Monitoring Network and Service Availability with Open-Source Software

T. Michael Silver

Silver describes the implementation of a monitoring system using an open-source software package to improve the availability of services and reduce the response time when troubles occur. He provides a brief overview of the literature available on monitoring library systems, and then describes the implementation of Nagios, an open-source network monitoring system, to monitor a regional library system's servers and wide area network. Particular attention is paid to using the plug-in architecture to monitor library services effectively. The author includes example displays and configuration files.

Editor's note: This article is the winner of the LITA/Ex Libris Writing Award, 2009.

Library IT departments have an obligation to provide reliable services both during and after normal business hours. The IT industry has developed guidelines for the management of IT services, but the library community has been slow to adopt these practices. The delay may be attributed to a number of factors, including a dependence on vendors and consultants for technical expertise, a reliance on librarians who have little formal training in IT best practices, and a focus on automation systems instead of infrastructure. Larger systems that employ dedicated IT professionals to manage the organization's technology resources likely implement best practices as a matter of course and see no need to discuss them within the library community.

In *The Practice of System and Network Administration*, Thomas A. Limoncelli, Christine J. Hogan, and Strata R. Chalup present a comprehensive look at best practices in managing systems and networks. Early in the book they provide a short list of first steps toward improving IT services, one of which is the implementation of some form of monitoring. They point out that without monitoring, systems can be down for extended periods before administrators notice or users report the problem.[1] They dedicate an entire chapter to monitoring services. In it, they discuss the two primary types of monitoring—real-time monitoring, which provides information on the current state of services, and historical monitoring, which provides long-term data on uptime, use, and performance.[2] While the software discussed in this article provides both types of monitoring, I focus on real-time monitoring and the value of problem identification and notification.

T. Michael Silver (michael.silver@ualberta.ca) is an MLIS student, School of Library and Information Studies, University of Alberta, Edmonton, Alberta, Canada.

Service monitoring does not appear frequently in library literature, and what is written often relates to single-purpose custom monitoring. An article in the September 2008 issue of *ITAL* describes the development and deployment of a wireless network, including a Perl script written to monitor the wireless network and associated services.[3] The script updates a webpage to display the results and sends an e-mail notifying staff of problems. An enterprise monitoring system could perform these tasks and present the results within the context of the complete infrastructure. It would require using advanced features because of the segregation of networks discussed in their article but would require little or no extra effort than it took to write the single-purpose script.

Dave Pattern at the University of Huddersfield shared another Perl script that monitors OPAC functionality.[4] Again, the script provided a single-purpose monitoring solution that could be integrated within a larger model. Below, I discuss how I modified his script to provide more meaningful monitoring of our OPAC than the stock webpage monitoring plug-in included with our open-source networks monitoring system, Nagios.

Service monitoring can consist of a variety of tests. In its simplest form, a ping test will verify that a host (server or device) is powered on and successfully connected to the network. Feher and Sondag used ping tests to monitor the availability of the routers and access points on their network, as do I for monitoring connectivity to remote locations.[5] A slightly more meaningful check would test for the establishment of a connection on a port. Feher and Sondag used this method to check the daemons in their network.[6] The step further would be to evaluate a service response, for example checking the status code returned by a Web server. Evaluating content forms the next level of meaning. Limoncelli, Hogan, and Chalup discuss end-to-end monitoring, where the monitoring system actually performs meaningful transactions and evaluates the results.[7]

Pattern's script, mentioned above, tests OPAC functionality by submitting a known keyword search and evaluating the response.[8] I implemented this after an incident where Nagios failed to alert me to a problem with the OPAC. The Web server returned a status code of 200 to the request for the search page. Users, however, want more from an OPAC, and attempts to search were unsuccessful because of problems with the index server. Modifying Pattern's original script, I was able to put together a custom check command that verifies a greater level of functionality by evaluating the number of results for the known search.

Software selection

Limoncelli, Hogan, and Chalup do not address specific

how-to issues and rarely mention specific products. Their book provides the foundational knowledge necessary to identify what must be done. In terms of monitoring, they leave the selection of an appropriate tool to the reader.[9] Myriad monitoring tools exist, both commercial and open-source. Some focus on network analysis, and some even target specific brands or model lines. The selection of a specific software package should depend on the services being monitored and the goals for the monitoring.

