GnuGroup International

Nagios
IT infrastructure monitoring tool

ILG
Insight GNU/Linux Group
Reinventing the way you,
Think,
Learn,
Work

NAGIOS

MODULE - 1

Index of module - 1

- Introduction
- History of Nagios
- System Requirements
- Prerequisites
- Licensing
- Downloading
- Advice for Beginners
- Quickstart installation
- Nagios Architecture
- Plugins
- Remote Monitoring Mechanisms
- Monitoring Windows
- Monitor Remote Linux Host
- Monitor Network Switch
- Configuration overview
- Object configuration overview
- Object Definitions
- Running Nagios
- Notifications
- Host Checks
- Service Check
- Active Check
- Passive Check
- States types
-
-
-

Introduction

•What is Nagios Core?

Nagios® CoreTM is an **Open Source system** and **network monitoring application**.

•It watches hosts and services that you specify,

- **alerting you** when things go bad and when they get better.
- Nagios Core was originally **designed to run under Linux**,
- although it should work under **most other unices as well**.

History of Nagios

- History Chart

http://www.nagios.org/about/history

System Requirements

The only software requirement for running Nagios Core is a machine

- Running Linux (or UNIX variant)
- Has network access
- A C compiler installed
- A web server (preferrably Apache)
- gd library version 1.6.3

Prerequisites

For Centos / Redhat distro

- Apache
- PHP
- GCC compiler
- GD development libraries

  !!!!!!!!!!!!!!!!!!Yum it !!!!!!!!!!!!!!!!!!!
- yum install httpd php
- yum install gcc glibc glibc-common
- yum install gd gd-devel

Licensing

Nagios Core is licensed under the terms of the

GNU General Public License Version 2

as published by the Free Software Foundation.

Downloading

http://www.nagios.org/download

Advice for Beginners

Relax – it's going to take some time....

- Use the quickstart instructions
- Read the documentation
- Seek the help of others

Quickstart installation

Prerequisites

Already Diccussed

Also disable selinux

Yum update

Already Diccussed

Quickstart installation

- ***User creation***

```
# su – l
# /usr/sbin/useradd -m nagios
# passwd nagios
```

- ***Group Creation***

1) Create a new ***nagcmd*** group for allowing external commands to be submitted through the web interface.

2) Add both the nagios user and the apache user to the group.

```
# /usr/sbin/groupadd nagcmd
# /usr/sbin/usermod -a -G nagcmd nagios
# /usr/sbin/usermod -a -G nagcmd apache
```

Quickstart installation

Download Nagios and the Plugins

http://www.nagios.org/download

Create a directory for storing the downloads.

```
# mkdir ~/downloads
# cd ~/downloads
```

```
# wget http://prdownloads.sourceforge.net/sourceforge/nagios/nagios-3.2.3.tar.gz
# wget http://prdownloads.sourceforge.net/sourceforge/nagiosplug/nagios-plugins-1.4.11.tar.gz
```

Quickstart installation

- **Compile and Install Nagios**

  Extract the Nagios source code tarball.

  ```
  # cd ~/downloads
  # tar xzf nagios-3.2.3.tar.gz
  # cd nagios-3.2.3
  ```

Quickstart installation

Compile and Install Nagios

Run the Nagios configure script, passing the name of the group you created earlier like so:

```
# ./configure --with-command-group=nagcmd
#  make all
```

Install binaries, init script, sample config files and set permissions on the external command directory.

```
# make install
# make install-init
# make install-config
# make install-commandmode
```

Don't start Nagios yet **- there's still more that needs to be done..**

Quickstart installation

Configure the Web Interface

Install the Nagios web config file in the Apache conf.d directory.

```
# make install-webconf
```

Create a nagiosadmin account for logging into the Nagios web interface.

Remember the password **you assign to this account - you'll need it later.**

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

Restart Apache to make the new settings take effect.

```
# service httpd restart
```

Quickstart installation

Compile & Install the Nagios Plugins

Extract the Nagios plugins source code tarball

```
# cd ~/downloads
# tar xzf nagios-plugins-1.4.11.tar.gz
# cd nagios-plugins-1.4.11
```

Compile and install the plugins.

```
# ./configure --with-nagios-user=nagios --with-nagios-group=nagios
# make
# make install
```

Quickstart installation

Start Nagios

Add Nagios to the list of system services and have it automatically start when the system boots.

```
# chkconfig --add nagios
# chkconfig nagios on
```

Add nagios service to runlevels

Verify the sample Nagios configuration files.

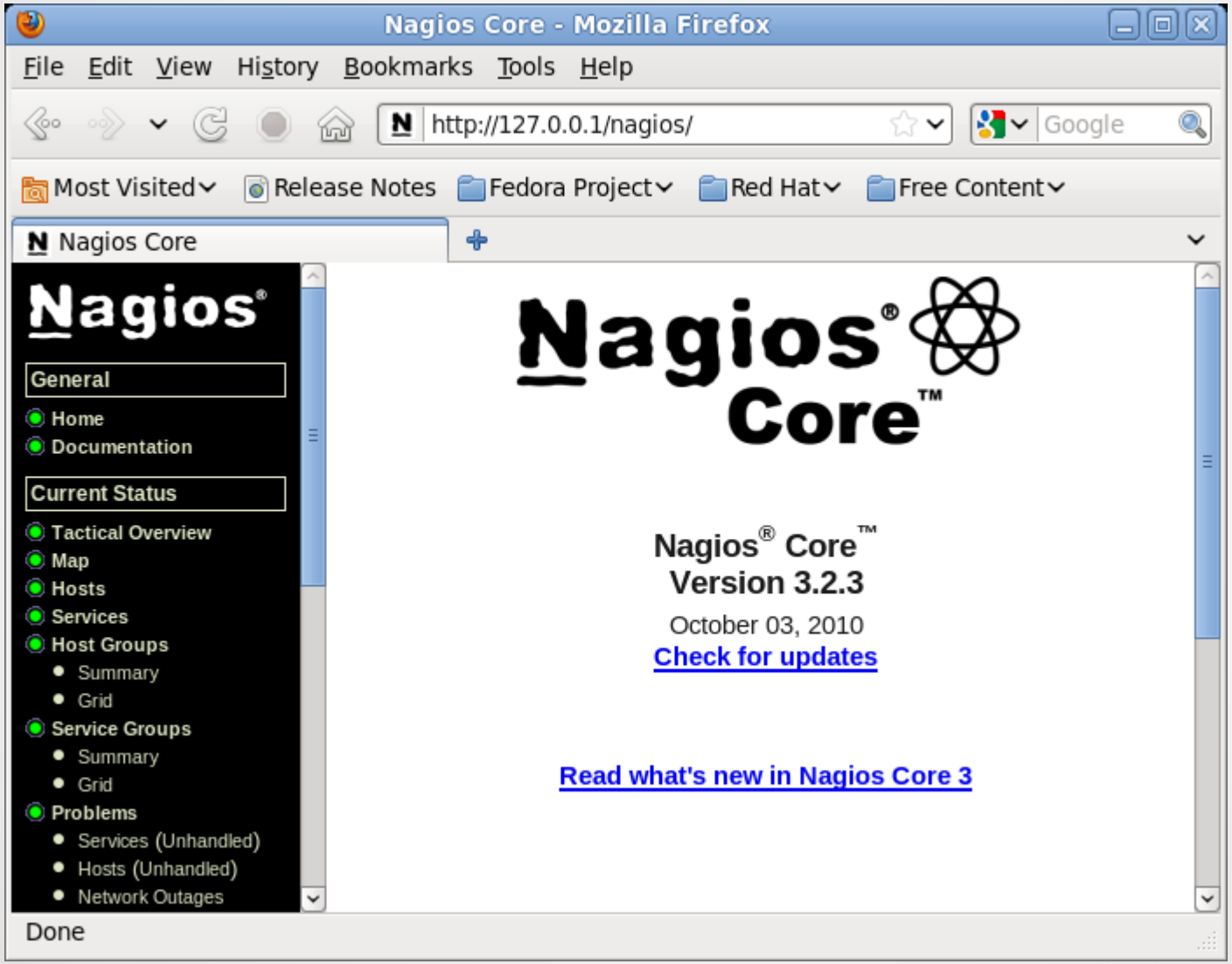
```
# /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

Verify config file

If there are no errors, start Nagios.

