(https://linuxacademy.com/cp/dashboard)
(/cp)

(https://www.cloudassessments.com/o/#/dashboard)

(https://scaleyourcode.com)

19 Latest Updates (https://linuxacademy.com/blog)Refer A Friend (/cp/referFriend)Support xe1phix

Navigation

Back to community

Evan Langlois (/cp/socialize/profile/user/elanglois)

- Bookmark this post
- Sticky this post
- Follow this user
- Share!
- Spam!

How To Guide

Introduction to Nagios
3/23/2017 - COURSE:

Nagios Certified Professional

✹ Instructor Approved

Table of Contents

What is Nagios? It's a tool that is going to make you a rockstar! Have you ever had someone ask you, "Hey, did you know web server 3 has been down all weekend?" If your response was anything other than, "Yeah, I took it out of the rotation for an upgrade", such as, "Uhmm ... It is?", then expect to be fired! It's your job to know! That's why you need Nagios!

Nagios is a server monitoring infrastructure. In the olds days we used to have to run horribly cryptics commands like *sar, ps, du, df,* and others, then parse the output with awk and sed and then figure out how to do something with the data. These commands were never standardized so scripts running under Solaris may not work as expected under SunOS! Things gradually evolved and Nagios came on the scene almost 20 years ago. Nagios is now used to monitor sites with hundreds of thousands of services and you can configure tiered monitoring if you need maximum scalability. Nagios is incredibly flexible, but its greatest strength is also its weakness. Its power comes from the fact that it is based on scripting, allowing you to monitor literally anything. It's notification system is also based on scripts, so you can do whatever you like with the information, even restart services, clean old logs, or whatever. You can even download pre-written scripts to extend Nagios in all sorts of ways. Check the Nagios website for a repository.

This power comes at a cost in added complexity. This document will cover the basics of installation and monitoring the most common aspects of your network. Once you get familiar with the basics, you can check Google to cover the rest. You will need a firm grasp on basic networking, the Unix permission model, how to set up basic mail and networking on your server, and some basic scripting before you can tackle this beast. After that, you'll find that the hooks available will let you easily manage Nagios with DevOps tools, get notifications right on your phone, and give the boss pretty graphs!

It also bears note that you can set up multiple users that have different access rights, and even hide parts of the system from a user. This means you can give your customers read access to their own systems, and only their own systems.

Expect problems. Installation will take at least a couple of hours if things go smoothly. If you run into issues, it could take quite a bit longer.

INSTALLATION

First, if you are doing this for your company and you use RedHat or CentOS servers, I recommend you purchase *Nagios XI*. It's a commercial version and comes pre-configured with extra features and abilities. It's the fastest and easiest way to get started. You can download a virtual machine image (with CentOS) free for 60 days, and you can even have professional installation done for you! Check out the site. Now, onto the hard way!

I normally recommend that you never break the package manager, but many distributions don't have Nagios in the repositories or the distribution is very old. If your distribution is up to date on the latest Nagios and they are known for timely updates, then use your package manager to install Nagios. At the time of this writing, Redhat/Centos and Debian/Ubuntu based distributions use a very old Nagios with a known root privilege escalation! Arch only has the nrpe and plugins to monitor the host from Nagios, not Nagios itself. Gentoo/Funtoo have current versions available to emerge. If you have to add non-standard repositories to get Nagios or the version is less than that described here, then I suggest installing Nagios from source using the instructions given below. You shouldn't trust binaries built from untrusted sources, and this is especially true with Nagios. If you install from source, you can use the "Check For Updates" link from the Web Interface to update your system. If you install from your package manager, continue to use the package manager to maintain Nagios - do not ever update a package from source that was installed by the package manager!

To keep things interesting, these instructions are written and tested on Debian Linux 8 (Jessie) running on a BeagleBone Black. It's my home network's DNS, dlna server, file server, dhcp server, backup server, etc. The Web server used is my favorite, Nginx. Nginx tends to be faster than Apache, uses fewer resources, and scales better. Nagios comes with sample config files for Apache. I'll give you my Nginx one, so you're covered using either web server. Installation should be almost identical on any Debian/Ubuntu based system. There will be minimal changes for other systems - mostly changing apt for your usual package manager and some of the package names in the early steps. After that, it's all the same.

Note: I will use a '\' at the end of a line to show that a long line has been wrapped and is all part of one command. You can type this all on one line without the slash, or just cut and paste since the slash is the shell's normal line continuation character.

