

Centreon Engine Documentation

Release 1.5.0

Centreon

December 17, 2015

Centreon Engine 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.

Centreon Engine was originally designed to run under Linux, although it should work under most other unices as well.

Some of the many features of Centreon Engine include:

- monitoring of network services (SMTP, IMAP, HTTP, NNTP, PING ...).
- monitoring of host resources (processor load, disk usage ...).
- simple plugin design that allows users to easily develop their own service checks.
- parallelized service checks.
- ability to define network host hierarchy using “parent” hosts, allowing detection of and distinction between hosts that are down and those that are unreachable.
- contact notifications when service or host problems occur and get resolved (via email, page, or user-defined method).
- ability to define event handlers to be run during service or host events for proactive problem resolution.
- automatic log file rotation.
- support for implementing redundant monitoring hosts.

System Requirements

The only requirement of running Centreon Engine is a machine running Linux (or Unix variant) that has network access and a C++ compiler installed (if installing from source code).

Contents:

1.1 Release notes

1.1.1 Centreon Engine 1.5

What's new

Timezone support

Hosts, services and contacts can now have a timezone. All time periods applied to these objects will be computed regarding their timeperiod.

Webservice module removal

The webservice module has now been removed from Centreon Engine.

Direct Centreon ID support

Most important objects can now have their Centreon ID directly set through a configuration directive. This change will reduce support code needed in Centreon Broker and provide more performance in this piece of software. However as a consequence of this change, Centreon Engine 1.5 requires at least Centreon Broker 2.11.

Failure prediction removal

The failure prediction configuration directives are now totally deprecated. This system inherited from Nagios never worked as intended. Only configuration fields were set in structures and this code was removed in this release.

1.1.2 Centreon Engine 1.4

What's new

Configuration reload

Centreon Engine do not need to be entirely restarted to apply new configuration files. With the reload mechanism, no interruption is necessary to create or remove hosts and services.

1.1.3 Centreon Engine 1.3

What's new

Qt library removal

We decided to stop using the Qt library. For the XML part, we now use the *xerces-c* library and for the other part, we use the *Centreon-Clib* library. This replacement allows us to have a better integration with all UNIX systems and better performance.

Performance improvement

- Some algorithms were replaced (*map* instead of *unordered map* when possible)
- The execution system now uses less resource (possibility to disable *use_setpgid*)

Variables deprecation

Some variables have been deprecated:

- *temp_file*, *temp_path*, *check_result_path* and *max_check_result_file_age* are used in *nagios* to get check result information from the file system, but in Centreon Engine the execution system writes directly in memory
- *enable_embedded_perl*, *use_embedded_perl_implicitly*, *p1_file* and *auth_file* are used in *nagios* by the embedded perl interpreter. In Centreon Engine, this mechanism is provided into the Perl connector. Engine.
- *lock_file*, *nagios_user* and *nagios_group* are removed because these mechanisms are not used by the Centreon Engine startup script
- *bare_update_check* and *check_for_updates* are removed. If you need to update Centreon Engine, use your package manager
- *free_child_process_memory* and *child_processes_fork_twice* are removed. Centreon Engine forks just once and child memory is always released by the system
- *daemon_dumps_core* is removed, you can create core dumps all the time if you are allowed to
- *log_archive_path* and *log_rotation_method* are removed. Centreon Engine relies on standard *logrotate* daemons

New *logrotate* script

The *logrotate* mechanism has changed. Now we use the classical UNIX mechanism to manage log rotation (ie. based on *logrotate* daemons).

Improved Centreon compatibility

Default permissions on the status file have been changed to allow Centreon reading it.

1.2 Installation

Centreon recommends using its official packages from the Centreon Enterprise Server (CES) repository. Most of Centreon's endorsed software are available as RPM packages.

Alternatively, you can build and install your own version of this software by following the *Using sources*.

1.2.1 Using packages

Centreon provides RPM for its products through Centreon Enterprise Server (CES). Open source products are freely available from our repository.

These packages are available for CentOS 5 and CentOS 6.

Prerequisites

In order to use RPM from the CES repository, you have to install the appropriate repository.

CentOS 5

Run the following command as privileged user

```
$ wget http://yum.centreon.com/standard/2.2/noarch/RPMS/ces-release-2.2-4.noarch.rpm
$ yum install --nogpgcheck ces-release-2.2-4.noarch.rpm
$ rm -f ces-release-2.2-4.noarch.rpm
$ yum clean all
```

CentOS 6

Run the following commands as privileged user

```
$ wget http://yum.centreon.com/standard/3.0/stable/noarch/RPMS/ces-release-3.0-1.noarch.rpm
$ yum install --nogpgcheck ces-release-3.0-1.noarch.rpm
$ rm -f ces-release-3.0-1.noarch.rpm
$ yum clean all
```

Install

Run the following commands as privileged user

```
$ yum install centreon-engine
```

All dependencies are automatically installed from Centreon repositories.

1.2.2 Using sources

To build Centreon Engine, you will need the following external dependencies:

- a C++ compilation environment.
- CMake (**>= 2.8**), a cross-platform build system.
- Centreon Clib (**>= 1.4**), The centreon core library.

This program is compatible only with Unix-like platforms (Linux, FreeBSD, Solaris, ...).

Prerequisites

If you decide to build Centreon Engine from sources, we heavily recommend that you create dedicated system user and group for security purposes.

On all systems the commands to create a user and a group both named **centreon-engine** are as follow (need to run these as root)

```
$ groupadd centreon-engine
$ useradd -g centreon-engine -m -r -d /var/lib/centreon-engine centreon-engine
```

Please note that these user and group will be used in the next steps. If you decide to change user and/or group name here, please do so in further steps too.

CentOS

In CentOS you need to add manually cmake. After that you can install binary packages. Either use the Package Manager or the yum tool to install them. You should check packages version when necessary.

Package required to build:

Software	Package Name	Description
C++ compilation environment	gcc gcc-c++ make	Mandatory tools to compile.
CMake (>= 2.8)	cmake	Read the build script and prepare sources for compilation.
Centreon Clib (>= 1.4)	centreon-clib-devel	Core library used by Centreon

1. Install basic compilation tools

```
$ yum install gcc gcc-c++ make
```

2. Install Centreon repository

You need to install Centreon Enterprise Server (CES) repos file as explained *Prerequisites* to use some specific package version.

3. Install cmake

```
$ yum install cmake
```

4. Install Centreon Clib

See the Centreon Clib [documentation](#).

Debian/Ubuntu

In recent Debian/Ubuntu versions, necessary software is available as binary packages from distribution repositories. Either use the Package Manager or the apt-get tool to install them. You should check packages version when necessary.

Package required to build:

Software	Package Name	Description
C++ compilation environment	build-essential	Mandatory tools to compile.
CMake (>= 2.8)	cmake	Read the build script and prepare sources for compilation.
Centreon Clib (>= 1.4)	centreon-clib-dev	Core library used by Centreon Connector.

1. Install compilation tools

```
$ apt-get install build-essential cmake
```

2. Install Centreon Clib

See the Centreon Clib [documentation](#).

OpenSUSE

In recent OpenSUSE versions, necessary software is available as binary packages from OpenSUSE repositories. Either use the Package Manager or the zypper tool to install them. You should check packages version when necessary.

Package required to build:

Software	Package Name	Description
C++ compilation environment	gcc gcc-c++ make	Mandatory tools to compile.
CMake (>= 2.8)	cmake	Read the build script and prepare sources for compilation.
Centreon Clib (>= 1.4)	centreon-clib-devel	Core library used by Centreon Connector.

1. Install compilation tools

```
$ zypper install gcc gcc-c++ make cmake
```

2. Install Centreon Clib

See the Centreon Clib [documentation](#).

Build

Get sources

Centreon Engine can be checked out from GitHub at <https://github.com/centreon/centreon-engine>. On a Linux box with git installed this is just a matter of

```
$ git clone https://github.com/centreon/centreon-engine
```

Or You can get the latest Centreon Engine's sources from its [download website](#) Once downloaded, extract it

```
$ tar xzf centreon-engine.tar.gz
```

Configuration

At the root of the project directory you'll find a build directory which holds build scripts. Generate the Makefile by running the following command

```
$ cd /path_to_centreon_engine/build
```

Your Centreon Engine can be tweaked to your particular needs using CMake's variable system. Variables can be set like this

```
$ cmake -D<variable1>=<value1> [-D<variable2>=<value2>] .
```

Here's the list of variables available and their description:

Variable	Description	Default value
WITH_BENCH	Build benchmarking tools.	OFF
WITH_CENTREON_CLIB_INCLUDE_DIR	Set the directory path of centreon-clib include.	auto detection
WITH_CENTREON_CLIB_LIBRARIES	Set the centreon-clib library to use.	auto detection
WITH_CENTREON_CLIB_LIBRARY_DIR	Set the centreon-clib library directory (don't use it if you use WITH_CENTREON_CLIB_LIBRARIES).	auto detection
WITH_GROUP	Set the group for Centreon Engine installation.	root
WITH_LOCK_FILE	Used by the startup script.	/var/lock/subsys/centengine.lock
WITH_LOG_ARCHIVE	Enable archive log files that have been rotated.	\${WITH_VAR_DIR}/archives
WITH_LOGROTATE_DIR	Dir to install logrotate files.	/etc/logrotate.d/
WITH_LOGROTATE_SCRIPT	Enable or disable install logrotate files.	OFF
WITH_PID_FILE	This file contains the process id (PID) number of the running Centreon Engine process.	/var/run/centengine.pid
WITH_PKGCONFIG_DIR	Dir to install pkg-config files.	\${WITH_PREFIX_LIB}/pkgconfig
WITH_PKGCONFIG_SCRIPT	Enable or disable install pkg-config files.	ON
WITH_PREFIX	Base directory for Centreon Engine installation. If other prefixes are expressed as relative paths, they are relative to this path.	/usr/local
WITH_PREFIX_BIN	Define specific directory for Centreon Engine binary.	\${WITH_PREFIX}/bin
WITH_PREFIX_CONF	Define specific directory for Centreon Engine configuration.	\${WITH_PREFIX}/etc
WITH_PREFIX_INC	Define specific directory for Centreon Engine headers.	\${WITH_PREFIX}/include/centreon
WITH_PREFIX_LIB	Define specific directory for Centreon Engine modules.	\${WITH_PREFIX}/lib/centreon-eng
WITH_RW_DIR	Use for files to need read/write access.	\${WITH_VAR_DIR}/rw
WITH_SAMPLE_CONFIG	Install sample configuration files.	ON
WITH_SHARED_LIB	Build shared library for the core library.	OFF
WITH_STARTUP_DIR	Define the startup directory.	Generally in /etc/init.d or /etc/init
WITH_STARTUP_SCRIPT	Generate and install startup script. Choices are 'auto', 'sysv' and 'upstart'.	auto
WITH_TESTING	Build unit test.	OFF
WITH_USER	Set the user for Centreon Engine installation.	root
WITH_VAR_DIR	Define specific directory for temporary Centreon Engine files.	\${WITH_PREFIX}/var

Example

```
$ cmake \
  -DWITH_PREFIX=/usr \
  -DWITH_PREFIX_BIN=/usr/sbin \
  -DWITH_PREFIX_CONF=/etc/centreon-engine \
  -DWITH_PREFIX_LIB=/usr/lib/centreon-engine \
  -DWITH_USER=centreon-engine \
  -DWITH_GROUP=centreon-engine \
  -DWITH_LOGROTATE_SCRIPT=1 \
  -DWITH_VAR_DIR=/var/log/centreon-engine \
```

```
-DWITH_RW_DIR=/var/lib/centreon-engine/rw \
-DWITH_STARTUP_DIR=/etc/init.d \
-DWITH_PKGCONFIG_SCRIPT=1 \
-DWITH_PKGCONFIG_DIR=/usr/lib/pkgconfig \
-DWITH_TESTING=0
```

At this step, the software will check for existence and usability of the rerequisites. If one cannot be found, an appropriate error message will be printed. Otherwise an installation summary will be printed.

Note: If you need to change the options you used to compile your software, you might want to remove the *CMake-Cache.txt* file that is in the *build* directory. This will remove cache entries that might have been computed during the last configuration step.

Compilation

Once properly configured, the compilation process is really simple

```
$ make
```

And wait until compilation completes.

Install

Once compiled, the following command must be run as privileged user to finish installation

```
$ make install
```

And wait for its completion.

Check-Up

After a successful installation, you should check for the existence of some of the following files.

File	Description
<code>\${WITH_PREFIX_BIN}/centengine</code>	Centreon Engine daemon.
<code>\${WITH_PREFIX_BIN}/centenginestats</code>	Centreon Engine statistic.
<code>\${WITH_PREFIX_CONF}/</code>	Centreon Engine sample configuration.
<code>\${WITH_PREFIX_LIB}/externalcmd.so</code>	External commands module.
<code>\${WITH_STARTUP_DIR}/centengine.conf</code>	Startup script for ubuntu.
<code>\${WITH_STARTUP_DIR}/centengine</code>	Startup script for other os.
<code>\${WITH_PREFIX_INC}/include/centreon-engine/</code>	All devel Centreon Engine's include.
<code>\${WITH_PKGCONFIG_DIR}/centengine.pc</code>	Centreon Engine pkg-config file.

1.3 User

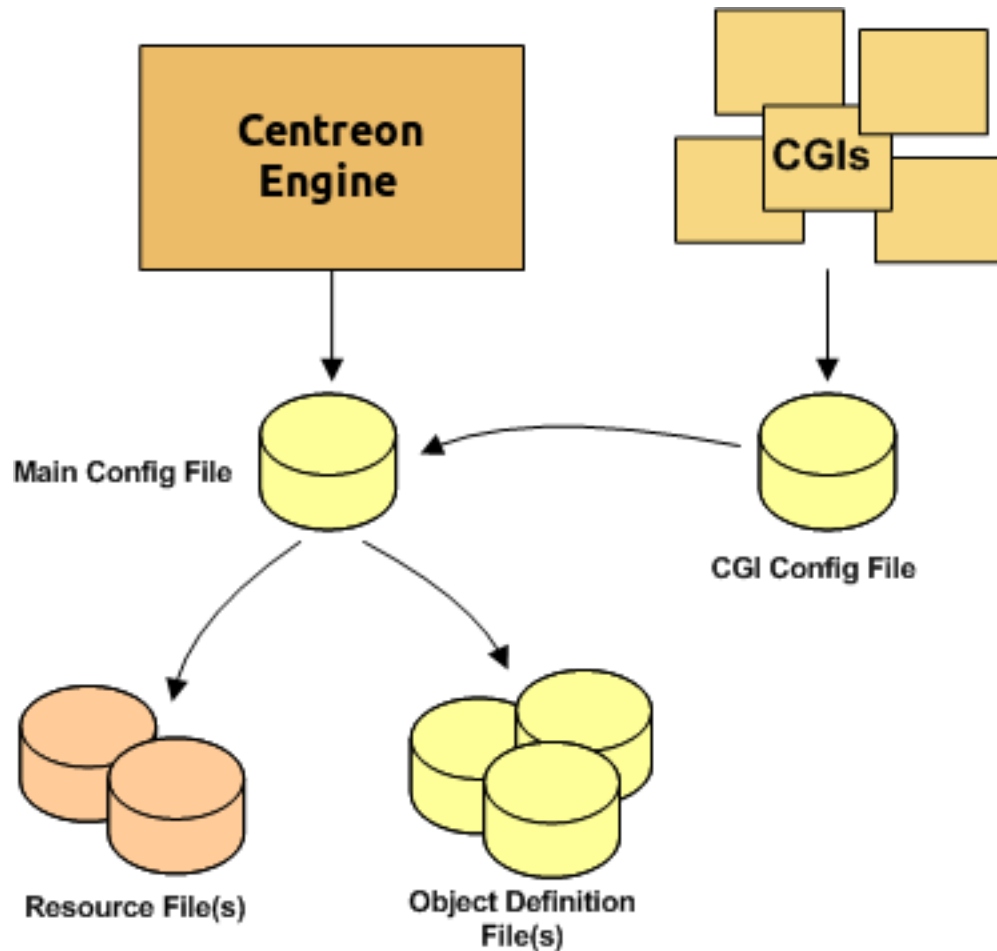
1.3.1 Configuration

Basics

Configuration Overview

Introduction There are several different configuration files that you're going to need to create or edit before you start monitoring anything. Be patient! Configuring Centreon Engine can take quite a while, especially if you're first-time user. Once you figure out how things work, it'll all be well worth your time. :-)

Note: Sample configuration files are installed in the `/etc/centreon-engine/` directory when you follow the *quickstart installation guide*.



Main Configuration File

The main configuration file contains a number of directives that affect how the Centreon Engine daemon operates. This is where you're going to want to get started in your configuration adventures.

Documentation for the main configuration file can be found *here*.

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).

You can specify one or more optional resource files by using the *resource_file* directive in your main configuration file.

Object Definition Files

Object definition files are used to define hosts, services, hostgroups, contacts, contactgroups, commands, etc. This is where you define all the things you want monitor and how you want to monitor them.

You can specify one or more object definition files by using the *cfg_file* and/or *cfg_dir* directives in your main configuration file.

An introduction to object definitions, and how they relate to each other, can be found *here*.

Main Configuration File Options

Notes 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
- Variables names must begin at the start of the line - no white space is allowed before the name
- Variable names are case-sensitive

Sample Configuration File

Note: A sample main configuration file */etc/centreon-engine/centengine.cfg* is installed for you when you follow the *quickstart installation guide*.

Config File Location The main configuration file is usually named *centengine.cfg* and located in the */etc/centreon-engine/* directory.

Configuration File Variables Below you will find descriptions of each main Centreon Engine configuration file option...

Log File This variable specifies where Centreon Engine should create its main log file. This should be the first variable that you define in your configuration file, as Centreon Engine 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.

Format	<code>log_file=<file_name></code>
Example	<code>log_file=/var/log/centreon-engine/centengine.log</code>

Object Configuration File This directive is used to specify an *object configuration file* containing object definitions that Centreon Engine should use for monitoring. Object configuration files contain definitions for hosts, host groups, contacts, contact groups, services, commands, etc. You can separate your configuration information into several files and specify multiple *cfg_file=* statements to have each of them processed.

For- mat	<code>cfg_file=<file_name></code>
Exam- ple	<code>cfg_file=/etc/centreon-engine/hosts.cfg</code> <code>cfg_file=/etc/centreon-engine/services.cfg</code> <code>cfg_file=/etc/centreon-engine/commands.cfg</code>

Object Configuration Directory This directive is used to specify a directory which contains *object configuration files* that Centreon Engine should use for monitoring. All files in the directory with a *.cfg* extension are processed as object config files. Additionally, Centreon Engine will recursively process all config files in subdirectories of the directory you specify here. You can separate your configuration files into different directories and specify multiple *cfg_dir=* statements to have all config files in each directory processed.

Format	<code>cfg_dir=<directory_name></code>
Exam- ple	<code>cfg_dir=/etc/centreon-engine/commands</code> <code>cfg_dir=/etc/centreon-engine/services</code> <code>cfg_dir=/etc/centreon-engine/hosts</code>

Object Cache File This directive is used to specify a file in which a cached copy of *object definitions* should be stored. The cache file is (re)created every time Centreon Engine is (re)started. It is intended to speed up config file caching and allow you to edit the source *object config files* while Centreon Engine is running without affecting the output displayed.

Format	object_cache_file=<file_name>
Example	object_cache_file=/var/log/centreon-engine/objects.cache

Precached Object File This directive is used to specify a file in which a pre-processed, pre-cached copy of *object definitions* should be stored. This file can be used to drastically improve startup times in large/complex Centreon Engine installations. Read more information on how to speed up start times *here*.

Format	precached_object_file=<file_name>
Example	precached_object_file=/var/log/centreon-engine/objects.precache

Resource File This is used to specify an optional resource file that can contain \$USERn\$ *macro* definitions. \$USERn\$ macros are useful for storing usernames, passwords, and items commonly used in command definitions (like directory paths). You can include multiple resource files by adding multiple resource_file statements to the main config file - Centreon Engine will process them all. See the sample resource.cfg file in the sample-config/ subdirectory of the Centreon Engine distribution for an example of how to define \$USERn\$ macros.

Format	resource_file=<file_name>
Example	resource_file=/etc/centreon-engine/resource.cfg

Temp File This is a deprecated and ignored variable.

Format	temp_file=<file_name>
---------------	-----------------------

Status File This is the file that Centreon Engine uses to store the current status, comment, and downtime information. This file is deleted every time Centreon Engine stops and recreated when it starts.

Format	status_file=<file_name>
Example	status_file=/var/log/centreon-engine/status.dat

Status File Update Interval This setting determines how often (in seconds) that Centreon Engine will update status data in the *status file*. The minimum update interval is 1 second.