```
# service nagios start
```

Quickstart installation



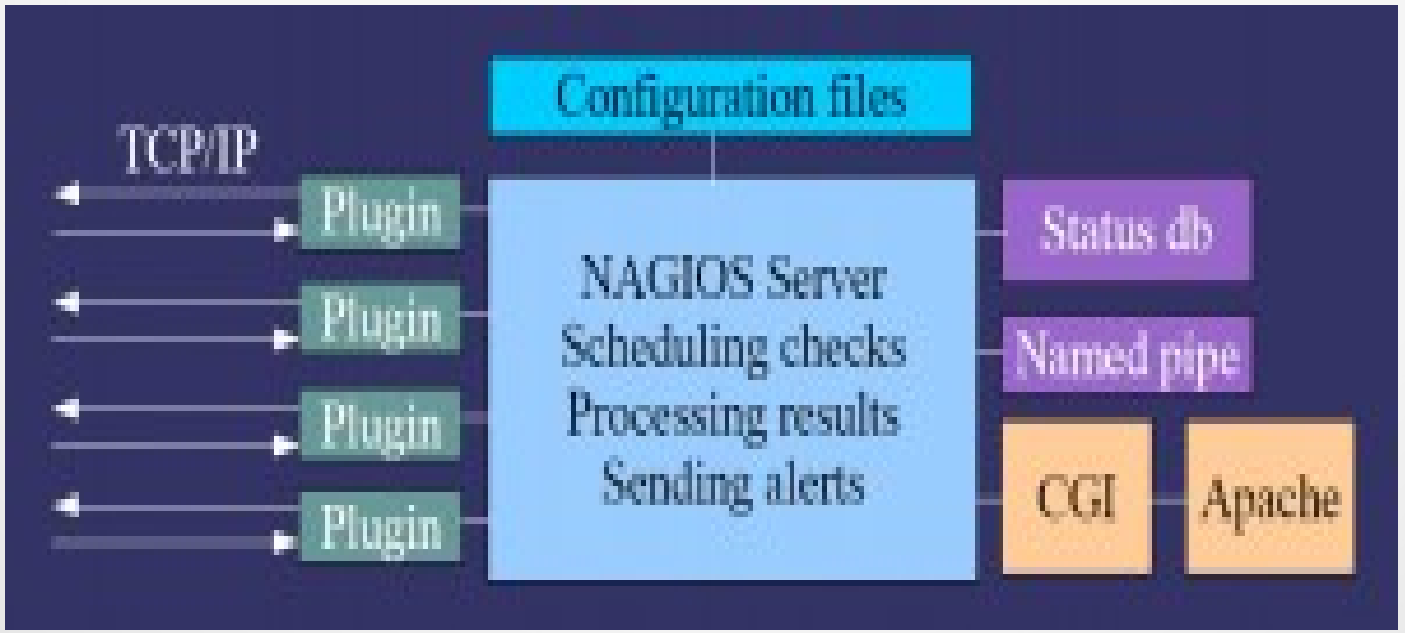
If you see me,
I am configured
And ready

Nagios Architecture

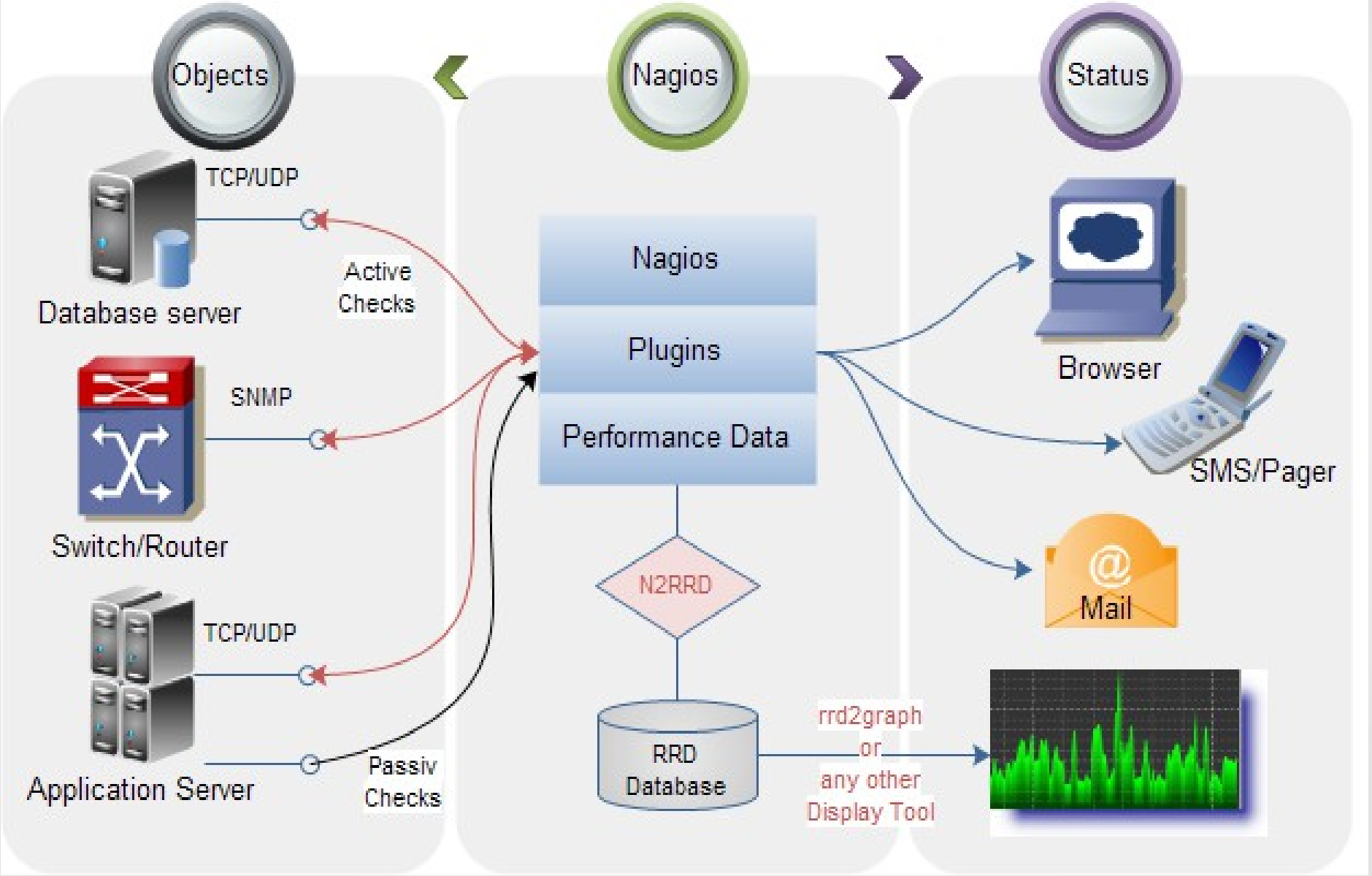
Nagios is built on a server / agents architecture.

- Usually, on a network, a Nagios server is running on a host,
- and **plugins are running on all the remote hosts** that need to be monitored.

These plugins send information to the server, which displays them in a GUI.



Nagios Architecture



Plugins

- **What Are Plugins?**

Nagios plugins are ***standalone extensions*** to Nagios Core

- provide low-level intelligence on how **to monitor anything and everything** with Nagios Core.
- Plugins *operate as standalone applications*, but are generally **designed to be executed by Nagios Core.**
- Plugins *process command-line arguments*, go about the business of performing a specific type of check,
- and **then return the results to Nagios Core for further processing.**

Plugins can either be **compiled binaries** (written in C, C++, etc) or **executable scripts** (shell, Perl, PHP, etc).

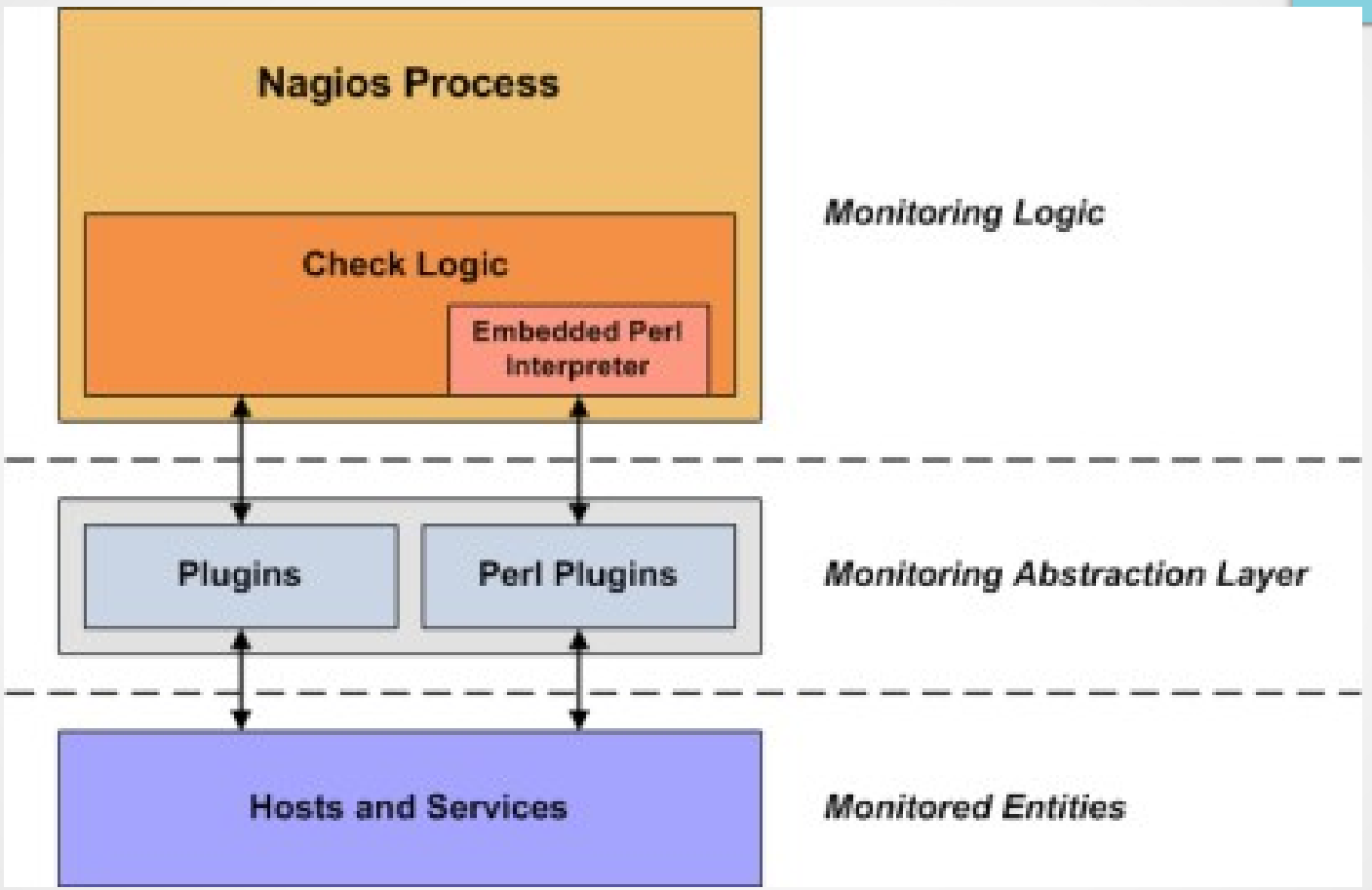
Plugins act as an **abstraction layer** between the **monitoring logic** present in the **Nagios daemon** and the actual services and hosts that are being monitored.

The upside of this type of plugin architecture:

That you can monitor just about anything you can think of.

If you can automate the process of checking something, you can monitor it with Nagios.

Plugins


Nagios Process
Check Logic
Embedded Perl Interpreter
Monitoring Logic
Plugins
Perl Plugins
Monitoring Abstraction Layer
Hosts and Services
Monitored Entities

Plugins

What Plugins Are Available?

There are plugins currently available to monitor many different kinds of devices and services, including:

- HTTP, POP3, IMAP, FTP, SSH, DHCP
- CPU Load, Disk Usage, Memory Usage, Current Users
- Unix/Linux, Windows, and Netware Servers
- Routers and Switches etc.

./check_http –help

– **Offical Plugins can be downloaded**

Nagios Plugins Project: http://nagiosplug.sourceforge.net/

Nagios Downloads Page: http://www.nagios.org/download/

NagiosExchange.org: http://www.nagiosexchange.org/

Remote Monitoring Mechanisms

(1) check_by_ssh

Add Nagios to the list of system services and have it automatically start when the system boots.

Since you want to execute a command

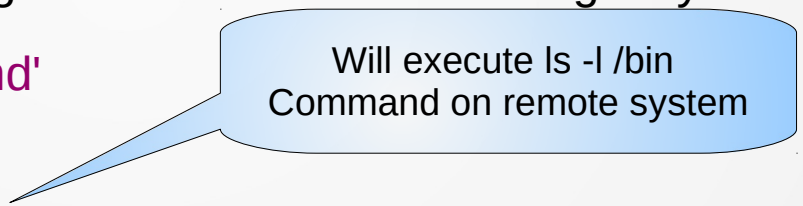
(a "check-command' or a nagios plugin to get the state of some process on the remote hosts),

The obvious solution is to run it through a remote shell on the target system -

ssh user@host 'command'

e.g

ssh ilg@node03 ls -l /bin



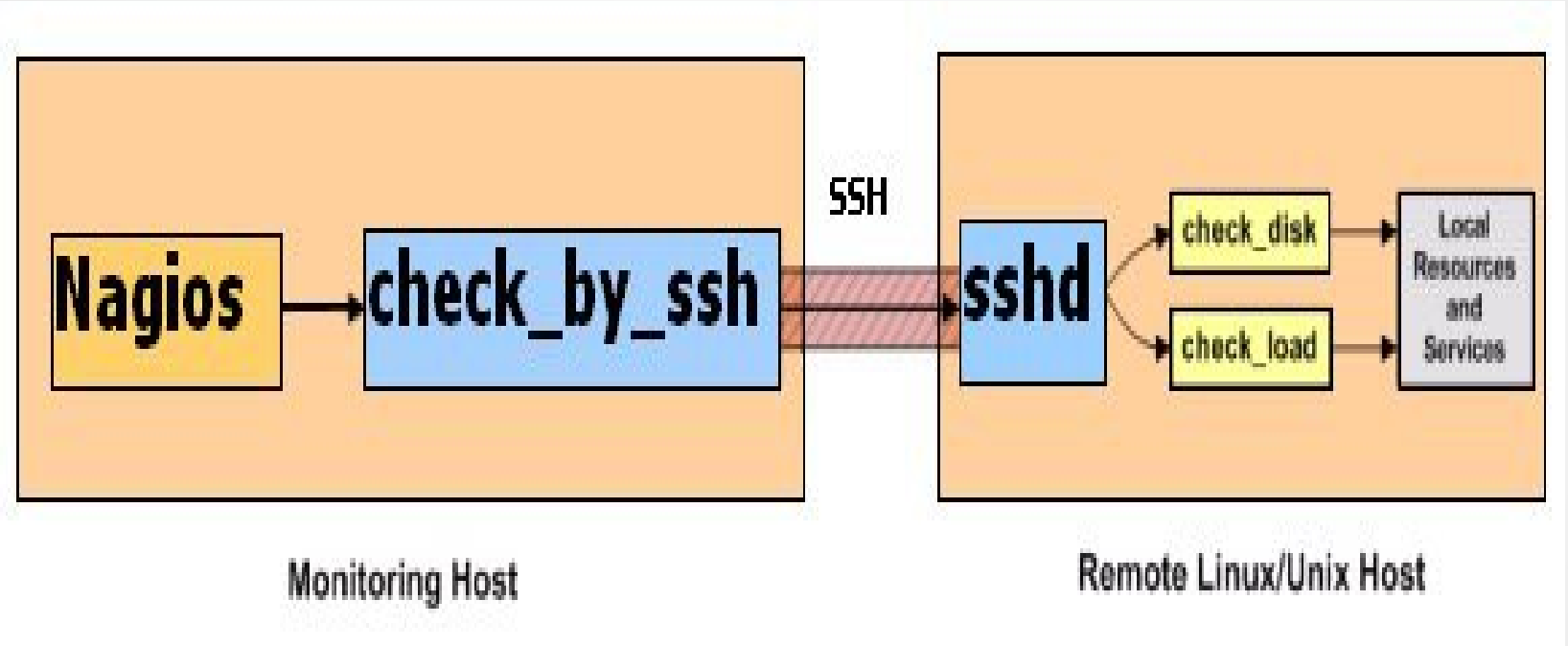
Nagios has a check_by_ssh command that works in a similar manner

This requires the target to have an ***ssh-server running***,

you need to set up nagios account on the remote host and arrange for non-interactive authentication (e.g key-based authentication, or by refering to username and passwords in a configuration file).

Remote Monitoring Mechanisms

Design Overview



Remote Monitoring Mechanisms

(2) nrpe

NRPE stands for *Nagios Remote Process Execution* :

The NRPE addon is designed to allow you to execute Nagios plugins on **remote Linux/Unix machines.**

The main reason for doing this is to allow Nagios to monitor "local" resources (like CPU load, memory usage, etc.) on remote machines.

Since these public resources are not usually exposed to external machines, an agent like NRPE must be installed on the remote Linux/Unix machines.

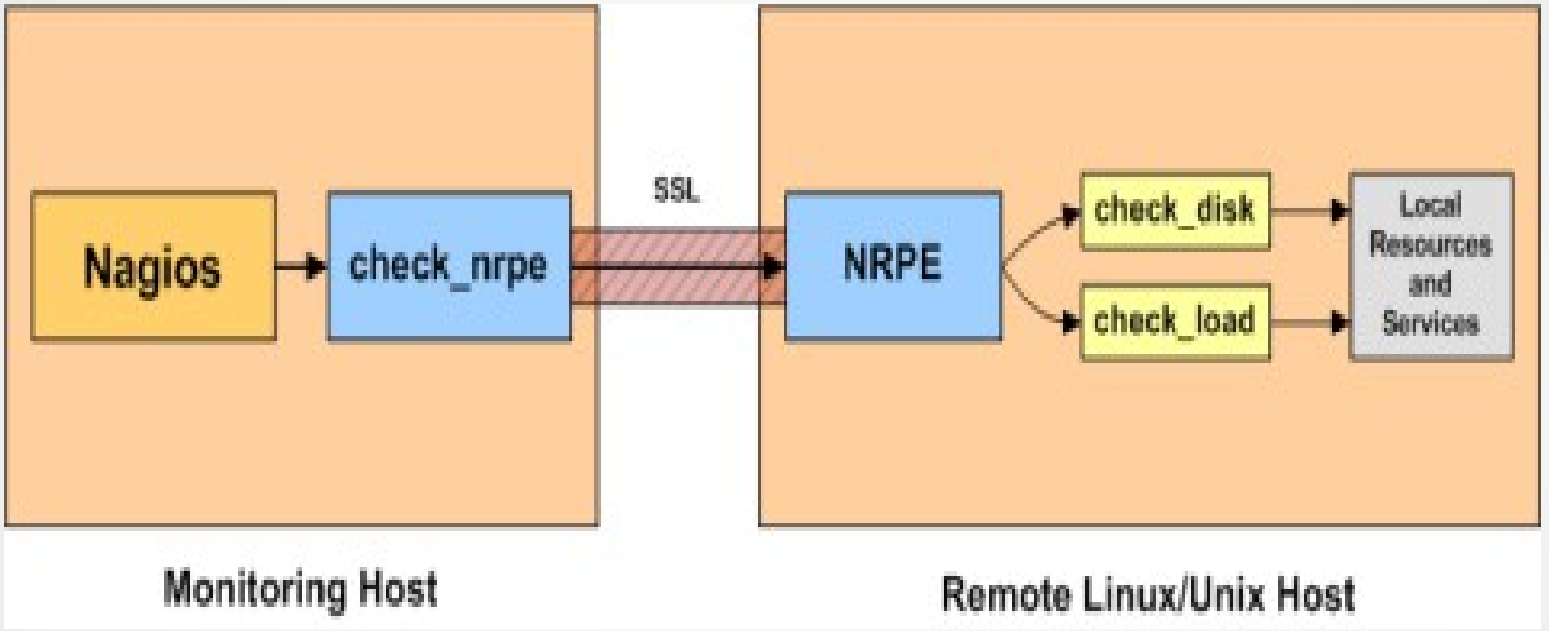
This is a "Remote Agent" architecture :

An **nrpe** agent on the target executes the requested check-commands and reports back the results :

Nagios (→ nagios host nrpe) → nrpe on remote host → command.

Remote Monitoring Mechanisms

- Design Overview



Remote Monitoring Mechanisms

(3) check_nt

Remote Agent mechanism specifically for Windows systems. Conceptually similar to check_by_ssh and check_nrpe.

Monitoring private services or attributes of a Windows machine requires that you install an agent on it.

This agent acts as a proxy between the Nagios plugin that does the monitoring and the actual service or attribute of the Windows machine.

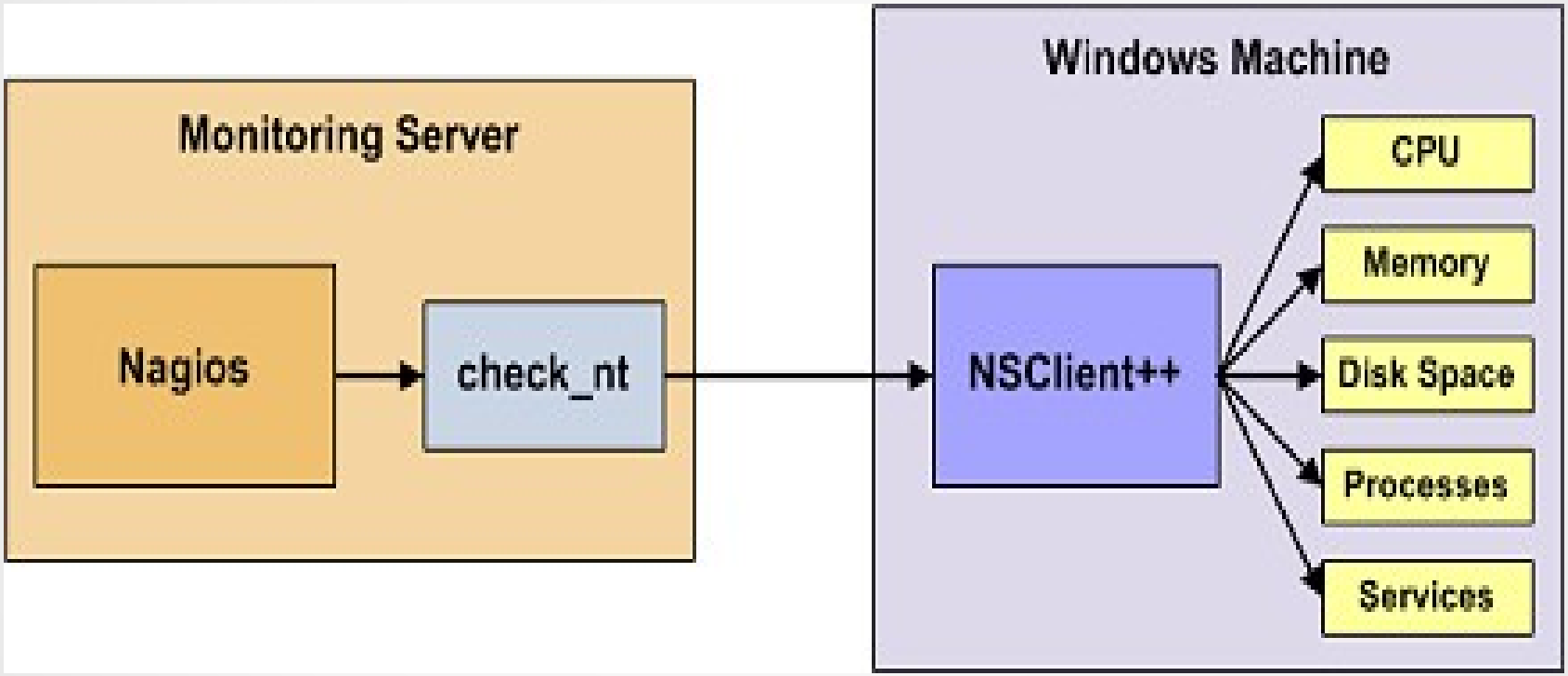
Without installing an agent on the Windows box, Nagios would be unable to monitor private services or attributes of the Windows box.

We will be installing the NSClient++ addon on the Windows machine and using the check_nt plugin to communicate with the NSClient++ addon.

Note: The check_nt plugin should already be installed on the Nagios server

Remote Monitoring Mechanisms

- Design Overview



Remote Monitoring Mechanisms

(4) NSCA

NSCA is Nagios Service Check Acceptor

It's an additional daemon on your Nagios host.

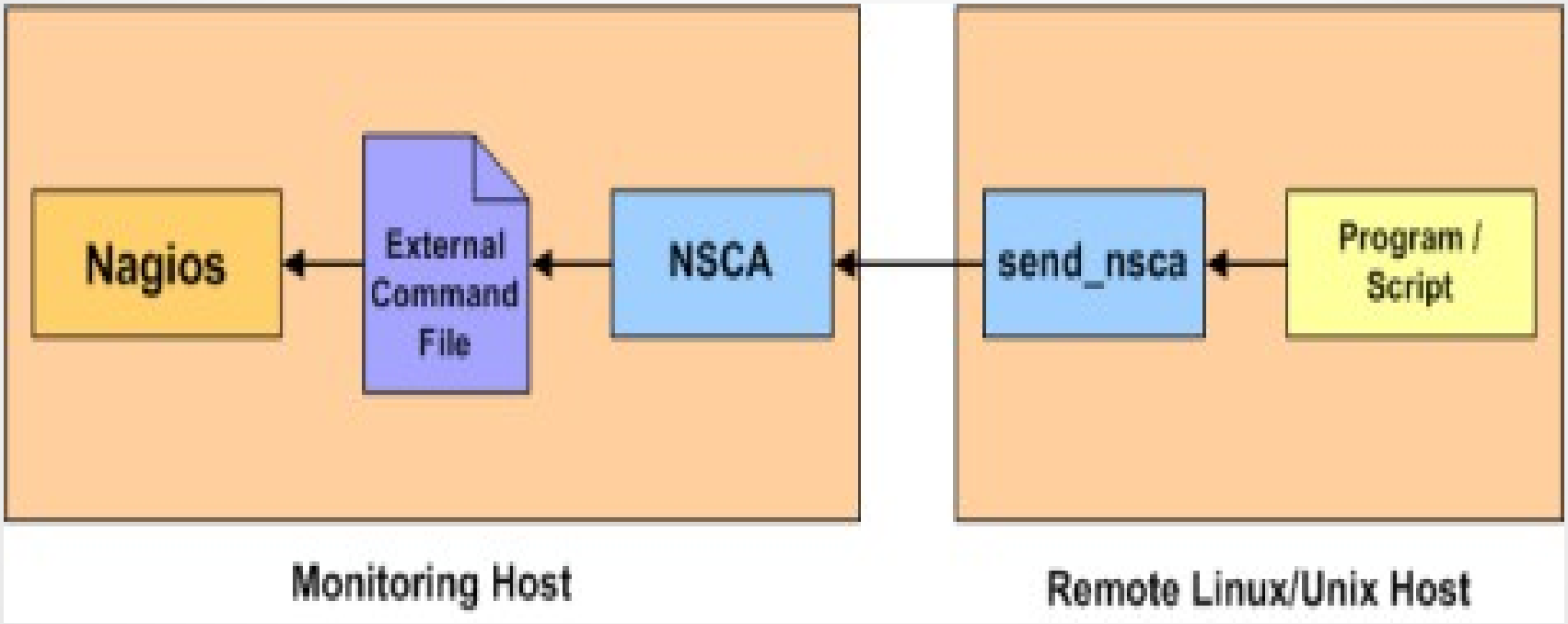
