Network Monitoring With Nagios

Adam Spencer Garside

Abstract

This article discusses how Nagios, an Open Source monitoring framework, can be used to react to potential system failures and proactively forsee future failure via trending.

1. Introduction

Nagios[1], an Open Source monitoring framework developed by Ethan Galstad and a team of developers, is used at Central Piedmont Community College to monitor routers, switches, firewalls, servers, and applications. The framework uses a plugin architecture layered around a robust scheduler to allow arbitrary checks written in arbitrary languages. Failed checks can generate alerts or trouble tickets allowing for quicker response to system failures. Furthermore, the framework provides trending and reporting as well as visualization tools for easily identifying problem spots as they are happening. A simple but concise CGI frontend (See Figure 1) is provided to allow authenticated users to manage most parts of the system, including scheduling downtime, adding comments to services, and acknowledging alerts and problems.

Nagios is developed on GNU/Linux systems but should work on any UNIX or Unix-compatible system (such as FreeBSD or OpenBSD). System requirements are very modest; before moving to an IBM HS20 blade server, we were monitoring 400+ hosts and over 600 individual services using a single Dell GX110 workstation.

This article will cover the following items

- Why CPCC chose Nagios
- How Nagios has helped CPCC better understand its network
- CPCC's Nagios Implementation
- Other software used with Nagios

2. Why CPCC chose Nagios

Before 2002, CPCC was using a relativly inexpensive commercial monitoring suite. We started to see limitations of the product when we needed to monitor not only service ports but also applications. The product had no ability to write custom checks and the inability to do a full application checks was a problem since serveral applications would fail but still pass open port checks.

We looked at a number of other solutions including enterprise packages from HP and Tivoli but found them either too expensive or restrictive. Since we had had good success with FOSS[1] packages in the past, we naturally looked in that arena. In 2002, Nagios was called Netsaint but provided the same features. A quick testing period showed that it was able to not only provide the monitoring functionality of our commercial solution but also to provide the necessary application level check customization we desired.

We quickly replaced our previous solution with Nagios and started creating customized checks. The main customization checks we wished to perform centered around Blackboard which was failing in ways that did not manifest themselves in open port checks. We wanted a check to perform the following functions:

- Connect to Blackboard
- Login as a test user

[1] Free and Open Source Software

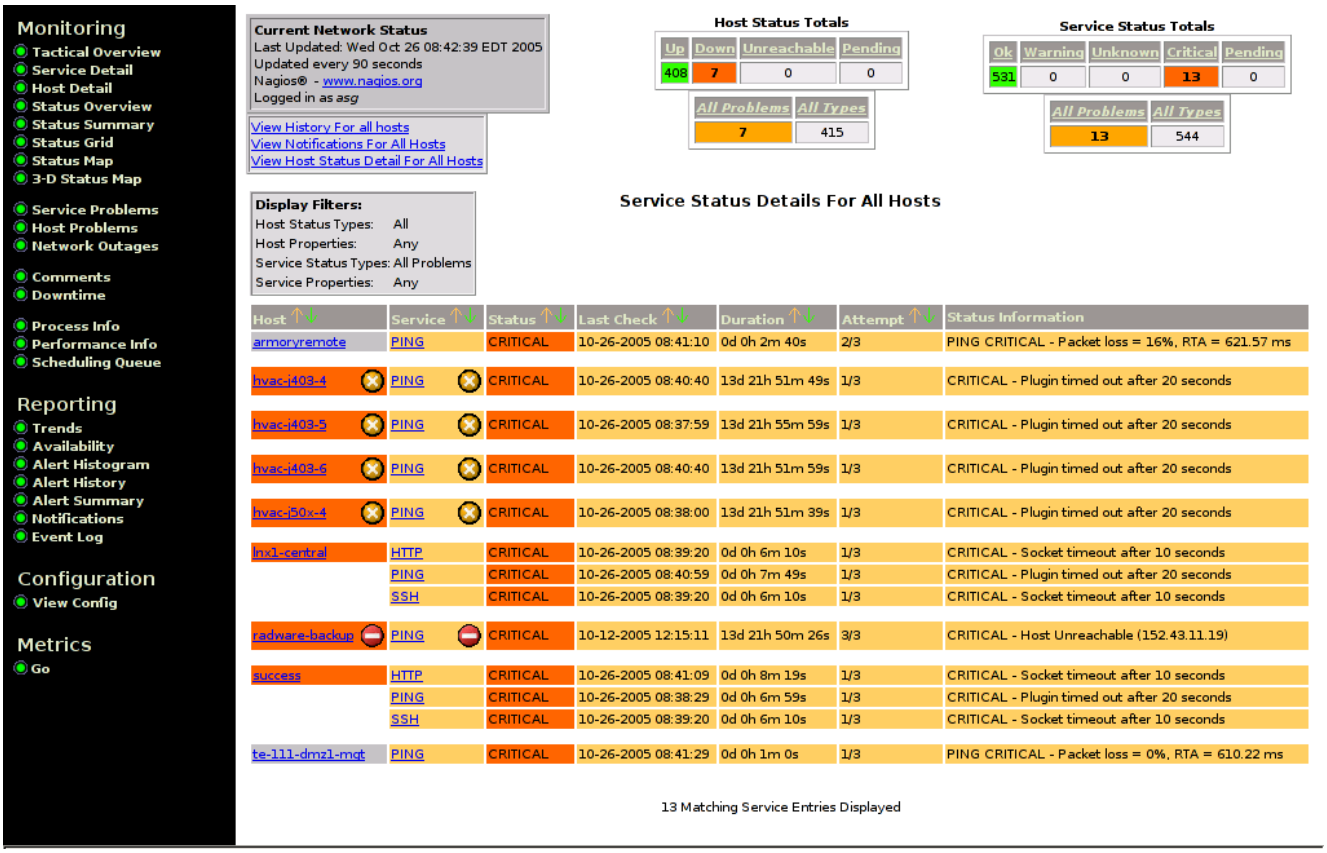

Figure 1: Nagios main screen

- Navigate to a number of key pages
- Logout

While we initially coded our checks in Java, we have since switched to Perl, since we now use Yale's CAS for Single Sign On and Perl's WWW::Mechanize library does a good job of handling the redirects from CAS back to Blackboard with little extra work. The following is a single check that connects to Blackboard, catches the redirect to CAS automatically, authenticates to CAS with a test user, returns to Blackboard and does a content check. Any failure results in the plugin failing causing Nagios to send an alert.

```
#!/usr/bin/perl -W
#
# Depends: libwww-mechanize-perl
#
#
# Login script to test Blackboard Front-ends
# Adam Garside <adam.garside@cpcc.edu>
# Matt Rubright <matt.rubright@cpcc.edu>
# Copyright 2005 - Central Piedmont Community College

use strict;
use WWW::Mechanize;
$|++;

my $host        = shift @ARGV;
```

```
my $username    = 'username';
my $password    = 'password';
my $casurl      = "http://$host/webapps/login";
my $mech        = WWW::Mechanize->new( timeout => 20, quiet => 1 );

# Simple error wrapper to fail back to Nagios
sub err {
  my $status = shift(@_);
  print "Login failed: $status\n";
  exit(2);
}

# Connect to CAS and Login
$mech->add_header( 'Host' => 'blackboard.cpcc.edu' );
$mech->get( $casurl );
err($mech->response->status_line) unless $mech->success;

$mech->submit_form(
        form_name       => 'login_form',
        fields          => {
                                username        => $username,
                                password        => $password
                           },
        );

# Follow return URL from Cas
my $link = $mech->find_link( text => 'here' );
my $linktext = $link->url_abs();
$linktext =~ s/blackboard\.cpcc\.edu/$host/;
$mech->add_header( 'Host' => 'blackboard.cpcc.edu' );
$mech->get( $linktext );
err($mech->response->status_line) unless $mech->success;

# Return to Blackboard page ( content frame )
$mech->add_header( 'Host' => 'blackboard.cpcc.edu' );
$mech->get( "http://$host/webapps/portal/tab/_1_1/index.jsp" );
err($mech->response->status_line) unless $mech->success;
my $content = $mech->content( format => "text" );
if ($content =~ m/Welcome, (:?\S)+/) {
   print "Login successful.\n";
   exit 0;
} else {
   err($mech->response->status_line);
}
```

That is a very small amount of custom code that provides a useful application level check. Similar checks are now used to monitor our Moodle LMS, Student and Faculty webmail systems, HVAC controllers and even environmental conditions of our datacenter and switching closets.

3. How Nagios has helped CPCC better understand its network

Before we started using Nagios, we would often find that an application or even a whole server was hung or had crashed and only find out when someone noticed. Now, we instantly know at a glance what issues are occuring and the state of the network as a whole. We also have alerts sent to various pagers to let system administrators and techs deal with problems at any time of the day or night. Our helpdesk has limited access to check the availability of systems and see alert acknowlegements so they can make informative decisions and send global notifications to students and faculty, often without the need for escalation.

