        sunday          00:00-24:00
        monday          00:00-24:00
        tuesday         00:00-24:00
        wednesday       00:00-24:00
        thursday        00:00-24:00
        friday          00:00-24:00
        saturday        00:00-24:00
        }

define timeperiod{
        timeperiod_name bosstime
        alias           All the time except for Sunday
        monday          00:00-24:00
        tuesday         00:00-24:00
        wednesday       00:00-24:00
        thursday        00:00-24:00
        friday          00:00-24:00
```

```
        saturday        00:00-24:00
        }
```

We have two timeperiods that pertain to our IT department. The *24x7* timeperiod includes all days of the week at all times. The *bosstime* timeperiod has all the days of the week, except Sunday. The *bosstime* timeperiod will be used to describe when Billy Bob, our faithful manager, can be notified. The *24x7* timeperiod will be used to specify when the rest of our team should be notified.

Sample contacts.cfg

```
define contact{
        contact_name                    bbob
        alias                           Billy Bob - IT Manager
        service_notification_period     bosstime
        host_notification_period        bosstime
        service_notification_options    w,u,c,r
        host_notification_options       d,r
        service_notification_commands   notify-by-email
        host_notification_commands      host-notify-by-email
        email                           bbob@widgets-inc.com
        }

define contact{
        contact_name                    tthompson
        alias                           Tom Thompson - Network Engineer
        service_notification_period     24x7
        host_notification_period        24x7
        service_notification_options    w,u,c,r
        host_notification_options       d,r
        service_notification_commands   notify-by-email
        host_notification_commands      host-notify-by-email
        email                           bbob@widgets-inc.com
        }

define contact{
        contact_name                    ssimms
        alias                           Sally Simms - Server Administrator
        service_notification_period     24x7
        host_notification_period        24x7
        service_notification_options    w,u,c,r
        host_notification_options       d,r
        service_notification_commands   notify-by-email
```


Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

```
        host_notification_commands      host-notify-by-email
        email                           bbob@widgets-inc.com
        }
```

We define each member of the IT department as a contact for Nagios. For each contact, we specify the events that will trigger a notification to each contact, along with what time they are allowed to be notified. You can see that for Billy Bob, our faithful manager, we've set his notification period to *bosstime*. We also specify which commands to use when notifying in regards to host or service events.

Sample contactgroups.cfg

```
define contactgroup{
        contactgroup_name       netengineers
        alias                   Network Engineers
        members                 tthompson
        }

define contactgroup{
        contactgroup_name       serveradmins
        alias                   Server Administrators
        members                 ssimms
        }

define contactgroup{
        contactgroup_name       managers
        alias                   IT Managers
        members                 bbob
        }
```

We have three contact groups in our IT department. The first, *netengineers*, contains Tom Thompson, our dutiful network engineer. In the future, as Widgets Incorporated grows, we can add new contacts into our contact groups. For now, however, each contact group contains only one individual. The *managers* contact group contains Billy Bob, our faithful manager.

Sample hosts.cfg

```
define host{
        name                            generic-host
        notifications_enabled           1
        event_handler_enabled           1
        check_command                   check-host-alive
        max_check_attempts              10
        check_period                    24x7
```


Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

```
        notification_interval           120
        notification_period             24x7
        notification_options            d,r
        register                        0
        }

define host{
        use                     generic-host
        host_name               monitor
        alias                   Nagios Server
        address                 127.0.0.1
        contact_groups          serveradmins
        }

define host {
        use                     generic-host
        host_name               dbsrv1
        alias                   Database Server
        address                 192.168.1.50
        contact_groups          serveradmins
}

define host {
        use                     generic-host
        host_name               filesrv1
        alias                   File Server
        address                 192.168.1.51
        contact_groups          serveradmins
}

define host {
        use                     generic-host
        host_name               itswitch
        alias                   IT Dept Switch
        address                 192.168.1.100
        contact_groups          netengineers
}

define host {
        use                     generic-host
        host_name               salesswitch
```

```
        parents                 itswitch
        alias                   Sales Dept Switch
        address                 192.168.1.101
        contact_groups          netengineers
}

define host {
        use                     generic-host
        host_name               accountswitch
        parents                 itswitch
        alias                   Accounting Switch
        address                 192.168.1.102
        contact_groups          netengineers
}

define host {
        use                     generic-host
        host_name               firewall
        parents                 itswitch
        alias                   WAN Firewall
        address                 192.168.1.1
        contact_groups          netengineers
}

define host {
        use                     generic-host
        host_name               wanrouter
        parents                 firewall
        alias                   Internet Router
        address                 10.10.20.200
        contact_groups          netengineers
}
```

Each device in our network has a host definition. The first host definition is actually a template. The template defines all the shared directives all our hosts will share. The *name* directive of our host template specifies the name to use in our host definitions. The *register* directive specifies this definition as a template. For each of our host definitions, we use the use directive to specifcy which template we want. The *parents* directive specifies which host is between Nagios and the target device. The *contact_groups* directive shows which contact groups are responsible for these devices. The web interface uses this directive when deciding which objects to show to the person using the interface.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

Sample hostgroups.cfg

```
define hostgroup {
        hostgroup_name  servers
        alias           Servers
        members         filesrv1,dbsrv1,monitor
}

define hostgroup {
        hostgroup_name  networking
        alias           Networking Equipment
        members         itswitch,salesswitch,accountswitch,firewall,wanrouter
}
```

The devices at Widgets Incorporated are separated into two groups. The *servers* group contains all the servers in the department. The *networking* group contains all the networking gear, including switches, firewalls, and routers.

Sample services.cfg