Format	status_update_interval=<seconds>
Example	status_update_interval=15

Notifications Option This option determines whether or not Centreon Engine will send out *notifications* when it initially (re)starts. If this option is disabled, Centreon Engine will not send out notifications for any host or service.

Format	enable_notifications=<0/1>
Example	enable_notifications=1

Note: If you have *state retention* enabled, Centreon Engine will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the *state retention file*), unless you disable the *use_retained_program_state* option. If you want to change this option when state retention is active (and the *use_retained_program_state* is enabled), you'll have to use the appropriate *external command* or change it via the web interface. Values are as follows:

- 0 = Disable notifications
- 1 = Enable notifications (default)

Service Check Execution Option This option determines whether or not Centreon Engine will execute service checks when it initially (re)starts. If this option is disabled, Centreon Engine will not actively execute any service checks and will remain in a sort of “sleep” mode (it can still accept *passive checks* unless you’ve *disabled them*). This option is most often used when configuring backup monitoring servers, as described in the documentation on *redundancy*, or when setting up a *distributed* monitoring environment.

Format	execute_service_checks=<0/1>
Example	execute_service_checks=1

Note: If you have *state retention* enabled, Centreon Engine will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the *state retention file*), unless you disable the *use_retained_program_state* option. If you want to change this option when state retention is active (and the *use_retained_program_state* is enabled), you’ll have to use the appropriate *external command* or change it via the web interface. Values are as follows:

- 0 = Don’t execute service checks
- 1 = Execute service checks (default)

Passive Service Check Acceptance Option This option determines whether or not Centreon Engine will accept *passive service checks* when it initially (re)starts. If this option is disabled, Centreon Engine will not accept any passive service checks.

Format	accept_passive_service_checks=<0/1>
Example	accept_passive_service_checks=1

Note: If you have *state retention* enabled, Centreon Engine will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the *state retention file*), unless you disable the *use_retained_program_state* option. If you want to change this option when state retention is active (and the *use_retained_program_state* is enabled), you’ll have to use the appropriate *external command* or change it via the web interface. Values are as follows:

- 0 = Don’t accept passive service checks
- 1 = Accept passive service checks (default)

Host Check Execution Option This option determines whether or not Centreon Engine will execute on-demand and regularly scheduled host checks when it initially (re)starts. If this option is disabled, Centreon Engine will not actively execute any host checks, although it can still accept *passive host checks* unless you’ve *disabled them*). This option is most often used when configuring backup monitoring servers, as described in the documentation on *redundancy*, or when setting up a *distributed* monitoring environment.

Format	execute_host_checks=<0/1>
Example	execute_host_checks=1

Note: If you have *state retention* enabled, Centreon Engine will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the *state retention file*), unless you disable the *use_retained_program_state* option. If you want to change this option when state retention is active (and the *use_retained_program_state* is enabled), you’ll have to use the appropriate *external command* or change it via the web interface. Values are as follows:

- 0 = Don’t execute host checks
- 1 = Execute host checks (default)

Passive Host Check Acceptance Option This option determines whether or not Centreon Engine will accept *passive host checks* when it initially (re)starts. If this option is disabled, Centreon Engine will not accept any passive host checks.

Format	accept_passive_host_checks=<0/1>
Example	accept_passive_host_checks=1

Note: If you have *state retention* enabled, Centreon Engine will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the *state retention file*), unless you disable the *use_retained_program_state* option. If you want to change this option when state retention is active (and the *use_retained_program_state* is enabled), you'll have to use the appropriate *external command* or change it via the web interface. Values are as follows:

- 0 = Don't accept passive host checks
- 1 = Accept passive host checks (default)

Event Handler Option This option determines whether or not Centreon Engine will run *event handlers* when it initially (re)starts. If this option is disabled, Centreon Engine will not run any host or service event handlers.

Format	enable_event_handlers=<0/1>
Example	enable_event_handlers=1

Note: If you have *state retention* enabled, Centreon Engine will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the *state retention file*), unless you disable the *use_retained_program_state* option. If you want to change this option when state retention is active (and the *use_retained_program_state* is enabled), you'll have to use the appropriate *external command* or change it via the web interface. Values are as follows:

- 0 = Disable event handlers
- 1 = Enable event handlers (default)

Log Rotation Method This is a deprecated and ignored variable. Use logrotate daemon.

Format	log_rotation_method=<n/h/d/w/m>
---------------	---------------------------------

Log Archive Path This is a deprecated and ignored variable.

Format	log_archive_path=<path>
---------------	-------------------------

External Command Check Option This option determines whether or not Centreon Engine will check the *command file* for commands that should be executed. More information on external commands can be found [here](#).

- 0 = Don't check external commands
- 1 = Check external commands (default)

Format	check_external_commands=<0/1>
Example	check_external_commands=1

External Command Check Interval If you specify a number with an "s" appended to it (i.e. 30s), this is the number of seconds to wait between external command checks. If you leave off the "s", this is the number of "time units" to wait between external command checks. Unless you've changed the *interval_length* value (as defined below) from the default value of 60, this number will mean minutes.

Format	command_check_interval=<xxx>[s]
Example	command_check_interval=1

Note: By setting this value to -1, Centreon Engine will check for external commands as often as possible. Each time Centreon Engine checks for external commands it will read and process all commands present in the *command file* before continuing on with its other duties. More information on external commands can be found [here](#).

External Command File This is the file that Centreon Engine will check for external commands to process. The external command file is implemented as a named pipe (FIFO), which is created when Centreon Engine starts and removed when it shuts down. If the file exists when Centreon Engine starts, the Centreon Engine process will terminate with an error message. More information on external commands can be found [here](#).

Format	command_file=<file_name>
Example	command_file=/var/log/centreon-engine/rw/centengine.cmd

External Command Buffer Slots

Format	external_command_buffer_slots=<#>
Example	external_command_buffer_slots=512

Note: This is an advanced feature. This option determines how many buffer slots Centreon Engine will reserve for caching external commands that have been read from the external command file by a worker thread, but have not yet been processed by the main thread of the Centreon Engine daemon. Each slot can hold one external command, so this option essentially determines how many commands can be buffered. For installations where you process a large number of passive checks (e.g. *distributed setups*), you may need to increase this number.

State Retention Option This option determines whether or not Centreon Engine will retain state information for hosts and services between program restarts. If you enable this option, you should supply a value for the *state_retention_file* variable. When enabled, Centreon Engine will save all state information for hosts and service before it shuts down (or restarts) and will read in previously saved state information when it starts up again.

- 0 = Don't retain state information
- 1 = Retain state information (default)

Format	retain_state_information=<0/1>
Example	retain_state_information=1

State Retention File This is the file that Centreon Engine will use for storing status, downtime, and comment information before it shuts down. When Centreon Engine is restarted it will use the information stored in this file for setting the initial states of services and hosts before it starts monitoring anything. In order to make Centreon Engine retain state information between program restarts, you must enable the *retain_state_information* option.

Format	state_retention_file=<file_name>
Example	state_retention_file=/var/log/centreon-engine/retention.dat

Automatic State Retention Update Interval This setting determines how often (in minutes) that Centreon Engine will automatically save retention data during normal operation. If you set this value to 0, Centreon Engine will not save retention data at regular intervals, but it will still save retention data before shutting down or restarting. If you have disabled state retention (with the *retain_state_information* option), this option has no effect.

Format	retention_update_interval=<minutes>
Example	retention_update_interval=60

Use Retained Program State Option This setting determines whether or not Centreon Engine will set various program-wide state variables based on the values saved in the retention file. Some of these program-wide state variables that are normally saved across program restarts if state retention is enabled include the *enable_notifications*, *enable_flap_detection*, *enable_event_handlers*, *execute_service_checks*, and *accept_passive_service_checks* options. If you do not have *state retention* enabled, this option has no effect.

- 0 = Don't use retained program state
- 1 = Use retained program state (default)

Format	use_retained_program_state=<0/1>
Example	use_retained_program_state=1

Use Retained Scheduling Info Option This setting determines whether or not Centreon Engine will retain scheduling info (next check times) for hosts and services when it restarts. If you are adding a large number (or percentage) of hosts and services, I would recommend disabling this option when you first restart Centreon Engine, as it can adversely skew the spread of initial checks. Otherwise you will probably want to leave it enabled.

- 0 = Don't use retained scheduling info
- 1 = Use retained scheduling info (default)

Format	use_retained_scheduling_info=<0/1>
Example	use_retained_scheduling_info=1

Retained Host and Service Attribute Masks They are a deprecated and ignored variables.

Format	retained_host_attribute_mask=<number> retained_service_attribute_mask=<number>
---------------	--

Retained Process Attribute Masks They are a deprecated and ignored variables.

Format	retained_process_host_attribute_mask=<number> retained_process_service_attribute_mask=<number>
---------------	--

Retained Contact Attribute Masks These options determine which contact attributes are NOT retained across program restarts. There are two masks because there are often separate host and service contact attributes that can be changed. The values for these options are a bitwise AND of values specified by the "MODATTR_" definitions in the include/common.h source code file. By default, all process attributes are retained.

Format	retained_contact_host_attribute_mask=<number> retained_contact_service_attribute_mask=<number>
Example	retained_contact_host_attribute_mask=0 retained_contact_service_attribute_mask=0

Note: This is an advanced feature. You'll need to read the Centreon Engine source code to use this option effectively.

Syslog Logging Option This variable determines whether messages are logged to the syslog facility on your local host. Values are as follows:

- 0 = Don't use syslog facility
- 1 = Use syslog facility

Format	use_syslog=<0/1>
Example	use_syslog=1

Notification Logging Option This variable determines whether or not notification messages are logged. If you have a lot of contacts or regular service failures your log file will grow relatively quickly. Use this option to keep contact notifications from being logged.

- 0 = Don't log notifications
- 1 = Log notifications

Format	log_notifications=<0/1>
Example	log_notifications=1

Service Check Retry Logging Option This variable determines whether or not service check retries are logged. Service check retries occur when a service check results in a non-OK state, but you have configured Centreon Engine to retry the service more than once before responding to the error. Services in this situation are considered to be in “soft” states. Logging service check retries is mostly useful when attempting to debug Centreon Engine or test out service *event handlers*.

- 0 = Don’t log service check retries
- 1 = Log service check retries

Format	log_service_retries=<0/1>
Example	log_service_retries=1

Host Check Retry Logging Option This variable determines whether or not host check retries are logged. Logging host check retries is mostly useful when attempting to debug Centreon Engine or test out host *event handlers*.

- 0 = Don’t log host check retries
- 1 = Log host check retries

Format	log_host_retries=<0/1>
Example	log_host_retries=1

Event Handler Logging Option This variable determines whether or not service and host *event handlers* are logged. Event handlers are optional commands that can be run whenever a service or hosts changes state. Logging event handlers is most useful when debugging Centreon Engine or first trying out your event handler scripts.

- 0 = Don’t log event handlers
- 1 = Log event handlers

Format	log_event_handlers=<0/1>
Example	log_event_handlers=1

Initial States Logging Option This variable determines whether or not Centreon Engine will force all initial host and service states to be logged, even if they result in an OK state. Initial service and host states are normally only logged when there is a problem on the first check. Enabling this option is useful if you are using an application that scans the log file to determine long-term state statistics for services and hosts.

- 0 = Don’t log initial states (default)
- 1 = Log initial states

Format	log_initial_states=<0/1>
Example	log_initial_states=1

External Command Logging Option This variable determines whether or not Centreon Engine will log *external commands* that it receives from the *external command file*.

Format	log_external_commands=<0/1>
Example	log_external_commands=1

Note: This option does not control whether or not *passive service checks* (which are a type of external command) get logged. To enable or disable logging of passive checks, use the *log_passive_checks* option.

- 0 = Don’t log external commands
- 1 = Log external commands (default)

Passive Check Logging Option This variable determines whether or not Centreon Engine will log *passive host and service checks* that it receives from the *external command file*. If you are setting up a *distributed monitoring environment* or plan on handling a large number of passive checks on a regular basis, you may wish to disable this option so your log file doesn't get too large.

- 0 = Don't log passive checks
- 1 = Log passive checks (default)

Format	log_passive_checks=<0/1>
Example	log_passive_checks=1

Global Host Event Handler Option This option allows you to specify a host event handler command that is to be run for every host state change. The global event handler is executed immediately prior to the event handler that you have optionally specified in each host definition. The command argument is the short name of a command that you define in your *object configuration file*. The maximum amount of time that this command can run is controlled by the *event_handler_timeout* option. More information on event handlers can be found [here](#).

Format	global_host_event_handler=<command>
Example	global_host_event_handler=log-host-event-to-db

Global Service Event Handler Option This option allows you to specify a service event handler command that is to be run for every service state change. The global event handler is executed immediately prior to the event handler that you have optionally specified in each service definition. The command argument is the short name of a command that you define in your *object configuration file*. The maximum amount of time that this command can run is controlled by the *event_handler_timeout* option. More information on event handlers can be found [here](#).

Format	global_service_event_handler=<command>
Example	global_service_event_handler=log-service-event-to-db

Inter-Check Sleep Time This is the number of seconds that Centreon Engine will sleep before checking to see if the next service or host check in the scheduling queue should be executed.

Format	sleep_time=<seconds>
Example	sleep_time=1

Note: That Centreon Engine will only sleep after it “catches up” with queued service checks that have fallen behind.

Service Inter-Check Delay Method This option allows you to control how service checks are initially “spread out” in the event queue. Using a “smart” delay calculation (the default) will cause Centreon Engine to calculate an average check interval and spread initial checks of all services out over that interval, thereby helping to eliminate CPU load spikes. Using no delay is generally not recommended, as it will cause all service checks to be scheduled for execution at the same time. This means that you will generally have large CPU spikes when the services are all executed in parallel. More information on how to estimate how the inter-check delay affects service check scheduling can be found [here](#). Values are as follows:

- n = Don't use any delay - schedule all service checks to run immediately (i.e. at the same time!)
- d = Use a “dumb” delay of 1 second between service checks
- s = Use a “smart” delay calculation to spread service checks out evenly (default)
- x.xx = Use a user-supplied inter-check delay of x.xx seconds

Format	service_inter_check_delay_method=<n/d/s/x.xx>
Example	service_inter_check_delay_method=s

Maximum Service Check Spread This option determines the maximum number of minutes from when Centreon Engine starts that all services (that are scheduled to be regularly checked) are checked. This option will automatically adjust the *service* inter-check delay method” (if necessary) to ensure that the initial checks of all services occur within the timeframe you specify. In general, this option will not have an affect on service check scheduling if scheduling information is being retained using the *use_retained_scheduling_info* option. Default value is 30 (minutes).

Format	max_service_check_spread=<minutes>
Example	max_service_check_spread=30

Service Interleave Factor This variable determines how service checks are interleaved. Interleaving allows for a more even distribution of service checks, reduced load on remote hosts, and faster overall detection of host problems. Setting this value to 1 is equivalent to not interleaving the service checks (this is how versions of Centreon Engine previous to 0.0.5 worked). Set this value to s (smart) for automatic calculation of the interleave factor unless you have a specific reason to change it. You should see that the service check results are spread out as they begin to appear. More information on how interleaving works can be found *here*.

- x = A number greater than or equal to 1 that specifies the interleave factor to use. An interleave factor of 1 is equivalent to not interleaving the service checks.
- s = Use a “smart” interleave factor calculation (default)

Format	service_interleave_factor=<slx>
Example	service_interleave_factor=s

Maximum Concurrent Service Checks This option allows you to specify the maximum number of service checks that can be run in parallel at any given time. Specifying a value of 1 for this variable essentially prevents any service checks from being run in parallel. Specifying a value of 0 (the default) does not place any restrictions on the number of concurrent checks. You’ll have to modify this value based on the system resources you have available on the machine that runs Centreon Engine, as it directly affects the maximum load that will be imposed on the system (processor utilization, memory, etc.). More information on how to estimate how many concurrent checks you should allow can be found *here*.

Format	max_concurrent_checks=<max_checks>
Example	max_concurrent_checks=20

Check Result Reaper Frequency This option allows you to control the frequency in seconds of check result “reaper” events. “Reaper” events process the results from host and service checks that have finished executing. These events constitute the core of the monitoring logic in Centreon Engine.

Format	check_result_reaper_frequency=<frequency_in_seconds>
Example	check_result_reaper_frequency=5

Maximum Check Result Reaper Time This option allows you to control the maximum amount of time in seconds that host and service check result “reaper” events are allowed to run. “Reaper” events process the results from host and service checks that have finished executing. If there are a lot of results to process, reaper events may take a long time to finish, which might delay timely execution of new host and service checks. This variable allows you to limit the amount of time that an individual reaper event will run before it hands control back over to Centreon Engine for other portions of the monitoring logic.

Format	max_check_result_reaper_time=<seconds>
Example	max_check_result_reaper_time=30

Use Check Result Path This option enable or disable compatibility mode to use check result path.

Format	use_check_result_path=<0/1>
Example	use_check_result_path=0

Check Result Path This options determines which directory Nagios will use to temporarily store host and service check results before they are processed. This directory should not be used to store any other files, as Nagios will periodically clean this directory of old file (see the `max_check_result_file_age` option for more information).

Format	check_result_path=<path>
Example	check_result_path=/tmp

Note: This options is deprecated.

Max Check Result File Age This options determines the maximum age in seconds that Nagios will consider check result files found in the `check_result_path` directory to be valid. Check result files that are older that this threshold will be deleted by Nagios and the check results they contain will not be processed. By using a value of zero (0) with this option, Nagios will process all check result files - even if they're older than your hardware :-).

Format	max_check_result_file_age=<seconds>
Example	max_check_result_file_age=3600

Note: This options is deprecated.

Host Inter-Check Delay Method This option allows you to control how host checks that are scheduled to be checked on a regular basis are initially “spread out” in the event queue. Using a “smart” delay calculation (the default) will cause Centreon Engine to calculate an average check interval and spread initial checks of all hosts out over that interval, thereby helping to eliminate CPU load spikes. Using no delay is generally not recommended. Using no delay will cause all host checks to be scheduled for execution at the same time. More information on how to estimate how the inter-check delay affects host check scheduling can be found [here](#). Values are as follows:

- n = Don't use any delay - schedule all host checks to run immediately (i.e. at the same time!)
- d = Use a “dumb” delay of 1 second between host checks
- s = Use a “smart” delay calculation to spread host checks out evenly (default)
- x.xx = Use a user-supplied inter-check delay of x.xx seconds

Format	host_inter_check_delay_method=<n/d/s/x.xx>
Example	host_inter_check_delay_method=s

Maximum Host Check Spread This option determines the maximum number of minutes from when Centreon Engine starts that all hosts (that are scheduled to be regularly checked) are checked. This option will automatically adjust the *host inter-check delay method* (if necessary) to ensure that the initial checks of all hosts occur within the timeframe you specify. In general, this option will not have an affect on host check scheduling if scheduling information is being retained using the *use_retained_scheduling_info* option. Default value is 30 (minutes).

Format	max_host_check_spread=<minutes>
Example	max_host_check_spread=30

Timing Interval Length This is the number of seconds per “unit interval” used for timing in the scheduling queue, re-notifications, etc. “Units intervals” are used in the object configuration file to determine how often to run a service check, how often to re-notify a contact, etc.

Format	interval_length=<seconds>
Example	interval_length=60

Note: The default value for this is set to 60, which means that a “unit value” of 1 in the object configuration file will mean 60 seconds (1 minute). I have not really tested other values for this variable, so proceed at your own risk if you decide to do so!

Auto-Rescheduling Option This option determines whether or not Centreon Engine will attempt to automatically reschedule active host and service checks to “smooth” them out over time. This can help to balance the load on the monitoring server, as it will attempt to keep the time between consecutive checks consistent, at the expense of executing checks on a more rigid schedule.

Format	auto_reschedule_checks=<0/1>
Example	auto_reschedule_checks=1

Note: This is an experimental feature and may be removed in future versions. Enabling this option can degrade performance - rather than increase it - if used improperly!

Auto-Rescheduling Interval This option determines how often (in seconds) Centreon Engine will attempt to automatically reschedule checks. This option only has an effect if the *auto_reschedule_checks* option is enabled. Default is 30 seconds.

Format	auto_rescheduling_interval=<seconds>
Example	auto_rescheduling_interval=30

Note: This is an experimental feature and may be removed in future versions. Enabling the auto-rescheduling option can degrade performance - rather than increase it - if used improperly!

Auto-Rescheduling Window This option determines the “window” of time (in seconds) that Centreon Engine will look at when automatically rescheduling checks. Only host and service checks that occur in the next X seconds (determined by this variable) will be rescheduled. This option only has an effect if the *auto_reschedule_checks* option is enabled. Default is 180 seconds (3 minutes).