Accepts and processes check results submitted by other Nagios hosts, or any other program that manages to deliver an "NSCA message" that says something useful about

Service,
Process,
Software or
Hardware state on a (remote) host.

As such, NSCA is instrumental in remote and distributed monitoring.

Remote Monitoring Mechanisms

- Design Overview



Monitoring Windows Machines

Overview

Following three steps will happen on a very high level when Nagios (installed on the nagios-server) monitors a service (for e.g. disk space usage) on the remote Windows host..

- Nagios will **execute check_nt command on nagios-server** and request it to **monitor disk usage** on **remote windows host.**
- The **check_nt on the nagios-server will contact the NSClient++** service on remote windows host and request it to execute the **USEDDISKSPACE on the remote host.**
- The results of the **USEDDISKSPACE command** will be returned back by NSClient++ daemon to the check_nt on nagios-server.

Monitoring Windows Machines

Following flow summarizes the above explanation:

- **Nagios Server (check_nt)** ——> Remote host (NSClient++) ——> **USEDDISKSPACE**
- **Nagios Server (check_nt)** <—— Remote host (NSClient++) <—— **USEDDISKSPACE (returns disk space usage)**

Monitoring Windows Machines

- **II. 4steps to setup nagios on remote windows host**

  1. Install NSClient++ on the remote windows server

  Download NSCP 0.3.1 (NSClient++-Win32-0.3.1.msi) from NSClient++ Project. NSClient++ is an **open source windows service** that allows **performance metrics** to be gathered by Nagios for windows services.

- Go through the following five NSClient++ installation steps to get the installation completed.

  (1) NSClient++ Welcome Screen

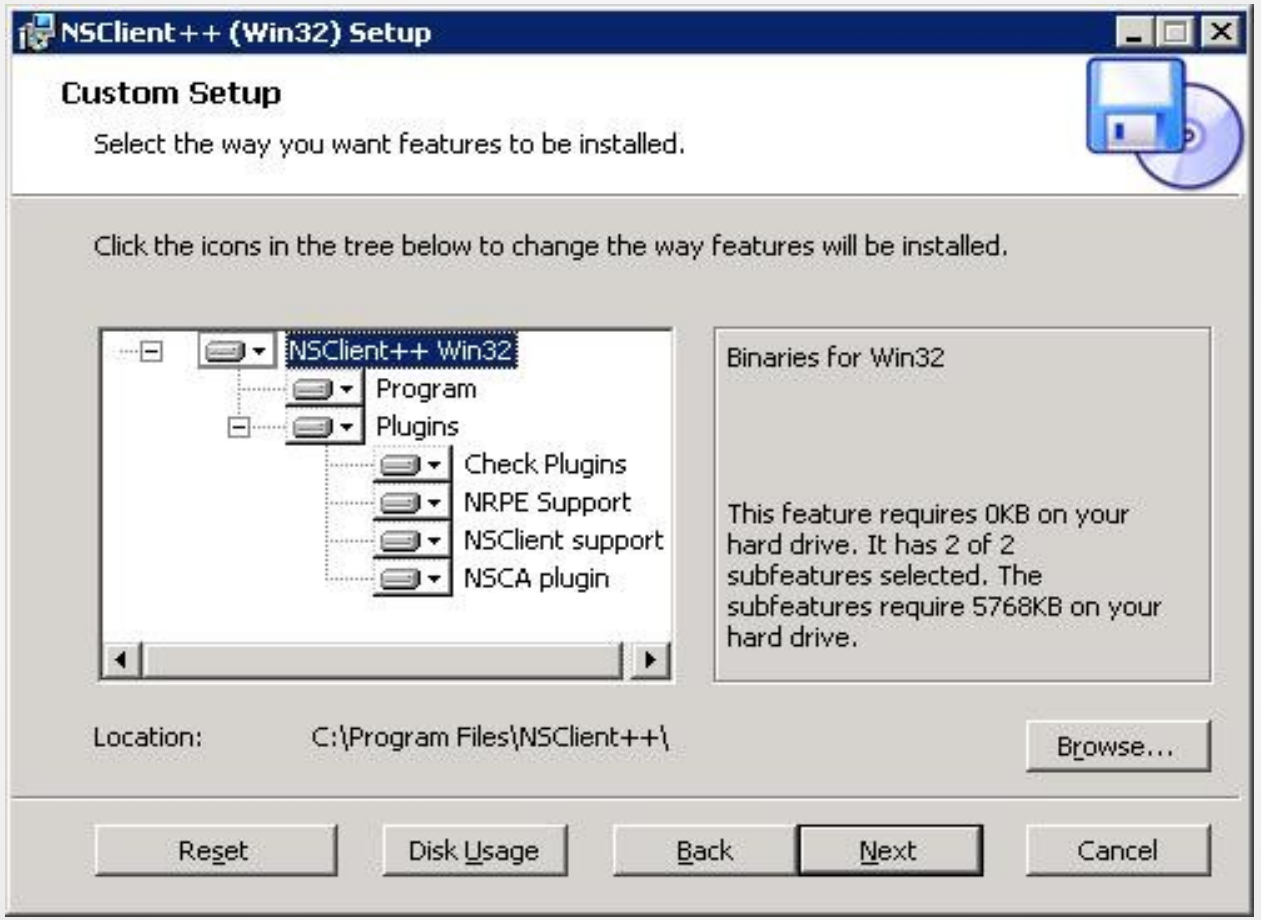
  (2) License Agreement Screen

  (3) Select Installation option and location. Use the default option and click next.

  (4) Ready to Install Screen. Click on Install to get it started.

  (5) Installation completed Screen.

Monitoring Windows Machines



Monitoring Windows Machines

3. **Modify the NSC.ini**

(1) Modify NSC.ini and uncomment *.dll: Edit the C:\Program Files\NSClient++\NSC.ini file and **uncomment everything** under [modules] **except RemoteConfiguration.dll** and CheckWMI.dll1.

[modules]

;# NSCLIENT++ MODULES

;# A list with DLLs to load at startup.

; You will need to enable some of these for NSClient++ to work.

; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

; * *

; * N O T I C E ! ! ! - Y O U H A V E T O E D I T T H I S *

; * *

Monitoring Windows Machines

; !

FileLogger.dll

CheckSystem.dll

CheckDisk.dll

NSClientListener.dll

NRPEListener.dll

SysTray.dll

CheckEventLog.dll

CheckHelpers.dll

;CheckWMI.dll

Monitoring Windows Machines

;

; RemoteConfiguration IS AN EXTREM EARLY IDEA SO DONT USE FOR PRODUCTION ENVIROMNEMTS!

;RemoteConfiguration.dll

; NSCA Agent is a new beta module use with care!

NSCAAgent.dll

; LUA script module used to write your own "check deamon" (sort of) early beta.

LUAScript.dll

; Script to check external scripts and/or internal aliases, early beta.

CheckExternalScripts.dll

; Check other hosts through NRPE extreme beta and probably a bit dangerous! :)

NRPEClient.dll

Monitoring Windows Machines

(3) **Modify NSC.ini** and **uncomment port.** Edit the C:\Program Files\NSClient++\NSC.ini file and uncomment the port# under [NSClient] section

;# NSCLIENT PORT NUMBER

; This is the port the NSClientListener.dll will listen to.

port=12489

Monitoring Windows Machines

4) Modify NSC.ini and specify password. You can also specify a password the nagios server needs to use to remotely access the NSClient++ agent.

[Settings]

;# OBFUSCATED PASSWORD

; This is the same as the password option but here you can store the password in an obfuscated manner.

; *NOTICE* obfuscation is *NOT* the same as encryption, someone with access to this file can still figure out the

; password. Its just a bit harder to do it at first glance.

;obfuscated_password=Jw0KAUUdXlAAUwASDAAB

;

;# PASSWORD

Monitoring Windows Machines

; This is the password (-s) that is required to access NSClient remotely. If you leave this blank everyone will be able to access the daemon remotly.

password=My2Secure$Password

4. Start the NSClient++ Service

Start the NSClient++ service

either from the Control Panel -> Administrative tools -> Services -> Select “NSClientpp (Nagios) 0.3.1.14 2008-03-12 w32″ and click on start (or) Click on “Start -> All Programs -> NSClient++ -> Start NSClient++ (Win32) . **Please note that this will start the NSClient++ as a windows service.**

Later if you modify anything in the NSC.ini file, you should restart the “NSClientpp (Nagios) 0.3.1.14 2008-03-12 w32″ from the windows service.

Monitoring Windows Machines

Verify that the windows-server template is enabled under
/usr/local/nagios/etc/objects/templates.cfg