```
define service {
        name                            generic-service
        active_checks_enabled           1
        passive_checks_enabled          1
        parallelize_check               1
        obsess_over_service             1
        notifications_enabled           1
        event_handler_enabled           1
        flap_detection_enabled          1
        is_volatile                     0
        check_period                    24x7
        max_check_attempts              4
        normal_check_interval           5
        retry_check_interval            1
        notification_interval           960
        notification_period             24x7
        register                        0
}

define service {
  use                           generic-service
  host_name                     monitor, filesrv1, dbsrv1
  service_description           PING
  check_command                 check_ping!100.0,20%!500.0,60%
```


Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

```
  contact_groups                serveradmins
}
define service {
  use                           generic-service
  host_name                     itswitch,salesswitch,accountswitch,firewall,wanrouter
  service_description           PING
  check_command                 check_ping!100.0,20%!500.0,60%
  contact_groups                netengineers
}
```

Like our *hosts.cfg*, the first definition in this file is really a template. The *generic-service* template defines all the directives that will be shared by our services. Our second service definition actually defines multiple services. The service definition specifies multiple hosts in the *host_name* directive, meaning this service will belong on all of these hosts.

Sample escalations.cfg

```
define hostescalation {
        hostgroup_name          servers, networking
        contact_groups          managers
        first_notification      5
        last_notification       0
        notification_interval   60
        escalation_period       24x7
        escalation_options      d,r
}


define serviceescalation {
        hostgroup_name          servers, networking
        contact_groups          managers
        service_description     PING
        first_notification      5
        last_notification       0
        notification_interval   60
        escalation_period       24x7
        escalation_options      w,c,r
}
```

Our escalation rules are simple. If something isn't fixed after four notifications have been sent out regarding a problem, notify Billy Bob (unless it's Sunday, of course). We have a hostescalation definition to define our escalation for hosts, as well as a serviceescalation for our services.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

The preceding configuration files get Widgets Incorporated running with a fairly minimal configuration. There is, of course, much more functionality that we could put in our configuration files, such as the storing of performance data, checking for resources on the remote devices, and checking for flapping devices and services. To find out how to configure these tasks, review the Nagios documentation.

Running Nagios

When we compiled and installed Nagios by hand, we used the *install-init* make target to install initialization scripts into our */etc/init.d* directory. As root, we can run the */etc/init.d/nagios* script to start and stop Nagios. Let's start Nagios now using this script.

```
localhost: /etc/init.d # ./nagios start
Starting network monitor: nagios
```

If you want Nagios to start automatically when your server starts, then you'll need to create the appropriate symbolic links. For example:

```
localhost:~ # ln –s /etc/init.d/nagios /etc/init.d/rc3.d/S99nagios
localhost:~ # ln –s /etc/init.d/nagios /etc/init.d/rc3.d/K99nagios
```

The above command will have Nagios start automatically when the system enters run level 3 (which is multiuser mode).

Once Nagios is up and running, you can verify its operation by using the web interface. However, before using the web interface, some configuration changes need to be made to Apache, and Nagios's CGI behavior needs to be defined in its CGI configuration file.

Configuring the Web Interface

When we compiled and installed Nagios, all the files required to interact with Nagios with a web browser were installed under */usr/local/nagios*. However, before we can access Nagios through our web browser, we must configure it to provide access to these resources. The web server we're using for our installation is Apache. A sample configuration snippet is created in the *sample-config/* subdirectory in the directory you extracted Nagios originally. You could copy the contents of the *httpd.conf* file in this directory into your Apache's main configuration file or follow the steps below to do it manually.

If you used the default settings when running Nagios's configure script, the following snippet will work. This snippet of code must be put in your Apache's configuration file.

```
ScriptAlias /nagios/cgi-bin /usr/local/nagios/sbin


<Directory "/usr/local/nagios/sbin">
    Options ExecCGI
```


Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

```
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile /usr/local/nagios/etc/htpasswd.users
    Require valid-user
</Directory>

Alias /nagios /usr/local/nagios/share

<Directory "/usr/local/nagios/share">
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile /usr/local/nagios/etc/htpasswd.users
    Require valid-user
</Directory>
```

This configuration snippet does a few things to Apache. First, it creates a *ScriptAlias* that will take any script references from the URL */nagios/cgi-bin* and then find the script instead in the path */usr/local/nagios/sbin*. In order to make scripts run, we set up a Directory directive to allow CGI script execution via the *ExecCGI* option. We also set up the basics of http authentication, using */usr/local/nagios/etc/htpasswd.users* as the source of our authentication, and we made an alias to point */nagios* to */usr/local/nagios/share*, which holds all the nonscript resources for the web interface such as documentation, images, and CSS stylesheets.

Once this snippet is inserted into Apache's configuration, restart Apache. Usually, a *httpd* script is located in */etc/init.d*.