Format	auto_rescheduling_window=<seconds>
Example	auto_rescheduling_window=180

Note: This is an experimental feature and may be removed in future versions. Enabling the auto-rescheduling option can degrade performance - rather than increase it - if used improperly!

Aggressive Host Checking Option Centreon Engine tries to be smart about how and when it checks the status of hosts. In general, disabling this option will allow Centreon Engine to make some smarter decisions and check hosts a bit faster. Enabling this option will increase the amount of time required to check hosts, but may improve reliability a bit. Unless you have problems with Centreon Engine not recognizing that a host recovered, I would suggest not enabling this option.

- 0 = Don't use aggressive host checking (default)
- 1 = Use aggressive host checking

Format	use_aggressive_host_checking=<0/1>
Example	use_aggressive_host_checking=0

Translate Passive Host Checks Option This option determines whether or not Centreon Engine will translate DOWN/UNREACHABLE passive host check results to their “correct” state from the viewpoint of the local Centreon Engine instance. This can be very useful in distributed and failover monitoring installations. More information on passive check state translation can be found *here*.

- 0 = Disable check translation (default)
- 1 = Enable check translation

Format	translate_passive_host_checks=<0/1>
Example	translate_passive_host_checks=1

Passive Host Checks Are SOFT Option This option determines whether or not Centreon Engine will treat *passive host checks* as HARD states or SOFT states. By default, a passive host check result will put a host into a *HARD state type*. You can change this behavior by enabling this option.

- 0 = Passive host checks are HARD (default)
- 1 = Passive host checks are SOFT

Format	passive_host_checks_are_soft=<0/1>
Example	passive_host_checks_are_soft=1

Predictive Host Dependency Checks Option This option determines whether or not Centreon Engine will execute predictive checks of hosts that are being depended upon (as defined in *host dependencies*”) for a particular host when it changes state. Predictive checks help ensure that the dependency logic is as accurate as possible. More information on how predictive checks work can be found *here*.

- 0 = Disable predictive checks
- 1 = Enable predictive checks (default)

Format	enable_predictive_host_dependency_checks=<0/1>
Example	enable_predictive_host_dependency_checks=1

Predictive Service Dependency Checks Option This option determines whether or not Centreon Engine will execute predictive checks of services that are being depended upon (as defined in *service dependencies*) for a particular service when it changes state. Predictive checks help ensure that the dependency logic is as accurate as possible. More information on how predictive checks work can be found *here*.

- 0 = Disable predictive checks
- 1 = Enable predictive checks (default)

Format	enable_predictive_service_dependency_checks=<0/1>
Example	enable_predictive_service_dependency_checks=1

Cached Host Check Horizon This option determines the maximum amount of time (in seconds) that the state of a previous host check is considered current. Cached host states (from host checks that were performed more recently than the time specified by this value) can improve host check performance immensely. Too high of a value for this option may result in (temporarily) inaccurate host states, while a low value may result in a performance hit for host checks. Use a value of 0 if you want to disable host check caching. More information on cached checks can be found *here*.

Format	cached_host_check_horizon=<seconds>
Example	cached_host_check_horizon=15

Cached Service Check Horizon This option determines the maximum amount of time (in seconds) that the state of a previous service check is considered current. Cached service states (from service checks that were performed more recently than the time specified by this value) can improve service check performance when a lot of *service dependencies* are used. Too high of a value for this option may result in inaccuracies in the service dependency logic. Use a value of 0 if you want to disable service check caching. More information on cached checks can be found [here](#).

Format	cached_service_check_horizon=<seconds>
Example	cached_service_check_horizon=15

Large Installation Tweaks Option This option determines whether or not the Centreon Engine daemon will take several shortcuts to improve performance. These shortcuts result in the loss of a few features, but larger installations will likely see a lot of benefit from doing so.

- 0 = Don't use tweaks (default)
- 1 = Use tweaks

Format	use_large_installation_tweaks=<0/1>
Example	use_large_installation_tweaks=0

Use Setpgid This option allow to change plugin process group into they own process group id. This option protect Centreon Engine process from child miss used or bug.

For example, if we use nagios check_ping, check_dns, check_dig or check_rbl, don't disable this option, because, these checks can call kill -KILL 0 on timeout (this is a bug from these plugins) and kill the engine if the PGID is the same as the engine.

For maximum performance, this option must be disable.

- 0 = Don't use setpgid
- 1 = Use setpgid (default)

Format	use_setpgid=<0/1>
Example	use_setpgid=1

Child Process Memory Option This is a deprecated and ignored variable.

Format	free_child_process_memory=<0/1>
---------------	---------------------------------

Child Processes Fork Twice This is a deprecated and ignored variable.

Format	child_processes_fork_twice=<0/1>
---------------	----------------------------------

Environment Macros Option This option determines whether or not the Centreon Engine daemon will make all standard *macros* available as environment variables to your check, notification, event handler, etc. commands. In large Centreon Engine installations this can be problematic because it takes additional memory and (more importantly) CPU to compute the values of all macros and make them available to the environment.

- 0 = Don't make macros available as environment variables
- 1 = Make macros available as environment variables (default)

Format	enable_environment_macros=<0/1>
Example	enable_environment_macros=0

Flap Detection Option This option determines whether or not Centreon Engine will try and detect hosts and services that are “flapping”. Flapping occurs when a host or service changes between states too frequently, resulting in a barrage of notifications being sent out. When Centreon Engine detects that a host or service is flapping, it will temporarily suppress notifications for that host/service until it stops flapping. Flap detection is very experimental at this point, so use this feature with caution! More information on how flap detection and handling works can be found *here*.

Format	enable_flap_detection=<0/1>
Example	enable_flap_detection=0

Note: If you have *state retention* enabled, Centreon Engine will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the *state retention file*), unless you disable the *use_retained_program_state* option. If you want to change this option when state retention is active (and the *use_retained_program_state* is enabled), you’ll have to use the appropriate *external command* or change it via the web interface.

- 0 = Don’t enable flap detection (default)
- 1 = Enable flap detection

Low Service Flap Threshold This option is used to set the low threshold for detection of service flapping. For more information on how flap detection and handling works (and how this option affects things) read *this*.

Format	low_service_flap_threshold=<percent>
Example	low_service_flap_threshold=25.0

High Service Flap Threshold This option is used to set the high threshold for detection of service flapping. For more information on how flap detection and handling works (and how this option affects things) read *this*.

Format	high_service_flap_threshold=<percent>
Example	high_service_flap_threshold=50.0

Low Host Flap Threshold This option is used to set the low threshold for detection of host flapping. For more information on how flap detection and handling works (and how this option affects things) read *this*.

Format	low_host_flap_threshold=<percent>
Example	low_host_flap_threshold=25.0

High Host Flap Threshold This option is used to set the high threshold for detection of host flapping. For more information on how flap detection and handling works (and how this option affects things) read *this*.

Format	high_host_flap_threshold=<percent>
Example	high_host_flap_threshold=50.0

Soft State Dependencies Option This option determines whether or not Centreon Engine will use soft state information when checking *host and service dependencies*. Normally Centreon Engine will only use the latest hard host or service state when checking dependencies. If you want it to use the latest state (regardless of whether its a soft or hard *state type*), enable this option.

- 0 = Don’t use soft state dependencies (default)
- 1 = Use soft state dependencies

Format	soft_state_dependencies=<0/1>
Example	soft_state_dependencies=0

Service Check Timeout This is the maximum number of seconds that Centreon Engine will allow service checks to run. If checks exceed this limit, they are killed and a CRITICAL state is returned. A timeout error will also be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off plugins which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each service check normally finishes executing within this time limit. If a service check runs longer than this limit, Centreon Engine will kill it off thinking it is a runaway processes.

Format	service_check_timeout=<seconds>
Example	service_check_timeout=60

Host Check Timeout This is the maximum number of seconds that Centreon Engine will allow host checks to run. If checks exceed this limit, they are killed and a CRITICAL state is returned and the host will be assumed to be DOWN. A timeout error will also be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off plugins which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each host check normally finishes executing within this time limit. If a host check runs longer than this limit, Centreon Engine will kill it off thinking it is a runaway processes.

Format	host_check_timeout=<seconds>
Example	host_check_timeout=60

Event Handler Timeout This is the maximum number of seconds that Centreon Engine will allow *event handlers* to be run. If an event handler exceeds this time limit it will be killed and a warning will be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off commands which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each event handler command normally finishes executing within this time limit. If an event handler runs longer than this limit, Centreon Engine will kill it off thinking it is a runaway processes.

Format	event_handler_timeout=<seconds>
Example	event_handler_timeout=60

Notification Timeout This is the maximum number of seconds that Centreon Engine will allow notification commands to be run. If a notification command exceeds this time limit it will be killed and a warning will be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off commands which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each notification command finishes executing within this time limit. If a notification command runs longer than this limit, Centreon Engine will kill it off thinking it is a runaway processes.

Format	notification_timeout=<seconds>
Example	notification_timeout=60

Obsessive Compulsive Service Processor Timeout This is the maximum number of seconds that Centreon Engine will allow an *obsessive compulsive service processor* command to be run. If a command exceeds this time limit it will be killed and a warning will be logged.

Format	ocsp_timeout=<seconds>
Example	ocsp_timeout=5

Obsessive Compulsive Host Processor Timeout This is the maximum number of seconds that Centreon Engine will allow an *obsessive compulsive host processor* command to be run. If a command exceeds this time limit it will be killed and a warning will be logged.

Format	ochp_timeout=<seconds>
Example	ochp_timeout=5

Performance Data Processor Command Timeout This is the maximum number of seconds that Centreon Engine will allow a *host performance data processor command* or *service performance data processor command* to be run. If a command exceeds this time limit it will be killed and a warning will be logged.

Format	perfdata_timeout=<seconds>
Example	perfdata_timeout=5

Obsess Over Services Option This value determines whether or not Centreon Engine will “obsess” over service checks results and run the *obsessive compulsive service processor command* you define. I know - funny name, but it was all I could think of. This option is useful for performing *distributed monitoring*. If you’re not doing distributed monitoring, don’t enable this option.

- 0 = Don’t obsess over services (default)
- 1 = Obsess over services

Format	obsess_over_services=<0/1>
Example	obsess_over_services=1

Obsessive Compulsive Service Processor Command This option allows you to specify a command to be run after every service check, which can be useful in *distributed monitoring*. This command is executed after any *event handler* or *notification* commands. The command argument is the short name of a *command definition* that you define in your object configuration file. The maximum amount of time that this command can run is controlled by the *ocsp_timeout* option. More information on distributed monitoring can be found [here](#). This command is only executed if the *obsess_over_services* option is enabled globally and if the *obsess_over_service* directive in the *service definition* is enabled.

Format	ocsp_command=<command>
Example	ocsp_command=obsessive_service_handler

Obsess Over Hosts Option This value determines whether or not Centreon Engine will “obsess” over host checks results and run the *obsessive compulsive host processor command* you define. I know - funny name, but it was all I could think of. This option is useful for performing *distributed monitoring*. If you’re not doing distributed monitoring, don’t enable this option.

- 0 = Don’t obsess over hosts (default)
- 1 = Obsess over hosts

Format	obsess_over_hosts=<0/1>
Example	obsess_over_hosts=1

Obsessive Compulsive Host Processor Command This option allows you to specify a command to be run after every host check, which can be useful in *distributed monitoring*. This command is executed after any *event handler* or *notification* commands. The command argument is the short name of a *command definition* that you define in your object configuration file. The maximum amount of time that this command can run is controlled by the *ochp_timeout* option. More information on distributed monitoring can be found [here](#). This command is only executed if the *obsess_over_hosts* option is enabled globally and if the *obsess_over_host* directive in the *host definition* is enabled.

Format	ochp_command=<command>
Example	ochp_command=obsessive_host_handler

Performance Data Processing Option This value determines whether or not Centreon Engine will process host and service check *performance data*.

- 0 = Don't process performance data (default)
- 1 = Process performance data

Format	process_performance_data=<0/1>
Example	process_performance_data=1

Host Performance Data Processing Command This option allows you to specify a command to be run after every host check to process host *performance data* that may be returned from the check. The command argument is the short name of a *command definition* that you define in your object configuration file. This command is only executed if the *process_performance_data* option is enabled globally and if the *process_perf_data* directive in the *host definition* is enabled.

Format	host_perfdata_command=<command>
Example	host_perfdata_command=process-host-perfdata

Service Performance Data Processing Command This option allows you to specify a command to be run after every service check to process service *performance data* that may be returned from the check. The command argument is the short name of a *command definition* that you define in your object configuration file. This command is only executed if the *process_performance_data* option is enabled globally and if the *process_perf_data* directive in the *service definition* is enabled.

Format	service_perfdata_command=<command>
Example	service_perfdata_command=process-service-perfdata

Host Performance Data File This option allows you to specify a file to which host *performance data* will be written after every host check. Data will be written to the performance file as specified by the *host_perfdata_file_template* option. Performance data is only written to this file if the *process_performance_data* option is enabled globally and if the *process_perf_data* directive in the *host definition* is enabled.

Format	host_perfdata_file=<file_name>
Example	host_perfdata_file=/var/log/centreon-engine/host-perfdata.dat

Service Performance Data File This option allows you to specify a file to which service *performance data* will be written after every service check. Data will be written to the performance file as specified by the *service_perfdata_file_template* option. Performance data is only written to this file if the *process_performance_data* option is enabled globally and if the *process_perf_data* directive in the *service definition* is enabled.

Format	service_perfdata_file=<file_name>
Example	service_perfdata_file=/var/log/centreon-engine/service-perfdata.dat

Host Performance Data File Template This option determines what (and how) data is written to the *host performance data file*. The template may contain *macros*, special characters (t for tab, r for carriage return, n for newline) and plain text. A newline is automatically added after each write to the performance data file.

Format	host_perfdata_file_template=<template>
Example	host_perfdata_file_template=[HOSTPERFDATA]\t\$HOSTNAME\$\t\$HOSTOUTPUT\$\t\$HOSTPERFDATA\$

Service Performance Data File Template This option determines what (and how) data is written to the *service performance data file*. The template may contain *macros*, special characters (t for tab, r for carriage return, n for newline) and plain text. A newline is automatically added after each write to the performance data file.

Format	service_perfdata_file_template=<template>
Example	service_perfdata_file_template=[SERVICEPERFDATA]\t\$HOSTNAME\t\$\$SERVICEDESC\t\$\$SERVICEOUTPUT\t\$\$SERV

Host Performance Data File Mode This option determines how the *host performance data file* is opened. Unless the file is a named pipe you'll probably want to use the default mode of append.

- a = Open file in append mode (default)
- w = Open file in write mode
- p = Open in non-blocking read/write mode (useful when writing to pipes)

Format	host_perfdata_file_mode=<mode>
Example	host_perfdata_file_mode=a

Service Performance Data File Mode This option determines how the *service performance data file* is opened. Unless the file is a named pipe you'll probably want to use the default mode of append.

- a = Open file in append mode (default)
- w = Open file in write mode
- p = Open in non-blocking read/write mode (useful when writing to pipes)

Format	service_perfdata_file_mode=<mode>
Example	service_perfdata_file_mode=a

Host Performance Data File Processing Interval This option allows you to specify the interval (in seconds) at which the *host performance data file* is processed using the *host performance data file processing command*. A value of 0 indicates that the performance data file should not be processed at regular intervals.

Format	host_perfdata_file_processing_interval=<seconds>
Example	host_perfdata_file_processing_interval=0

Service Performance Data File Processing Interval This option allows you to specify the interval (in seconds) at which the *service performance data file* is processed using the *service performance data file processing command*. A value of 0 indicates that the performance data file should not be processed at regular intervals.

Format	service_perfdata_file_processing_interval=<seconds>
Example	service_perfdata_file_processing_interval=0

Host Performance Data File Processing Command This option allows you to specify the command that should be executed to process the *host performance data file*. The command argument is the short name of a *command definition* that you define in your object configuration file. The interval at which this command is executed is determined by the *host_perfdata_file_processing_interval* directive.

Format	host_perfdata_file_processing_command=<command>
Example	host_perfdata_file_processing_command=process-host-perfdata-file

Service Performance Data File Processing Command This option allows you to specify the command that should be executed to process the *service* performance data file”. The command argument is the short name of a *command definition* that you define in your object configuration file. The interval at which this command is executed is determined by the *service_perfdata_file_processing_interval* directive.

Format	service_perfdata_file_processing_command=<command>
Example	service_perfdata_file_processing_command=process-service-perfdata-file

Orphaned Service Check Option This option allows you to enable or disable checks for orphaned service checks. Orphaned service checks are checks which have been executed and have been removed from the event queue, but have not had any results reported in a long time. Since no results have come back in for the service, it is not rescheduled in the event queue. This can cause service checks to stop being executed. Normally it is very rare for this to happen - it might happen if an external user or process killed off the process that was being used to execute a service check. If this option is enabled and Centreon Engine finds that results for a particular service check have not come back, it will log an error message and reschedule the service check. If you start seeing service checks that never seem to get rescheduled, enable this option and see if you notice any log messages about orphaned services.

- 0 = Don’t check for orphaned service checks
- 1 = Check for orphaned service checks (default)

Format	check_for_orphaned_services=<0/1>
Example	check_for_orphaned_services=1

Orphaned Host Check Option This option allows you to enable or disable checks for orphaned host checks. Orphaned host checks are checks which have been executed and have been removed from the event queue, but have not had any results reported in a long time. Since no results have come back in for the host, it is not rescheduled in the event queue. This can cause host checks to stop being executed. Normally it is very rare for this to happen - it might happen if an external user or process killed off the process that was being used to execute a host check. If this option is enabled and Centreon Engine finds that results for a particular host check have not come back, it will log an error message and reschedule the host check. If you start seeing host checks that never seem to get rescheduled, enable this option and see if you notice any log messages about orphaned hosts.

- 0 = Don’t check for orphaned host checks
- 1 = Check for orphaned host checks (default)

Format	check_for_orphaned_hosts=<0/1>
Example	check_for_orphaned_hosts=1

Service Freshness Checking Option This option determines whether or not Centreon Engine will periodically check the “freshness” of service checks. Enabling this option is useful for helping to ensure that *passive service checks* are received in a timely manner. More information on freshness checking can be found [here](#).

- 0 = Don’t check service freshness
- 1 = Check service freshness (default)

Format	check_service_freshness=<0/1>
Example	check_service_freshness=0

Service Freshness Check Interval This setting determines how often (in seconds) Centreon Engine will periodically check the “freshness” of service check results. If you have disabled service freshness checking (with the *check_service_freshness* option), this option has no effect. More information on freshness checking can be found [here](#).

Format	service_freshness_check_interval=<seconds>
Example	service_freshness_check_interval=60

Host Freshness Checking Option This option determines whether or not Centreon Engine will periodically check the “freshness” of host checks. Enabling this option is useful for helping to ensure that *passive host checks* are received in a timely manner. More information on freshness checking can be found *here*.

- 0 = Don’t check host freshness
- 1 = Check host freshness (default)

Format	check_host_freshness=<0/1>
Example	check_host_freshness=0

Host Freshness Check Interval This setting determines how often (in seconds) Centreon Engine will periodically check the “freshness” of host check results. If you have disabled host freshness checking (with the *check_host_freshness* option), this option has no effect. More information on freshness checking can be found *here*.

Format	host_freshness_check_interval=<seconds>
Example	host_freshness_check_interval=60

Additional Freshness Threshold Latency Option This option determines the number of seconds Centreon Engine will add to any host or services freshness threshold it automatically calculates (e.g. those not specified explicitly by the user). More information on freshness checking can be found *here*.

Format	additional_freshness_latency=<#>
Example	additional_freshness_latency=15

Date Format This option allows you to specify what kind of date/time format Centreon Engine should use in the web interface and date/time *macros*. Possible options (along with example output) include:

Option	Output Format	Sample Output
us	MM/DD/YYYY HH:MM:SS	06/30/2002 03:15:00
euro	DD/MM/YYYY HH:MM:SS	30/06/2002 03:15:00
iso8601	YYYY-MM-DD HH:MM:SS	2002-06-30 03:15:00
strict-iso8601	YYYY-MM-DDTHH:MM:SS	2002-06-30T03:15:00

Format	date_format=<option>
Example	date_format=us

Timezone Option This option allows you to override the default timezone that this instance of Centreon Engine runs in. Useful if you have multiple instances of Centreon Engine that need to run from the same server, but have different local times associated with them. If not specified, Centreon Engine will use the system configured timezone.

Format	use_timezone=<tz>
Example	use_timezone=US/Mountain

Illegal Object Name Characters This option allows you to specify illegal characters that cannot be used in host names, service descriptions, or names of other object types. Centreon Engine will allow you to use most characters in object definitions, but I recommend not using the characters shown in the example above. Doing may give you problems in the web interface, notification commands, etc.