```
# Windows host definition template - This is NOT a real host, just a template!

define host{

name                windows-server; The name of this host template

use                 generic-host    ; Inherit default values from the generic-   host template

check_period     24x7   ; By default, Windows servers are  monitored round the clock

check_interval         5              ; Actively check the server every 5  minutes
```

Monitoring Windows Machines

```
retry_interval                1                ; Schedule host check retries at 1
                                                 minute intervals
max_check_attempts            10               ; Check each server 10 times
                                               (max)
check_command                 check-host-alive  ; Default command to check if
                                                 servers are "alive"
notification_period           24x7              ; Send notification out at any time
                                                 - day or night
notification_interval         30                ; Resend notifications every 30
                                                 minutes
notification_options          d,r              ; Only send notifications for
                                                specific host states
contact_groups                admins           ; Notifications get sent to the
                                                admins by default
```

Monitoring Windows Machines

hostgroups windows-servers ; Host groups that Windows servers should be a member of

register 0 ; DONT REGISTER THIS – ITS JUST A TEMPLATE

}

2. Uncomment windows.cfg in /usr/local/nagios/etc/nagios.cfg

Definitions for monitoring a Windows machine

cfg_file=/usr/local/nagios/etc/objects/windows.cfg

Monitoring Windows Machines

address 192.168.1.4 ; IP address of the remote windows host

}

4. Define windows services that should be monitored.

Following are the default windows services that are already enabled in the sample windows.cfg. Make sure to update the host_name on these services to reflect the host_name defined in the above step.

define service{

use generic-service

host_name remote-windows-host

service_description NSClient++ Version

check_command check_nt!CLIENTVERSION

}

Monitoring Windows Machines

```
define service{
use                     generic-service
host_name               remote-windows-host
service_description     Uptime
check_command           check_nt!UPTIME
}
define service{
use                     generic-service
host_name               remote-windows-host
service_description     CPU Load
check_command           check_nt!CPULOAD!-l 5,80,90
}
```

Monitoring Windows Machines

```
define service{
use                     generic-service
host_name               remote-windows-host
service_description     Uptime
check_command           check_nt!UPTIME
}
define service{
use                     generic-service
host_name               remote-windows-host
service_description     CPU Load
check_command           check_nt!CPULOAD!-l 5,80,90
}
```

Monitoring Windows Machines

5. Enable Password Protection

If you specified a password in the NSC.ini file of the NSClient++ configuration file on the Windows machine,

you'll need to **modify the check_nt command** definition to include the password.

Modify the **/usr/local/nagios/etc/commands.cfg** file **and add password** as shown below.

```
define command{

command_name  check_nt

command_line    $USER1$/check_nt -H $HOSTADDRESS$ -p 12489 -s My2Secure$Password -v $ARG1$ $ARG2$

}
```

Monitoring Windows Machines

6. Verify Configuration and Restart Nagios.

Verify the nagios configuration files as shown below.

[nagios-server]#/**usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg**

Total Warnings: 0

Total Errors: 0

Things look okay - No serious problems were detected during the pre-flight check

Monitor Remote Linux Host

I. Overview:

- Following three steps will happen on a very high level when Nagios (installed on the nagios-servers) monitors a service (for e.g. disk space usage) on the remote Linux host.

- **Nagios will execute check_nrpe command** on nagios-server and request it to monitor disk usage on remote host using check_disk command.

- The **check_nrpe on the nagios-server will contact the NRPE daemon** on remote host and request it to execute the check_disk on remote host.

- The results of the **check_disk command** will be returned back by **NRPE daemon** to the **check_nrpe** on nagios-server.

Monitor Remote Linux Host

II. 7 steps to install Nagios Plugins and NRPE on the remote host.

1. Download Nagios Plugins and NRPE Add-on

Download following files from Nagios.org and move to /home/downloads:

- **nagios-plugins-1.4.11.tar.gz**
- **nrpe-2.12.tar.gz**

Monitor Remote Linux Host

2. Create nagios account

- [remotehost]# useradd nagios
- [remotehost]# passwd usersecret

3. Install nagios-plugin

- [remotehost]# cd /home/downloads
- [remotehost]# tar xvfz nagios-plugins-1.4.11.tar.gz
- [remotehost]# cd nagios-plugins-1.4.11
- [remotehost]# export LDFLAGS=-ldl

Monitor Remote Linux Host

- [remotehost]# **./configure --with-nagios-user=nagios --with-nagios-group=nagios --enable-redhat-pthread-workaround**
- [remotehost]# make
- [remotehost]# make install
- [remotehost]# chown nagios.nagios /usr/local/nagios
- [remotehost]# chown -R nagios.nagios /usr/local/nagios/libexec/

Note: **On Red Hat, For me the ./configure command was hanging with the the message: “checking for redhat spopen problem…”. Add --enable-redhat-pthread-workaround to the ./configure command as a work-around for the above problem.**

Monitor Remote Linux Host

5. Setup NRPE to run as daemon (i.e as part of xinetd):

- Modify the /etc/xinetd.d/nrpe to add the ip-address of the Nagios monitoring server to the only_from directive.

  Note that there is a space after the 127.0.0.1 and the nagios monitoring server ip-address (in this example, nagios monitoring server ip-address is: 192.168.1.2)

  only_from = 127.0.0.1 **192.168.1.2**

- Modify the **/etc/services** and add the following at the end of the file.

  nrpe 5666/tcp # NRPE

- Start the service

  [remotehost]#**service xinetd restart**

Monitor Remote Linux Host

- **Verify whether NRPE is listening**

  [remotehost]# netstat -at | grep nrpe

  tcp 0 0 *:nrpe *:* LISTEN

- **Verify to make sure the NRPE is functioning properly**

  [remotehost]# /usr/local/nagios/libexec/check_nrpe -H localhost

  NRPE v2.12

- **6. Modify the /usr/local/nagios/etc/nrpe.cfg**

  The nrpe.cfg file located on the remote host contains the commands that are needed to check the services on the remote host.

- By default the nrpe.cfg comes with few standard check commands as samples.

- check_users and check_load are shown below as an example.

Monitor Remote Linux Host

command[check_users]=/usr/local/nagios/libexec/check_users -w 5 -c 10

command[check_load]=/usr/local/nagios/libexec/check_load -w 15,10,5 -c 30,25,20

In all the check commands, the "**-w" stands for "Warning"** and "**-c" stands for "Critical".**

- for e.g. in the check_disk command below, if the available disk space gets to 20% of less, nagios will send warning message.
- If it gets to 10% or less, nagios will send critical message. Change the value of "-c" and "-w" parameter below depending on your environment.

command[check_disk]=/usr/local/nagios/libexec/check_disk -w 20% -c 10% -p /dev/hda1

Monitor Remote Linux Host

III. 4 Configuration steps on the Nagios monitoring server to monitor remote host:.

- **1. Download NRPE Add-on**
  - Download nrpe-2.12.tar.gz from Nagios.org and move to /home/downloads:te:
  -

2. Install check_nrpe on the nagios monitoring server

- [nagios-server]# tar xvfz nrpe-2.12.tar.gz
- [nagios-server]# cd nrpe-2.1.2
- [nagios-server]# ./configure
- [nagios-server]# make all

Monitor Remote Linux Host

- [nagios-server]# make install-plugin
- ./configure will give a configuration summary as shown below:
- *** Configuration summary for nrpe 2.12 05-31-2008 ***:
- General Options:
- ————————————-
- NRPE port: 5666
- NRPE user: nagios
- NRPE group: nagios
- Nagios user: nagios
- Nagios group: nagios

Monitor Remote Linux Host

Note: **I got the “checking for SSL headers… configure: error: Cannot find ssl headers”** error message while performing ./configure. Install openssl-devel as shown below and run the ./configure again to fix the problem.

- [nagios-server]# rpm -ivh openssl-devel-0.9.7a-43.16.i386.rpm krb5-devel-1.3.4-47.i386.rpm zlib-devel-1.2.1.2-1.2.i386.rpm e2fsprogs-devel-1.35-12.5.
- el4.i386.rpm
- warning: openssl-devel-0.9.7a-43.16.i386.rpm: V3 DSA signature: NOKEY, key ID db42a60e
- Preparing… ### [100%]
- 1:e2fsprogs-devel ### [25%]
- 2:krb5-devel ### [50%]

Monitor Remote Linux Host

host definition sample:

- define host{
- use linux-server
- host_name remotehost
- alias Remote Host
- address 192.168.1.3
- contact_groups admins
- }

Monitor Remote Linux Host

Service definition sample:

-
- define service{
- use generic-service
- service_description Root Partition
- contact_groups admins
- check_command check_nrpe!check_disk
- }

Note: In all the above examples, replace remotehost with the corresponding hostname of your remotehost.

Monitor Remote Linux Host

4. Restart the nagios service

Restart the nagios as shown below and login to the nagios web (http://nagios-server/nagios/) to verify the status of the remotehost linux sever that was added to nagios for monitoring.

- [nagios-server]# **service nagios reload**

Monitor Network Switch and Ports

3. Add a new host for the switch to be monitered

In this example, I've defined a host to monitor the **core switch** in the **/usr/local/nagios/etc/objects/switch.cfg** file.

Change the address directive to your switch ip-address accordingly.

- define host{
- use generic-switch
- host_name **core-switch**
- alias Cisco Core Switch
- address **192.168.1.50**
- hostgroups switches
- }

Monitor Network Switch and Ports

4. Add common services for all switches

Displaying the **uptime of the switch** and

verifying whether **switch is alive** are common services for all switches.

- So, define these services under the switches hostgroup_name as shown below.
- # Service **definition to ping the switch using check_ping**
- define service{
- use generic-service
- hostgroup_name switches
- service_description PING
- check_command check_ping!200.0,20%!600.0,60%
- normal_check_interval 5
- retry_check_interval 1
- }

Monitor Network Switch and Ports