Wikipedia lists thirty-five different products, of which eighteen are commercial (some with free versions with reduced functionality or features); fourteen are open-source projects under a General Public License or similar license (some with commercial support available but without different feature sets or licenses); and three offer different versions under different licenses.[10] Von Hagen and Jones suggest two of them: Nagios and Zabbix.[11]

I selected the Nagios open-source product (http://www.nagios.org). The software has an established history of active development, a large and active user community, a significant number of included and user-contributed extensions, and multiple books published on its use. Commercial support is available from a company founded by the creator and lead developer as well as other authorized solution providers. Monitoring appliances based on Nagios are available, as are sensors designed to interoperate with Nagios. Because of the flexibility of a software design that uses a plug-in architecture, service checks for library-specific applications can be implemented. If a check or action can be scripted using practically any protocol or programming language, Nagios can monitor it. Nagios also provides a variety of information displays, as shown in appendixes A–E.

Installation

The Nagios system provides an extremely flexible solution to monitor hosts and services. The object-orientation and use of plug-ins allows administrators to monitor any aspect of their infrastructure or services using standard plug-ins, user-contributed plug-ins, or custom scripts. Additionally, the open-source nature of the package allows independent development of extensions to add features or integrate the software with other tools. Community sites such as MonitoringExchange (formerly Nagios Exchange), Nagios Community, and Nagios Wiki provide repositories of documentation, plug-ins, extensions, and other tools designed to work with Nagios.[12] But that flexibility comes at a cost—Nagios has a steep learning curve, and user-contributed plug-ins often require the installation of other software, most notably Perl modules.

Nagios runs on a variety of Linux, Unix, and Berkeley Software Distribution (BSD) operating systems. For testing, I used a standard Linux server distribution installed on a virtual machine. Virtualization provides an easy way to test software, especially if an alternate operating system is needed. If given sufficient resources, a virtual machine is capable of running the production instance of Nagios.

After installing and updating the operating system, I installed the following packages:

- Apache Web server
- Perl
- GD development library, needed to produce graphs and status maps
- libpng-devel and libjpeg-devel, both needed by the GD library
- gcc and GNU make, which are needed to compile some plug-ins and Perl modules

Most major Linux and BSD distributions include Nagios in their software repositories for easy installation using the native package management system. Although the software in the repositories is often not the most recent version, using these repositories simplifies the installation process. If a reasonably recent version of the software is available from a repository, I will install from there. Some software packages are either outdated or not available, and I manually install these. Detailed installation instructions are available on the Nagios website, in several books, and on the previously mentioned websites.[13] The documentation for version 3 includes a number of quick-start guides.[14] Most package managers will take care of some of the setup, including modifying the Apache configuration file to create an alias available at http://server.name/nagios. I prepared the remainder of this article using the latest stable versions of Nagios (3.0.6) and the plug-ins (1.4.13) at the time of writing.

Configuration

Nagios configuration relies on an object model, which allows a great deal of flexibility but can be complex. Planning your configuration beforehand is highly recommended.

Nagios has two main configuration files, cgi.cfg and nagios.cfg. The former is primarily used by the Web interface to authenticate users and control access, and it defines whether authentication is used and which users can access what functions. The latter is the main configuration file and controls all other program operations. The cfg_file and cfg_dir directives allow the configuration to be split into manageable groupsusing additional recourse files and the object definition files (see figure 1). The flexibility offered allows a variety of different structures. I group network

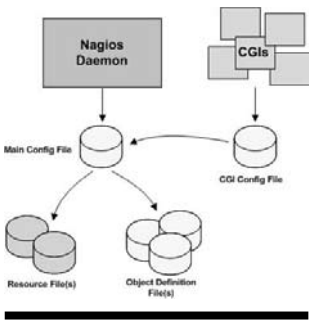


Figure 1. Nagios configuration relationships. Copyright © 2009 Ethan Galstead, Nagios Enterprises. Used with permission.

devices into groups but create individual files for each server.

Nagios uses an object-oriented design. The objects in Nagios are displayed in table 1.

A complete review of Nagios configuration is beyond the scope of this article. The documentation installed with Nagios covers it in great detail. Special attention should be paid to the concepts of templates and object inheritance as they are vital to creating a manageable configuration. The discussion below provides a brief introduction, while appendixes F–J provide concrete examples of working configuration files.

Table 1. Nagios objects

Object	Used for
hosts	servers or devices being monitored
hostgroups	groups of hosts
services	services being monitored
servicegroups	groups of services
timeperiods	scheduling of checks and notifications
commands	checking hosts and services
	notifying contacts
	processing performance data
	event handling
contacts	individuals to alert
contactgroups	groups of contacts

cgi.cfg

The cgi.cfg file controls the Web interface and its associated CGI (Common Gateway Interface) programs. During testing, I often turn off authentication by setting use_authentication to 0 if the Web interface is not accessible from the Internet. There also are various configuration directives that provide greater control over which users can access which features. The users are defined in the /etc/nagios/htpasswd.users file. A summary of commands to control entries is presented in table 2.