Format	illegal_object_name_chars=<chars...>
Example	illegal_object_name_chars='~!\$%^&*"'<>?,()=

Illegal Macro Output Characters This option allows you to specify illegal characters that should be stripped from *macros* before being used in notifications, event handlers, and other commands. This DOES NOT affect macros used in service or host check commands. You can choose to not strip out the characters shown in the example above, but I recommend you do not do this. Some of these characters are interpreted by the shell (i.e. the backtick) and can lead to security problems. The following macros are stripped of the characters you specify:

\$HOSTOUTPUT\$, \$HOSTPERFDATA\$, \$HOSTACKAUTHOR\$, \$HOSTACKCOMMENTS\$, \$SERVICEOUTPUT\$, \$SERVICEPERFDATA\$,

Format	illegal_macro_output_chars=<chars...>
Example	illegal_macro_output_chars='~\$^&” ’<>

Regular Expression Matching Option This option determines whether or not various directives in your *object definitions* will be processed as regular expressions. More information on how this works can be found [here](#).

- 0 = Don't use regular expression matching (default)
- 1 = Use regular expression matching

Format	use_regexp_matching=<0/1>
Example	use_regexp_matching=0

True Regular Expression Matching Option If you've enabled regular expression matching of various object directives using the *use_regexp_matching* option, this option will determine when object directives are treated as regular expressions. If this option is disabled (the default), directives will only be treated as regular expressions if they contain *, ?, +, or \.. If this option is enabled, all appropriate directives will be treated as regular expression - be careful when enabling this! More information on how this works can be found [here](#).

- 0 = Don't use true regular expression matching (default)
- 1 = Use true regular expression matching

Format	use_true_regexp_matching=<0/1>
Example	use_true_regexp_matching=0

Administrator Email Address This is the email address for the administrator of the local machine (i.e. the one that Centreon Engine is running on).

This value can be used in notification commands by using the \$ADMINEMAIL\$ *macro*.

Format	admin_email=<email_address>
Example	admin_email=root@localhost.localdomain

Administrator Pager This is the pager number (or pager email gateway) for the administrator of the local machine (i.e. the one that Centreon Engine is running on). The pager number/address can be used in notification commands by using the \$ADMINPAGER\$ *macro*.

Format	admin_pager=<pager_number_or_pager_email_gateway>
Example	admin_pager=pageroot@localhost.localdomain

Event Broker Options This option controls what (if any) data gets sent to the event broker and, in turn, to any loaded event broker modules. This is an advanced option. When in doubt, either broker nothing (if not using event broker modules) or broker everything (if using event broker modules). Possible values are shown below.

- 0 = Broker nothing
- -1 = Broker everything

- **# = See `BROKER_*` definitions in source code (`include/broker.h`) for** other values that can be OR'ed together

Format	<code>event_broker_options=<#></code>
Example	<code>event_broker_options=-1</code>

Event Broker Modules This directive is used to specify an event broker module that should be loaded by Centreon Engine at startup. Use multiple directives if you want to load more than one module. Arguments that should be passed to the module at startup are separated from the module path by a space.

Format	<code>broker_module=<modulepath> [moduleargs]</code>
Example	<code>broker_module=/usr/local/centreon-engine/bin/ndomod.o cfg_file=/etc/centreon-engine/ndomod.cfg</code>

Note: Do NOT overwrite modules while they are being used by Centreon Engine or Centreon Engine will crash in a fiery display of SEGFAULT glory. This is a bug/limitation either in `dlopen()`, the kernel, and/or the filesystem. And maybe Centreon Engine...

The correct/safe way of updating a module is by using one of these methods:

- Shutdown Centreon Engine, replace the module file, restart Centreon Engine
- While Centreon Engine is running... delete the original module file, move the new module file into place, restart Centreon Engine

Debug File This option determines where Centreon Engine should write debugging information. What (if any) information is written is determined by the `debug_level` and `debug_verbosity` options. You can have Centreon Engine automatically rotate the debug file when it reaches a certain size by using the `max_debug_file_size` option.

Format	<code>debug_file=<file_name></code>
Example	<code>debug_file=/var/log/centreon-engine/centengine.debug</code>

Debug Level This option determines what type of information Centreon Engine should write to the `debug_file`. This value is a logical OR of the values below.

- -1 = Log everything
- 0 = Log nothing (default)
- 1 = Function enter/exit information
- 2 = Config information
- 4 = Process information
- 8 = Scheduled event information
- 16 = Host/service check information
- 32 = Notification information
- 64 = Event broker information

Format	<code>debug_level=<#></code>
Example	<code>debug_level=24</code>

Debug Verbosity This option determines how much debugging information Centreon Engine should write to the `debug_file`.

- 0 = Basic information

- 1 = More detailed information (default)
- 2 = Highly detailed information

Format	<code>debug_verbosity=<#></code>
Example	<code>debug_verbosity=1</code>

Maximum Debug File Size This option determines the maximum size (in bytes) of the *debug file*. If the file grows larger than this size, it will be renamed with a .old extension. If a file already exists with a .old extension it will automatically be deleted. This helps ensure your disk space usage doesn't get out of control when debugging Centreon Engine.

Format	<code>max_debug_file_size=<#></code>
Example	<code>max_debug_file_size=1000000</code>

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

More information on what objects are and how they relate to each other can be found below.

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.

Note: When you follow *quickstart installation guide*, several sample object configuration files are placed in `/etc/centreon-engine/objects/`. You can use these sample files to see how object inheritance works and learn how to define your own object definitions.

How Are Objects Defined? Objects are defined in a flexible template format, which can make it much easier to manage your Centreon Engine configuration in the long term. Basic information on how to define objects in your configuration files can be found *here*.

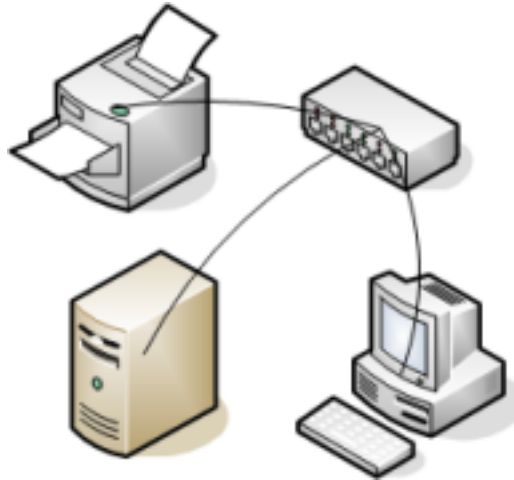
Once you get familiar with the basics of how to define objects, you should read up on *object inheritance*, as it will make your configuration more robust for the future. Seasoned users can exploit some advanced features of object definitions as described in the documentation on *object tricks*.

Objects Explained Some of the main object types are explained in greater detail below...

Host are one of the central objects in the monitoring logic. Important attributes of hosts are as follows:

- Hosts are usually physical devices on your network (servers, workstations, routers, switches, printers, etc).
- Hosts have an address of some kind (e.g. an IP or MAC address).
- Hosts have one or more services associated with them.
- Hosts can have parent/child relationships with other hosts, often representing real-world network connections, which is used in the *network reachability* logic.

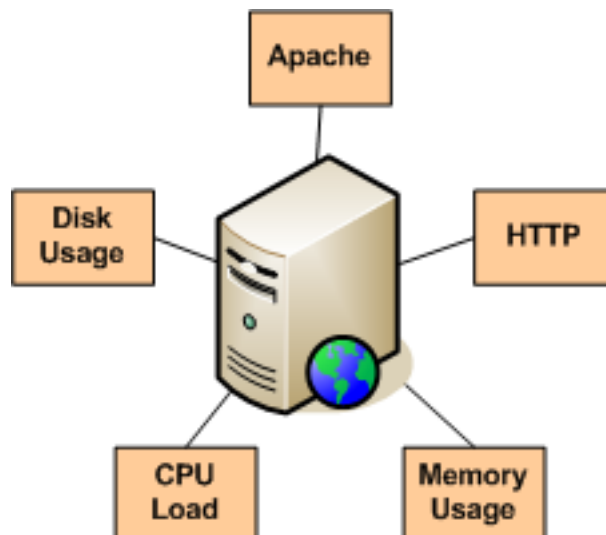
Host Groups are groups of one or more hosts. Host groups can make it easier to (1) view the status of related hosts in the Centreon Engine web interface and (2) simplify your configuration through the use of *object tricks*.



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.)

Services Groups are groups of one or more services. Service groups can make it easier to (1) view the status of related services in the Centreon Engine web interface and (2) simplify your configuration through the use of *object tricks*.



Contacts are people involved in the notification process:

- Contacts have one or more notification methods (cellphone, pager, email, instant messaging, etc.)
- Contacts receive notifications for hosts and service they are responsible for *Contacts Groups* are groups of one or more contacts. Contact groups can make it easier to define all the people who get notified when certain host or service problems occur.



Timeperiods are used to control:

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

Information on how timeperiods work can be found [here](#).



Commands are used to tell Centreon Engine what programs, scripts, etc. it should execute to perform:

- Host and service checks
- Notifications
- Event handlers
- and more...



Object Definitions

Introduction One of the features of Centreon Engine' object configuration format is that you can create object definitions that inherit properties from other object definitions. An explanation of how object inheritance works can be found [here](#). I strongly suggest that you familiarize yourself with object inheritance once you read over the documentation presented below, as it will make the job of creating and maintaining object definitions much easier than it otherwise would be. Also, read up on the *object tricks* that offer shortcuts for otherwise tedious configuration tasks.

Note: 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
-

Retention Notes It is important to point out that several directives in host, service, and contact definitions may not be picked up by Centreon Engine when you change them in your configuration files. Object directives that can exhibit this behavior are marked with an asterisk (*). The reason for this behavior is due to the fact that Centreon Engine chooses to honor values stored in the *state retention file* over values found in the config files, assuming you have *state retention* enabled on a program-wide basis and the value of the directive is changed during runtime with an *external command*. One way to get around this problem is to disable the retention of non-status information using the *retain_nonstatus_information* directive in the host, service, and contact definitions. Disabling this directive will cause Centreon Engine to take the initial values for these directives from your config files, rather than from the state retention file when it (re)starts.

Sample Configuration Files

Note: Sample object configuration files are installed in the `/etc/centreon-engine/` directory when you follow the *quickstart installation guide*.

Object Types

- *Host definitions*
- *Host group definitions*
- *Service definitions*
- *Service group definitions*
- *Contact definitions*
- *Contact group definitions*
- *Time period definitions*
- *Command definitions*
- *Connector definitions*
- *Service dependency definitions*
- *Service escalation definitions*
- *Host dependency definitions*
- *Host escalation definitions*
- *Extended host information definitions*
- *Extended service information definitions*

Host Definition

Description A host definition is used to define a physical server, workstation, device, etc. that resides on your network.

Definition Format

Note: Optional directives are comment (line start with #).

```
define host{
    host_name          host_name
    alias              alias
    # display_name     display_name
    address            address
    # parents          host_names
    # hostgroups        hostgroup_names
    # check_command     command_name
    # initial_state    [o,d,u]
    max_check_attempts #
    # check_interval   #
    # retry_interval   #
    # active_checks_enabled [0/1]
    # passive_checks_enabled [0/1]
    check_period       timeperiod_name
    # obsess_over_host [0/1]
    # check_freshness  [0/1]
    # freshness_threshold #
    # event_handler     command_name
    # event_handler_enabled [0/1]
    # low_flap_threshold #
    # high_flap_threshold #
    # flap_detection_enabled [0/1]
    # flap_detection_options [o,d,u]
    # process_perf_data [0/1]
    # retain_status_information [0/1]
    # retain_nonstatus_information [0/1]
    contacts           contacts
    contact_groups      contact_groups
    notification_interval #
    # first_notification_delay #
    notification_period timeperiod_name
    # notification_options [d,u,r,f,s]
    # notifications_enabled [0/1]
    # stalking_options    [o,d,u]
    # notes               note_string
    # notes_url           url
    # action_url          url
    # icon_image          image_file
    # icon_image_alt      alt_string
    # vrmml_image         image_file
    # statusmap_image     image_file
    # 2d_coords           x_coord,y_coord
    # 3d_coords           x_coord,y_coord,z_coord
}
```

Example Definition

```
define host{
    host_name          bogus-router
    alias              Bogus Router #1
    address            192.168.1.254
    parents            server-backbone
    check_command       check-host-alive
    check_interval      5
}
```

```

retry_interval          1
max_check_attempts      5
check_period            24x7
process_perf_data       0
retain_nonstatus_information 0
contact_groups          router-admins
notification_interval    30
notification_period      24x7
notification_options     d,u,r
}

```

host_name	This directive is used to define a short name used to identify the host. It is used in host group and service definitions to reference this particular host. Hosts can have multiple services (which are monitored) associated with them. When used properly, the \$HOSTNAME\$ <i>macro</i> will contain this short name.
alias	This directive is used to define a longer name or description used to identify the host. It is provided in order to allow you to more easily identify a particular host. When used properly, the \$HOSTALIAS\$ <i>macro</i> will contain this alias/description.
address	This directive is used to define the address of the host. Normally, this is an IP address, although it could really be anything you want (so long as it can be used to check the status of the host). You can use a FQDN to identify the host instead of an IP address, but if DNS services are not available this could cause problems. When used properly, the \$HOSTADDRESS\$ <i>macro</i> will contain this address. .. note: If you do not specify an address directive in a host definition, you should use a word of caution about doing this, however * if DNS will be unable to resolve the host name.
display_name	This directive is used to define an alternate name that should be displayed in the web interface for this host. If not specified, this defaults to the value you specify for the host_name directive.

Continued on next page

Table 1.1 – continued from previous page

parents	<p>This directive is used to define a comma-delimited list of short names of the “parent” hosts for this particular host. Parent hosts are typically routers, switches, firewalls, etc. that lie between the monitoring host and a remote hosts. A router, switch, etc. which is closest to the remote host is considered to be that host’s “parent”. Read the “Determining Status and Reachability of Network Hosts” document located <i>here</i> for more information. If this host is on the same network segment as the host doing the monitoring (without any intermediate routers, etc.) the host is considered to be on the local network and will not have a parent host. Leave this value blank if the host does not have a parent host (i.e. it is on the same segment as the Centreon Engine host). The order in which you specify parent hosts has no effect on how things are monitored.</p>
hostgroups	<p>This directive is used to identify the short name(s) of the <i>hostgroup(s)</i> that the host belongs to. Multiple <i>hostgroups</i> should be separated by commas. This directive may be used as an alternative to (or in addition to) using the <i>members</i> directive in <i>hostgroup</i> definitions.</p>
check_command	<p>This directive is used to specify the short name of the <i>command</i> that should be used to check if the host is up or down. Typically, this command would try and ping the host to see if it is “alive”. The command must return a status of OK (0) or Centreon Engine will assume the host is down. If you leave this argument blank, the host will not be actively checked. Thus, Centreon Engine will likely always assume the host is up (it may show up as being in a “PENDING” state in the web interface). This is useful if you are monitoring printers or other devices that are frequently turned off. The maximum amount of time that the notification command can run is controlled by the <i>host_check_timeout</i> option.</p>
initial_state	<p>By default Centreon Engine will assume that all hosts are in UP states when it starts. You can override the initial state for a host by using this directive. Valid options are: o = UP, d = DOWN, and u = UNREACHABLE.</p>
max_check_attempts	<p>This directive is used to define the number of times that Centreon Engine will retry the host check command if it returns any state other than an OK state. Setting this value to 1 will cause Centreon Engine to generate an alert without retrying the host check. .. note: If you do not want to check the status of the host host check, just leave the check_command option blank</p>
check_interval	<p>This directive is used to define the number of “time units” between regularly scheduled checks of the host. Unless you’ve changed the <i>interval_length</i> directive from the default value of 60, this number will mean minutes. More information on this value can be found in the <i>check scheduling</i> documentation.</p>

Continued on next page

Table 1.1 – continued from previous page

retry_interval	This directive is used to define the number of “time units” to wait before scheduling a re-check of the hosts. Hosts are rescheduled at the retry interval when they have changed to a non-UP state. Once the host has been retried max_check_attempts times without a change in its status, it will revert to being scheduled at its “normal” rate as defined by the check_interval value. Unless you’ve changed the <i>interval_length</i> directive from the default value of 60, this number will mean minutes. More information on this value can be found in the <i>check scheduling</i> documentation.
active_checks_enabled	* This directive is used to determine whether or not active checks (either regularly scheduled or on-demand) of this host are enabled. Values: 0 = disable active host checks, 1 = enable active host checks (default).
passive_checks_enabled	* This directive is used to determine whether or not passive checks are enabled for this host. Values: 0 = disable passive host checks, 1 = enable passive host checks (default).
check_period	This directive is used to specify the short name of the <i>time period</i> during which active checks of this host can be made.
obsess_over_host	* This directive determines whether or not checks for the host will be “obsessed” over using the <i>ochp_command</i> .
check_freshness	* This directive is used to determine whether or not <i>freshness checks</i> are enabled for this host. Values: 0 = disable freshness checks, 1 = enable freshness checks (default).
freshness_threshold	This directive is used to specify the freshness threshold (in seconds) for this host. If you set this directive to a value of 0, Centreon Engine will determine a freshness threshold to use automatically.
event_handler	This directive is used to specify the short name of the <i>command</i> that should be run whenever a change in the state of the host is detected (i.e. whenever it goes down or recovers). Read the documentation on <i>event handlers</i> for a more detailed explanation of how to write scripts for handling events. The maximum amount of time that the event handler command can run is controlled by the <i>event_handler_timeout</i> option.
event_handler_enabled	* This directive is used to determine whether or not the event handler for this host is enabled. Values: 0 = disable host event handler, 1 = enable host event handler.
low_flap_threshold	This directive is used to specify the low state change threshold used in flap detection for this host. More information on flap detection can be found <i>here</i> . If you set this directive to a value of 0, the program-wide value specified by the <i>low_host_flap_threshold</i> directive will be used.

Continued on next page

Table 1.1 – continued from previous page

high_flap_threshold	This directive is used to specify the high state change threshold used in flap detection for this host. More information on flap detection can be found <i>here</i> . If you set this directive to a value of 0, the program-wide value specified by the <i>high_host_flap_threshold</i> directive will be used.
flap_detection_enabled	* This directive is used to determine whether or not flap detection is enabled for this host. More information on flap detection can be found <i>here</i> . Values: 0 = disable host flap detection, 1 = enable host flap detection.
flap_detection_options	This directive is used to determine what host states the <i>flap detection logic</i> will use for this host. Valid options are a combination of one or more of the following: o = UP states, d = DOWN states, u = UNREACHABLE states.
process_perf_data	* This directive is used to determine whether or not the processing of performance data is enabled for this host. Values: 0 = disable performance data processing, 1 = enable performance data processing.
retain_status_information	This directive is used to determine whether or not status-related information about the host is retained across program restarts. This is only useful if you have enabled state retention using the <i>retain_state_information</i> directive. Value: 0 = disable status information retention, 1 = enable status information retention.
retain_nonstatus_information	This directive is used to determine whether or not non-status information about the host is retained across program restarts. This is only useful if you have enabled state retention using the <i>retain_state_information</i> directive. Value: 0 = disable non-status information retention, 1 = enable non-status information retention.
contacts	This is a list of the short names of the <i>contacts</i> that should be notified whenever there are problems (or recoveries) with this host. Multiple contacts should be separated by commas. Useful if you want notifications to go to just a few people and don't want to configure <i>contact groups</i> . You must specify at least one contact or contact group in each host definition.
contact_groups	This is a list of the short names of the <i>contact groups</i> that should be notified whenever there are problems (or recoveries) with this host. Multiple contact groups should be separated by commas. You must specify at least one contact or contact group in each host definition.
notification_interval	This directive is used to define the number of "time units" to wait before re-notifying a contact that this service is still down or unreachable. Unless you've changed the <i>interval_length</i> directive from the default value of 60, this number will mean minutes. If you set this value to 0, Centreon Engine will not re-notify contacts about problems for this host - only one problem notification will be sent out.