Step 1: Pre-install Steps

First, make sure your web server is installed and can serve basic static pages and make sure your mail system can send mail (such as by using the "*mail*" command line tool). If you just need to relay mail through another system or this isn't a mission critical installation, you can get up and running fast by installing SSMTP rather than going through the trouble of setting up *Postfix* (my choice) or *Exim*. However, SSMTP does not verify SSL certificates. It just blindly accepts them! Shown is how you relay mail to a gmail account with SSMTP.

```
sudo apt-get install ssmtp mailutils
sudo -e /etc/ssmtp/ssmtp.conf
```

Change it to look like this:

```
# root is the person who gets all mail for userids < 1000
root=yourname@gmail.com
# Here is the gmail configuration (or change it to your private smtp server)
mailhub=smtp.gmail.com:587
AuthUser=yourname@gmail.com
AuthPass=YourPasswordHere
UseTLS=YES
UseSTARTTLS=YES
sudo -e /etc/ssmtp/revalias
# sSMTP aliases
#
# Format: local_account:outgoing_address:mailhub
#
# where [:port] is an optional port number that defaults to 25.
root:yourname@gmail.com:smtp.gmail.com:587
nagios:yourname@gmail.com:smtp.gmail.com:587
```

You'll need a user (nagios) and a group (nagcmd) for the system to run under. You can turn off logins for this user once everything is installed.

```
sudo useradd nagios
sudo groupadd nagcmd
sudo usermod -a -G nagcmd nagios
sudo usermod -a -G nagcmd www-data
```

Now, install the prerequisites required for a full nagios installation.

```
sudo apt-get install build-essential wget unzip php5-fpm fcgiwrap \
openssl libssl-dev libgd2-xpm-dev xinetd apache2-utils mrtg
```

Step 2: Installing Nagios

Go to the Nagios website : https://www.nagios.org/downloads/nagios-core/ (https://www.nagios.org/downloads/nagios-core/) and fill out the form to get to the download section. When you see the link for nagios-core, right click it and find the latest release. Select Copy Link Address. Return to your terminal session and paste the address after *wget*. Then, uncompress the newly downloaded image.

```
wget https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.2.4.tar.gz
tar -xvf nagios-4.2.4.tar.gz
cd nagios-4.2.4
```

Now, we need to run our usual *configure*/*make* pass. If you have multiple CPU cores, you can increase compile speed by adding '-j X' to your make commands. Replace 'X' with the number of CPU cores you have (or number of cores + 1). If you omit X, there won\'t be a limit to the number of threads created with -j. On a 16 core server with hyper-threading (32 virtual CPUs) use '-j 32'. This applies any time you invoke *make*.

```
./configure --with-nagios-group=nagios --with-command-group=nagcmd
make all
sudo make install
sudo make install-init
sudo make install-commandmode
sudo make install-config #- for sample configs
sudo make install-exfoliation #- or classicui, as the default theme
```

Now, if you are using Apache, you can grab the apache config file and install it, then you can skip the next step.

```
sudo make install-webconf #- for apache
```

Step 3: Configure Nginx (optional)

If you are using Nginx, here is a config file. I have set this up so that "nagios" is a virtual host. Set the *server_name* as it should appear in the server portion of your URL and be sure your DNS can resolve it!

Since the following example uses a virtual host and not a subdirectory of a host you will need to change */usr/local/nagios/etc/cgi.cfg*. The *cgi.cfg* file is where you configure the cgi web interface. Look for a line that says *url_html_path* and change it from */nagios* to just */*

```
# You should look at the following URL's in order to grasp a solid
# understanding of Nginx configuration files in order to fully unleash
# the power of Nginx.
# http://wiki.nginx.org/Pitfalls
# http://wiki.nginx.org/QuickStart
# http://wiki.nginx.org/Configuration
#
# Generally, you will want to move this file somewhere, and start with a clean
# file but keep this around for reference. Or just disable in sites-enabled.
#
# Please see /usr/share/doc/nginx-doc/examples/ for more detailed examples.
##
# Default server configuration
#
server {
 listen 80;
 server_name nagios;
 access_log /var/log/nginx/nagios.access.log;
 error_log /var/log/nginx/nagios.error.log info;
 expires 31d;
 root /usr/local/nagios/share;
 index index.php index.html;
 auth_basic "Nagios Restricted Access";
 auth_basic_user_file /usr/local/nagios/etc/htpasswd.users;
 location ~ \.cgi$ {
    root /usr/local/nagios/sbin;
    rewrite ^/nagios/cgi-bin/(.*)$ /$1;
    include /etc/nginx/fastcgi_params;
    fastcgi_param AUTH_USER $remote_user;
    fastcgi_param REMOTE_USER $remote_user;
    fastcgi_param SCRIPT_FILENAME /usr/local/nagios/sbin/$fastcgi_script_name;
    fastcgi_pass fcgiwrap;
 }
 #- magic needed to make joomla-style URLs work
 location ~ [^/]\.php(/|$) {
    gzip off;
    fastcgi_split_path_info ^(.+\.php)(/.*)$;
    if (!-f $document_root$fastcgi_script_name) {
        return 404;
    }
    fastcgi_keep_conn on;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_pass php;
    fastcgi_index index.php;
    include /etc/nginx/fastcgi_params;
 }
}
```