The Web interface includes other features, such as sounds, status map displays, and integration with other products. Discussion of these directives is beyond the scope of this article. The cgi.cfg file provided with the software is well commented, and the Nagios documentation provides additional information. A number of screenshots from the Web interface are provided in the appendixes, including status displays and reporting.

nagios.cfg

The nagios.cfg file controls the operation of everything except the Web interface. Although it is possible to have a single monolithic configuration file, organizing the configuration into manageable files works better. The two main directives of note are cfg_file, which defines a single file that should be included, and cfg_dir, which includes all files in the specified directory with a .cfg extension. A third type of file that gets included is resource.cfg, which defines various macros for use in commands.

Organizing the object files takes some thought. I monitor more than one hundred services on roughly seventy hosts, so the method of organizing the files was of more than academic interest. I use the following configuration files:

- commands.cfg, containing command definitions
- contacts.cfg, containing the list of contacts and associated information, such as e-mail address, (see appendix H)
- groups.cfg, containing all groups—hostgroups, servicegroups, and contactgroups, (see appendix G)
- templates.cfg, containing all object templates, (see appendix F)
- timeperiods.cfg, containing the time ranges for checks and notifications

All devices and servers that I monitor are placed in directories using the cfg_dir directive:

Servers—Contains server configurations. Each file includes the host and service configurations for a physical or virtual server.

Devices—Contains device information. I create individual files for devices with service monitoring that goes beyond simple ping tests for connectiv-

Table 2. Sample commands for managing the htpasswd.users file

Create or modify an entry, with password entered at a prompt: htpasswd /etc/nagios/htpasswd.users <username>
Create or modify an entry using password from the command line: htpasswd -b /etc/nagios/htpasswd.users <username> <password>
Delete an entry from the file: htpasswd -D /etc/nagios/htpasswd.users <username>

ity. Devices monitored solely for connectivity are grouped logically into a single file. For example, we monitor connectivity with fifty remote locations, and all fifty of them are placed in a single file.

The resource.cfg file uses two macros to define the path to plug-ins and event handlers. Thirty other macros are available. Because the CGI programs do not read the resource file, restrictive permissions can be applied to them, enabling some of the macros to be used for usernames and passwords needed in check commands. Placing sensitive information in service configurations exposes them to the Web server, creating a security issue.

Configuration

The appendixes include the object configuration files for a simple monitoring situation. A switch is monitored using a simple ping test (see appendix J), while an opac server on the other side of the switch is monitored for both Web and Z39.50 operations (see appendix I). Note that the opac configuration includes a parents directive that tells Nagios that a problem with the gateway-switch will affect connectivity with the opac server. I monitor fifty remote sites. If my router is down, a single notification regarding my router provides more information if it is not buried in a storm of notifications about the remote sites.
The Web port, Web service, and opac search services demonstrate different levels of monitoring. The Web port simply attempts to establish a connection to port 80 without evaluating anything beyond a successful connection. The Web service check requests a specific page from the Web server and evaluates only the status code returned by the server. It displays a warning because I configured the check to download a file that does not exist. The Web server is running because it returns an error code, hence the warning status. The opac search uses a known search to evaluate the result content, specifically whether the correct number of results is returned for a known search.

I used a number of templates in the creation of this configuration. Templates reduce the amount of repetitive typing by allowing the reuse of directives. Templates can be chained, as seen in the host templates. The opac definition uses the Linux-server template, which in turn uses the generic-host template. The host definition inherits the directives of the template it uses, overriding any elements in both and adding new elements. In practical terms, generic-host directives are read first. Linux-server directives are applied next. If there is a conflict, the Linux-server directive takes precedence. Finally, opac is read. Again, any conflicts are resolved in favor of the last configuration read, in this case opac.

Plug-ins and service checks

The nagios plugins package provides numerous plug-ins, including the check-host-alive, check_ping, check_tcp, and check_http commands. Using the plug-ins is straightforward, as demonstrated in the appendixes. Most plug-ins will provide some information on use if executed with—help supplied as an argument to the command. By default, the plug-ins are installed in /usr/lib/nagios/plugins. Some distributions may install them in a different directory.