Continued on next page

Table 1.1 – continued from previous page

first_notification_delay	This directive is used to define the number of “time units” to wait before sending out the first problem notification when this host enters a non-UP state. Unless you’ve changed the <i>interval_length</i> directive from the default value of 60, this number will mean minutes. If you set this value to 0, Centreon Engine will start sending out notifications immediately.
notification_period	This directive is used to specify the short name of the <i>time period</i> during which notifications of events for this host can be sent out to contacts. If a host goes down, becomes unreachable, or recovers during a time which is not covered by the time period, no notifications will be sent out.
notification_options	This directive is used to determine when notifications for the host should be sent out. Valid options are a combination of one or more of the following: d = send notifications on a DOWN state, u = send notifications on an UNREACHABLE state, r = send notifications on recoveries (OK state), f = send notifications when the host starts and stops <i>flapping</i> , and s = send notifications when <i>scheduled downtime</i> starts and ends. If you specify n (none) as an option, no host notifications will be sent out. If you do not specify any notification options, Centreon Engine will assume that you want notifications to be sent out for all possible states. Example: If you specify d,r in this field, notifications will only be sent out when the host goes DOWN and when it recovers from a DOWN state.
notifications_enabled	* This directive is used to determine whether or not notifications for this host are enabled. Values: 0 = disable host notifications, 1 = enable host notifications.
stalking_options	This directive determines which host states “stalking” is enabled for. Valid options are a combination of one or more of the following: o = stalk on UP states, d = stalk on DOWN states, and u = stalk on UNREACHABLE states. More information on state stalking can be found <i>here</i> .
notes	This directive is used to define an optional string of notes pertaining to the host.
notes_url	This variable is used to define an optional URL that can be used to provide more information about the host. Any valid
URL can be used. This can be	very useful if you want to make detailed information on the host, emergency contact methods, etc. available to other support staff.
action_url	This directive is used to define an optional URL that can be used to provide more actions to be performed on the host. Any valid URL can be used.

Continued on next page

Table 1.1 – continued from previous page

icon_image	This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. The image will look best if it is 40x40 pixels in size. Images for hosts are assumed to be in the <code>logos/</code> subdirectory in your HTML images directory (i.e. <code>/usr/local/centengine/share/images/logos</code>).
icon_image_alt	This variable is used to define an optional string that is used in the ALT tag of the image specified by the <code><icon_image></code> argument.
vrml_image	This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be used as the texture map for the specified host. Unlike the image you use for the <code><icon_image></code> variable, this one should probably not have any transparency. If it does, the host object will look a bit wierd. Images for hosts are assumed to be in the <code>logos/</code> subdirectory in your HTML images directory (i.e. <code>/usr/local/centengine/share/images/logos</code>).
statusmap_image	This variable is used to define the name of an image that should be associated with this host. You can specify a JPEG, PNG, and GIF image if you want, although I would strongly suggest using a GD2 format image, as other image formats will result in a lot of wasted CPU time when the statusmap image is generated. GD2 images can be created from PNG images by using the <code>pngtogd2</code> utility supplied with Thomas Boutell's gd library . The GD2 images should be created in uncompressed format in order to minimize CPU load. Images for hosts are assumed to be in the <code>logos/</code> subdirectory in your HTML images directory (i.e. <code>/usr/local/centengine/share/images/logos</code>).
2d_coords	This variable is used to define coordinates to use when drawing the host. Coordinates should be given in positive integers, as they correspond to physical pixels in the generated image. The origin for drawing (0,0) is in the upper left hand corner of the image and extends in the positive x direction (to the right) along the top of the image and in the positive y direction (down) along the left hand side of the image. For reference, the size of the icons drawn is usually about 40x40 pixels (text takes a little extra space). The coordinates you specify here are for the upper left hand corner of the host icon that is drawn. Note: Don't worry about what the maximum x and y coordinates that you can use are.
3d_coords	This variable is used to define coordinates to use when drawing the host. Coordinates can be positive or negative real numbers. The origin for drawing is (0.0,0.0,0.0). For reference, the size of the host cubes drawn is 0.5 units on each side (text takes a little more space). The coordinates you specify here are used as the center of the host cube.

Continued on next page

Table 1.1 – continued from previous page

timezone	Timezone in which the host resides. This will affect its check and notification periods. The timezone must be formatted using the POSIX TZ variable format. That is for most users this will look like : <i>America/New_York</i> (notice the colon).
----------	--

Directive Descriptions

Host Group Definition

Description A host group definition is used to group one or more hosts together for simplifying configuration with *object tricks*.

Definition Format

Note: Optional directives are comment (line start with #).

```
define hostgroup{
    hostgroup_name    hostgroup_name
    alias             alias
    # members         hosts
    # hostgroup_members hostgroups
    # notes           note_string
    # notes_url       url
    # action_url      url
}
```

Example Definition

```
define hostgroup{
    hostgroup_name novell-servers
    alias          Novell Servers
    members        netware1,netware2,netware3,netware4
}
```

Directive Descriptions

host-group_name	This directive is used to define a short name used to identify the host group.
alias	This directive is used to define is a longer name or description used to identify the host group provided in order to allow you to more easily identify a particular host group.
members	This is a list of the short names of <i>hosts</i> that should be included in this group. Multiple host names should be separated by commas. This directive may be used as an alternative to (or in addition to) the <i>hostgroups</i> directive in <i>host definitions</i> .
host-group_members	This optional directive can be used to include hosts from other “sub” host groups in this host group. Specify a comma-delimited list of short names of other host groups whose members should be included in this group.
notes	This directive is used to define an optional string of notes pertaining to the host. If you specify a note here, you will see the it.
notes_url	This variable is used to define an optional URL that can be used to provide more information on the host group. Any valid URL can be used. This can be very useful if you want to make detailed information on the host group, emergency contact methods, etc. available to other support staff.
action_url	This directive is used to define an optional URL that can be used to provide more actions to be performed on the host group. Any valid URL can be used.

Service Definition

Description A service definition is used to identify a “service” that runs on a host. The term “service” is used very loosely. It can mean an actual service that runs on the host (POP, SMTP, HTTP, etc.) or some other type of metric associated with the host (response to a ping, number of logged in users, free disk space, etc.). The different arguments to a service definition are outlined below.

Definition Format

Note: Optional directives are comment (line start with #).

```
define service{
    host_name                host_name
    # hostgroup_name         hostgroup_name
    service_description      service_description
    # display_name           display_name
    # servicegroups          servicegroup_names
    # is_volatile            [0/1]
    check_command            command_name
    # initial_state          [o,w,u,c]
    max_check_attempts      #
    check_interval          #
    retry_interval          #
    # active_checks_enabled [0/1]
    # passive_checks_enabled [0/1]
    check_period            timeperiod_name
    # obsess_over_service   [0/1]
    # check_freshness       [0/1]
    # freshness_threshold   #
    # event_handler         command_name
    # event_handler_enabled [0/1]
    # low_flap_threshold    #
    # high_flap_threshold  #
    # flap_detection_enabled [0/1]
    # flap_detection_options [o,w,c,u]
    # process_perf_data     [0/1]
    # retain_status_information [0/1]
    # retain_nonstatus_information [0/1]
    notification_interval   #
    # first_notification_delay #
    notification_period      timeperiod_name
    # notification_options   [w,u,c,r,f,s]
    # notifications_enabled  [0/1]
    contacts                 contacts
    contact_groups           contact_groups
    # stalking_options       [o,w,u,c]
    # notes                  note_string
    # notes_url              url
    # action_url             url
    # icon_image             image_file
    # icon_image_alt         alt_string
}
```

Example Definition

```
define service{
    host_name                linux-server
```

```

service_description check-disk-sda1
check_command check-disk!/dev/sda1
max_check_attempts 5
check_interval 5
retry_interval 3
check_period 24x7
notification_interval 30
notification_period 24x7
notification_options w,c,r
contact_groups linux-admins
}

```

host_name	This directive is used to specify the short name(s) of the <i>host(s)</i> that the service “runs” on or is associated with.
hostgroup_name	This directive is used to specify the short name(s) of the <i>hostgroup(s)</i> that the service “runs” on or is associated with.
service_description;	This directive is used to define the description of the service, which may contain spaces, dashes, and colon characters.
display_name	This directive is used to define an alternate name that should be displayed in the web interface for this service.
servicegroups	This directive is used to identify the short name(s) of the <i>servicegroup(s)</i> that the service belongs to. Multiple values are allowed.
is_volatile	This directive is used to denote whether the service is “volatile”. Services are normally not volatile. More information is available in the Centreon Engine documentation.
check_command	This directive is used to specify the short name of the <i>command</i> that Centreon Engine will run in order to check the service.
initial_state	By default Centreon Engine will assume that all services are in OK states when it starts. You can override this default by using this directive.
max_check_attempts	This directive is used to define the number of times that Centreon Engine will retry the service check command if it fails.
check_interval	This directive is used to define the number of “time units” to wait before scheduling the next “regular” check of the service.
retry_interval	This directive is used to define the number of “time units” to wait before scheduling a re-check of the service after a failed check.
active_checks_enabled	* This directive is used to determine whether or not active checks of this service are enabled. Values: 0 = disabled, 1 = enabled.
passive_checks_enabled	* This directive is used to determine whether or not passive checks of this service are enabled. Values: 0 = disabled, 1 = enabled.
check_period	This directive is used to specify the short name of the <i>time period</i> during which active checks of this service are performed.
obsess_over_service	* This directive determines whether or not checks for the service will be “obsessed” over using the <i>ocsp_command</i> directive.
check_freshness	* This directive is used to determine whether or not <i>freshness checks</i> are enabled for this service. Values: 0 = disabled, 1 = enabled.
freshness_threshold	This directive is used to specify the freshness threshold (in seconds) for this service. If you set this directive to 0, the freshness threshold is the same as the global default.
event_handler	This directive is used to specify the short name of the <i>command</i> that should be run whenever a change in the state of the service occurs.
event_handler_enabled	This directive is used to determine whether or not the event handler for this service is enabled. Values: 0 = disabled, 1 = enabled.
low_flap_threshold	This directive is used to specify the low state change threshold used in flap detection for this service. More information is available in the Centreon Engine documentation.
high_flap_threshold	This directive is used to specify the high state change threshold used in flap detection for this service. More information is available in the Centreon Engine documentation.
flap_detection_enabled	* This directive is used to determine whether or not flap detection is enabled for this service. More information is available in the Centreon Engine documentation.
flap_detection_options	This directive is used to determine what service states the <i>flap detection logic</i> will use for this service. Valid options are a combination of the following: c, f, r, s, t, u, v, w, x, y, z.
process_perf_data	* This directive is used to determine whether or not the processing of performance data is enabled for this service. Values: 0 = disabled, 1 = enabled.
retain_status_information	This directive is used to determine whether or not status-related information about the service is retained in the service’s history.
retain_nonstatus_information	This directive is used to determine whether or not non-status information about the service is retained in the service’s history.
notification_interval	This directive is used to define the number of “time units” to wait before re-notifying a contact that the service is in a problem state.
first_notification_delay	This directive is used to define the number of “time units” to wait before sending out the first problem notification to a contact.
notification_period	This directive is used to specify the short name of the <i>time period</i> during which notifications of events are sent to contacts.
notification_options	This directive is used to determine when notifications for the service should be sent out. Valid options are a combination of the following: a, c, f, r, s, t, u, v, w, x, y, z.
notifications_enabled	* This directive is used to determine whether or not notifications for this service are enabled. Values: 0 = disabled, 1 = enabled.
contacts	This is a list of the short names of the <i>contacts</i> that should be notified whenever there are problems (or successes) with the service.
contact_groups	This is a list of the short names of the <i>contact groups</i> that should be notified whenever there are problems (or successes) with the service.
stalking_options	This directive determines which service states “stalking” is enabled for. Valid options are a combination of the following: c, f, r, s, t, u, v, w, x, y, z.
notes	This directive is used to define an optional string of notes pertaining to the service.
notes_url	This directive is used to define an optional URL that can be used to provide more information about the service.
action_url	This directive is used to define an optional URL that can be used to provide more actions to be performed on the service.
icon_image	This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with the service.
icon_image_alt	This variable is used to define an optional string that is used in the ALT tag of the image specified by the <i>icon_image</i> directive.
timezone	Timezone in which the service resides. This will affect its check and notification periods. Services use the local timezone of the host.

Directive Descriptions

Service Group Definition

Description A service group definition is used to group one or more services together for simplifying configuration with *object tricks*.

Definition Format

Note: Optional directives are comment (line start with #).

```
define servicegroup{
  servicegroup_name    servicegroup_name
  alias                alias
  # members            services
  # servicegroup_members servicegroups
  # notes              note_string
  # notes_url          url
  # action_url         url
}
```

Example Definition

```
define servicegroup{
  servicegroup_name dbservices
  alias              Database Services
  members            ms1,SQL Server,ms1,SQL Server Agent,ms1,SQL DTC
}
```

Directive Descriptions

service-group_name	This directive is used to define a short name used to identify the service group.
alias	This directive is used to define is a longer name or description used to identify the service g It is provided in order to allow you to more easily identify a particular service group.
members	This is a list of the descriptions of <i>services</i> (and the names of their corresponding hosts) tha should be included in this group. Host and service names should be separated by commas. " directive may be used as an alternative to the servicegroups directive in <i>service</i> definitions". format of the member directive is as follows (note that a host name must precede a service name/description):members=<host1>,<service1>,<host2>,<service2>,...,<hostn>,<servicen
service-group_members	This optional directive can be used to include services from other "sub" service groups in th service group. Specify a comma-delimited list of short names of other service groups whose members should be included in this group.
notes	This directive is used to define an optional string of notes pertaining to the service group.
notes_url	This directive is used to define an optional URL that can be used to provide more informati about the service group. Any valid URL can be used. This can be very useful if you want to detailed information on the service group, emergency contact methods, etc. available to othe support staff.
action_url	This directive is used to define an optional URL that can be used to provide more actions to performed on the service group. Any valid URL can be used.

Contact Definition

Description A contact definition is used to identify someone who should be contacted in the event of a problem on your network.

The different arguments to a contact definition are described below.

Definition Format

Note: Optional directives are comment (line start with #).

```
define contact{
    contact_name                contact_name
    # alias                     alias
    contactgroups               contactgroup_names
    host_notifications_enabled  [0/1]
    service_notifications_enabled [0/1]
    host_notification_period    timeperiod_name
    service_notification_period timeperiod_name
    host_notification_options   [d,u,r,f,s,n]
    service_notification_options [w,u,c,r,f,s,n]
    host_notification_commands  command_name
    service_notification_commands command_name
    # email                     email_address
    # pager                     pager_number or pager_email_gateway
    # addressx                  additional_contact_address
    # can_submit_commands       [0/1]
    # retain_status_information [0/1]
    # retain_nonstatus_information [0/1]
}
```

Example Definition

```
define contact{
    contact_name                jdoe
    alias                       John Doe
    host_notifications_enabled  1
    service_notifications_enabled 1
    service_notification_period 24x7
    host_notification_period    24x7
    service_notification_options w,u,c,r
    host_notification_options   d,u,r
    service_notification_commands notify-by-email
    host_notification_commands  host-notify-by-email
    email                       jdoe@localhost.localdomain
    pager                       555-5555@pagergateway.localhost.localdomain
    address1                     xxxxx.xyyy@icq.com
    address2                     555-555-5555
    can_submit_commands         1
}
```

Directive Descriptions

contact_name	This directive is used to define a short name used to identify the contact. It is referenced in <i>contact group</i> definitions. Under the right circumstances, the <code>\$CONTACTNAME\$ macro</code> will contain this value.
alias	This directive is used to define a longer name or description for the contact. Under the right circumstances, the <code>\$CONTACTALIAS\$ macro</code> will contain this value. If not specified, <code>contact_name</code> will be used as the alias.
contactgroups	This directive is used to identify the short name(s) of the <i>contactgroup(s)</i> that the contact belongs to. Multiple <i>contactgroups</i> should be separated by commas. This directive may be used as an alternative to (or in addition to) using the <i>members</i> directive in <i>contactgroup</i> definitions.
host_notifications_enabled	This directive is used to determine whether or not the contact will receive notifications about host problems and recoveries. Values: 0 = don't send notifications, 1 = send notifications.
service_notifications_enabled	This directive is used to determine whether or not the contact will receive notifications about service problems and recoveries. Values: 0 = don't send notifications, 1 = send notifications.
host_notification_period	This directive is used to specify the short name of the <i>time period</i> during which the contact can be notified about host problems or recoveries. You can think of this as an “on call” period for host notifications for the contact. Read the documentation on <i>time periods</i> for more information on how this works and potential problems that may result from improper use.
service_notification_period	This directive is used to specify the short name of the <i>time period</i> during which the contact can be notified about service problems or recoveries. You can think of this as an “on call” time for service notifications for the contact. Read the documentation on <i>time periods</i> for more information on how this works and potential problems that may result from improper use.
host_notification_commands	This directive is used to define a list of the short names of the <i>commands</i> used to notify the contact of a host problem or recovery. Multiple notification commands should be separated by commas. All notification commands are executed when the contact needs to be notified. The maximum amount of time that a notification command can run is controlled by the <i>notification_timeout</i> option.
host_notification_options	This directive is used to define the host states for which notifications can be sent out to the contact. Valid options are a combination of one or more of the following: d = notify on DOWN host states, u = notify on UNREACHABLE host states, r = notify on host recoveries (UP states), f = notify when the host starts and stops <i>flapping</i> , and s = send notifications when host or service <i>scheduled downtime</i> starts and ends. If you specify n (none) as an option, the contact will not receive any type of host notifications.
service_notification_options	This directive is used to define the service states for which notifications can be sent out to the contact. Valid options are a combination of one or more of the following: w = notify on WARNING service states, u = notify on UNKNOWN service states, c = notify on CRITICAL service states, r = notify on service recoveries (OK states), and f = notify when the service starts and stops <i>flapping</i> . If you specify n (none) as an option, the contact will not receive any type of service notifications.
service_notification_commands	This directive is used to define a list of the short names of the <i>commands</i> used to notify the contact of a service problem or recovery. Multiple notification commands should be separated by commas. All notification commands are executed when the contact needs to be notified. The maximum amount of time that a notification command can run is controlled by the <i>notification_timeout</i> option.
email	This directive is used to define an email address for the contact. Depending on how you configure your notification commands, it can be used to send out an alert email to the contact. Under the right circumstances, the <code>\$CONTACTEMAIL\$ macro</code> will contain this value.
pager	This directive is used to define a pager number for the contact. It can also be an email address to a pager gateway (i.e. pagejoe@pagenet.com). Depending on how you configure your notification commands, it can be used to send out an alert page to the contact. Under the right circumstances, the <code>\$CONTACTPAGER\$ macro</code> will contain this value.
addressx	Address directives are used to define additional “addresses” for the contact. These addresses can be anything - cell phone numbers, instant messaging addresses, etc. Depending on how you configure your notification commands, they can be used to send out an alert to the contact. Up to six addresses can be defined using these directives (<code>address1</code> through <code>address6</code>). The <code>\$CONTACTADDRESSx\$ macro</code> will contain this value.
can_submit_commands	This directive is used to determine whether or not the contact can submit external commands to Centreon Engine. Values: 0 = don't allow contact to submit commands, 1 = allow contact to submit commands.

1.3. User

Contact Group Definition

Description A contact group definition is used to group one or more *contacts* together for the purpose of sending out alert/recovery *notifications*.

Definition Format

Note: Optional directives are comment (line start with #).

```
define contactgroup{
    contactgroup_name    contactgroup_name
    alias                alias
    # members            contacts
    # contactgroup_members contactgroups
}
```

Example Definition

```
define contactgroup{
    contactgroup_name novell-admins
    alias              Novell Administrators
    members            jdoe,rtobert,tzach
}
```

Directive Descriptions

contact-group_name	This directive is a short name used to identify the contact group.
alias	This directive is used to define a longer name or description used to identify the contact group.
members	This optional directive is used to define a list of the short names of <i>contacts</i> that should be included in this group. Multiple contact names should be separated by commas. This directive may be used as an alternative to (or in addition to) using the <i>contactgroups</i> directive in <i>contactgroup</i> definitions.
contact-group_members	This optional directive can be used to include contacts from other “sub” contact groups in the contact group. Specify a comma-delimited list of short names of other contact groups whose members should be included in this group.

Time Period Definition

Description A time period is a list of times during various days that are considered to be “valid” times for notifications and service checks. It consists of time ranges for each day of the week that “rotate” once the week has come to an end. Different types of exceptions to the normal weekly time are supported, including: specific weekdays, days of generic months, days of specific months, and calendar dates.