```
localhost:~ # /etc/init.d/httpd restart
```

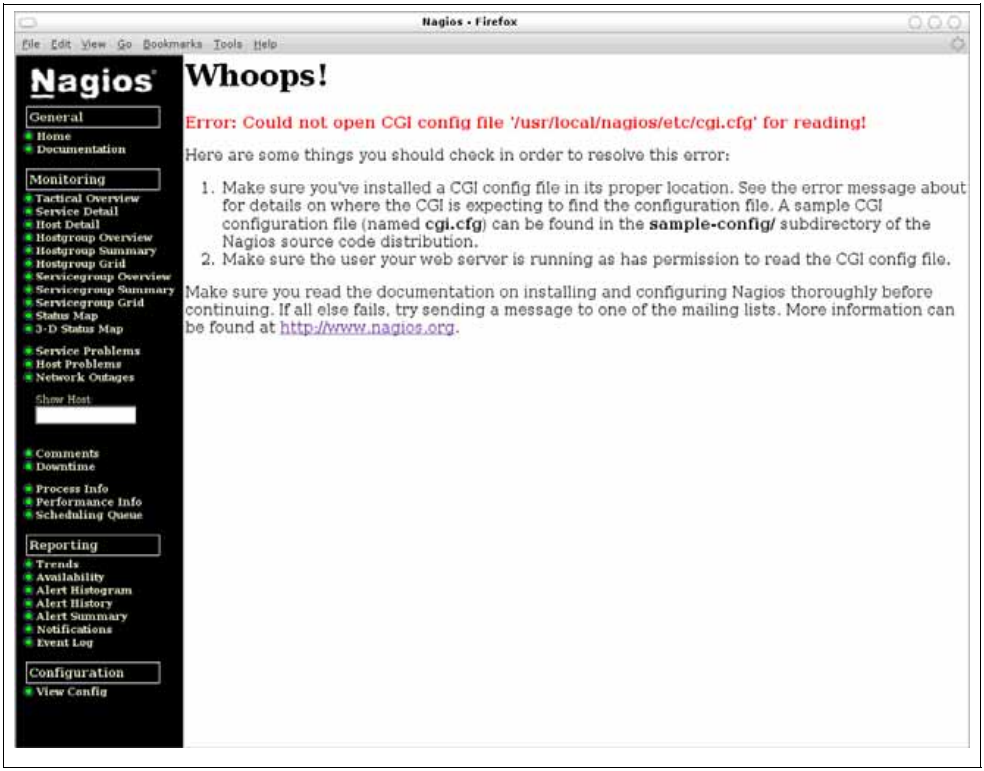
If there are any problems with the configuration, Apache refuses to restart and points you to the problem. Otherwise, Apache starts up, and any requests to a url to the server that is in the path */nagios* pops up an http authentication window request. Since we have not yet created and configured our authentication file, we'll need to do that now. As stated previously, the authentication file we will use is */usr/local/nagios/etc/htpasswd.users*. The program *htpasswd*, which is distributed with Apache, is used to create and configure this file. We will create our file initially and our first user record.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

```
localhost:~ # htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
New password: admin
Re-type new password: admin
```

These steps create a new user, *nagiosadmin*, with the password admin. This is not entirely secure and should be changed later on; however, for our basic purposes, this is fine to test our installation. For more information on *htpasswd*, see its man(1) page.

Now that our user is created, logging in as admin when visiting */nagios* on our webserver presents the welcome page to Nagios. However, clicking on any of the monitoring links presents a commonly seen error to new installations.



Common Nagios CGI error when first installing

The error states that the web interface could not read the CGI config file */usr/local/nagios/etc/cgi.cfg*. Yes, it's true. Nagios requires yet another configuration file for operation. The installation process of Nagios put a sample CGI configuration file in */usr/local/nagios/etc* called *cgi.cfg-sample*. The best thing to do is rename it to simply *cgi.cfg* and then use that as a starting point for

customizing the Nagios web interface behavior.Using the sample configuration file is a great start; however, there are some important parameters in that file that need special mention.

Common CGI Configuration File Parameters

main_config_file
: The web interface needs to know where your main configuration file is in order to match up status information to proper objects. The configuration files also dictate what human-readable labels to give to our various objects (via the *alias* parameter in most). In our installation, the path is */usr/local/nagios/etc/nagios.cfg*, but if you ever move your configuration files, this parameter's value will need to change as well.

physical_html_path
: The CGI scripts that power the web interface often have to reference images and documentation. The *physical_html_path* parameters point to the location on the filesystem that has these resources. Our default value is */usr/local/nagios/share/*, which is fine. When none of your images are showing up and the web interface just looks plain ugly, make sure this parameter points to the right location.

url_html_path
: Remember the alias we put in Apache's configuration file? The CGI scripts need to know what the URL path is to the rest of the scripts in order to properly display links. In our installation, *url_html_path* would be */nagios*. If you changed the alias in the Apache configuration files, you'll need to change this value as well.

show_context_help
: While you are new to the web interface, it's important to have *show_context_help* set to 1. By doing so, the web interface provides helpful descriptions of the actions and objects you are looking at when you hover over them. I always have it set to 1 because the help never really gets in the way.

use_authentication
: When we configured Apache, we set up http authentication. It makes sense to have Nagios use authentication to validate actions when using the web interface. For small installations, it *may* be reasonable to not even use authentication. In that case, setting *use_authentication* to 0 causes the web interface to not need proper authentication to run. It also shows all information to anyone and allows anyone to perform any operation available in the web interface. When using best practices, however, it is best to always use authentication, no matter how small the installation and support team.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

refresh_rate
: In order for the information displayed to be kept up to date, the Nagios web interface reloads the page in regular intervals. The *refresh_rate* value, in seconds, is used to determine that refresh rate. By setting it smaller than the default value of 90 seconds, the information refresh rate can be more aggressive; however, if you have numerous browsers viewing the web interface, it could become taxing on the web server with so many requests.

authorized_for_all_services, authorized_for_all_hosts
: When using authentication, it may be suitable for a certain user to be able to view all services and hosts. These parameters specify the comma-separated list of users that can perform such behavior. The user provided doesn't even need to be a contact in the Nagios configuration files. As long as a valid http-authenticated user is logged in and belongs in this list, he is considered omnipotent.

There are other useful parameters you can set in this configuration file. For more information on those parameters, refer to the Nagios documentation. In order to see how the CGI configuration file would look in a real environment, let's take a look at the *cgi.cfg* file for Widgets Incorporated, the company we configured earlier.

Sample cgi.cfg