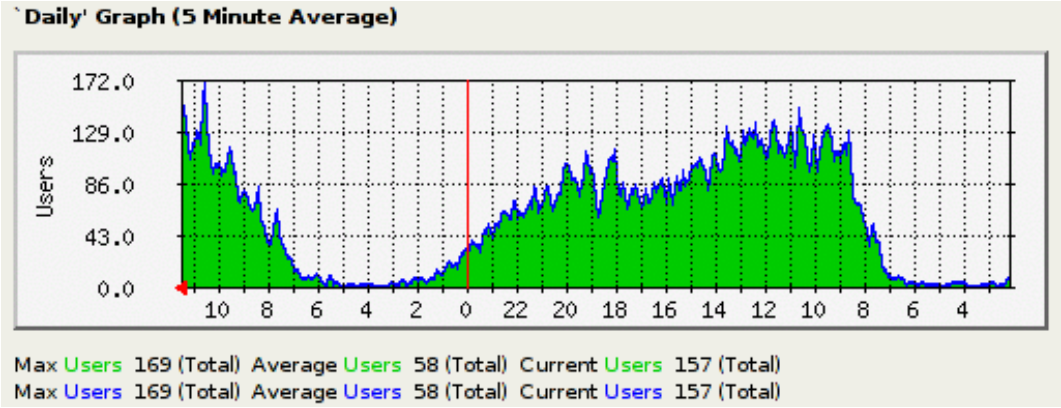

Figure 2: Daily Blackboard users graph generated by MRTG

4. CPCC's Nagios Implementation

CPCC now runs Nagios on a single SAN booted IBM HS20 blade server running Debian GNU/Linux 3.1 (sarge). Even performing close to 1000 checks at this time, the system is only 2% utilized allowing for future growth and expansion.

Each defined Nagios host is given a URL which the interface uses to provide a custom page when the hostname is clicked in various contexts. This is used to specify system informational items such as inventory tags, load graphs (See Figure 2) generated with MRTG[2], physical location, etc. We currently have static data for most systems but expect to be re-implementing the system informational pages using a secure wiki to better facilitate collaboration.

5. Other software used with Nagios

While Nagios provides a full monitorning framework, several other FOSS packages complement it. These include:

- syslog-ng[3] – a more powerful Unix logging daemon
- sec[4] – a simple event correleation daemon

Though the standard and custom Nagios plugins will provide most of the monitoring you will need, there are some systems that cannot be monitored effectively because they don't have the necessary interfaces, for example switches. Most networking hardware can send a SNMP trap when an error occurs and Nagios handles this. However, some events are the result of a number of smaller events that occur in some order or within a certain temporal threshold. In these cases, the use of network logging and event correlation are useful.

All Unix systems ship with some form syslog or syslog compatible logger. Such loggers can also log system events to remote loggers which allows for centralized log management and analysis. Even Windows systems can log to remote Unix loggers.

syslog-ng provides a number of features beyond traditional syslog including

- arbitrary partitioning based on system names, facilities, and even regular expressions
- multiple custom sources, including tcp listeners (for using secure tunnelled loggin), and fifos
- multiple custom outputs sinks

When using syslog-ng on a central logging server, more organized partitioning of saved logs becomes possible making analysis easier. Furthermore, because syslog-ng can output to fifos as well as files, it can be used to provide an analysis stream without taking up drive space. This becomes important when doing event correlation.

sec, or Simple Event Correlator, is a small and very elegant Perl daemon that can monitor multiple files and fifos for events which are defined using a number of builtin primitives. These primitives are as simple as matching a regex or static line of content but can be combined to provide complex correlation processing, such as looking for multiple failed attempts

to connect to a ssh daemon within a certain period of time from a single machine and running a script to block the offending machine at the router or firewall level. Since the Nagios scheduler uses a fifo for event itemization, sec can be used to inject alerts directly based on events.

6. Conclusion

Nagios and other FOSS tools have been invaluable in providing Central Piedmont Community College with a unified overview or network health and have allowed network and system administrators to respond to issues before they are noticed by faculty and staff. While some work is required to setup and maintain a monitoring system, the benefits are vast.

References

[1] *http://nagios.org/*,

[2] *http://people.ee.ethz.ch/ oetiker/webtools/mrtg/*

[3] *http://www.balabit.com/products/syslog-ng/*

[4] *http://kodu.neti.ee/ risto/sec/*