Definition Format

Note: Optional directives are comment (line start with #).

```
define timeperiod{
    timeperiod_name    timeperiod_name
    alias              alias
    # [weekday]        timeranges
    # [exception]      timeranges
    # exclude          [timeperiod1,timeperiod2,...,timeperiodn]
}
```

Example Definitions

```
define timeperiod{
    timeperiod_name nonworkhours
    alias            Non-Work Hours
    sunday           00:00-24:00           ; Every Sunday of every week
    monday           00:00-09:00,17:00-24:00 ; Every Monday of every week
    tuesday          00:00-09:00,17:00-24:00 ; Every Tuesday of every week
    wednesday        00:00-09:00,17:00-24:00 ; Every Wednesday of every week
    thursday         00:00-09:00,17:00-24:00 ; Every Thursday of every week
    friday           00:00-09:00,17:00-24:00 ; Every Friday of every week
    saturday         00:00-24:00           ; Every Saturday of every week
}

define timeperiod{
    timeperiod_name      misc-single-days
    alias                Misc Single Days
    1999-01-28           00:00-24:00 ; January 28th, 1999
    monday 3             00:00-24:00 ; 3rd Monday of every month
    day 2                00:00-24:00 ; 2nd day of every month
    february 10          00:00-24:00 ; February 10th of every year
    february -1          00:00-24:00 ; Last day in February of every year
    friday -2            00:00-24:00 ; 2nd to last Friday of every month
    thursday -1 november 00:00-24:00 ; Last Thursday in November of every year
}

define timeperiod{
    timeperiod_name      misc-date-ranges
    alias                Misc Date Ranges
    2007-01-01 - 2008-02-01 00:00-24:00 ; January 1st, 2007 to February 1st, 2008
    monday 3 - thursday 4   00:00-24:00 ; 3rd Monday to 4th Thursday of every month
    day 1 - 15              00:00-24:00 ; 1st to 15th day of every month
    day 20 - -1             00:00-24:00 ; 20th to the last day of every month
    july 10 - 15            00:00-24:00 ; July 10th to July 15th of every year
    april 10 - may 15       00:00-24:00 ; April 10th to May 15th of every year
    tuesday 1 april - friday 2 may 00:00-24:00 ; 1st Tuesday in April to 2nd Friday in May of every year
}

define timeperiod{
    timeperiod_name      misc-skip-ranges
    alias                Misc Skip Ranges
    2007-01-01 - 2008-02-01 / 3 00:00-24:00 ; Every 3 days from January 1st, 2007 to February 1st, 2008
    2008-04-01 / 7              00:00-24:00 ; Every 7 days from April 1st, 2008 (continuing forever)
    monday 3 - thursday 4 / 2   00:00-24:00 ; Every other day from 3rd Monday to 4th Thursday of every month
    day 1 - 15 / 5              00:00-24:00 ; Every 5 days from the 1st to the 15th day of every month
    july 10 - 15 / 2            00:00-24:00 ; Every other day from July 10th to July 15th of every year
    tuesday 1 april - friday 2 may / 6 00:00-24:00 ; Every 6 days from the 1st Tuesday in April to the 2nd Friday in May of every year
}
```

Directive Descriptions	timeperiod_name	This directive is the short name used to identify the time period.
	alias	This directive is a longer name or description used to identify the time period.
	[weekday]	The weekday directives (“sunday” through “saturday”) are comma-delimited lists of time ranges which are “valid” times for a particular day of the week. Notice that there are seven different days for which you can define time ranges (Sunday through Saturday). Each time range is in the form of HH:MM-HH:MM, where hours are specified on a 24 hour clock. For example, 00:15-24:00 means 12:15am in the morning for this day until 12:00am midnight (a 23 hour, 45 minute total time range). If you wish to exclude an entire day from the timeperiod, simply do not include it in the timeperiod definition.
	[exception]	You can specify several different types of exceptions to the standard rotating weekday schedule. Exceptions can take a number of different forms including single days of a specific or generic month, single weekdays in a month, or single calendar dates. You can also specify a range of days/dates and even specify skip intervals to obtain functionality described by “every 3 days between these dates”. Rather than list all the possible formats for exception strings, I’ll let you look at the example timeperiod definitions above to see what’s possible. :-). Weekdays and different types of exceptions all have different levels of precedence, so it’s important to understand how they can affect each other. More information on this can be found in the documentation on <i>timeperiods</i> .
	exclude	This directive is used to specify the short names of other timeperiod definitions whose time ranges should be excluded from this timeperiod. Multiple timeperiod names should be separated with a comma.

Command Definition

Description A command definition is just that. It defines a command. Commands that can be defined include service checks, service notifications, service event handlers, host checks, host notifications, and host event handlers. Command definitions can contain *macros*, but you must make sure that you include only those macros that are “valid” for the circumstances when the command will be used. More information on what macros are available and when they are “valid” can be found *here*. The different arguments to a command definition are outlined below.

Definition Format

Note: Optional directives are comment (line start with #).

```
define command{
    command_name    command_name
    command_line    command_line
    # connector     connector_name
}
```

Example Definition

```
define command{
    command_name    check_pop
    command_line    /usr/lib/nagios/plugins/check_pop -H $HOSTADDRESS$
}
```


Directive Descriptions

command_name	This directive is the short name used to identify the command. It is referenced in <i>contact</i> , <i>host</i> , and <i>service</i> definitions (in notification, check, and event handler routines), among other places.
command_line	This directive is used to define what is actually executed by Centreon Engine when the command is used for service or host checks, notifications, or <i>event handlers</i> . Before the command line is executed, all valid macros are replaced with their respective values. See the documentation on macros for determining when you can use different macros. Note that the command line is never rounded in quotes. Also, if you want to pass a dollar sign (\$) on the command line, you have to escape it with another dollar sign. .. note: You may not include a semicolon (;) in the command line or comment. You can work around this by using the <code>:ref: 'resource file <main_cfg_opts> command_line directive in place of <command_line> :ref: '\$ARGn\$ macros <user_config_opts> individual arguments from the command line. (host check command, service event handler definitions are processed during</code>
connector	Note: Centreon-Engine does not support the shell commands in <code>command_line</code> . You need to define a connector without shell features. This directive is used for link a command with a connector. When this directive is not empty, the command is replaced by the connector. When the connector is empty, the <code>command_line</code> argument is use.

Connector Definition

Description A connector is just like a command with better performances. A connector run on background and it is never close. A connector is define by a name and a command line.

Definition Format

Note: Optional directives are comment (line start with #).

```
define connector{
    connector_name connector_name
    connector_line connector_line
}
```

Example Definition

```
define connector{
    connector_name connector_icmp
    connector_line /usr/lib/nagios/plugins/connector_icmp
}
```

Directive Descriptions

connec- tor_name	This directive is the short name used to identify the connector. It is referenced in <i>command</i> definitions.
connec- tor_line	This directive is used to define the path of the binary connector and the optional argument. It is possible to use the Centreon-Engine macros.

Service Dependency Definition

Description Service dependencies are an advanced feature of Centreon Engine that allow you to suppress notifications and active checks of services based on the status of one or more other services. Service dependencies are optional and are mainly targeted at advanced users who have complicated monitoring setups. More information on how service dependencies work (read this!) can be found *here*.

Definition Format

Note: Optional directives are comment (line start with #). However, you must supply at least one type of criteria for the definition to be of much use.

```
define servicedependency{
    dependent_host_name          host_name
    # dependent_hostgroup_name    hostgroup_name
    dependent_service_description service_description
    host_name                    host_name
    # hostgroup_name              hostgroup_name
    service_description          service_description
    # inherits_parent             [0/1]
    # execution_failure_criteria  [o,w,u,c,p,n]
    # notification_failure_criteria [o,w,u,c,p,n]
    # dependency_period           timeperiod_name
}
```

Example Definition

```
define servicedependency{
    host_name          WWW1
    service_description Apache Web Server
    dependent_host_name WWW1
    dependent_service_description Main Web Site
    execution_failure_criteria n
    notification_failure_criteria w,u,c
}
```

Directive Descriptions

dependant_host_name	This directive is used to identify the short name(s) of the <i>host(s)</i> that the dependent service “runs” on or is associated with. Multiple hosts should be separated by commas.
dependant_hostgroup_name	This directive is used to specify the short name(s) of the <i>hostgroup(s)</i> that the dependent service “runs” on or is associated with. Multiple hostgroups should be separated by commas. The <i>dependant_hostgroup</i> may be used instead of, or in addition to, the <i>dependant_host</i> directive.
dependant_service_description	This directive is used to identify the description of the dependent <i>service</i> .
host_name	This directive is used to identify the short name(s) of the <i>host(s)</i> that the service that is being depended upon (also referred to as the master service) “runs” on or is associated with. Multiple hosts should be separated by commas.
hostgroup_name	This directive is used to identify the short name(s) of the <i>hostgroup(s)</i> that the service that is being depended upon (also referred to as the master service) “runs” on or is associated with. Multiple hostgroups should be separated by commas. The <i>hostgroup_name</i> may be used instead of, or in addition to, the <i>host_name</i> directive.
service_description	This directive is used to identify the description of the <i>service</i> that is being depended upon (also referred to as the master service).
inherits_parent	This directive indicates whether or not the dependency inherits dependencies of the service that is being depended upon (also referred to as the master service). In other words, if the master service is dependent upon other services and any one of those dependencies fail, the dependency will also fail.
execution_failure_criteria	This directive is used to specify the criteria that determine when the dependent service should not be actively checked. If the master service is in one of the failure states we specify, the dependent service will not be actively checked. Valid options are a combination of one or more of the following (multiple options are separated with commas): o = fail on an OK state, w = fail on a WARNING state, u = fail on an UNKNOWN state, c = fail on a CRITICAL state, and p = fail on a pending state (e.g. the service has not yet been checked). If you specify n (none) as an option, the execution dependency will never fail and checks of the dependent service will always be actively checked (if other conditions allow for it to be). Example: If you specify o,c,u in this field, the dependent service will not be actively checked if the master service is in either an OK, a CRITICAL, or an UNKNOWN state.
notification_failure_criteria	This directive is used to define the criteria that determine when notifications for the dependent service should not be sent out. If the master service is in one of the failure states we specify, notifications for the dependent service will not be sent to contacts. Valid options are a combination of one or more of the following: o = fail on an OK state, w = fail on a WARNING state, u = fail on an UNKNOWN state, c = fail on a CRITICAL state, and p = fail on a pending state (e.g. the service has not yet been checked). If you specify n (none) as an option, the notification dependency will never fail and notifications for the dependent service will always be sent out. Example: If you specify w in this field, the notifications for the dependent service will not be sent out if the master service is in a WARNING state.
dependency_period	This directive is used to specify the short name of the <i>time period</i> during which this dependency is valid. If this directive is not specified, the dependency is considered to be valid during all times.

Service Escalation Definition

Description Service escalations are completely optional and are used to escalate notifications for a particular service. More information on how notification escalations work can be found [here](#).

Definition Format

Note: Optional directives are comment (line start with #).

```

define serviceescalation{
    host_name                host_name
    # hostgroup_name          hostgroup_name
    service_description        service_description
    contacts                   contacts
    contact_groups              contactgroup_name
    first_notification          #
    last_notification           #
    notification_interval       #
    # escalation_period         timeperiod_name
    # escalation_options        [w,u,c,r]
}

```

Example Definition

```

define serviceescalation{
    host_name                nt-3
    service_description        Processor Load
    first_notification         4
    last_notification          0
    notification_interval      30
    contact_groups              all-nt-admins,themanagers
}

```

Descriptions Directive Descriptions

1.3. User

host_name	This directive is used to identify the <i>host(s)</i> that the <i>service</i> escalation is associated with.
hostgroup_name	This directive is used to specify the <i>hostgroup(s)</i> that the service escalation applies to or is associated with. Multiple <i>hostgroup(s)</i> may be separated by commas. The <i>hostgroup(s)</i> may be used instead of, or in addition to, the <i>host(s)</i> .
service_description	This directive is used to identify the <i>service</i> the escalation should apply to.
first_notification	This directive is a number that identifies the first notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will be used if the service is in a non-OK state for a third notification to go out.
last_notification	This directive is a number that identifies the last notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will be used if more than five notifications are sent out. Setting this value to 0 means to keep the escalation entry forever (no matter how many notifications are sent out).
contacts	This is a list of the short names of the contacts that should be notified whenever there is a problem (failures or coveries) with this service. Multiple contacts may be separated by commas. Useful if you want to go to just a few people and don't want to notify all <i>contact groups</i> . You must specify at least one <i>contact group</i> in each service escalation definition.
contact_groups	This directive is used to identify the <i>contact group</i> that should be notified whenever a notification is escalated. Multiple <i>contact groups</i> may be separated by commas. You must specify at least one <i>contact</i> or <i>contact group</i> in each service escalation definition.
notification_interval	This directive is used to determine the interval between notifications should be made while the escalation is valid. If you specify a value of 0, the escalation is valid. If you specify a value of 0, the escalation is valid, but no more problem notifications from the host. Notifications are sent out again when the host covers. This is useful if you want to limit the number of notifications sent out after a certain amount of time. If multiple escalation entries are defined for a service, from all escalation entries.
escalation_period	This directive is used to specify the <i>time period</i> during which this escalation is valid. If this directive is not specified, the escalation is valid during all times.
escalation_options	This directive is used to define the options when this service escalation is used. The options are used only if the service is in one of the states defined by this directive. If this directive is not specified, the escalation is considered valid for all service states. Valid options are: r = (recovery) state, w = escalate on a warning state, c = (critical) state.

Host Dependency Definition

Description Host dependencies are an advanced feature of Centreon Engine that allow you to suppress notifications for hosts based on the status of one or more other hosts. Host dependencies are optional and are mainly targeted at advanced users who have complicated monitoring setups. More information on how host dependencies work (read this!) can be found *here*.

Definition Format

Note: Optional directives are comment (line start with #).

```
define hostdependency{
    dependent_host_name      host_name
    # dependent_hostgroup_name  hostgroup_name
    host_name                host_name
    # hostgroup_name          hostgroup_name
    # inherits_parent         [0/1]
    # execution_failure_criteria [o,d,u,p,n]
    # notification_failure_criteria [o,d,u,p,n]
    # dependency_period       timeperiod_name
}
```

Example Definition

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

Directive Descriptions

depend- ent_host_name	This directive is used to identify the short name(s) of the dependent <i>host(s)</i> . Multiple hosts should be separated by commas.
depend- ent_hostgroup_name	This directive is used to identify the short name(s) of the dependent <i>hostgroup(s)</i> . Multiple hostgroups should be separated by commas. The dependent_hostgroup_name may be used instead of, or in addition to, the dependent_host_name directive.
host_name	This directive is used to identify the short name(s) of the <i>host(s)</i> that is being depended upon (also referred to as the master host). Multiple hosts should be separated by commas.
host- group_name	This directive is used to identify the short name(s) of the <i>hostgroup(s)</i> that is being depended upon (also referred to as the master host). Multiple hostgroups should be separated by commas. The hostgroup_name may be used instead of, or in addition to, the host_name directive.
inherits_parent	This directive indicates whether or not the dependency inherits dependencies of the host it is being depended upon (also referred to as the master host). In other words, if the master host is dependent upon other hosts and any one of those dependencies fail, this dependency will fail.
execu- tion_failure_criteria	This directive is used to specify the criteria that determine when the dependent host should be actively checked. If the master host is in one of the failure states we specify, the dependent host will not be actively checked. Valid options are a combination of one or more of the following (multiple options are separated with commas): o = fail on an UP state, d = fail on a DOWN state, u = fail on an UNREACHABLE state, and p = fail on a pending state (e.g. the host has not yet been checked). If you specify n (none) as an option, the execution dependency will never fail and the dependent host will always be actively checked (if other conditions allow for it to be). Example: If you specify u,d in this field, the dependent host will not be actively checked if the master host is in either an UNREACHABLE or DOWN state.
notifica- tion_failure_criteria	This directive is used to define the criteria that determine when notifications for the dependent host should not be sent out. If the master host is in one of the failure states we specify, notifications for the dependent host will not be sent to contacts. Valid options are a combination of one or more of the following: o = fail on an UP state, d = fail on a DOWN state, u = fail on an UNREACHABLE state, and p = fail on a pending state (e.g. the host has not yet been checked). If you specify n (none) as an option, the notification dependency will never fail and notifications for the dependent host will always be sent out. Example: If you specify d in this field, the notifications for the dependent host will not be sent out if the master host is in a DOWN state.
depend- ency_period	This directive is used to specify the short name of the <i>time period</i> during which this dependency is valid. If this directive is not specified, the dependency is considered to be valid during all times.

Host Escalation Definition

Description Host escalations are completely optional and are used to escalate notifications for a particular host. More information on how notification escalations work can be found [here](#).

Definition Format

Note: Optional directives are comment (line start with #).

```
define hostescalation{
    host_name                host_name
    # hostgroup_name         hostgroup_name
    contacts                 contacts
    contact_groups           contactgroup_name
    first_notification        #
    last_notification         #
    notification_interval     #
    # escalation_period       timeperiod_name
}
```

```
# escalation_options      [d,u,r]
}
```

Example Definition

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


Directive Descriptions

host_name	This directive is used to identify the short name of the <i>host</i> that the escalation should apply to.
hostgroup_name	This directive is used to identify the short name(s) of the <i>hostgroup(s)</i> that the escalation should apply to. If multiple hostgroups should be separated by commas. If not used, the escalation will apply to all hosts that are members of the specified hostgroup(s).
first_notification	This directive is a number that identifies the first notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the host is down or unreachable long enough for a third notification to go out.
last_notification	This directive is a number that identifies the last notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five notifications are sent out for this host. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications are sent out).
contacts	This is a list of the short names of the <i>contacts</i> that should be notified whenever there are problems (notifications or discoveries) with this host. Multiple contacts should be separated by commas. Useful if you want notifications to go to just a few people and don't want to contact all "contact groups". You must specify at least one contact or contact group in each host escalation definition.
contact_groups	This directive is used to identify the short name(s) of the <i>contact group</i> that should be notified when the host notification is escalated. Multiple contact groups should be separated by commas. You must specify at least one contact or contact group in each host escalation definition.
notification_interval	This directive is used to determine the interval at which notifications should be made while this escalation is valid. If you specify a value of 0 for the interval, the Centreon Engine will send the first notification when the escalation definition is valid, but will then prevent any more problem notifications from being sent out for this host. Notifications are sent out again until the host recovers. This is useful if you want to stop having notifications sent out after a certain amount of time. ... If multiple escalation entries for a host are defined, the value from all escalation entries is used.
escalation_period	This directive is used to specify the short name of the <i>time period</i> during which this escalation is valid. If this directive is not specified, the escalation is considered to be valid during all times.
escalation_options	This directive is used to define the criteria that determine when this host escalation is used. The escalation is only valid if the host is in one of the states specified in this directive. If this directive is not specified in a host escalation, the escalation is considered to be valid for all host states. Valid options are a combination of any or more of the following: r = escalate on an UP (recovered) state, d = escalate on a DOWN state, and u = escalate on an UNREACHABLE state. Example: If you specify "r,d,u" in this field, the escalation will only be used if the host is in a DOWN state.

1.3. User

MERETHIS 12 AVENUE RASPAIL FR94290 GENTILLY

www.centreon.com

Extended Host Information Definition

Description Extended host information entries are basically used to make the output. They have no effect on monitoring and are completely optional.

Note: As of Centreon Engine 1.x, all directives contained in extended host information definitions are also available in *host definitions*. Thus, you can choose to define the directives below in your host definitions if it makes your configuration simpler. Separate extended host information definitions will continue to be supported for backward compatability.