```
main_config_file=/usr/local/nagios/etc/nagios.cfg
physical_html_path=/usr/local/nagios/share
url_html_path=/nagios
show_context_help=1
use_authentication=1
refresh_rate=90
```

We accept the defaults that Nagios provides; however, we do want context-based help shown when possible. Also, it's always important to use authentication. The refresh rate of 90 specifies that each page of the web interface will refresh every minute and a half.

When using the configuration files for Widgets Incorporated, it's important to create htpasswd users for each of the contacts. Do this for each of the department members. We will use Widgets Incorporated as our company when discussing the following scenarios.

Using the Web Interface

Once the *cgi.cfg* configuration file is created and you are logged in with proper http authentication, you will be presented with the Nagios web interface. If you are logged in with a username that matches a contact in your Nagios configuration

files, you will be presented with information that pertains only to those hostgroups, hosts, and services that you are a valid contact for. If you are not a valid contact in the Nagios configuration files, and you are not set as a user that is *authorized_for_all_services* or *authorized_for_all_hosts* in the *cgi.cfg* file, then you will receive an error that you do not have access to any information.

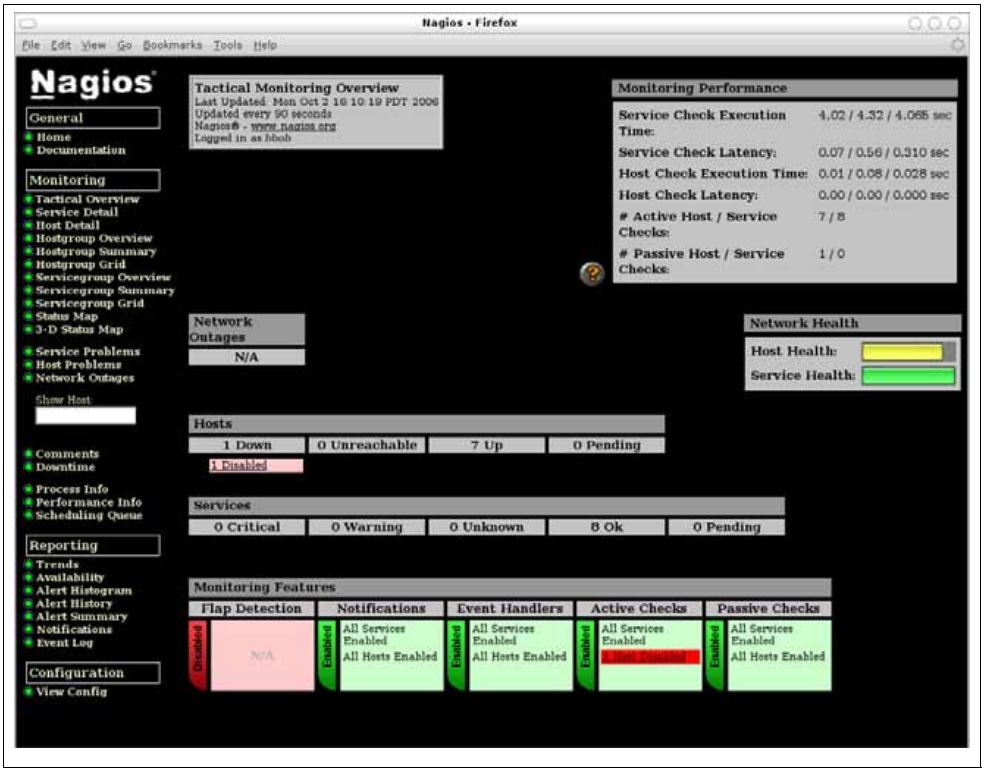
There are many destinations in the web interface that provide a great deal of information; however, they are all well described in the Nagios documentation. What we will do instead is cover some common scenarios that occur and how you would interact with Nagios as a result.

Warning

The Nagios web interface is written as a C-based CGI application. It is because of this implementation that it does not have the idea of "application-stat.e" It means that with each request of the web interface, it has completely forgotten what it previously did. Furthermore, with each request, the CGI application has to completely re-read Nagios's configuration files and status data. With Nagios 1.x, the web interface parsed the configuration files and the status files. With Nagios 2.x, the process has improved somewhat in which the interface now reads cache files which speeds up the process. However, even with this improvement, the wait for each request to take grows exponentially with the growth of your Nagios monitored infrastructure. I've seen these issues become strongly apparent when the infrastructure went over about 500 monitored nodes. Your mileage may vary.

Scenario #1: Viewing the Overall Health of Your Network