The plugins folder contains a subfolder with user-contributed scripts that have proven useful. Most of these plug-ins are Perl scripts, many of which require additional Perl modules available from the Comprehensive Perl Archive Network (CPAN). The check_hip_search plug-in (appendix K) used in the examples requires additional modules. Installing Perl modules is best accomplished using the CPAN Perl module. Detailed instructions on module installation are available online.[15] Some general tips:

- Gcc and make should be installed before trying to install Perl modules, regardless of whether you are installing manually or using CPAN. Most modules are provided as source code, which may require compiling before use. CPAN automates this process but requires the presence of these packages.
- Alternately, many Linux distributions provide Perl module packages. Using repositories to install usually works well assuming the repository has all the needed modules. In my experience, that is rarely the case.

- Many modules depend on other modules, sometimes requiring multiple install steps. Both CPAN and distribution package managers usually satisfy dependencies automatically. Manual installation requires the installer to satisfy the dependencies one by one.
- Most plug-ins provide information on required software, including modules, in a readme file or in the source code for the script. In the absence of such documentation, running the script on the command line usually produces an error containing the name of the missing module.
- Testing should be done using the nagios user. Using another user account, especially the root user, to create directories, copy files, and run programs creates folders and files that are not accessible to the nagios user. The best practice is to use the nagios user for as much of the configuration and testing as possible. The lists and forums frequently include questions from new users that have successfully installed, configured, and tested Nagios as the root user and are confused when Nagios fails to start or function properly.

Advanced topics

Once the system is running, more advanced features can be explored. The documentation describes many such enhancements, but the following may be particularly useful depending on the situation.

- Nagios provides access control through the combination of settings in the cgi.cfg and htpasswd.users files. Library administration and staff, as well as patrons, may appreciate the ability to see the status of the various systems. However, care should be taken to avoid disclosing sensitive information regarding the network or passwords, or allowing access to CGI programs that perform actions.
- Nagios permits the establishment of dependency relationships. Host dependencies may be useful in some rare circumstances not covered by the parent–child relationships mentioned above, but service dependencies provide a method of connecting services in a meaningful manner. For example, certain OPAC functions are dependent on ILS services. Defining these relationships takes both time and thought, which may be worthwhile depending on any given situation.
- Event handlers allow Nagios to initiate certain actions after a state change. If Nagios notices that a particular service is down, it can run a script or program to attempt to correct the problem. Care should be taken when creating these scripts as service restarts may delete or overwrite information critical to solving a problem, or worsen the actual situation if an attempt to restart a service or reboot a server fails.
- Nagios provides notification escalations, permitting the automatic notification of problems that last longer than a certain time. For example, a service escalation could send the first three alerts to the admin group. If properly configured, the fourth alert would be sent to the managers group as well as the admin group. In addition to escalating issues to management, this feature can be used to establish a series of responders for multiple on-call personnel.
- Nagios can work in tandem with remote machines. In addition to custom scripts using Secure Shell (SSH), the Nagios Remote Plug-in Executor (NRPE) add-on allows the execution of plug-ins on remote machines, while the Nagios Service Check Acceptor (NSCA) add-on allows a remote host to submit check results to the Nagios server for processing. Implementing Nagios on the Feher and Sondag wireless network mentioned earlier would require one of these options because the wireless network is not accessible from the external network. These add-ons also allow for distributed monitoring, sharing the load among a number of servers while still providing the administrators with a single interface to the entire monitored network. The Nagios Exchange (http://exchange.nagios.org/) contains similar user-contributed programs for Windows.
- Nagios can be configured to provide redundant or failover monitoring. Limoncelli, Hogan, and Chalup call this metamonitoring and describe when it is needed and how it can be implemented, suggesting self-monitoring by the host or having a second monitoring system that only monitors the main system.[16] Nagios permits more complex configurations, allowing for either two servers operating in parallel, only one of which sends notifications unless the main server fails, or two servers communicating to share the monitoring load.
- Alternative means of notification increase access to information on the status of the network. I implemented another open-source software package, QuickPage, which allows Nagios text messages to be sent from a computer to a pager or cell phone.[17] Appendix L shows a screenshot of a Firefox extension that displays host and service problems in the status bar of my browser and provides optional audio alerts.[18] The Nagios community has developed a number of alternatives, including specialized Web interfaces and RSS feed generators.[19]

Appropriate use

Monitoring uses bandwidth and adds to the load of machines being monitored. Accordingly, an IT department should only monitor its own servers and devices, or those for which it has permission to do so. Imagine what would happen if all the users of a service such as WorldCat started monitoring it! The additional load would be noticeable and could conceivably disrupt service.

Aside from reasons connected with being a good "netizen," monitoring appears similar to port-scanning, a technique used to discover network vulnerabilities. An organization that blithely monitors devices without the owner's permission may find their traffic is throttled back or blocked entirely. If a library has a definite need to monitor another service, obtaining permission to do so is a vital first step. If permission is withheld, the service level agreement between the library and its service provider or vendor should be reevaluated to ensure that the provider has an appropriate system in place to respond to problems.