Definition Format

Note: Optional directives are comment (line start with #). However, you need to supply at least one optional variable in each definition for it to be of much use.

```
define hostextinfo{
    host_name      host_name
    # notes        note_string
    # notes_url     url
    # action_url    url
    # icon_image    image_file
    # icon_image_alt alt_string
    # vrml_image    image_file
    # statusmap_image image_file
    # 2d_coords     x_coord,y_coord
    # 3d_coords     x_coord,y_coord,z_coord
}
```

Example Definition

```
define hostextinfo{
    host_name      netware1
    notes          This is the primary Netware file server
    notes_url      http://webserver.localhost.localdomain/hostinfo.pl?host=netware1
    icon_image     novell40.png
    icon_image_alt IntranetWare 4.11
    vrml_image     novell40.png
    statusmap_image novell40.gd2
    2d_coords      100,250
    3d_coords      100.0,50.0,75.0
}
```

Variable Descriptions

host_name	This variable is used to identify the short name of the <i>host</i> which the data is associated with.
notes	This directive is used to define an optional string of notes pertaining to the host.
notes_url	This variable is used to define an optional URL that can be used to provide more information about the host. Any valid URL can be used. This can be very useful if you want to make detailed information on the host, emergency contact methods, etc. available to other support staff.
action_url	This directive is used to define an optional URL that can be used to provide more actions to be performed on the host. Any valid URL can be used.
icon_image	This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. The image will look best if it is 40x40 pixels in size. Images for hosts are assumed to be in the <code>logos/</code> subdirectory in your HTML images directory (i.e. <code>/usr/local/centengine/share/images/logos</code>).
icon_image_alt	This variable is used to define an optional string that is used in the ALT tag of the image specified in the <code><icon_image></code> argument.
vrmf_image	This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be used as the texture map for the specified host. Unlike the image you specify for the <code><icon_image></code> variable, this one should probably not have any transparency. If it does, the host object will look a bit wierd. Images for hosts are assumed to be in the <code>logos/</code> subdirectory in your HTML images directory (i.e. <code>/usr/local/centengine/share/images/logos</code>).
statusmap_image	This variable is used to define the name of an image that should be associated with this host. You can specify a JPEG, PNG, and GIF image if you want, although I would strongly suggest using a GIF format image, as other image formats will result in a lot of wasted CPU time when the statusmap image is generated. GD2 images can be created from PNG images by using the <code>pngtogd2</code> utility supplied with Thomas Boutell's gd library . The GD2 images should be created in uncompressed format in order to minimize CPU load. The image will look best if it is 40x40 pixels in size. Images for hosts are assumed to be in the <code>logos/</code> subdirectory in your HTML images directory (i.e. <code>/usr/local/centengine/share/images/logos</code>).
2d_coords	This variable is used to define coordinates to use when drawing the host. Coordinates should be given in positive integers, as they correspond to physical pixels in the generated image. The origin for drawing (0,0) is in the upper left hand corner of the image and extends in the positive x direction (to the right) along the top of the image and in the positive y direction (down) along the left hand side of the image. For reference, the size of the icons drawn is usually about 40x40 pixels (text takes a little extra space). The coordinates you specify here are for the upper left hand corner of the host that is drawn. Note: Don't worry about what the maximum x and y coordinates that you can use.
3d_coords	This variable is used to define coordinates to use when drawing the host. Coordinates can be positive or negative real numbers. The origin for drawing is (0.0,0.0,0.0). For reference, the size of the host cubes drawn is 0.5 units on each side (text takes a little more space). The coordinates you specify here are used as the center of the host cube.

Extended Service Information Definition

Description Extended service information entries are basically used to make the output. They have no effect on monitoring and are completely optional.

Note: As of Centreon Engine 3.x, all directives contained in extended service information definitions are also available in *service definitions*. Thus, you can choose to define the directives below in your service definitions if it makes your configuration simpler. Separate extended service information definitions will continue to be supported for backward compatibility.

Definition Format

Note: Optional directives are comment (line start with #). However, you need to supply at least one optional variable in each definition for it to be of much use.

```

define serviceextinfo{
    host_name          host_name
    service_description service_description
    # notes             note_string
    # notes_url         url
    # action_url        url
    # icon_image        image_file
    # icon_image_alt    alt_string
}

```

Example Definition

```

define serviceextinfo{
    host_name          linux2
    service_description Log Anomalies
    notes Security-related log anomalies on secondary Linux server
    notes_url          http://webserver.localhost.localdomain/serviceinfo.pl?host=linux2&service=Log Anomalies
    icon_image         security.png
    icon_image_alt     Security-Related Alerts
}

```

Variable Descriptions

host_name	This directive is used to identify the short name of the host that the <i>service</i> is associated with.
service_description	This directive is description of the <i>service</i> which the data is associated with.
notes	This directive is used to define an optional string of notes pertaining to the service.
notes_url	This directive is used to define an optional URL that can be used to provide more information on the service. Any valid URL can be used. This can be very useful if you want to make detailed information on the service, emergency contact methods, etc. available to other support staff.
action_url	This directive is used to define an optional URL that can be used to provide more actions to be performed on the service. Any valid URL can be used.
icon_image	This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. The image will look best if it is 40x40 pixels in size. Images for hosts are assumed to be in the logos/ subdirectory in your HTML images directory (i.e. /usr/local/centengine/share/images/logos).
icon_image_alt	This variable is used to define an optional string that is used in the ALT tag of the image specified by the <icon_image> argument.

Object Inheritance

Introduction This documentation attempts to explain object inheritance and how it can be used in your *object definitions*.

If you are confused about how recursion and inheritance work after reading this, take a look at the sample object config files provided in the Centreon Engine distribution. If that still doesn't help, drop an email message with a detailed description of your problem to the centengine-users mailing list.

Basics There are three variables affecting recursion and inheritance that are present in all object definitions. They are indicated in red as follows:

```

define someobjecttype{
    object-specific variables ...
    name      template_name
    use      name_of_template_to_use
}

```

```

    register [0/1]
}

```

The first variable is name. Its just a “template” name that can be referenced in other object definitions so they can inherit the objects properties/variables. Template names must be unique amongst objects of the same type, so you can’t have two or more host definitions that have “hosttemplate” as their template name.

The second variable is use. This is where you specify the name of the template object that you want to inherit properties/variables from. The name you specify for this variable must be defined as another object’s template named (using the name variable).

The third variable is register. This variable is used to indicate whether or not the object definition should be “registered” with Centreon Engine. By default, all object definitions are registered. If you are using a partial object definition as a template, you would want to prevent it from being registered (an example of this is provided later). Values are as follows: 0 = do NOT register object definition, 1 = register object definition (this is the default). This variable is NOT inherited; every (partial) object definition used as a template must explicitly set the register directive to be 0. This prevents the need to override an inherited register directive with a value of 1 for every object that should be registered.

Local Variables vs. Inherited Variables One important thing to understand with inheritance is that “local” object variables always take precedence over variables defined in the template object. Take a look at the following example of two host definitions (not all required variables have been supplied):

```

define host{
    host_name          bighost1
    check_command       check-host-alive
    notification_options d,u,r
    max_check_attempts 5
    name               hosttemplate1
}

define host{
    host_name          bighost2
    max_check_attempts 3
    use                hosttemplate1
}

```

You’ll note that the definition for host bighost1 has been defined as having hosttemplate1 as its template name. The definition for host bighost2 is using the definition of bighost1 as its template object. Once Centreon Engine processes this data, the resulting definition of host bighost2 would be equivalent to this definition:

```

define host{
    host_name          bighost2
    check_command       check-host-alive
    notification_options d,u,r
    max_check_attempts 3
}

```

You can see that the check_command and notification_options variables were inherited from the template object (where host bighost1 was defined). However, the host_name and max_check_attempts variables were not inherited from the template object because they were defined locally. Remember, locally defined variables override variables that would normally be inherited from a template object. That should be a fairly easy concept to understand.

Note: If you would like local string variables to be appended to inherited string values, you can do so. Read more about how to accomplish this *below*.

Inheritance Chaining Objects can inherit properties/variables from multiple levels of template objects. Take the following example:

```
define host{
    host_name          bighost1
    check_command       check-host-alive
    notification_options d,u,r
    max_check_attempts 5
    name               hosttemplate1
}

define host{
    host_name          bighost2
    max_check_attempts 3
    use                hosttemplate1
    name               hosttemplate2
}

define host{
    host_name bighost3
    use       hosttemplate2
}
```

You'll notice that the definition of host bighost3 inherits variables from the definition of host bighost2, which in turn inherits variables from the definition of host bighost1. Once Centreon Engine processes this configuration data, the resulting host definitions are equivalent to the following:

```
define host{
    host_name          bighost1
    check_command       check-host-alive
    notification_options d,u,r
    max_check_attempts 5
}

define host{
    host_name          bighost2
    check_command       check-host-alive
    notification_options d,u,r
    max_check_attempts 3
}

define host{
    host_name          bighost3
    check_command       check-host-alive
    notification_options d,u,r
    max_check_attempts 3
}
```

There is no inherent limit on how “deep” inheritance can go, but you'll probably want to limit yourself to at most a few levels in order to maintain sanity.

Using Incomplete Object Definitions as Templates It is possible to use incomplete object definitions as templates for use by other object definitions. By “incomplete” definition, I mean that all required variables in the object have not been supplied in the object definition. It may sound odd to use incomplete definitions as templates, but it is in fact recommended that you use them. Why? Well, they can serve as a set of defaults for use in all other object definitions. Take the following example:

```

define host{
    check_command      check-host-alive
    notification_options d,u,r
    max_check_attempts 5
    name               generichosttemplate
    register           0
}

define host{
    host_name bighost1
    address   192.168.1.3
    use       generichosttemplate
}

define host{
    host_name bighost2
    address   192.168.1.4
    use       generichosttemplate
}

```

Notice that the first host definition is incomplete because it is missing the required `host_name` variable. We don't need to supply a host name because we just want to use this definition as a generic host template. In order to prevent this definition from being registered with Centreon Engine as a normal host, we set the `register` variable to 0.

The definitions of hosts `bighost1` and `bighost2` inherit their values from the generic host definition. The only variable we've chosen to override is the `address` variable. This means that both hosts will have the exact same properties, except for their `host_name` and `address` variables. Once Centreon Engine processes the config data in the example, the resulting host definitions would be equivalent to specifying the following:

```

define host{
    host_name      bighost1
    address        192.168.1.3
    check_command  check-host-alive
    notification_options d,u,r
    max_check_attempts 5
}

define host{
    host_name      bighost2
    address        192.168.1.4
    check_command  check-host-alive
    notification_options d,u,r
    max_check_attempts 5
}

```

At the very least, using a template definition for default variables will save you a lot of typing. It'll also save you a lot of headaches later if you want to change the default values of variables for a large number of hosts.

Custom Object Variables

Any *custom object variables* that you define in your host, service, or contact definition templates will be inherited just like other standard variables. Take the following example:

```

define host{
    _customvar1      somevalue ; <-- Custom host variable
    _snmp_community  public ; <-- Custom host variable
    name             generichosttemplate
    register         0
}

```

```

}

define host{
    host_name bighost1
    address 192.168.1.3
    use generichosttemplate
}

```

The host `bighost1` will inherit the custom host variables `_customvar1` and `_snmp_community`, as well as their respective values, from the `generichosttemplate` definition. The effective result is a definition for `bighost1` that looks like this:

```

define host{
    host_name      bighost1
    address        192.168.1.3
    _customvar1    somevalue
    _snmp_community public
}

```

Cancelling Inheritance of String Values In some cases you may not want your host, service, or contact definitions to inherit values of string variables from the templates they reference. If this is the case, you can specify “null” (without quotes) as the value of the variable that you do not want to inherit. Take the following example:

```

define host{
    event_handler my-event-handler-command
    name          generichosttemplate
    register      0
}

define host{
    host_name      bighost1
    address        192.168.1.3
    event_handler  null
    use            generichosttemplate
}

```

In this case, the host `bighost1` will not inherit the value of the `event_handler` variable that is defined in the `generichosttemplate`. The resulting effective definition of `bighost1` is the following:

```

define host{
    host_name bighost1
    address 192.168.1.3
}

```

Additive Inheritance of String Values Centreon Engine gives preference to local variables instead of values inherited from templates. In most cases local variable values override those that are defined in templates. In some cases it makes sense to allow Centreon Engine to use the values of inherited and local variables together.

This “additive inheritance” can be accomplished by prepending the local variable value with a plus sign (+). This feature is only available for standard (non-custom) variables that contain string values. Take the following example:

```

define host{
    hostgroups all-servers
    name        generichosttemplate
    register    0
}

```



```
define host{
    host_name    linuxserver1
    hostgroups   +linux-servers,web-servers
    use          generichosttemplate
}
```

In this case, the host linuxserver1 will append the value of its local hostgroups variable to that from generichosttemplate. The resulting effective definition of linuxserver1 is the following:

```
define host{
    host_name    linuxserver1
    hostgroups   all-servers,linux-servers,web-servers
}
```

Implied Inheritance Normally you have to either explicitly specify the value of a required variable in an object definition or inherit it from a template. There are a few exceptions to this rule, where Centreon Engine will assume that you want to use a value that instead comes from a related object. For example, the values of some service variables will be copied from the host the service is associated with if you don't otherwise specify them.

The following table lists the object variables that will be implicitly inherited from related objects if you don't explicitly specify their value in your object definition or inherit them from a template.

Object Type	Object Variable	Implied Source
Services	contacts	contacts in the associated host definition
Services	contact_groups	contact_groups in the associated host definition
Services	notification_interval	notification_interval in the associated host definition
Services	notification_period	notification_period in the associated host definition
Services	timezone	timezone in the associated host definition
Host Escalations	contacts	contacts in the associated host definition
Host Escalations	contact_groups	contact_groups in the associated host definition
Host Escalations	notification_interval	notification_interval in the associated host definition
Host Escalations	escalation_period	notification_period in the associated host definition
Service Escalations	contacts	contacts in the associated service definition
Service Escalations	contact_groups	contact_groups in the associated service definition
Service Escalations	notification_interval	notification_interval in the associated service definition
Service Escalations	escalation_period	notification_period in the associated service definition

Implied/Additive Inheritance in Escalations Service and host escalation definitions can make use of a special rule that combines the features of implied and additive inheritance. If escalations 1) do not inherit the values of their contact_groups or contacts directives from another escalation template and 2) their contact_groups or contacts directives begin with a plus sign (+), then the values of their corresponding host or service definition's contact_groups or contacts directives will be used in the additive inheritance logic.

Here's an example:

```
define host{
    name          linux-server
    contact_groups linux-admins
    ...
}

define hostescalation{
    host_name      linux-server
    contact_groups +management
    ...
}
```

This is a much simpler equivalent to:

```
define hostescalation{
    host_name      linux-server
    contact_groups linux-admins,management
    ...
}
```

Important values Service templates can make use of a special rule which gives precedence to their `check_command` value. If the `check_command` is prefixed with an exclamation mark (!), then the template's `check_command` is marked as important and will be used over the `check_command` defined for the service (this is styled after CSS syntax, which uses ! as an important attribute).

Why is this useful? It is mainly useful when setting a different `check_command` for distributed systems. You may want to set a freshness threshold and a `check_command` that forces the service into a failed state, but this doesn't work with the normal templating system. Using this important flag allows the custom `check_command` to be written, but a general distributed template can be used to overrule the `check_command` when used on a central Centreon Engine server.

For instance:

```
# On master
define service{
    name                service-distributed
    register            0
    active_checks_enabled 0
    check_freshness     1
    check_command       !set_to_stale
}

# On slave
define service{
    name                service-distributed
    register            0
    active_checks_enabled 1
}

# Service definition, used by master and slave
define service{
    host_name          host1
    service_description serviceA
    check_command       check_http...
    use                 service-distributed
    ...
}
```

Multiple Inheritance Sources Thus far, all examples of inheritance have shown object definitions inheriting variables/values from just a single source. You are also able to inherit variables/values from multiple sources for more complex configurations, as shown below:

```
# Generic host template
define host{
    name                generic-host
    active_checks_enabled 1
    check_interval      10
    ...
    register            0
}
```

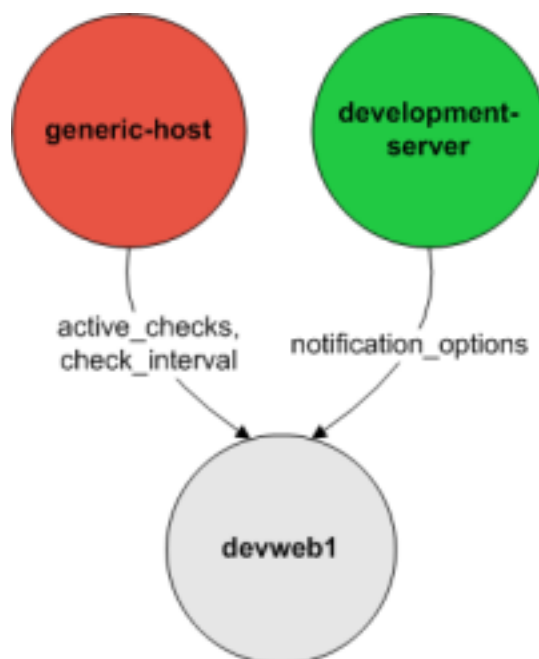
```

}

# Development web server template
define host{
    name                development-server
    check_interval       15
    notification_options d,u,r
    ...
    register             0
}

# Development web server
define host{
    use                 generic-host,development-server
    host_name devweb1
    ...
}

```



In the example above, devweb1 is inheriting variables/values from two sources: generic-host and development-server. You'll notice that a check_interval variable is defined in both sources. Since generic-host was the first template specified in devweb1's use directive, its value for the check_interval variable is inherited by the devweb1 host. After inheritance, the effective definition of devweb1 would be as follows:

```

# Development web server
define host{
    host_name            devweb1
    active_checks_enabled 1
    check_interval       10
    notification_options d,u,r
    ...
}

```

Precedence With Multiple Inheritance Sources When you use multiple inheritance sources, it is important to know how Centreon Engine handles variables that are defined in multiple sources. In these cases Centreon Engine

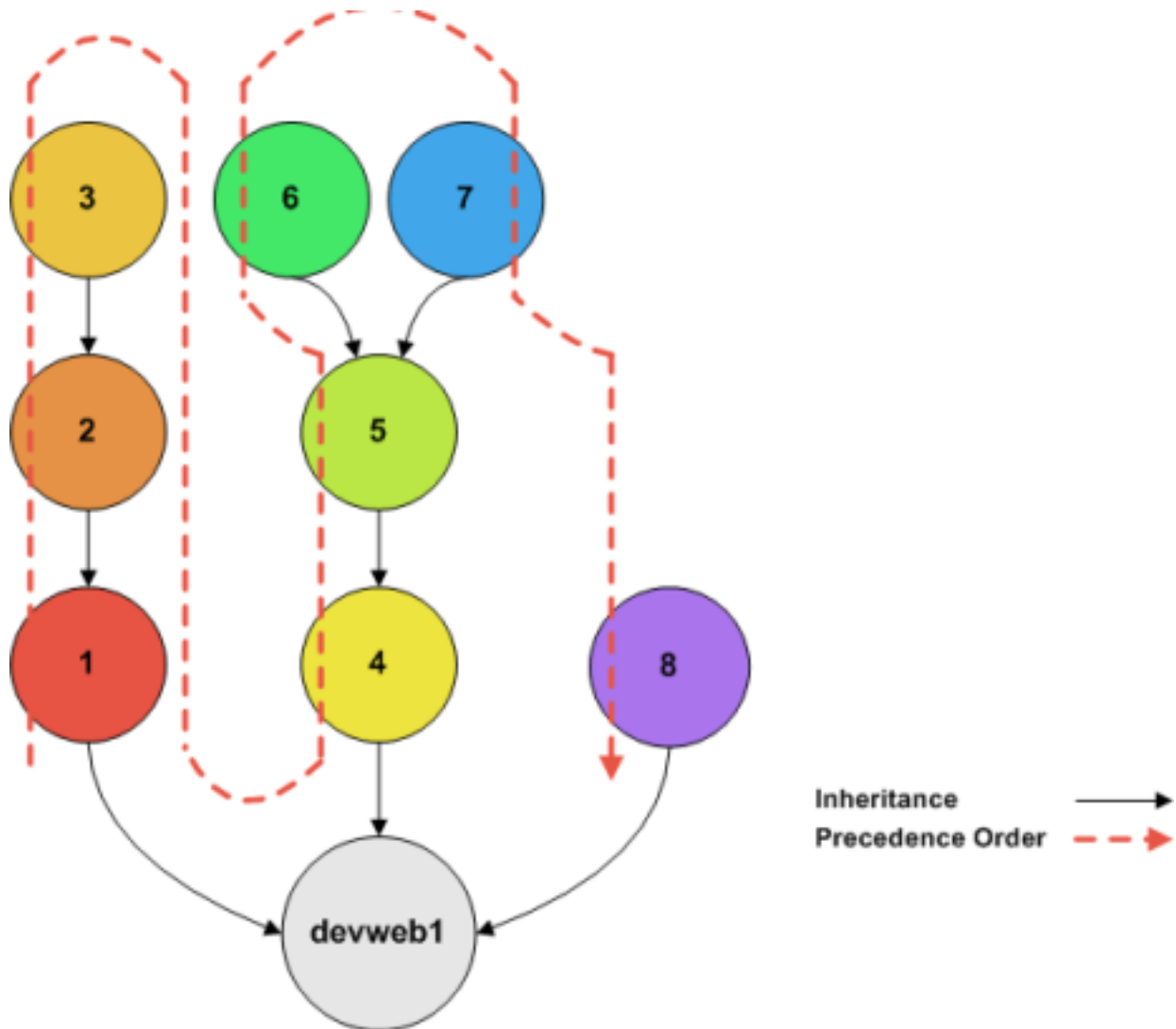
will use the variable/value from the first source that is specified in the use directive. Since inheritance sources can themselves inherit variables/values from one or more other sources, it can get tricky to figure out what variable/value pairs take precedence.

Consider the following host definition that references three templates:

```
# Development web server
define host{
    use      1, 4, 8
    host_name devweb1
    ...
}
```

If some of those referenced templates themselves inherit variables/values from one or more other templates, the precedence rules are shown to the right.