Plop that into sites.available and make a link in *sites.enabled*. Then you'll need to define the php and fcgiwrap gateways. Basically, tell Nginx where the sockets are to pass this stuff on. Inside the main */etc/nginx/nginx.conf*, inside the *http {}* section, add these lines:

```
upstream php {
    server unix:/var/run/php5-fpm.sock;
}
upstream fcgiwrap {
    server unix:/var/run/fcgiwrap.socket;
}
```

Now, let\'s fire up a few servers and make sure they all work:

```
sudo systemctl start php5-fpm
sudo systemctl enable php5-fpm
sudo systemctl start fcgiwrap
sudo systemctl enable fcgiwrap
sudo systemctl reload-or-restart nginx
sudo systemctl status -l nginx
```

If that last command shows a running nginx, you are ready to configure Nagios!

Nagios Basic Configuration

Still with me?

OK, let's install the basic nagios plugin package. These are plugins used to monitor various services. You'll install these on both the monitoring machine and the machine to be monitored, although I'll configure the monitored machines a bit differently in the next step. Grab it from the same site as you downloaded nagios, copy the URL and paste it into your monitoring server.

```
wget https://nagios-plugins.org/download/nagios-plugins-2.1.4.tar.gz
tar -xvf nagios-plugins-2.1.4.tar.gz
cd nagios-plugins
./configure --with-nagios-user=nagios --with-nagios-group=nagcmd
make && make install
```

Everything in Nagios is installed under */usr/local/nagios* and generally is owned by the *nagios* user with a group of *nagcmd*. It's log files will be under */usr/local/nagios/var/nagios.log* and its configuration is in */usr/local/nagios/etc.* Get it? Nagios contains a number of CGIs, PHP files, plus a back-end service. Let's take a look at how it's all set up.

First, let's look at the main config file */usr/local/nagios/etc/nagios.cfg*. Go ahead and open it and take a look. You'll see that you define where it logs here plus the user and group it should be using (double check that they are nagios and nagcmd, respectively). You'll also notice that it defines a bunch of object files. These are just more config files. You'll also see you can load whole directories should you wish to have 1 config file per host (making it easier to manage with external tools). Let's look into these object files in more depth.

```
cd objects
```

Nagios Object Files

Let's start with an important one, contacts. You'll see this is where we set up the users. I've set the first user to have a contact_name of nagiosadmin. You'll see where to add your email address and full name. But what about a password?

```
sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
```

Next file worth looking at is templates. Here you will see that when we make definitions, such as a contact, there is a "use" line that says "generic-contact". You can think of this as object inheritance. What makes a definition a template is the "register" line that says 0. Basically, you can "use" any other definition as a starting point for a new definition, and unregistered definitions are templates only. This will let you easily define "classes" of servers that are all monitored the same way. There are templates for all the basics. We won't edit these.

Move on to timeperiods and you'll see where you can modify times of day, holidays, and things like that. You'll be able to turn notifications on and off based on these periods.

Next is commands. This one is really important! Here is where you define all the commands that Nagios runs. These can be scripts and an absolutely huge number of macros will be defined by Nagios when the script runs. There are two commands for notifications, one is used when a host goes down, the other is used to send notifications about specific services. The default just sends a simple email. Feel free to replace this with a complex script in the language of your choice. You'll also see all the service definitions. The $USER1$ macro refers to */usr/local/nagios/libexec*. You'll see all the plugins in there. You can add your own scripts to this directory as well.

Many commands will have arguments that specify levels for warnings and critical levels. If you aren't sure what all the arguments are, just run the command manually from the *libexec* directory. It will give you all the parameters. In fact, browse the directory now to get a feel for all the stuff you can monitor, and this is just with the basic plugins. The exchange has more!