Billy Bob comes into the office Monday morning. He has a weekly meeting with the rest of the senior staff to discuss the current status of the company. In order to prepare for this meeting, he wants to get an up-to-date look at the health of the IT infrastructure. He opens up the Nagios interface and logs in with his username (which matches the contact name in Nagios; in this case, *bbob*). He then clicks on *Tactical Overview* in the navigation pane on the right. He is presented with the following screen.

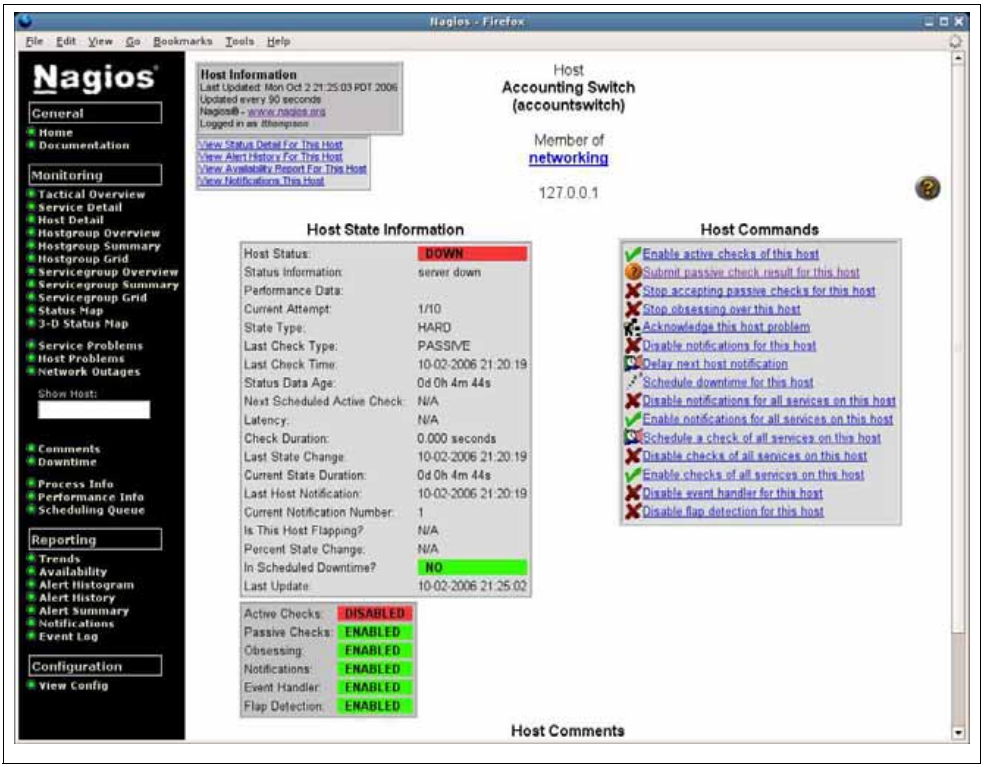


The Tactical Monitoring Overview Page

In the *Tactical Monitoring Overview* screen, Billy Bob is presented with the overall health of the network. This screen specifies what problems there are and if the problem is acknowledge by someone in his staff. In our case, we notice that one host is down. We also notice that active checks are disabled for one host. Billy Bob can only hope that his competent staff is handling the issue.

Scenario #2: Responding to a Problem Notification

Tom Thompson, our competent Network Engineer, receives an email alert stating there's a problem with the Accounting Switch. He logs into the Nagios web interface and enters *accountswitch* into the *Show Host* input field in the navigation sidebar. This takes him to the Host Status page shown below.



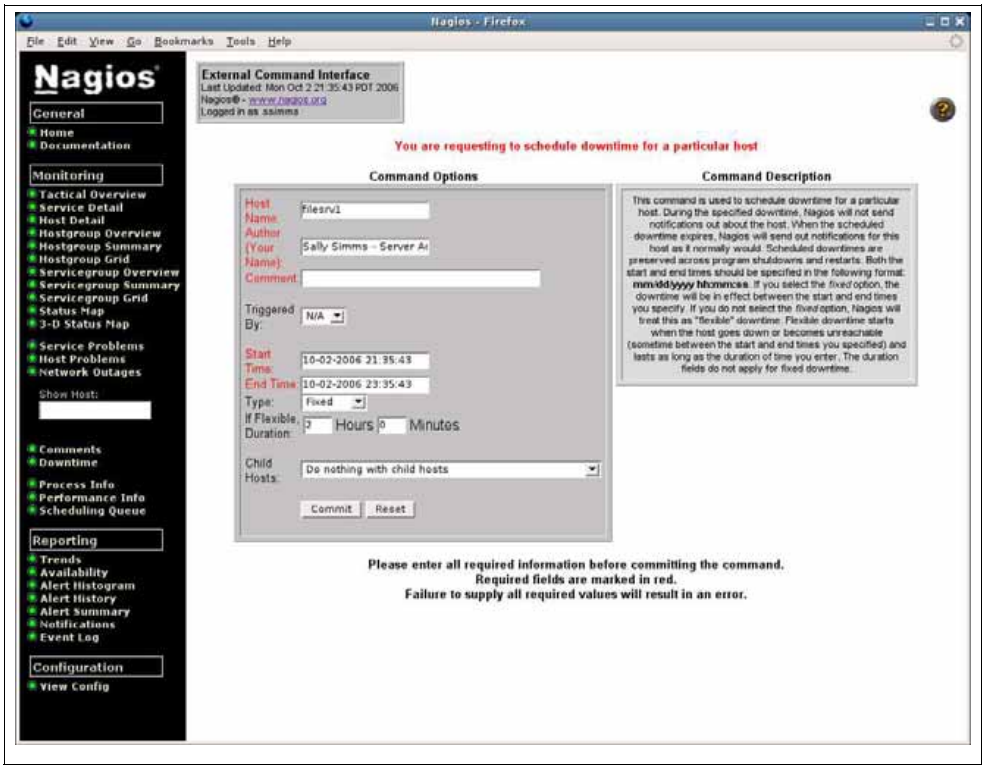
The Host Detail screen, showing our host in DOWN state

The screen shows that the Host Status is DOWN. Tom wants to acknowledge the problem so the other team members are aware that he is working on it. When a problem is acknowledged, future visits to this host will show that. Furthermore, if chosen, an acknowledgment notification will be sent to all contacts for this host. This action will alert the rest of the team that you have acknowledged the problem and will be working on it. There is a list of commands on the righthand side, and each command allows us to perform operations on this host. By choosing the link "Acknowledge this host problem," we will be taken to a screen to enter some more information regarding the acknowledgment. After filling out the form, the acknowledement will be sent out and the interface will be updated. Also looking at the status page for the *accountswitch* host, Tom notices that Active Checks are disabled for this host. This isn't the right behavior, so Tom decides to re-enable Active checks by clicking on the "Enable active checks of this host link." This instructs Nagios to continue actively checking this host. Once the problem is corrected, Nagios will update this host with an OK state.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

Scenario #3: Scheduling Downtime for a Host or Service