Benefits

The system-administration books provide an accurate overview of the benefits of monitoring, but personally reaping those benefits provides a qualitative background to the experience. I was able to justify the time spent on setting up monitoring the first day of production. One of the available plug-ins monitors Sybase database servers. It was one of the first contributed plug-ins I implemented because of past experiences with our production database running out of free space, causing the system to become nonfunctional. This happened twice, approximately a year apart. Each time, the integrated library system was down while the vendor addressed the issue. When I enabled the Sybase service checks, Nagios immediately returned a warning for the free space. The advance warning allowed me to work with the vendor to extend the database volume with no downtime for our users. That single event convinced the library director of the value of the system.

Since that time, Nagios has proven its worth in alerting IT staff to problem situations, providing information on outage patterns both for in-house troubleshooting and discussions with service providers.

Conclusion

Monitoring systems and services provides IT staff with a vital tool in providing quality customer service and managing systems. Installing and configuring such a system involves a learning curve and takes both time and computing resources. My experiences with Nagios have convinced me that the return on investment more than justifies the costs.

References

1. Thomas A. Limoncelli, Christina J. Hogan, and Strata R. Chalup, *The Practice of System and Network Administration*, 2nd ed. (Upper Saddle River, N.J.: Addison-Wesley, 2007): 36.

2. Ibid., 523–42.

3. James Feher and Tyler Sondag, "Administering an Open-Source Wireless Network," *Information Technology & Libraries* 27, no. 3 (Sept. 2008): 44–54.

4. Dave Pattern, "Keeping an Eye on Your HIP," online posting, Jan. 23, 2007, Self-Plagiarism is Style, http://www.daveyp.com/blog/archives/164 (accessed Nov. 20, 2008).

5. Feher and Sondag, "Administering an Open-Source Wireless Network," 45–54.

6. Ibid., 48, 53–54.

7. Limoncelli, Hogan, and Chalup, *The Practice of System and Network Administration*, 539–40.

8. Pattern, "Keeping an Eye on Your HIP."

9. Limoncelli, Hogan, and Chalup, *The Practice of System and Network Administration*, xxv.

10. "Comparison of Network Monitoring Systems," Wikipedia, The Free Encyclopedia, Dec. 9, 2008, http://en.wikipedia.org/wiki/Comparison_of_network_monitoring_systems (accessed Dec. 10, 2008).

11. William Von Hagen and Brian K. Jones, *Linux Server Hacks*, Vol. 2 (Sebastopol, Calif.: O'Reilly, 2005): 371–74 (Zabbix), 382–87 (Nagios).

12. MonitoringExchange, http://www.monitoringexchange.org/ (accessed Dec. 23, 2009); Nagios Community, http://community.nagios.org (accessed Dec. 23, 2009); Nagios Wiki, http://www.nagioswiki.org/ (accessed Dec. 23, 2009).

13. "Nagios Documentation," Nagios, Mar. 4, 2008, http://www.nagios.org/docs/ (accessed Dec. 8, 2008); David Josephsen, *Building a Monitoring Infrastructure with Nagios* (Upper Saddle River, N.J.: Prentice Hall, 2007); Wolfgang Barth, *Nagios: System and Network Monitoring*, U.S. ed. (San Francisco: Open Source Press; No Starch Press, 2006).

14. Ethan Galstead, "Nagios Quickstart Installation Guides," Nagios 3.x Documentation, Nov. 30, 2008, http://nagios.sourceforge.net/docs/3_0/quickstart.html (accessed Dec. 3, 2008).

15. The Perl Directory, (http://www.perl.org/) contains complete information on Perl. Specific information on using CPAN is available in "How Do I Install a Module from CPAN?" perlfaq8, Nov. 7, 2007, http://perldoc.perl.org/perlfaq8.html (accessed Dec. 4, 2008).

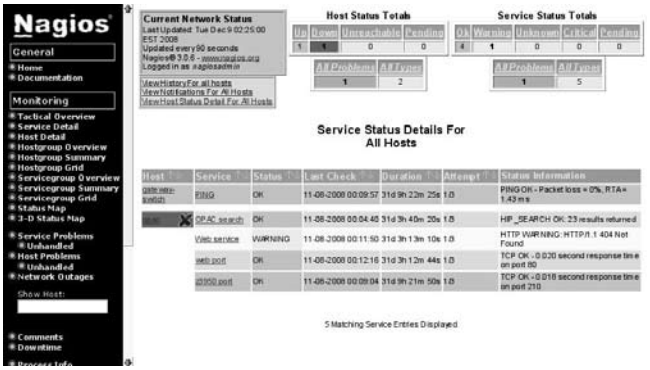
16. Limoncelli, Hogan, and Chalup, *The Practice of System and Network Administration*, 539–40.