You'll note a command called *check_ping*. This plugin will call the system's *ping* command. Technically, Linux doesn't have an API for sending ICMP packets so you use the raw socket interface. This interface can't be used by a regular user. You'll have to test the ping command as an unprivileged user and see if it works. If it works, it's likely been marked as a *setuid* command and owned by root. Other distributions don't do this and normal users can't ping. This means Nagios can't ping a host to see if it's alive! If this is the case for you, there are two methods to solve this, depending on your security views.

The *ping* command is small and isn't likely to change, it's well tested, and the chances of a security problem are small. Make sure it's not writable and mark it setuid root like this: *chmod u+s /bin/ping*

However ... if you run a security scan or do an upgrade that looks at this command in your filesystem, it can cause issues because you have changed a system file and allowed all users to execute this command as root! The alternative is to leave *ping* alone and let nagios run the command via *sudo*. This is what I did. Use *visudo* to edit */etc/sudoers* and add this entry:

```
nagios ALL=NOPASSWD: /bin/ping,/usr/local/nagios/libexec/check_ping
```

And then, edit your command definition in */usr/local/nagios/etc/objects/commands.cfg* to run *sudo* by just inserting it in front of the command! There are two commands, *check_ping* and *check_host_alive*. The former uses arguments and the later is more basic.

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

Do not attempt to just run nagios as root or mark it as setuid root! Nagios is a huge package and it faces the Internet. It runs without privilege for a reason!

The rest of the objects are hosts. Let's begin with the localhost file. We don't need to make any changes just yet, but open it up and have a look. You'll see that hosts can inherit templates and are defined like any other object. They can be placed into groups to be monitored together (such as clusters of web servers or by customer). Each host has its own list of services to monitor on that particular host, allowing you to customize the particular parameters on a per-host basis. You'll see that the command to perform a particular check shows the command (as given in *commands.cfg*) followed by arguments. An exclamation point separates arguments. Each argument will be named $ARG1$, $ARG2$, and so forth.

Starting Nagios

Before starting or restarting Nagios, you'll want to have it check it\'s config files

```
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

Now you can start Nagios and check it's status to make sure it's running.

```
sudo systemctl start nagios
sudo systemctl status nagios
```

if everything looks okay, enable it to start on boot

```
sudo systemctl enable nagios
```

If everything is OK, you can pop open a web browser and go to the URL! You should see Nagios monitoring itself. If Nagios says something like Bad Gateway, then check the paths to your *php5-fpm* and *fcgi* sockets and edit the upstream command in *nginx.conf*. You can check using this command:

```
sudo netstat -anp | grep -E 'fcgi|php'
```

Make sure you have a working web interface before you go any further. You may need to invoke the omnipotent Google for help. Once that's done, you'll start monitoring things other than just yourself! If you see broken images and css files failing to load, then you need to configure */usr/local/nagios/etc/cgi.cfg* as mentioned in *Step 3*.

Monitoring Remote Hosts

Routers & Switches

You can monitor a lot of hosts with just the *check_ping* or *check_host_alive* commands. You can make sure any host with an IP address is alive with just one of these commands. Smarter routers have more features, such as SNMP. If you monitor the traffic of these systems with *MRTG* or *Cacti* (a great graphing and monitoring tool that also uses RRD graphs which you can check with plugins on the Nagios Exchange) then you can get traffic data right out of the mrtg log files with Nagios and generate alerts if the traffic is above a threshold. You'll note that the *check_mrtg* file just takes a path to the mrtg log file and bandwidth thresholds for warnings and critical levels.

SNMP can monitor the CPU and Memory usage of smart switches and routers and there are examples of all these in the *switch.cfg* file. Here are some more examples of some services for monitoring a Cisco ASA 5520 whose hostname is "mcp" *(big bonus points if you can guess the origin of that name)*. Cisco has some weird MIB's so you may have to dig around for the right MIB definition files. If you have trouble with SNMP, use *snmpwalk* to verify the basic connectivity and permissions.