```
# Service definition to monitor switch uptime using check_snmp
define service{
use                     generic-service
hostgroup_name          switches
service_description     Uptime
check_command           check_snmp!-C public -o sysUpTime.0
}
```

Monitor Network Switch and Ports

6. Add service to monitor an active switch port

Use check_snmp to monitor the specific port as shown below.

The following two services monitors port#1 and port#5. To add additional ports, change the value ifOperStatus.n accordingly. i.e n defines the port#.

- # Monitor status of port number 1 on the Cisco core switch
- define service{
- use generic-service
- host_name core-switch
- service_description Port 1 Link Status
- check_command check_snmp!-C public -o ifOperStatus.1 -r 1 -m RFC1213-MIB
- }

Monitor Network Switch and Ports

```
# Monitor status of port number 5 on the Cisco core switch
define service{
use                     generic-service
host_name               core-switch
service_description     Port 5 Link Status
check_command   check_snmp!-C public -o ifOperStatus.5 -r 1 -m RFC1213-MIB
}
```

Monitor Network Switch and Ports

7. Add services to monitor multiple switch ports together

Sometimes you may need to **monitor the status of multiple ports** combined together.

- i.e Nagios should **send an alert, even if one of the port is down**.

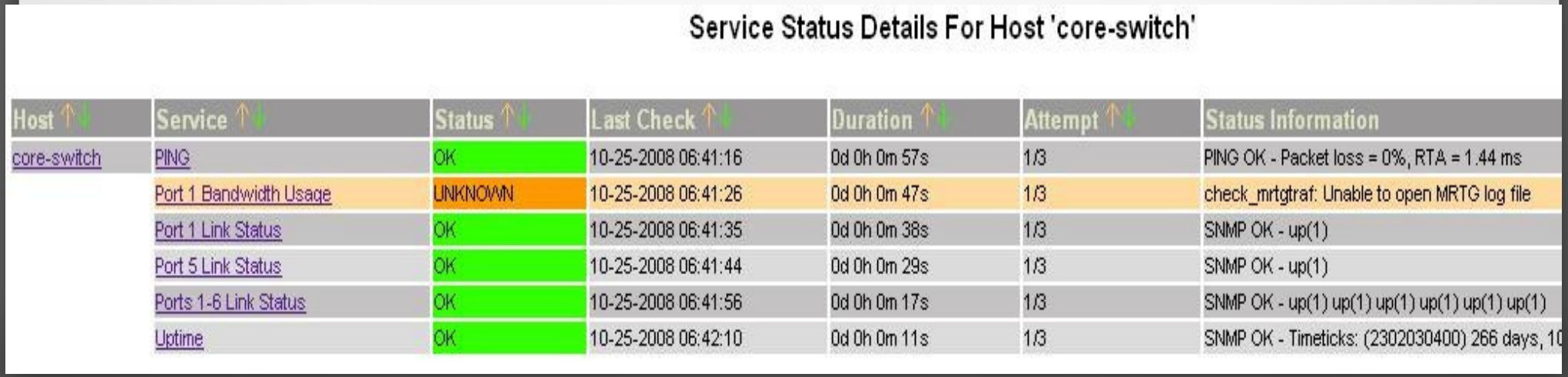
In this case, define the following service to monitor multiple ports.

- # Monitor ports 1 - 6 on the Cisco core switch.
- define service{
- use generic-service
- host_name core-switch
- service_description Ports 1-6 Link Status
- check_command check_snmp!-C public -o ifOperStatus.1 -r 1 -m RFC1213-MIB, -o ifOperStatus.2 -r 1 -m RFC1213-MIB, -o ifOperStatus.3 -r 1 -m RFC1213-MIB, -o ifOperStatus.4 -r 1 -m RFC1213-MIB, -o ifOperStatus.5 -r 1 -m RFC1213-MIB, -o ifOperStatus.6 -r 1 -m RFC1213-MIB

Monitor Network Switch and Ports

- # /etc/rc.d/init.d/nagios stop
- Stopping nagios: .done.
- # /etc/rc.d/init.d/nagios start
- Starting nagios: done.

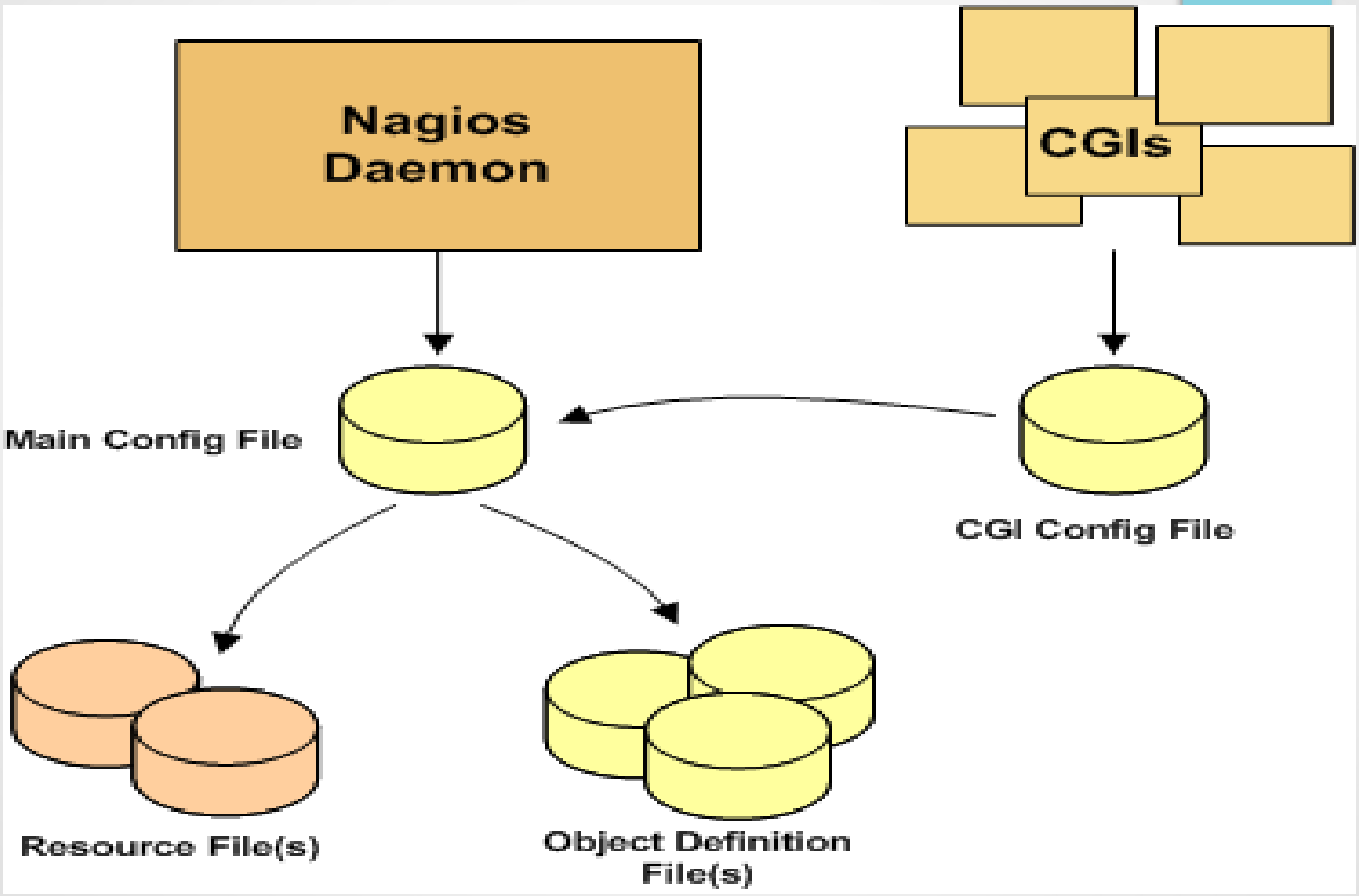
Verify the status of the switch from the Nagios web UI: http://{nagios-server}/nagios as shown below:



Service Status Details For Host 'core-switch'

Host	Service	Status	Last Check	Duration	Attempt	Status Information
core-switch	PING	OK	10-25-2008 06:41:16	0d 0h 0m 57s	1/3	PING OK - Packet loss = 0%, RTA = 1.44 ms
	Port 1 Bandwidth Usage	UNKNOWN	10-25-2008 06:41:26	0d 0h 0m 47s	1/3	check_mrtgtraf: Unable to open MRTG log file
	Port 1 Link Status	OK	10-25-2008 06:41:35	0d 0h 0m 38s	1/3	SNMP OK - up(1)
	Port 5 Link Status	OK	10-25-2008 06:41:44	0d 0h 0m 29s	1/3	SNMP OK - up(1)
	Ports 1-6 Link Status	OK	10-25-2008 06:41:56	0d 0h 0m 17s	1/3	SNMP OK - up(1) up(1) up(1) up(1) up(1) up(1)
	Uptime	OK	10-25-2008 06:42:10	0d 0h 0m 11s	1/3	SNMP OK - Timeticks: (2302030400) 266 days, 10

Configuration overview



Configuration overview

Resource File(s)

Resource files can be used to **store user-defined macros.**

The main point of having resource files is to **use them to store sensitive configuration information (like passwords),** without making them available to the CGIs.

Format: **resource_file=<file_name>**

Example:

resource_file=/usr/local/nagios/etc/resource.cfg

Configuration overview

Main Configuration File

The main configuration file contains a number of directives that affect how the Nagios daemon operates.

This **config file is read by both the Nagios daemon** and the CGIs.

This is where you're going to want to get started in your configuration adventures.

- **Config File Location**

  The main configuration file is usually named **nagios.cfg** and located in the **/usr/local/nagios/etc/ directory.**

Configuration overview

CGI Configuration File

The CGI configuration file contains **a number of directives that affect the operation of the CGIs.**

It also contains a reference the main configuration file,

so the CGIs know how you've configured Nagios and

where your object defintions are stored.

A sample CGI configuration file (/usr/local/nagios/etc/cgi.cfg) is installed for you when you follow the quickstart installation guide.

Configuration overview

Main configuration file options

Log File

Format: **log_file=<file_name>**

- Example:

  log_file=/usr/local/nagios/var/nagios.log

  This variable specifies where Nagios should create its main log file.

- This should be the first variable that you define in your configuration file, as Nagios will try to write errors that it finds in the rest of your configuration data to this file.

- If you have log rotation enabled, this file will automatically be rotated every hour, day, week, or month.

Configuration overview

Main configuration file options

- **Temp File**

  Format: temp_file=<file_name>

  Example:

  temp_file=/usr/local/nagios/var/nagios.tmp

  This is a temporary file that Nagios periodically creates to use when updating comment data, status data, etc.

Configuration overview

Main configuration file options

Status File

Format: **status_file=<file_name>**

Example:

status_file=/usr/local/nagios/var/status.dat

This is the file that Nagios uses to store the current status, comment, and downtime information.

This file is used by the CGIs so that current monitoring status can be reported via a web interface.

The CGIs must have read access to this file in order to function properly.

Note: This file is deleted every time Nagios stops and recreated when it starts.

Object configuration overview

What Are Objects?

Objects are all the elements that are involved in the monitoring and notification logic. Types of objects include:

- **Services**
- **Service Groups**
- **Hosts**
- **Host Groups**
- **Contacts**
- **Contact Groups**
- **Commands**
- **Time Periods**
- **Notification Escalations**
- **Notification and Execution Dependencies**

Object configuration overview

Where Are Objects Defined?

Objects can be defined in one or more configuration files and/or directories that you specify using the **cfg_file** and/or **cfg_dir** directives in the main configuration file.

How Are Objects Defined?

Objects are defined in a flexible template format, which can make it much easier to manage your Nagios configuration in the long term.

Object configuration overview

Host Groups are groups of one or more hosts.

Host groups can make it easier to

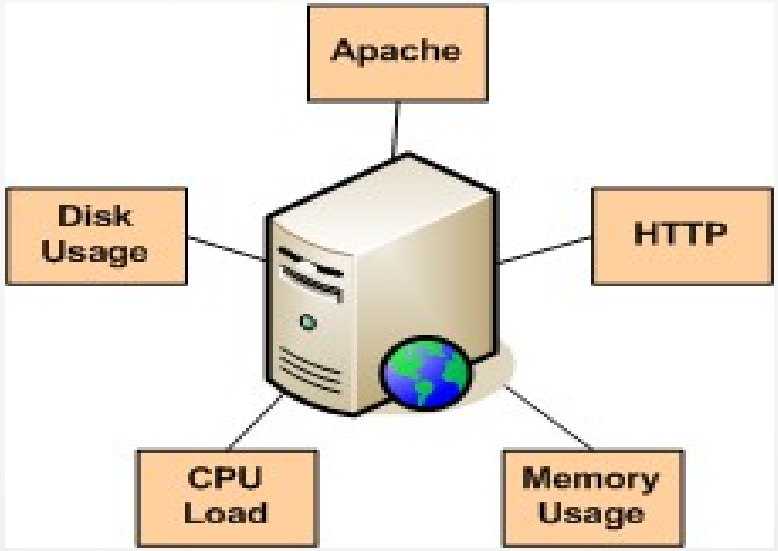
(1) view the status of related hosts in the Nagios web interface and

(2) simplify your configuration through the use of object tricks.

Object configuration overview

Services are one of the central objects in the monitoring logic.

- Services are associated with hosts and can be:
- **Attributes of a host (CPU load, disk usage, uptime, etc.)**
- **Services provided by the host (HTTP, POP3, FTP, SSH, etc.)**
- **Other things associated with the host (DNS records, etc.)**



Object configuration overview

Service Groups are groups of one or more services.

Service groups can make it easier to

(1) view the status of related services in the Nagios web interface and

(2) simplify your configuration through the use of object tricks.

Object configuration overview

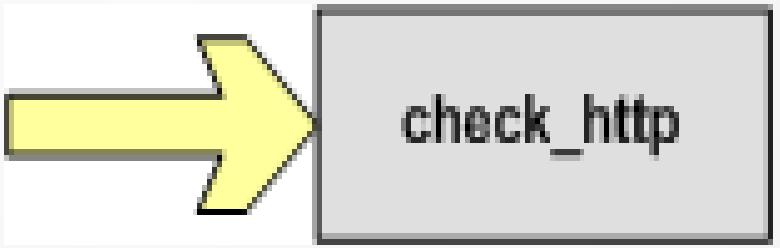
Timeperiods are are used to control:

- When hosts and services can be monitored
- When contacts can receive notifications

Object configuration overview

Commands are used to tell Nagios what programs, scripts, etc. it should

- execute to perform:
- Host and service checks
- Notifications
- Event handlers
- and more...



Object Definitions

Introduction

With Nagios, you're going to monitor hosts and services.

Being a flexible framework, Nagios lets you define those yourself, as "object definitions" in configuration files

When creating and/or editing configuration files, keep the following in mind:

- Lines that start with a '#' character are taken to be comments and are not processed
- Directive names are case-sensitive
- Characters that appear after a semicolon (;) in configuration lines are treated as comments and are not processed

Path /usr/local/nagios/etc/object/

Object Definitions

Host Definition

"Host" monitoring only tells you if a host is up, i.e. responding to a ping from the Nagios server.

A sample definition of a host is as follows:

```
define host
{
host_name                       linuxbox01
hostgroups                      linuxservers
alias                           Linux Server 01
address                         10.0.2.1
check_command                   check-host-alive
check_interval                  5
retry_interval                  1
max_check_attempts5
check_period                    24x7
contact_groups                  linux-admins
notification_interval           30
notification_period             24x7
notification_options            d,u,r
}
```

The slide defines a Linux box that will use the **check-host-alive command** to make sure the box is up and running.

The test will be performed every five minutes,

and after five failed tests,

it will assume the host is down.

If it is down, a notification will be sent out every 30 minutes.

Object Definitions

Determining Status and Reachability of Network Hosts

- Nagios is able to determine whether the hosts you're monitoring are in a **DOWN or UNREACHABLE state.**

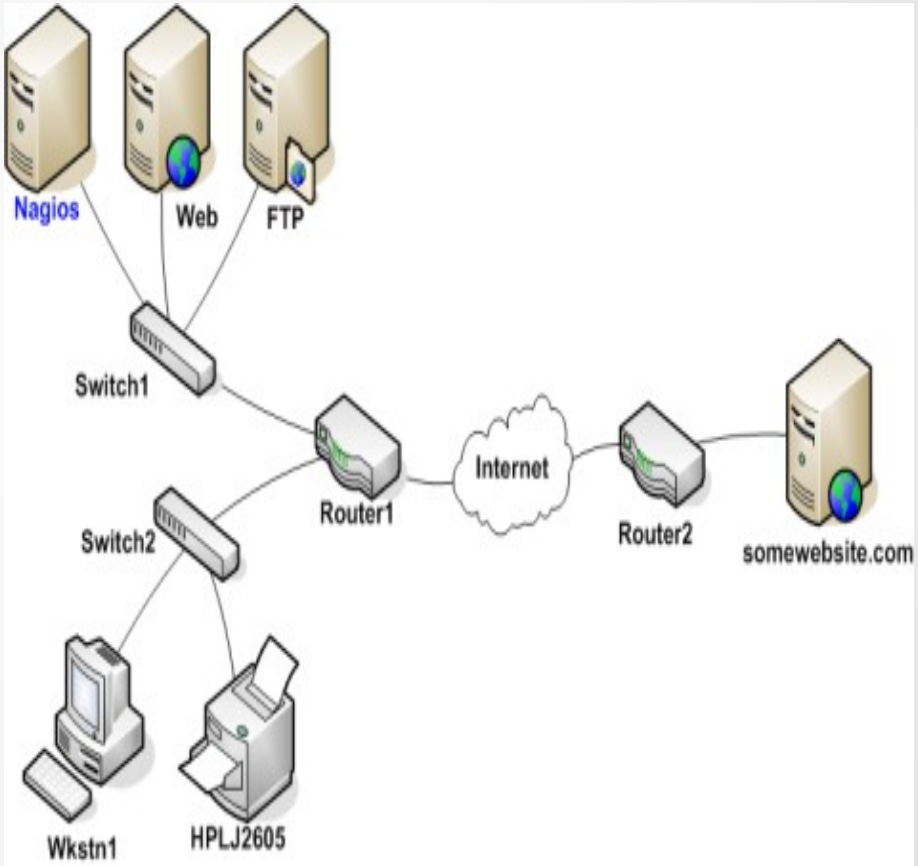
  These are very different (although related) states and can help you quickly determine the root cause of network problems.

  Here's how the reachability logic works to distinguish between these two states...

Object Definitions

Determining Status and Reachability of Network Hosts

- Take a look at the simple network diagram. For this example, lets assume you're monitoring all the hosts (server, routers, switches, etc) that are pictured. Nagios is installed and running on the Nagios host.

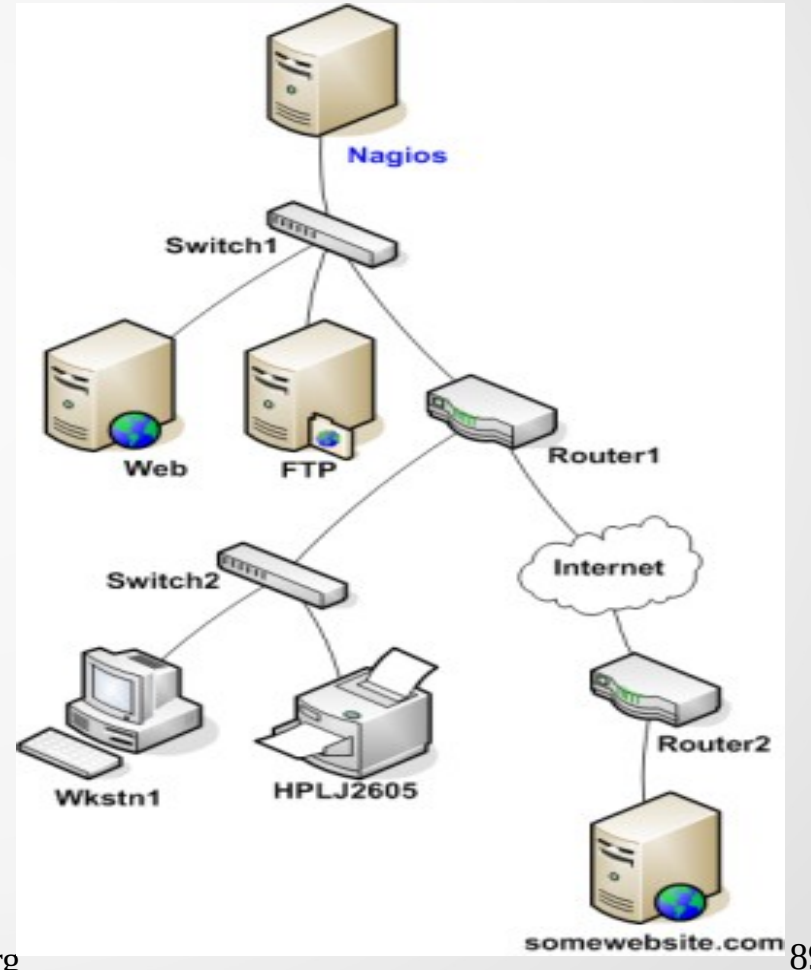


Object Definitions

Defining Parent/Child Relationships

- Now that you know what the parent/child
- relationships look like for hosts that are
- being monitored,
- how do you configure Nagios to reflect them?

- **The parents directive in your host definitions**
- **allows you to do this.**



Object Definitions

Defining Parent/Child Relationships

define host{

host_name **Nagios** ; <-- The local host has no parent - it is the topmost host

}

Object Definitions

```
define host
{
host_name          Web
parents            Switch1
}
```

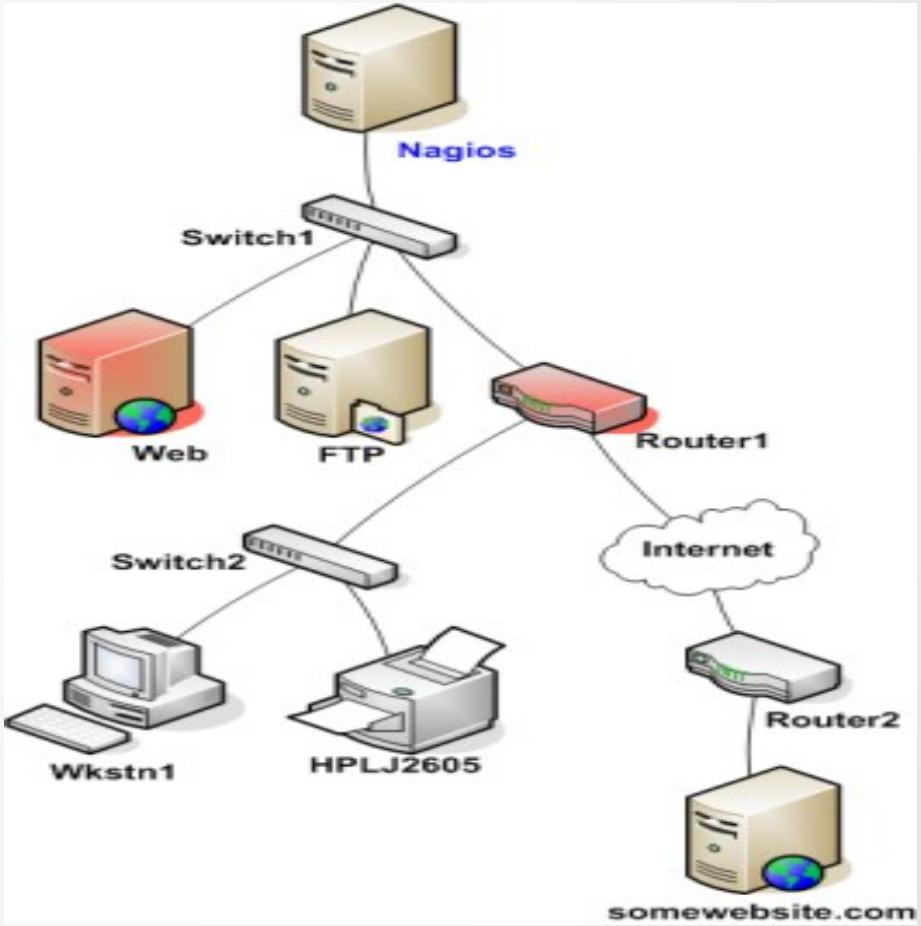
```
define host
{
host_name          ftp
parents            Switch1
}
```