17. Thomas Dwyer III, QPage Solutions, http://www.qpage.org/ (accessed Dec. 9, 2008).

18. Petr Šimek, "Nagioschecker," Google Code, Aug. 12, 2008, http://code.google.com/p/nagioschecker/ (accessed Dec. 8, 2008).

19. "Notifications," MonitoringExchange, http://www.monitoringexchange.org/inventory/Utilities/AddOn-Projects/Notifications (accessed Dec. 23, 2009).

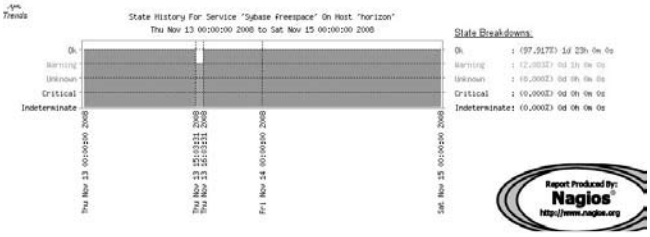
Appendix A. Service detail display from test system



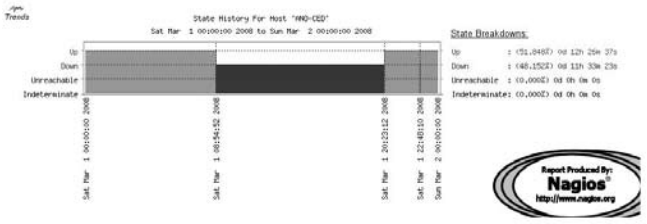
Appendix B. Service details for OPAC (hip) and ILS (horizon) servers from production system

hip	HIP search	OK	12-07-2008 17:06:50	39d 3h 6m 48s	1/4	HIP_SEARCH OK: 23 results returned
	Search daemon	OK	12-07-2008 17:03:35	39d 3h 7m 17s	1/4	TCP OK - 0.019 second response time on port 12502
	Z39.50	OK	12-07-2008 17:06:50	41d 1h 44m 45s	1/4	OK: found 22 records in 1 seconds
horizon	Sybase HIP logins	OK	12-07-2008 17:03:35	221d 5h 44m 44s	1/4	OK: Number of logins within thresholds (100 logins)
	Sybase connect	OK	12-07-2008 17:06:50	185d 3h 25m 41s	1/4	OK: Connect okay
	Sybase freespace	OK	12-07-2008 17:03:36	24d 1h 4m 32s	1/4	OK: Free space within thresholds (31.39% free)
	Sybase non-HIP logins	OK	12-07-2008 17:06:50	185d 3h 10m 41s	1/4	OK: Number of logins within thresholds (6 logins)

Appendix C. Sybase freespace trends for a specified period



Appendix D. Connectivity history for a specified period



Appendix E. Availability report for host shown in Appendix D

Host State Breakdowns:

State	Type / Reason	Time	% Total Time	% Known Time
UP	Unscheduled	0d 12h 26m 37s	51.848%	51.848%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**0d 12h 26m 37s**	**51.848%**	**51.848%**
DOWN	Unscheduled	0d 11h 33m 23s	48.152%	48.152%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**0d 11h 33m 23s**	**48.152%**	**48.152%**
UNREACHABLE	Unscheduled	0d 0h 0m 0s	0.000%	0.000%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**0d 0h 0m 0s**	**0.000%**	**0.000%**
Undetermined	Nagios Not Running	0d 0h 0m 0s	0.000%	
	Insufficient Data	0d 0h 0m 0s	0.000%	
	Total	0d 0h 0m 0s	0.000%	
All	Total	1d 0h 0m 0s	100.000%	100.000%

Appendix F. templates.cfg file

```
##############################################################################
# TEMPLATES.CFG - SAMPLE OBJECT TEMPLATES
##############################################################################

##############################################################################
# CONTACT TEMPLATES
##############################################################################
```