```
# Monitor uptime via SNMP
define service{
    use generic-service ; Inherit values from a template
    host_name mcp
    service_description Uptime
    check_command check_snmp!-C public -o \
    DISMAN-EVENT-MIB::sysUpTimeInstance
}
# Monitor Port 1 status via SNMP
define service{
    use generic-service ; Inherit values from a template
    host_name mcp
    service_description Port 1 Link Status
    check_command check_snmp!-C public -o ifOperStatus.1 -r 1 -m \
IF-MIB
}
# Monitor bandwidth via MRTG logs
define service{
   use generic-service ; Inherit values from a template
   host_name mcp
   service_description Port 1 Bandwidth Usage
   check_command \
check_local_mrtgtraf!/var/www/html/mrtg/mcp_2.log!AVG!1000000,1000000!5000000,5000000!10
}
```

Printers

You monitor printers the same way as switches, using *check_ping*. You can also add *check_hpjd* (HP JetDirect). Or, if you have an odd printer (or other device), you can use *check_tcp* to make sure a particular port is open. In my *printer.cfg* I have a ping, and this:

```
define service{
    use generic-service
    host_name pantum
    service_description Printer Status ; The service description
    check_command check_tcp!9100 ; The command used to monitor
    normal_check_interval 10 ; Check the service every 10 minutes
    retry_check_interval 1 ; Re-check the service every minute
}
```

Windows Machines

Sooner or later you will have to deal with Windows machines. I don't like to, so I'll keep this brief. There is a windows.cfg file as one of the object files that shows examples for windows machines. You will need to install a program called *NSClient++* on the machine to be monitored. Be aware that the Windows firewall may block your ping requests so you'll have to configure it or turn it off. Also note that it appears that service names are case sensitive, so if you use the example configuration where it looks to see if *Explorer.exe* is running, and it fails, you'll need to open the task manager and get the correct process name. On Windows 7 it was *explorer.exe*, all lower-case.

The *NSClient++* program installs as a Windows service with a nice easy GUI installer and config tool. It can be scripted using most scripting tools for Windows, even Visual Basic.

Linux & Unix Hosts

And now we are down to it. There are a number of ways to remotely execute commands on a Linux host. In fact, *check_ssh* is still around and can be used if you are paranoid about security. However, ssh has a relatively lengthy key exchange protocol and it's not exactly efficient when you need to constantly setup and teardown multiple connections. Setting up an ssh tunnel might work, but an alternative called NRPE (Nagios Remote Plugin Executor) has been devised.

NRPE can be configured to only accept connections from a specific IP address and it encrypts all connections with SSL. It can also verify the SSL keys to verify that the host is authentic. You just tell it what commands to run through the connection. Only commands that the local host has configured will be allowed and you can determine if arguments will be sent by Nagios or if you will supply them locally (local is more secure). Also NRPE runs as its own user which can be disallowed from running a shell, and it doesn't need *sudo* access because it doesn't need to run *ping*. In fact, you'll notice I set *nrpe* and the nagios plugins it will run to use a user and group of "*nrpe*". Even though these users are not on the same host, it's best to keep the roles separate. The *nagios* user can run *sudo*, *nrpe* can not.

First, Install the nagios-plugins package (note the different config from before):

```
tar -xvf nagios-plugins-2.1.4.tar.gz
cd nagios-plugins
./configure --with-nagios-user=nrpe --with-nagios-group=nrpe
make && sudo make install
```

Now, you will need to grab the nrpe package (always the same site) and install it:

```
tar -xvf nrpe-3.0.1.tar.gz
cd nrpe-3.0.1
./configure --with-nrpe-user=nrpe --with-nrpe-group=nrpe
make -j nrpe
sudo make install-groups-users
sudo make install-daemon
sudo make install-config
sudo make install-init
```

You'll want to change *allowed_hosts* in */usr/local/nagios/etc/nrpe.cfg* to your nagios server's IP address. You should also change any commands you want to execute in this file as well. I use the local argument versions. Pay attention to things like drive device names and remember that the service name needs to match the service descriptions on your nagios server.

Install the nrpe package on the nagios server as well. Here you should use something more like this configuration:

```
configure --with-nagios-user=nagios --with-nagios-group=nagcmd
make -j check_nrpe
sudo make install-plugin
sudo make install-daemon
```

You can pass both sets of options to configure so that you can configure it once and then share the directory to install different parts on different systems. To use NRPE, you'll need to modify commands.cfg on the Nagios server to include the *check_nrpe* command. Add these lines:

```
# 'check_nrpe' command definition
define command {
    command_name check_nrpe
    command_line $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
}
```

Now you can add services. I moved the linux-servers group out of *localhost.cfg* and put it into a new configuration file called *unixhosts.cfg*. Below is an example file. Once you get nrpe configured on your remote host, start it in the usual manner ("make install-init" installed a systemd service file for you).