Sally has determined that the *filesrv1* server needs to be removed from the network for maintenance. She wants to let the rest of the team know that the server will be in scheduled downtime. When a host or service is in a period of scheduled downtime, Nagios suppresses any notifications in regards to that host or service. Sally logs into the Nagios interface and heads to the host status page for *filesrv1*. She clicks on the "Schedule downtime for this host link," resulting in the form below.



Scheduled downtime form

At this form, you can schedule two types of downtime. A *Fixed* downtime represents a static period of time in which the host will be in maintenance. If you want to schedule a fixed downtime, you'll need to put in the Start Time and End Time of the downtime. The other type of downtime, *Flexible* downtime, will have Nagios start the downtime when the host or service begins to fail. The downtime period will then last as long as the Flexible Duration you've set up. Furthermore, if the downtime is for a host, you can specify the downtime to occur for all hosts that have this host as a *parent*. Sally knows that she'll begin maintenance at 2:00 a.m in

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

the morning (she's one of the stronger ones) and will be completed at 4:00 a.m. Sally makes sure the type is *fixed* and enters the appropriate Start and End Times. After she presses Commit, Nagios schedules the downtime. When the host enters the scheduled downtime, a construction sign appears next to the host in the web interface to represent the scheduled downtime.

There is a lot more you can do with the web interface than what is covered in the previous simple scenarios; however, they are well documented in the Nagios documentation, along with the context-sensitive help, if enabled. Furthermore, many third-party tools have been written to extend the functionality of the web interface, and of Nagios in general.

Extending Nagios

Nagios is a great tool to begin monitoring your network quickly and efficiently. However, there are other open source tools out there that compliment, even enhance, Nagios. Let's look at a few that you can get up and running quickly.

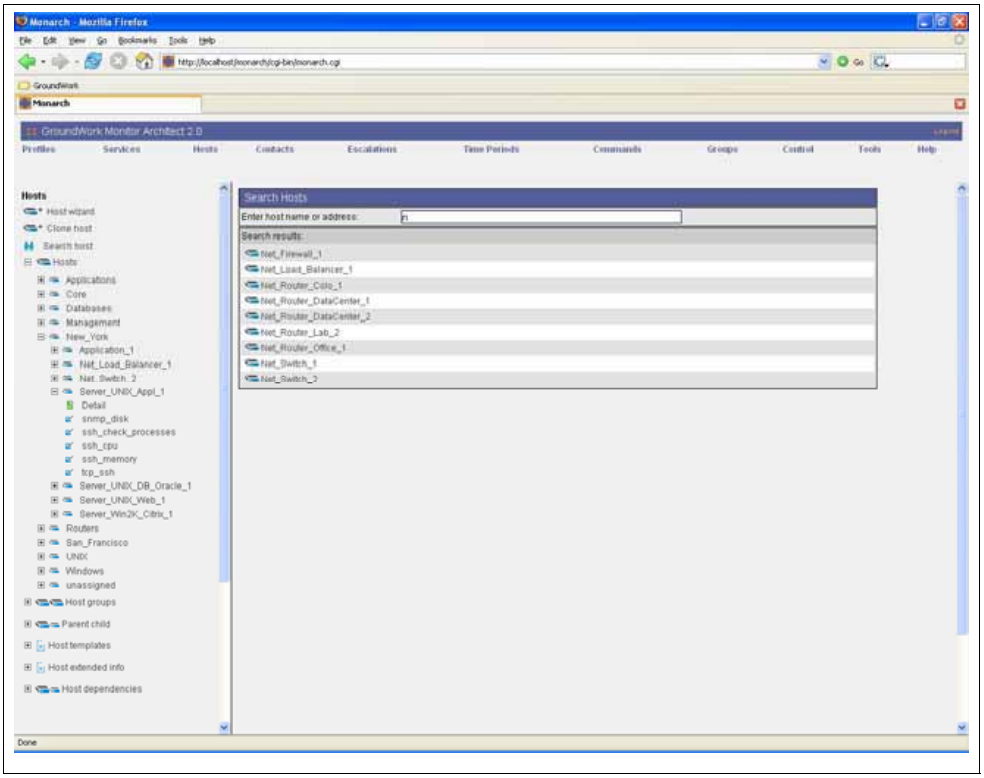
Configuration Tools

Writing your configuration files is a pain. It's bearable when you have only a handful of devices to monitor, but imagine the number growing into the hundreds. Many people, including myself, have tackled this problem with creating a myriad of configuration frontends to handle the hassle of configuration. We'll look briefly at a couple of the most well known; however, there are quite a few more available. Refer to Nagios Exchange (**http://www.nagiosexchange.org**) for a more complete list.