```
define host
{
host_name          Router1
parents            Switch1
}
```

```
define host
{
host_name          Switch2
parents   Router1
}
```

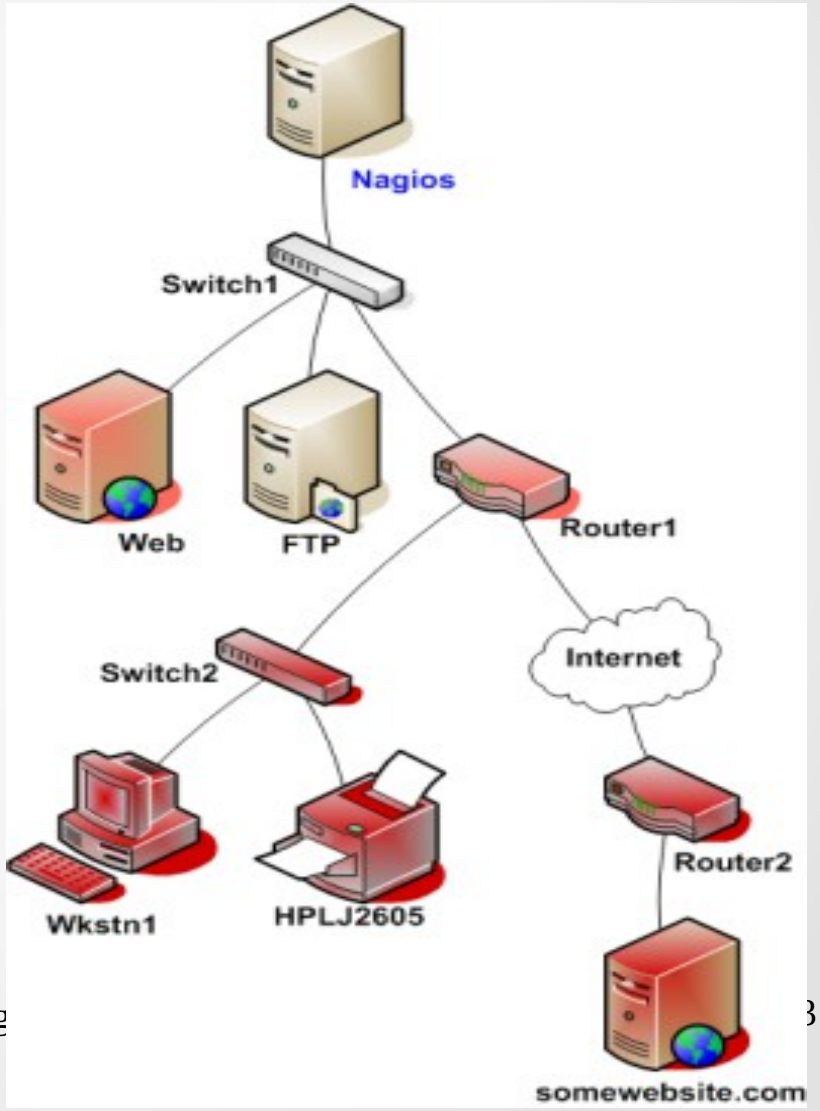
Object Definitions

- **Reachability Logic in Action**
- Now that you're configured Nagios with the proper parent/child relationships for your hosts,
- let's see what happen when problems arise.
- Assume that two hosts - **Web and Router1 - go offline...**



Object Definitions

- **Reachability Logic in Action**
- When hosts change state (i.e. from UP to DOWN),
- the host reachability logic in Nagios kicks in
- The reachability logic will initiate parallel checks of the parents and children of whatever hosts change state.
- This allows Nagios to quickly determine the current status of your network infrastructure when changes occur.
-

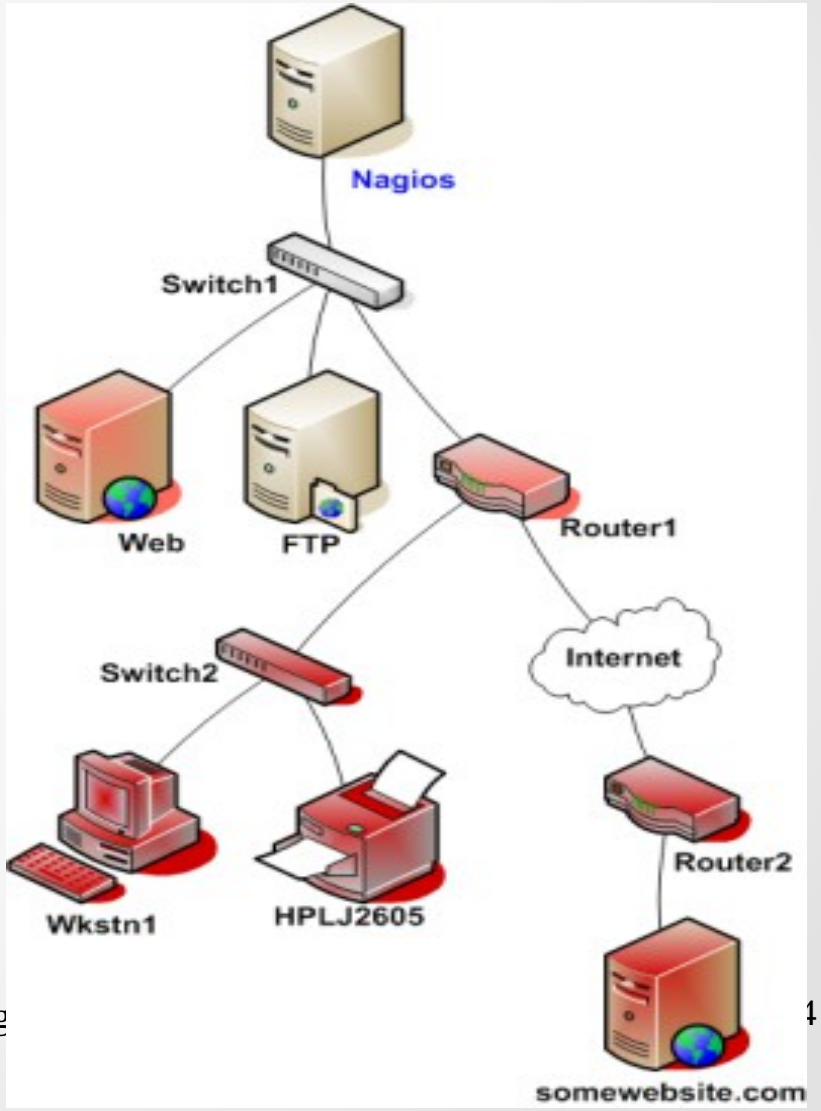


Object Definitions

- **Reachability Logic in Action**

Final conclusion

- Nagios will determine that Web and Router1 are both in DOWN states because the "path" to those hosts is not being blocked.
- Nagios will determine that all the hosts "beneath" Router1 are all in an UNREACHABLE state because Nagios can't reach them.
- Router1 is DOWN and is blocking the path to those other hosts
- Those hosts might be running fine, or they might be offline -
- Nagios doesn't know because it can't reach them. **Hence Nagios considers them to be UNREACHABLE instead of DOWN.**



Object Definitions

```
define hostgroup
{
  hostgroup_name          linux-servers
Alias                                  Linux servers
Members                   linuxbox1,linuxbox2
}
```

```
define hostgroup
{
  hostgroup_name          aix-servers
Alias                     AIX servers
Members                   aixbox1,aixbox2
}
```

```
define hostgroup
{
  hostgroup_name          unix-servers
Alias                     UNIX servers
Members                   llinux-servers,aix-servers
}
```

hostgroup_name Short, unique name of the host group

alias Descriptive name of the host group

members List of all hosts that should be a member of this group, separated by commas

hostgroup_members List of all other host groups whose members should also be members of this group, separated by commas

Object Definitions

- **Host Group Definition**
- Host groups can also be used when defining services or dependencies. For example,

- it is possible to tell Nagios that all Linux servers should have their SSH service monitored and

- all AIX servers should have a telnet accepting connections.

- It is also possible to define dependencies between hosts
  They are, in a way, similar to a parent-host relationship, but dependencies offer more complex configuration options.

- Nagios will only issue host and service checks if all dependant hosts are currently up

Object Definitions

Service Group Definition

Services can be grouped in a similar way to host objects

This can be done to manage services more conveniently.

It also aids in viewing service reports on the Nagios web interface.

Service groups are also used to configure dependencies in a more convenient way.

Object Definitions

Contact Definition

Contacts define people who can either be owners of specific machines, or people who should be contacted in case of problems.

- Depending on how your organization chooses to contact people in case of problems, the definition of a contact may vary a lot.
- A contact consists of a unique name, a descriptive name, and one or more email addresses and/or pager numbers.
- Contact definitions can also contain additional data specific to how a person can be contacted.

Contact Definition

Example

```
define contact
{
contact_name                    jdoe
alias                           John Doe
email                           john.doe@yourcompany.com
host_notification_period        24x7
service_notification_period     24x7
host_notification_options       d,u,r
service_notification_options    w,u,c,r
host_notification_commands      host-notify-by-email
service_notification_commands   notify-by-email
}
```

Object Definitions

Contact Group Definition

Contacts can be grouped. Usually, grouping is used to keep a list of which users are responsible for which tasks,

- and the group maps to job responsibilities for particular people.
- It also makes it possible to define people who should be responsible for
- handling problems at specific time periods,
- and Nagios will automatically contact the right people depending on the time at which a problem has occurred.
- It is also possible to specify different people or groups for handling host-related and service-related problems—

for example, hardware administrators for handling host problems and system administrators for handling service issues.

Object Definitions

Time Period Definition

- Timeperiod definitions allow you to control when various aspects of the monitoring and alerting logic can operate.

- For instance, you can restrict:
- When regularly scheduled host and service checks can be performed
- When notifications can be sent out
- When notification escalations can be used
- When dependencies are valid

Object Definitions

How Time Periods Work With Service Checks

- Without the implementation of time periods, Nagios would monitor all services that you had defined 24 hours a day, 7 days a week.
- While this is fine for most services that need monitoring, it doesn't work out so well for others.
- For instance, do you really need to monitor printers all the time when they're really only used during normal business hours?
- Perhaps you have development servers which you would prefer to have up, but aren't "mission critical" and therefore don't have to be monitored for problems over the weekend.
- Time period definitions now allow you to have more control over when such services may be checked...

Object Definitions

define timeperiod{

- timeperiod_name admin1-oncall
- alias Admin1 On Call
- 2009-04-17 / 14 15:00-24:00
- 2009-04-18 / 14 00:00-24:00
- 2009-04-19 / 14 00:00-24:00
- 2009-04-20 / 14 00:00-24:00
- 2009-04-21 / 14 00:00-24:00
- 2009-04-22 / 14 00:00-24:00
- 2009-04-23 / 14 00:00-24:00
- 2009-04-24 / 14 00:00-15:00
- }

define timeperiod{

- timeperiod_name admin2-oncall
- alias Admin2 On Call
- 2009-04-24 / 14 15:00-24:00
- 2009-04-25 / 14 00:00-24:00
- 2009-04-26 / 14 00:00-24:00
- 2009-04-27 / 14 00:00-24:00
- 2009-04-28 / 14 00:00-24:00
- 2009-04-29 / 14 00:00-24:00
- 2009-04-30 / 14 00:00-24:00
- 2009-05-01 / 14 00:00-15:00
- }

Object Definitions

Command Definition

Example Definition:

```
define command{
command_name check_pop
command_line /usr/local/nagios/libexec/check_pop -H $HOSTADDRESS$
}
```

Object Definitions

Service Dependency Definition

- Service dependencies work in a similar way as host dependencies.
- For services dependencies, you need to define a master service and a dependent service
- For example,
  you can tell Nagios that POP3 services on the emailservers host group depend on the LDAP service on the ldapserver host.

Service Dependency Definition

Example

```
define servicedependency
{
host_name                       ldapserver
service_description             LDAP
dependent_hostgroup_name        emailservers
dependent_service_description   POP3
execution_failure_criteria      c,u
notification_failure_criteria   c,u,w

}
```

Object Definitions

Host Dependency Definition

Example Definition:

```
define hostdependency{
host_name                       WWW1
dependent_host_name             DBASE1
notification_failure_criteria  d,u
}
```

Object Definitions

- **Service Escalation Definition**

  Service escalations are completely optional and are used to escalate notifications for a particular service.

```
define serviceescalation{
host_name webserver
service_description HTTP
first_notification 3
last_notification 5
notification_interval 90
contact_groups nt-admins,managers
}
```

```
define serviceescalation{
host_name
webserver
234
service_description HTTP
first_notification 6
last_notification 10
notification_interval 60
contact_groups nt-admins,managers,everyone
}
```

Object Definitions

- **Host escalations Definition**

  Host escalations are completely optional and are used to escalate notifications for a particular host.