Appendix F. templates.cfg file (cont.)

```
# Generic contact definition template - This is NOT a real contact, just
# a template!
define contact{
   name                            generic-contact

   service_notification_period   24x7
   host_notification_period      24x7
   service_notification_options  w,u,c,r,f,s
   host_notification_options     d,u,r,f,s
   service_notification_commands notify-service-by-email
   host_notification_commands    notify-host-by-email
   register                       0
   }

##############################################################################
# HOST TEMPLATES
##############################################################################

# Generic host definition template - This is NOT a real host, just
# a template!
define host{
   name                              generic-host
   notifications_enabled            1
   event_handler_enabled             1
   flap_detection_enabled           1
   failure_prediction_enabled        1
   process_perf_data                 1
   retain_status_information         1
   retain_nonstatus_information      1
   notification_period             24x7
   register                          0
   }

# Linux host definition template - This is NOT a real host, just a template!
define host{
      name                    linux-server
      use                    generic-host
      check_period            24x7
      check_interval          5
      retry_interval          1
      max_check_attempts     10
      check_command                check-host-alive
      notification_period    workhours
      notification_interval  120
      notification_options   d,u,r
      contact_groups         admins
      register               0
      }
```

Appendix F. templates.cfg file (cont.)

```
# Define a template for switches that we can reuse
define host{
      name                    generic-switch
      use                     generic-host
      check_period            24x7
      check_interval          5
      retry_interval          1
      max_check_attempts      10
      check_command           check-host-alive
      notification_period     24x7
      notification_interval   30
      notification_options    d,r
      contact_groups          admins
      register                0
      }

##############################################################################
# SERVICE TEMPLATES
##############################################################################

# Generic service definition template - This is NOT a real service,
# just a template!
define service{
   name                            generic-service
   active_checks_enabled           1
   passive_checks_enabled          1
   parallelize_check               1
   obsess_over_service             1
   check_freshness                 0
   notifications_enabled           1
   event_handler_enabled           1
   flap_detection_enabled          1
   failure_prediction_enabled      1
   process_perf_data               1
   retain_status_information       1
   retain_nonstatus_information    1
   is_volatile                     0
   check_period                    24x7
   max_check_attempts              3
   normal_check_interval           10
   retry_check_interval            2
   contact_groups                  admins
   notification_options              w,u,c,r
   notification_interval           60
   notification_period             24x7
   register                        0
   }
```

Appendix F. templates.cfg file (cont.)

```
# Define a ping service. This is NOT a real service, just a template!
define service{
      use                   generic-service
      name                  ping-service
      notification_options  n
      check_command          check_ping!1000.0,20%!2000.0,60%
      register              0
      }
```

Appendix G. groups.cfg file

```
##############################################################################
# CONTACT GROUP DEFINITIONS
##############################################################################
# We only have one contact in this simple configuration file, so there is
# no need to create more than one contact group.
define contactgroup{
        contactgroup_name       admins
        alias                   Nagios Administrators
        members                 nagiosadmin
        }

##############################################################################
# HOST GROUP DEFINITIONS
##############################################################################
# Define an optional hostgroup for Linux machines
define hostgroup{
        hostgroup_name  linux-servers ; The name of the hostgroup
        alias           Linux Servers ; Long name of the group
        }

# Create a new hostgroup for ILS servers
define hostgroup{
      hostgroup_name  ils-servers           ; The name of the hostgroup
      alias      ILS servers     ; Long name of the group
      }

# Create a new hostgroup for switches
define hostgroup{
      hostgroup_name  switches        ; The name of the hostgroup
      alias      Network Switches; Long name of the group
      }

##############################################################################
# SERVICE GROUP DEFINITIONS
##############################################################################
```

Appendix G. groups.cfg file (cont.)

```
# Define a service group for network connectivity
define servicegroup{
      servicegroup_name     network
      alias            Network infrastructure services
      }

# Define a servicegroup for ILS
define servicegroup{
      servicegroup_name     ils-services
      alias            ILS related services
      }
```

Appendix H. contacts.cfg

```
##############################################################################
# CONTACTS.CFG - SAMPLE CONTACT/CONTACTGROUP DEFINITIONS
##############################################################################

# Just one contact defined by default - the Nagios admin (that's you)
# This contact definition inherits a lot of default values from the
# 'generic-contact' template which is defined elsewhere.
define contact{
     contact_name     nagiosadmin
     use                generic-contact
     alias            Nagios Admin
     email            nagios@localhost
        }
```

Appendix I. opac.cfg

```
##############################################################################
# OPAC SERVER
##############################################################################

##############################################################################
# HOST DEFINITION
##############################################################################

# Define a host for the server we'll be monitoring
# Change the host_name, alias, and address to fit your situation
define host{
      use        linux-server
      host_name  opac
      parents    gateway-switch
      alias      OPAC server
```