Monarch

Monarch, an open source project supported by Groundwork Open Source, is one of the more popular configuration tools for Nagios. It implements the use of profiles, which are sets of configuration for hosts and services. Host profiles can also include the initial list of services to include. The use of profiles, along with Monarch's easy user interface, makes configuration a snap. Furthermore, because it is commercially supported, it is constantly maintained.

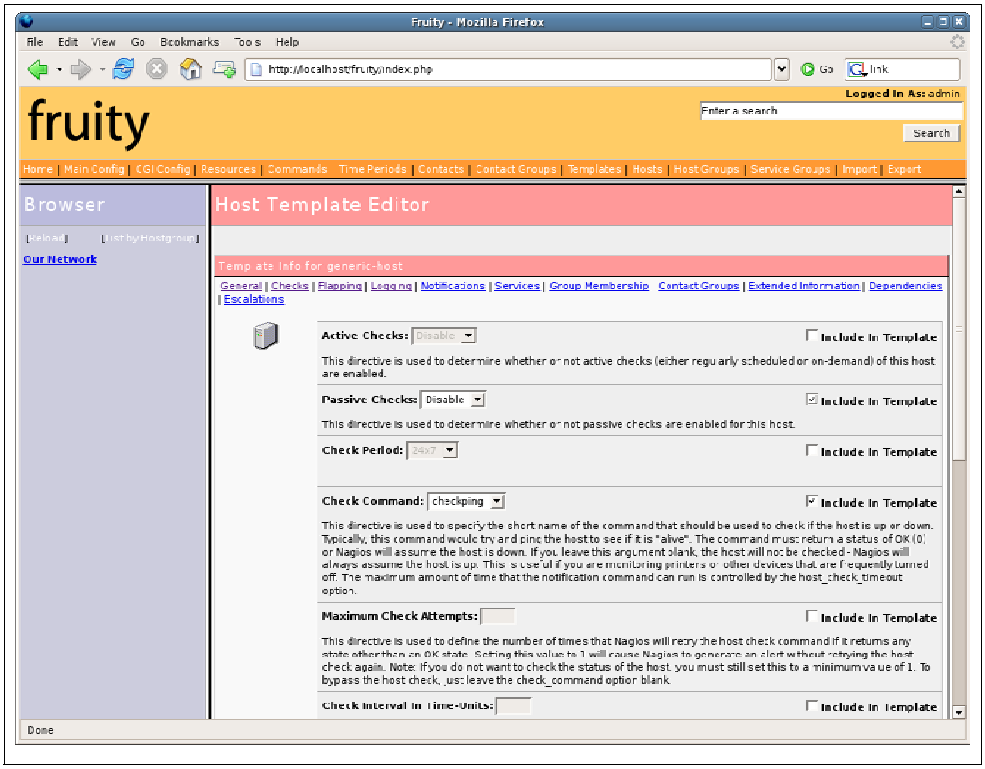
The navigation on the left gives you the ability to quickly find the items to modify and add quickly. Also included is a quick host search that will immediately pull up a list of hosts as you type. With these types of features, Monarch speeds up the time it takes to make modifications to your configuration changes.



Groundwork Monarch

Fruity

I wrote Fruity, so I will attempt not to be too biased in what I have to say about it. Fruity is also a Groundwork open source project is and also a configuration tool. Fruity uses Nagios templates extensively but also extends their capabilities. For example, while Nagios cannot specify what services to include in a host template, Fruity provides that functionality. You can also attach extended information, dependencies, and escalation rules to templates. Overriding template values is a cinch.



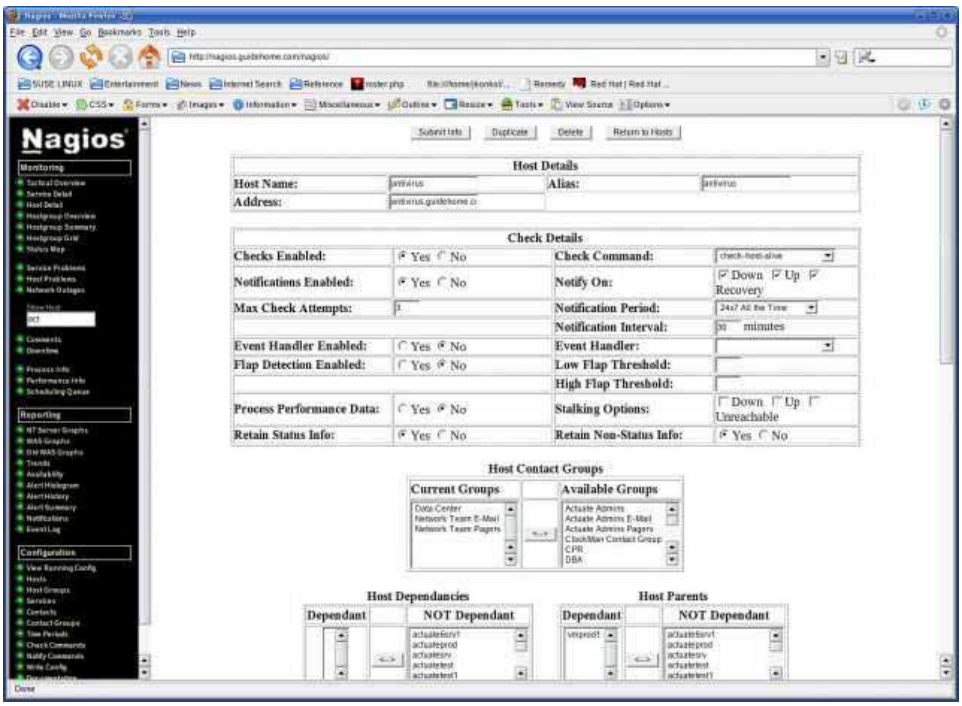
The Fruity Template Editor

Fruity has a very strong user community behind it. Users are constantly adding new functionality in the form of patches, which you can download from Fruity's project page at Sourceforge. The Fruity mailing list is also fairly active, with people happy to answer questions as well.

Nagios Web Config

Nagios Web Config is a utility that can fit inside Nagios's interface and provide a visual editor into the configuration files. It's mostly a form-based system that lets you put in the information through a browser instead of manually editing the configuration files. This feature makes it easier to extend the functionality of modifying the configuration files to other people.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01



Nagios Web Config

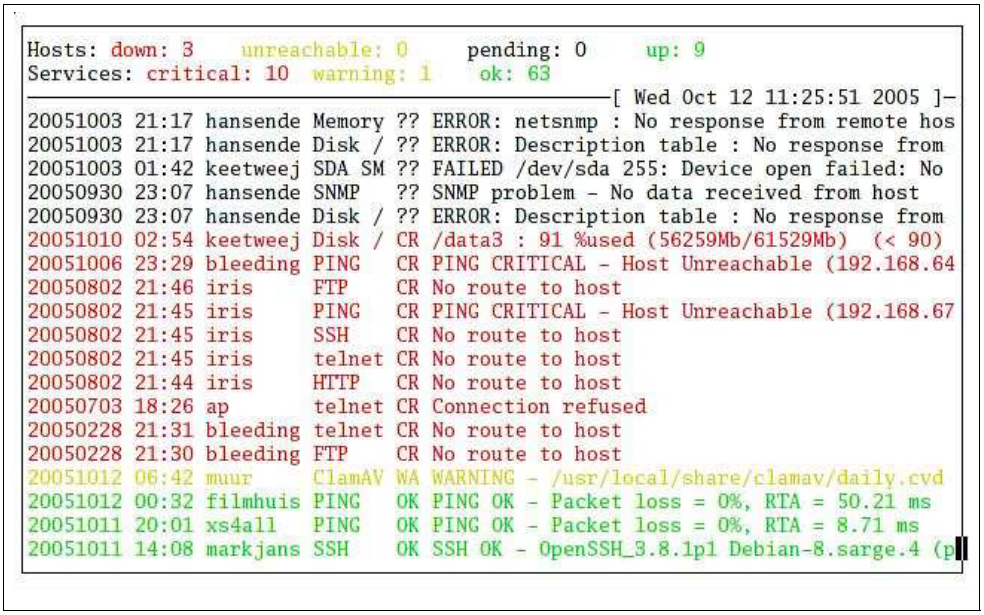
Frontends

The Nagios web interface is an excellent way to view your status-based data. There are, however, other ways to look at the data that Nagios gathers. For that, additional frontend applications are available.

Nagcon