```
define hostescalation{
host_name router-34
first_notification 5
last_notification 8
notification_interval 60
contact_groups all-router-admins
}

define hostescalation{
host_name router-34
first_notification 9
last_notification 20
notification_interval 60
contact_groups all-network-manager
}
```

Starting and Stopping Nagios

Start Nagios

1) **Init Script:**
   The easiest way to start the Nagios daemon is by using the init script like so:
   /etc/rc.d/init.d/nagios start

2) **Manually:** You can start the Nagios daemon manually with the -d command line option like so:
   /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg

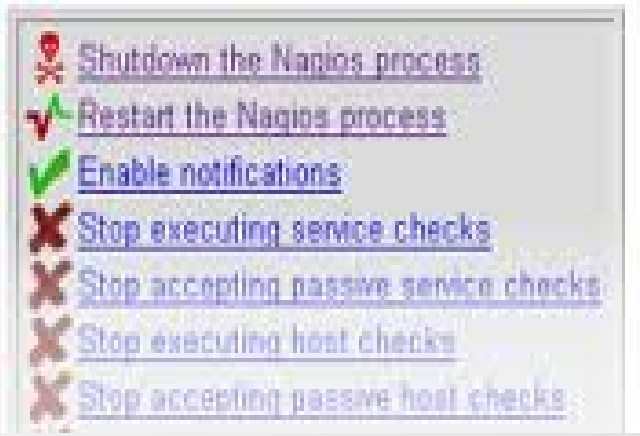
Starting and Stopping Nagios

Restarting Nagios

Restarting/reloading is nececessary when you modify your configuration files and want those changes to take effect.

1) Init Script: The easiest way to restart the Nagios daemon is by using the init script like so: /etc/rc.d/init.d/nagios reload

2) Web Interface: You can restart the Nagios through the web interface by clicking the "Process Info"navigation link and selecting "Restart the Nagios process":



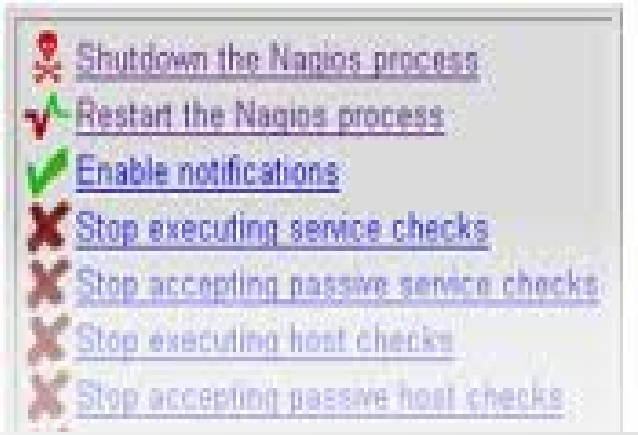
Starting and Stopping Nagios

Stopping Nagios

1) Init Script: he easiest way to stop the Nagios daemon is by using the init script like so: /etc/rc.d/init.d/nagios stop

2) Web Interface: You can stop the Nagios through the web interface by clicking the "Process Info" navigation link and selecting "Shutdown the Nagios process":

3) Manually: You can stop the Nagios process by sending it a SIGTERM signal like so:
kill <nagios_pid>



Notifications

Notifications may be sent out in one of the following situations:

- 1. The host has changed its state to DOWN or UNREACHABLE state; notification is sent out after first_notification_delay number of minutes specified in the corresponding host object.
- 2.The host remains in DOWN or UNREACHABLE state; notification is sent out every notification_interval number of minutes specified in the corresponding
- host object
- 3. Host recovers to an UP state; notification is sent out immediately and only once
- 4.Host starts or stops flapping; notification is sent out immediately
- 5. Host remains flapping; notification is sent out every notification_interval number of minutes specified in the corresponding host object

Notifications

Notifications may be sent out in one of the following situations:

6. Service has changed its state to WARNING, CRITICAL or UNKNOWN state; notification is sent out after first_notification_delay number of minutes specified in the corresponding service object

7. Service remains in WARNING, CRITICAL or UNKNOWN state; notification is sent out every notification_interval number of minutes specified in the corresponding service object

8. Service recovers to an OK state; notification is sent out immediately and only once

9. Service starts or stops flapping; notification is sent out immediately

10. Service remains flapping; notification is sent out every notification_ interval number of minutes specified in the corresponding service object

If one of these conditions occurs, Nagios starts evaluating whether information about it should be sent out and to whom.

Notifications

Who Gets Notified?

- Each host and service definition has a <contact_groups> option that specifies what contact groups receive notifications for that particular host or service.
- Contact groups can contain one or more individual contacts
- **Notification Methods**
  pager,
  Cellphone,
  email,
  instant message,
  audio alert,
  etc.

Host Checks

- When Are Host Checks Performed?

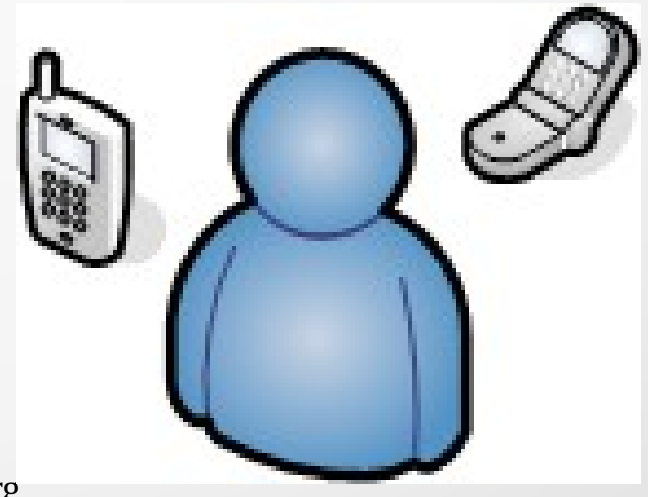
Hosts are checked by the Nagios daemon:

At regular intervals, as defined by the check_interval and retry_interval options in your host definitions.

On-demand when a service associated with the host changes state.

On-demand as needed as part of the host reachability logic.

On-demand as needed for predictive host dependency checks.



Service Checks

- When Are Service Checks Performed?

Services are checked by the Nagios daemon:

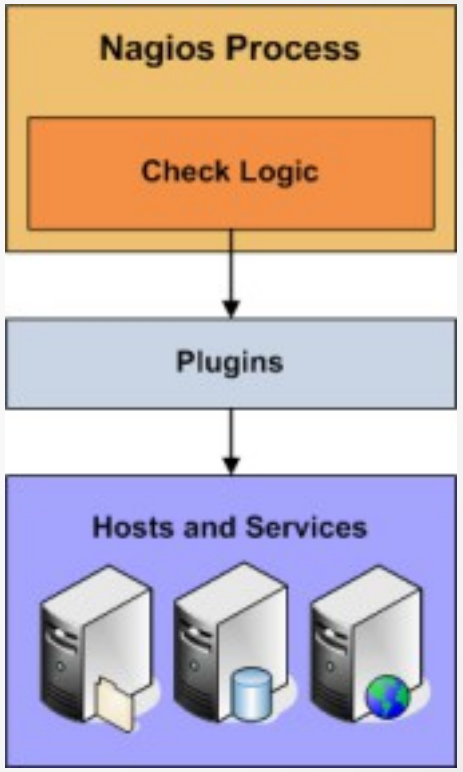
At regular intervals, as defined by the check_interval and retry_interval options in your service definitions.

On-demand as needed for predictive service dependency checks.

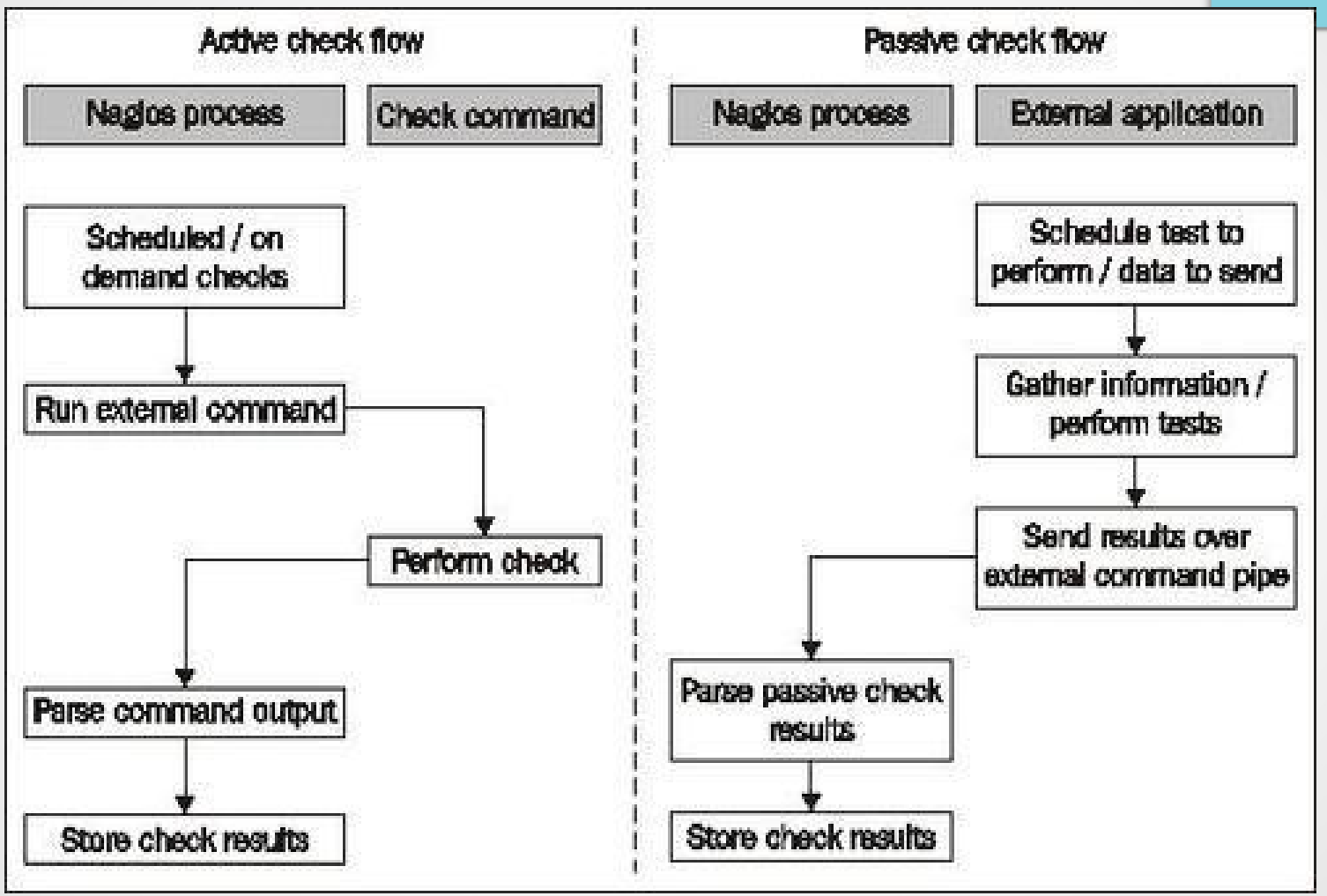
Active Checks

- Nagios is capable of monitoring hosts and services in t**wo ways: actively and passively**
- Active checks are the most common method for monitoring hosts and services. The main features of actives checks as as follows:
- Active checks are initiated by the Nagios process
- Active checks are run on a regularly scheduled basis

- **How Are Active Checks Performed?**
- Active checks are initiated by the check logic in the Nagios daemon.
- When Nagios needs to check the status of a host or service it will execute a plugin and pass it information about what needs to be checked.
- The plugin will then check the operational state of the host or service and report the results back to the Nagios daemon.
- Nagios will process the results of the host or service check and take appropriate action as necessary (e.g. send notifications, run event handlers, etc).



Passive Checks



State Types

Host States

- Hosts that are checked can be in one of three different states:
  - UP
  - DOWN
  - UNREACHABLE

Host State Determination

Host checks are performed by plugins, which can return a state of

- OK,
- WARNING,
- UNKNOWN, or
- CRITICAL.

State Types

Host States

Plugin Result	Preliminary Host State
OK	UP
WARNING	UP or DOWN*
UNKNOWN	DOWN
CRITICAL	DOWN

Preliminary Host State	Parent Host State	Final Host State
DOWN	At least one parent is UP	DOWN
DOWN	All parents are either DOWN or UNREACHABLE	UNREACHABLE

Host State Changes

Soft States

Soft states occur in the following situations...

1) When a service or host check results in a non-OK or non-UP state and the service check has not yet been (re)checked the number of times specified by the max_check_attempts directive in the service or host definition. This is called a soft error.

2) When a service or host recovers from a soft error. This is considered a soft recovery.

The following things occur when hosts or services experience SOFT state changes:

- The SOFT state is logged.
- Event handlers are executed to handle the SOFT state

- The only important thing that really happens during a soft state is the execution of event handlers.
- Using event handlers can be particularly useful if you want to try and proactively fix a problem before it turns into a HARD state.

State Types

Host State Changes

Hard States

Hard states occur for services in the following situations

1) When a service check results in a non-OK state and it has been (re)checked the number of times specified by the <max_check_attempts> option in the service definition. This is a hard error state

2) When a service recovers from a hard error state. This is considered to be a hard

Recovery

3) When a service check results in a non-OK state and its corresponding host is either

DOWN or UNREACHABLE.

.

what happens when a host or service is in a hard state:

- hanges from a hard OK state to a hard non-OK state
- changes from a hard non-OK state to a hard OK-state
- changes from a hard non-OK state of some kind to a hard non-OK state of another kind