```
sudo systemctl start nrpe
sudo systemctl enable nrpe
```

If you add new configuration files (like *unixhosts.cfg*), remember to tell the main *nagios.cfg* about it. Larger installations will likely want to just add directories for specific groups, give *nagios.cfg* the directory name (there is an example in the file) and then you can just toss in files into the directory and reload. For our example, add a line like the others:

```
# Definitions for monitoring linux/unix hosts
cfg_file=/usr/local/nagios/etc/objects/unixhosts.cfg
```

You can then add hosts to your unixhosts.cfg file (as below - adding to the group definition, not replacing it). And then check your Nagios configuration (as before):

```
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

And if it's good, tell Nagios to reload the new configuration entries:

```
sudo systemctl reload nagios
```

If you get weird segfaults running NRPE, then you probably mixed nrpe version 2 and 3. The *check_nrpe* from the nagios plugins package defaults to using version 3 of the protocol and if the client uses version 2, you get crashes. You can have nagios pass "-2" to the *check_nrpe* command to use the older protocol if you want to use older versions of nrpe, such as a version supplied by your distribution's package manager. This even lets you set this per-host. A simple *unixhosts.cfg* is below. Remember that the command listed as an argument to check_nrpe gets passed to the remote host and its definition is picked up by the nrpe config file on the remote system.

```
###############################################################################
###############################################################################
#
# HOST GROUP DEFINITION
#
###############################################################################
###############################################################################
# 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
    ; Comma separated list of hosts that belong to this group
    members localhost,taro
}
###############################################################################
###############################################################################
#
# HOST DEFINITION
#
###############################################################################
###############################################################################
# Define a host for the local machine
define host{
    use linux-server ; Name of host template to use
    host_name taro
    alias taro
    address 192.168.12.150
}
###############################################################################
###############################################################################
#
# SERVICE DEFINITIONS
#
###############################################################################
###############################################################################
define service{
    use local-service
    host_name taro
    service_description PING
    check_command check_ping!100.0,20%!500.0,60%
}
define service{
    use local-service
    host_name taro
    service_description Root Partition
    check_command check_nrpe!check_disk
}
define service{
    use local-service
    host_name taro
    service_description Current Users
    check_command check_nrpe!check_users
}
define service{
    use local-service
    host_name taro
    service_description Total Processes
    check_command check_nrpe!check_procs
}
define service{
    use local-service
    host_name taro
    service_description Current Load
    check_command check_nrpe!check_load
}
```

Conclusions

This tutorial is about 14 pages and just *begins* to introduce Nagios. You'll want to consider a few things in how your system gets notifications to you and how you customize that information. Nagios can send HTML email, so it's easy to send colors and even links so that you can acknowledge problems right from the email entry. I have mine sending me graphs and all sorts of things.

Now consider what happens when you are monitoring the mail server. The mail server goes down and you get notified ... by email? This is one of the areas where scripts that can apply some logic make more sense. Maybe you'll install an old voice modem and use it to dial your phone and tell you the mail server is out! Likely, it's easier to just set up a secondary email to be notified of mail server problems that uses a different mail server. You can also stick an Android app on your phone that logs in to the usual Web Interface and checks service status information, gives you an overview, and can do its own alarms and notifications, right on your phone, without ever using the email notification plugins. Free on Google Play!

Also remember to use the system to full capacity. If a system is going down for maintenance, open the nagios interface and schedule the downtime. This turns off notifications and lets the other admins know what it is going on. I've even heard of people using scripts that log into nagios (you can just fetch crafted URLs and perform commands by number) and turn off notifications when you manually shut down a machine. It then turns notifications back on when the system is done booting. Scripting is very powerful! Nagios works well with modern tools such as *Puppet* and *Chef* for managing large infrastructures. I also check upstream internet connectivity by including remote hosts as if they were local "dumb" switches that I ping, although options for checking http and other services are also available.

So, the next time someone says "Did you know Web Server 3 is down?" You can tell them it was only down for 4 minutes during an unusually high burst of traffic on BGP port 3 that caused the number of processes to reach dangerous levels and you are investigating the root cause based on the information logged by Nagios. And now you get that raise you wanted!

4 16 + −

... Show Previous Comments

Joan Segarra Casanovas (/cp/socialize/profile/user/jsegarrac) 4/12/2017
Brief but very useful tutorial. Thanks so much

0

uday kiran reddy billa (/cp/socialize/profile/user/ubilla) 11/14/2017
Very useful to get a picture about Nagios

0

Reply