Nagcon is a console-based utility that allows you to view the troubled services of your infrastructure. Nagcon is useful if you don't have a browser handy. It gives you the same information as the Service Problems page, but in a console screen that is constantly updating.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

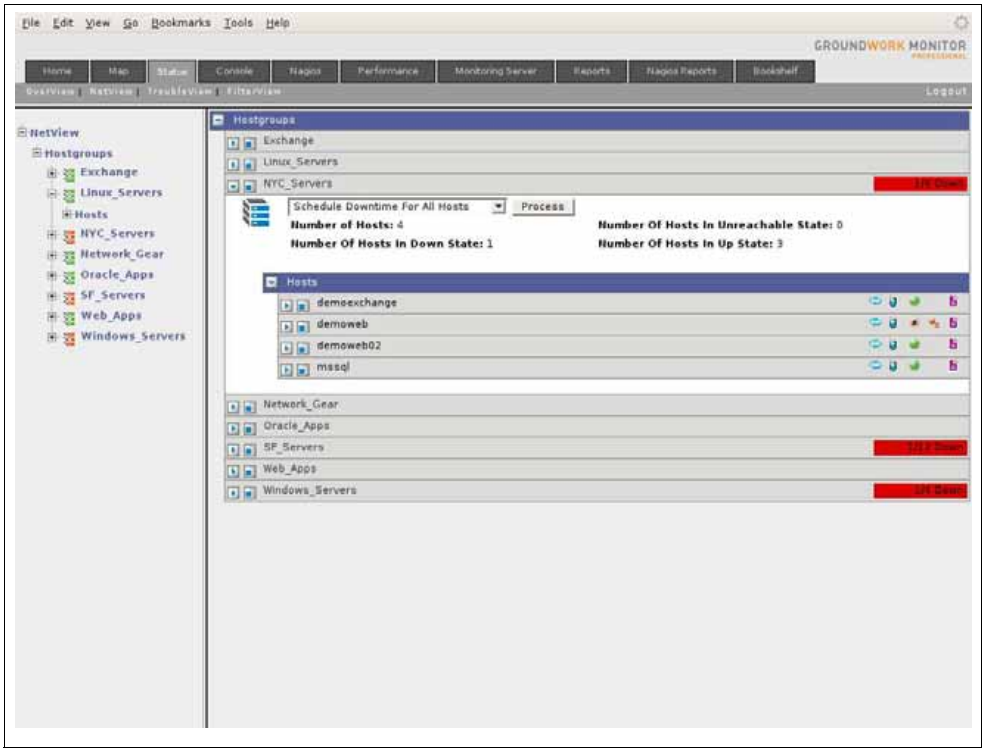


The NagCon application showing troubled items

Groundwork Status Viewer

Status Viewer is a replacement browser interface to Nagios data. It uses a drill-down metaphor for navigating your infrastructure, keeping all previous information on the screen. It also uses Perl scripts to feed up-to-date status information into a database so thatquerying for information is faster. On top of that, Status Viewer is written as a dynamic web application, modifying only the parts of the screen that change. The result is a faster-responding UI.

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01



Status Viewer running inside Groundwork Monitor

Going Forward

We covered the basics of installing, configuring, and using Nagios; however, your environment is more than likely more complex than Widgets Incorporated. There is a great deal more information in regards to configuring and using Nagios than what we could cover together. I'll leave you now with a few great online resources that can help walk you through some of the finer points of Nagios and how to use it in other situations.

Nagios Online Documentation

This is the official documentation. Every object directive is covered in detail. Advanced topics such as distributed monitoring and adaptive monitoring are covered here as well.

http://nagios.sourceforge.net/docs/2_0/

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

Nagios Mailing Lists

The best resource for assistance is the Nagios community itself. The Nagios User mailing list is always busy with people asking questions and other people answering them. However, before you ask, make sure the solution is not covered in the documentation.

http://www.nagios.org/support/mailinglists.php

Nagios Exchange

A good deal of additional add-on software is available through Nagios Exchange. Nagios Exchange has links to software in various categories to help you improve your Nagios installation.

www.nagiosexchange.org

Nagios Exchange Wiki

The Nagios Exchange Wiki offers extensive how-tos and tips on monitoring specifics hardware and software platforms. If you stumble across a tip of your own, share it on the Wiki.

http://www.nagiosexchange.org/Wiki.3.0.html

Short Cuts Network Monitoring with Nagios
Short Cuts Network Monitoring with Nagios By Taylor Dondich ISBN: 0596528191
Publisher: O'Reilly
Print Publication Date: 2006/10/01

Copyright

ISBN: 978-0-596-52819-5