Appendix I. opac.cfg (cont.)

```
      address    192.168.1.123
      }

##############################################################################
# SERVICE DEFINITIONS
##############################################################################

# Create a service for monitoring the HTTP port
define service{
      use                     generic-service
      host_name               opac
      service_description     web port
      check_command           check_tcp!80
      }

# Create a service for monitoring the web service
define service{
      use                    generic-service
      host_name               opac
      service_description     Web service
      check_command           check_http!-u/bogusfilethatdoesnotexist.html
      }

# Create a service for monitoring the opac search
define service{
      use                     generic-service
      host_name               opac
      service_description     OPAC search
      check_command           check_hip_search
      }

# Create a service for monitoring the Z39.50 port
define service{
      use                     generic-service
      host_name               opac
      service_description     z3950 port
      check_command           check_tcp!210
      }
```

Appendix J. switches.cfg

```
##############################################################################
# SWITCH.CFG - SAMPLE CONFIG FILE FOR MONITORING SWITCHES
##############################################################################

##############################################################################
# HOST DEFINITIONS
##############################################################################
```

Appendix J. switches.cfg

```
# Define the switch that we'll be monitoring
define host{
      use             generic-switch
      host_name  gateway-switch
      alias      Gateway Switch
      address          192.168.0.1
      hostgroups switches
      }

################################################################################
###
# SERVICE DEFINITIONS
################################################################################
###

# Create a service to PING to switches
# Note this entry will ping every host in the switches hostgroup
define service{
      use                           ping-service
      hostgroups                 switches
      service_description       PING
      normal_check_interval 5
      retry_check_interval  1
      }
```

Appendix K. check_hip_search script

```
#!/usr/bin/perl -w
#########################
# Check Horizon Information Portal (HIP) status.
# HIP is the web-based interface for Dynix and Horizon
# ILS systems by SirsiDynix corporation.
#
# This plugin is based on a standalone Perl script written
# by Dave Pattern. Please see
#    http://www.daveyp.com/blog/index.php/archives/164/
# for the original script.
#
# The original script and this derived work are covered by
# http://creativecommons.org/licenses/by-nc-sa/2.5/
#########################

   use strict;
   use LWP::UserAgent;       # Note the requirement for Perl module
LWP::UserAgent!
   use lib "/usr/lib/nagios/plugins";
   use utils qw($TIMEOUT %ERRORS);
```

Appendix K. check_hip_search script (cont.)

```
### Some configuration options

   my $hipServerHome      = "http://ipac.prl.ab.ca/ipac20/ipac.
jsp?profile=alap";
   my $hipServerSearch      = "http://ipac.prl.ab.ca/ipac20/ipac.jsp?menu=se
arch&aspect=subtab132&npp=10&ipp=20&spp=20&profile=alap&ri=&index=.GW&term=li
nux&x=18&y=13&aspect=subtab132&GetXML=true";
   my $hipSearchType      = "xml";
   my $httpProxy         = '';

### check home page is available...

{
   my $ua = LWP::UserAgent->new;
   $ua->timeout( 10 );
   if( $httpProxy ) { $ua->proxy( 'http', $httpProxy ) }
        my $response = $ua->get( $hipServerHome );
        my $status = $response->status_line;
        if( $response->is_success )
        {
        }
        else
        {
            print "HIP_SEARCH CRITICAL: $status\n";
            exit $ERRORS{'CRITICAL'};
        }
}

### check search page is returning results...
   {
      my $ua = LWP::UserAgent->new;
      $ua->timeout( 10 );
      if( $httpProxy )
         { $ua->proxy( 'http', $httpProxy ) }

      my $response = $ua->get( $hipServerSearch );
      my $status   = $response->status_line;

      if( $response->is_success )
      {
         my $results = 0;
         my $content = $response->content;

         if( lc( $hipSearchType ) eq 'html' )
         {
            if ( $content =~ /\<b\>(\d+?)\<\/b\>\ \;titles matched/ )
            {
               $results = $1;
```

Appendix K. check_hip_search script (cont.)

```
            }
         }

         if( lc( $hipSearchType ) eq 'xml' )
         {
            if( $content =~ /\<hits\>(\d+?)\<\/hits\>/ )
            {
               $results = $1;
            }
         }

### Modified section - original script triggered another function to
### save results to a temp file and email an administrator.
         unless( $results )
         {
            print "HIP_SEARCH CRITICAL: No results returned|results=0\n";
            exit $ERRORS{'CRITICAL'};
         }

         if ( $results )
         {
            print "HIP_SEARCH OK: $results results
returned|results=$results\n";
            exit $ERRORS{'OK'};
         }
   }
}
```

Appendix L. Nagios Checker display