Testing, trial, and error will help you better understand exactly how things work in complex inheritance situations like this. :-)



Understanding Macros and How They Work

Macros One of the main features that make Centreon Engine so flexible is the ability to use macros in command definitions. Macros allow you to reference information from hosts, services, and other sources in your commands.

Macro Substitution - How Macros Work Before Centreon Engine executes a command, it will replace any macros it finds in the command definition with their corresponding values. This macro substitution occurs for all types of commands that Centreon Engine executes - host and service checks, notifications, event handlers, etc.

Certain macros may themselves contain other macros. These include the \$HOSTNOTES\$, \$HOSTNOTESURL\$, \$HOSTACTIONURL\$, \$SERVICENOTES\$, \$SERVICENOTESURL\$, and \$SERVICEACTIONURL\$ macros.

Example 1: Host Address Macro When you use host and service macros in command definitions, they refer to values for the host or service for which the command is being run. Let's try an example. Assuming we are using a host definition and a check_ping command defined like this:

```
define host{
    host_name      linuxbox
    address        192.168.1.2
    check_command  check_ping
    ...
}

define command{
    command_name check_ping
    command_line  /usr/lib/nagios/plugins/check_ping -H $HOSTADDRESS$ -w 100.0,90% -c 200.0,60%
}
```

The expanded/final command line to be executed for the host's check command would look like this:

```
$ /usr/lib/nagios/plugins/check_ping -H 192.168.1.2 -w 100.0,90% -c 200.0,60%
```

Pretty simple, right? The beauty in this is that you can use a single command definition to check an unlimited number of hosts. Each host can be checked with the same command definition because each host's address is automatically substituted in the command line before execution.

Example 2: Command Argument Macros You can pass arguments to commands as well, which is quite handy if you'd like to keep your command definitions rather generic. Arguments are specified in the object (i.e. host or service) definition, by separating them from the command name with exclamation points (!) like so:

```
define service{
    host_name      linuxbox
    service_description PING
    check_command  check_ping!200.0,80%!400.0,40%
    ...
}
```

In the example above, the service check command has two arguments (which can be referenced with *ARGn* macros). The \$ARG1\$ macro will be "200.0,80%" and \$ARG2\$ will be "400.0,40%" (both without quotes). Assuming we are using the host definition given earlier and a check_ping command defined like this:

```
define command{
    command_name check_ping
    command_line  /usr/lib/nagios/plugins/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$
}
```

The expanded/final command line to be executed for the service's check command would look like this:

```
$ /usr/lib/nagios/plugins/check_ping -H 192.168.1.2 -w 200.0,80% -c 400.0,40%
```

Note: If you need to pass bang (!) characters in your command arguments, you can do so by escaping them with a backslash (). If you need to include backslashes in your command arguments, they should also be escaped with a backslash.

On-Demand Macros Normally when you use host and service macros in command definitions, they refer to values for the host or service for which the command is being run. For instance, if a host check command is being executed for a host named “linuxbox”, all the *standard host macros* will refer to values for that host (“linuxbox”).

If you would like to reference values for another host or service in a command (for which the command is not being run), you can use what are called “on-demand” macros. On-demand macros look like normal macros, except for the fact that they contain an identifier for the host or service from which they should get their value. Here’s the basic format for on-demand macros:

```
* **HOSTMACRONAME:** host_name
* **SERVICEMACRONAME:** host_name:service_description
```

Replace HOSTMACRONAME and SERVICEMACRONAME with the name of one of the standard host or service macros found *here*.

Note that the macro name is separated from the host or service identifier by a colon (:). For on-demand service macros, the service identifier consists of both a host name and a service description - these are separated by a colon (:) as well.

Note: On-demand service macros can contain an empty host name field. In this case the name of the host associated with the service will automatically be used.

Examples of on-demand host and service macros follow:

```
$HOSTDOWNTIME:myhost$ <--- On-demand host macro
$SERVICESTATEID:novellserver:DS Database$ <--- On-demand service macro
$SERVICESTATEID::CPU Load$ <--- On-demand service macro with blank host name field
```

On-demand macros are also available for hostgroup, servicegroup, contact, and contactgroup macros. For example:

```
$CONTACTEMAIL:john$ <--- On-demand contact macro
$CONTACTGROUPMEMBERS:linux-admins$ <--- On-demand contactgroup macro
$HOSTGROUPALIAS:linux-servers$ <--- On-demand hostgroup macro
$SERVICEGROUPALIAS:DNS-Cluster$ <--- On-demand servicegroup macro
```

On-Demand Group Macros You can obtain the values of a macro across all contacts, hosts, or services in a specific group by using a special format for your on-demand macro declaration. You do this by referencing a specific host group, service group, or contact group name in an on-demand macro, like so:

```
* **HOSTMACRONAME:** hostgroup_name:delimiter
* **SERVICEMACRONAME:**servicegroup_name:delimiter
* **CONTACTMACRONAME:**contactgroup_name:delimiter
```

Replace HOSTMACRONAME, SERVICEMACRONAME, and CONTACTMACRONAME with the name of one of the standard host, service, or contact macros found *here*. The delimiter you specify is used to separate macro values for each group member.

For example, the following macro will return a comma-separated list of host state ids for hosts that are members of the hg1 hostgroup:

```
$HOSTSTATEID:hg1:,$
```

This macro definition will return something that looks like this:

```
0,2,1,1,0,0,2
```

Custom Variable Macros Any *custom object variables* that you define in host, service, or contact definitions are also available as macros. Custom variable macros are named as follows:

- `$_HOSTvarname$`
- `$_SERVICEvarname$`
- `$_CONTACTvarname$`

Take the following host definition with a custom variable called “`_MACADDRESS`”:

```
define host{
    host_name linuxbox
    address 192.168.1.1
    _MACADDRESS 00:01:02:03:04:05
    ...
}
```

The `_MACADDRESS` custom variable would be available in a macro called `$_HOSTMACADDRESS$`. More information on custom object variables and how they can be used in macros can be found *here*.

Macro Cleansing Some macros are stripped of potentially dangerous shell metacharacters before being substituted into commands to be executed. Which characters are stripped from the macros depends on the setting of the *illegal_macro_output_chars* directive. The following macros are stripped of potentially dangerous characters:

- `HOSTOUTPUT`
- `LONGHOSTOUTPUT`
- `HOSTPERFDATA`
- `HOSTACKAUTHOR`
- `HOSTACKCOMMENT`
- `SERVICEOUTPUT`
- `LONGSERVICEOUTPUT`
- `SERVICEPERFDATA`
- `SERVICEACKAUTHOR`
- `SERVICEACKCOMMENT`

Additionally, any macros that contain *custom variables* are stripped for safety and security.

Macros as Environment Variables Most macros are made available as environment variables for easy reference by scripts or commands that are executed by Centreon Engine. For purposes of security and sanity, *USERn* and “on-demand” host and service macros are not made available as environment variables.

Environment variables that contain standard macros are named the same as their corresponding macro names (listed *here*), with `NAGIOS_` prepended to their names. For example, the `HOSTNAME` macro would be available as an environment variable named `NAGIOS_HOSTNAME`.

Available Macros A list of all the macros that are available in Centreon Engine, as well as a chart of when they can be used, can be found [here](#).

Standard Macros

Macro Validity Although macros can be used in all commands you define, not all macros may be “valid” in a particular type of command. For example, some macros may only be valid during service notification commands, whereas other may only be valid during host check commands. There are ten types of commands that Centreon Engine recognizes and treats differently. They are as follows:

- Service checks
- Service notifications
- Host checks
- Host notifications
- Service *event handlers* and/or a global service event handler
- Host *event handlers* and/or a global host event handler
- *OCSP* command
- *OCHP* command
- Service *performance data* commands
- Host *performance data* commands

The tables below list all macros currently available in Centreon Engine, along with a brief description of each and the types of commands in which they are valid. If a macro is used in a command in which it is invalid, it is replaced with an empty string. It should be noted that macros consist of all uppercase characters and are enclosed in \$ characters.

Macro Availability Chart

Legend	No	The macro is not available
	Yes	The macro is available

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers
HOSTNAME	Yes	Yes	Yes	Yes	Yes
HOSTDISPLAYNAME	Yes	Yes	Yes	Yes	Yes
HOSTALIAS	Yes	Yes	Yes	Yes	Yes
HOSTADDRESS	Yes	Yes	Yes	Yes	Yes
HOSTSTATE	Yes	Yes	Yes ¹	Yes	Yes
HOSTSTATEID	Yes	Yes	Yes ¹	Yes	Yes
LASTHOSTSTATE	Yes	Yes	Yes	Yes	Yes
LASTHOSTSTATEID	Yes	Yes	Yes	Yes	Yes
HOSTSTATETYPE	Yes	Yes	Yes ¹	Yes	Yes
HOSTATTEMPT	Yes	Yes	Yes	Yes	Yes
MAXHOSTATTEMPTS	Yes	Yes	Yes	Yes	Yes
HOSTEVENTID	Yes	Yes	Yes	Yes	Yes
LASTHOSTEVENTID	Yes	Yes	Yes	Yes	Yes
HOSTPROBLEMID	Yes	Yes	Yes	Yes	Yes
LASTHOSTPROBLEMID	Yes	Yes	Yes	Yes	Yes

Table 1.3 – continued from previous page

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers
HOSTLATENCY	Yes	Yes	Yes	Yes	Yes
HOSTEXECUTIONTIME	Yes	Yes	Yes ¹	Yes	Yes
HOSTDURATION	Yes	Yes	Yes	Yes	Yes
HOSTDURATIONSEC	Yes	Yes	Yes	Yes	Yes
HOSTDOWNTIME	Yes	Yes	Yes	Yes	Yes
HOSTPERCENTCHANGE	Yes	Yes	Yes	Yes	Yes
HOSTGROUPNAME	Yes	Yes	Yes	Yes	Yes
HOSTGROUPNAMES	Yes	Yes	Yes	Yes	Yes
LASTHOSTCHECK	Yes	Yes	Yes	Yes	Yes
LASTHOSTSTATECHANGE	Yes	Yes	Yes	Yes	Yes
LASTHOSTUP	Yes	Yes	Yes	Yes	Yes
LASTHOSTDOWN	Yes	Yes	Yes	Yes	Yes
LASTHOSTUNREACHABLE	Yes	Yes	Yes	Yes	Yes
HOSTOUTPUT	Yes	Yes	Yes ¹	Yes	Yes
LONGHOSTOUTPUT	Yes	Yes	Yes ¹	Yes	Yes
HOSTPERFDATA	Yes	Yes	Yes ¹	Yes	Yes
HOSTCHECKCOMMAND	Yes	Yes	Yes	Yes	Yes
HOSTACKAUTHOR ⁸	No	No	No	Yes	No
HOSTACKAUTHORNAME ⁸	No	No	No	Yes	No
HOSTACKAUTHORALIAS ⁸	No	No	No	Yes	No
HOSTACKCOMMENT ⁸	No	No	No	Yes	No
HOSTACTIONURL	Yes	Yes	Yes	Yes	Yes
HOSTNOTESURL	Yes	Yes	Yes	Yes	Yes
HOSTNOTES	Yes	Yes	Yes	Yes	Yes
TOTALHOSTSERVICES	Yes	Yes	Yes	Yes	Yes
TOTALHOSTSERVICESOK		Yes	Yes	Yes	Yes
TOTALHOSTSERVICESWARNING		Yes	Yes	Yes	Yes
TOTALHOSTSERVICESUNKNOWN		Yes	Yes	Yes	Yes
TOTALHOSTSERVICESCRITICAL		Yes	Yes	Yes	Yes

Host Macros³

	Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OSCP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Host Group Macros	HOST-GROUPALIAS ⁵	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	HOST-GROUP-MEMBERS ⁵	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	HOST-GROUP-NOTES ⁵	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	HOST-GROUP-NOTESURL ⁵	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	HOST-GROUPACTIONURL ⁵	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers
SERVICEDESC	Yes	Yes		No	Yes
SERVICEDISPLAYNAME	Yes	Yes		No	Yes
SERVICESTATE ²	Yes	Yes		No	Yes
SERVICESTATEID ²	Yes	Yes		No	Yes
LASTSERVICESTATE	Yes	Yes		No	Yes
LASTSERVICESTATEID	Yes	Yes		No	Yes
SERVICESTATETYPE	Yes	Yes		No	Yes
SERVICEATTEMPT	Yes	Yes		No	Yes
MAXSERVICEATTEMPTS	Yes	Yes		No	Yes
SERVICEISVOLATILE	Yes	Yes		No	Yes
SERVICEEVENTID	Yes	Yes		No	Yes
LASTSERVICEEVENTID	Yes	Yes		No	Yes
SERVICEPROBLEMID	Yes	Yes		No	Yes
LASTSERVICEPROBLEMID	Yes	Yes		No	Yes
SERVICELATENCY	Yes	Yes		No	Yes
SERVICEEXECUTIONTIME ²	Yes	Yes		No	Yes
SERVICEDURATION	Yes	Yes		No	Yes
SERVICEDURATIONSEC	Yes	Yes		No	Yes
SERVICEDOWNTIME	Yes	Yes		No	Yes
SERVICEPERCENTCHANGE	Yes	Yes		No	Yes
SERVICEGROUPNAME	Yes	Yes		No	Yes
SERVICEGROUPNAMES	Yes	Yes		No	Yes
LASTSERVICECHECK	Yes	Yes		No	Yes
LASTSERVICESTATECHANGE	Yes	Yes		No	Yes
LASTSERVICEOK	Yes	Yes		No	Yes
LASTSERVICEWARNING	Yes	Yes		No	Yes
LASTSERVICEUNKNOWN	Yes	Yes		No	Yes
LASTSERVICECRITICAL	Yes	Yes		No	Yes

Table 1.4 – continued from previous page

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers
SERVICEOUTPUT ²	Yes	Yes		No	No
LONGSERVICEOUTPUT ²	Yes	Yes		No	No
SERVICEPERFDATA ²	Yes	Yes		No	No
SERVICECHECKCOMMAND	Yes	Yes		No	No
SERVICEACKAUTHOR ⁸	No	Yes		No	No
SERVICEACKAUTHORNAME ⁸	No	Yes		No	No
SERVICEACKAUTHORALIAS ⁸	No	Yes		No	No
SERVICEACKCOMMENT ⁸	No	Yes		No	No
SERVICEACTIONURL	Yes	Yes		No	No
SERVICENOTESURL	Yes	Yes		No	No
SERVICENOTES	Yes	Yes		No	No

Service Macros

	Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data
Service Group Macros	SERVICE-GROUPALIAS ⁶	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	SERVICE-GROUP-MEMBERS ⁶	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	SERVICE-GROUP-NOTES ⁶	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	SERVICE-GROUP-NOTESURL ⁶	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	SERVICE-GROUPACTIONURL ⁶	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Contact Macros

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
CONTACT-NAME	No	Yes	No	Yes	No	No	No	No
CONTACT-TALIAS	No	Yes	No	Yes	No	No	No	No
CONTACT-EMAIL	No	Yes	No	Yes	No	No	No	No
CONTACT-PAGER	No	Yes	No	Yes	No	No	No	No
CONTACT-ADDRESS _n	No	Yes	No	Yes	No	No	No	No

Contact Group Macros

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data
CONTACT-GROUP-TALIAS ₇	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CONTACT-GROUP-MEMBERS ₇	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Summary Macros

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
TOTALHOSTSUP ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALHOSTS-DOWN ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALHOST-SUNREACH-ABLE ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALHOSTS-DOWNUNHANDLED ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALHOST-SUNREACH-ABLEUNHANDLED ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALHOST-PROBLEMS ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALHOST-PROBLEMSUNHANDLED	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALSERVICE-SOK ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALSERVICESWARNING ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALSERVICE-SCRITICAL ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALSERVICE-SUNKNOWN ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALSERVICESWARNINGUNHANDLED ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALSERVICE-SCRITICALUNHANDLED ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALSERVICE-SUNKNOWNUNHANDLED ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALSERVICEPROBLEMS ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
TOTALSERVICEPROBLEM-SUNHANDLED ₁₀	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes

1.3. User

Notification Macros

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
NOTIFICATIONTYPE	No	Yes	No	Yes	No	No	No	No
NOTIFICATIONRECIPIENTS	No	Yes	No	Yes	No	No	No	No
NOTIFICATIONIS-ESCALATED	No	Yes	No	Yes	No	No	No	No
NOTIFICATIONAUTHOR	No	Yes	No	Yes	No	No	No	No
NOTIFICATIONAUTHORNAME	No	Yes	No	Yes	No	No	No	No
NOTIFICATIONAUTHORALIAS	No	Yes	No	Yes	No	No	No	No
NOTIFICATIONCOMMENT	No	Yes	No	Yes	No	No	No	No
HOSTNOTIFICATIONNUMBER	No	Yes	No	Yes	No	No	No	No
HOSTNOTIFICATIONID	No	Yes	No	Yes	No	No	No	No
SERVICENOTIFICATIONNUMBER	No	Yes	No	Yes	No	No	No	No
SERVICENOTIFICATIONID	No	Yes	No	Yes	No	No	No	No

Date/Time Macros

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
LONG-DATE-TIME	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SHORT-DATE-TIME	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DATE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TIME	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TIMET	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IS-VALID-TIME ⁹	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
NEXTVALID-TIME ⁹	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

	Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
File Macros	MAIN-CONFIG-FILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	STATUS-DATAFILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	COMMENT-DATAFILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	DOWN-TIME-DATAFILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	RETENTION-DATAFILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	OBJECT-CACHEFILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	TEMP-FILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	TEMP-PATH	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	LOGFILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	RE-SOURCE-FILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	COMMAND-FILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	HOST-PERF-DATAFILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	SERVICEPERF-DATAFILE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

	Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Misc Macros	PROCESSTART-TIME	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	EVENTSTART-TIME	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	ADMIN-MAIL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	ADMIN-PAGER	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	ARGn	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	USERn	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Macro Descriptions

HOSTNAME	Short name for the host (i.e. “biglinuxbox”). This value is taken from the host_name directive in the <i>host definition</i> .
HOSTDISPLAYNAME	An alternate display name for the host. This value is taken from the display_name directive in the <i>host definition</i> .
HOSTALIAS	Long name/description for the host. This value is taken from the alias directive in the <i>host definition</i> .
HOSTADDRESS	Address of the host. This value is taken from the address directive in the <i>host definition</i> .
HOSTSTATE	A string indicating the current state of the host (“UP”, “DOWN”, or “UNREACHABLE”).
HOSTSTATEID	A number that corresponds to the current state of the host: 0=UP, 1=DOWN, 2=UNREACHABLE.
LASTHOSTSTATE	A string indicating the last state of the host (“UP”, “DOWN”, or “UNREACHABLE”).
LASTHOSTSTATEID	A number that corresponds to the last state of the host: 0=UP, 1=DOWN, 2=UNREACHABLE.
HOSTSTATETYPE	A string indicating the <i>state type</i> for the current host check (“HARD” or “SOFT”). Soft states occur when the host is in a soft state.
HOSTATTEMPT	The number of the current host check retry. For instance, if this is the second time that the host is being checked.
MAXHOSTATTEMPTS	The max check attempts as defined for the current host. Useful when writing host event handlers for “soft” states.
HOSTEVENTID	A globally unique number associated with the host’s current state. Every time a host (or service) experiences a state change, the event number is incremented.
LASTHOSTEVENTID	The previous (globally unique) event number that was given to the host.
HOSTPROBLEMID	A globally unique number associated with the host’s current problem state. Every time a host (or service) experiences a problem, the problem number is incremented.
LASTHOSTPROBLEMID	The previous (globally unique) problem number that was given to the host. Combined with event handlers, this can be used to track problems.
HOSTLATENCY	A (floating point) number indicating the number of seconds that a scheduled host check lagged behind its expected time.
HOSTEXECUTIONTIME	A (floating point) number indicating the number of seconds that the host check took to execute (i.e. the time from the start of the check to the end).
HOSTDURATION	A string indicating the amount of time that the host has spent in its current state. Format is “XXh YYm ZZs”.
HOSTDURATIONSEC	A number indicating the number of seconds that the host has spent in its current state.
HOSTDOWNTIME	A number indicating the current “downtime depth” for the host. If this host is currently in a period of soft downtime, this macro will be 1. If it is in a period of hard downtime, this macro will be 2. If it is not in a period of downtime, this macro will be 0.
HOSTPERCENTCHANGE	A (floating point) number indicating the percent state change the host has undergone. Percent state change is calculated as: $\frac{\text{current_state} - \text{previous_state}}{\text{previous_state}} \times 100$.
HOSTGROUPNAME	The short name of the hostgroup that this host belongs to. This value is taken from the hostgroup_name directive in the <i>hostgroup definition</i> .
HOSTGROUPNAMES	A comma separated list of the short names of all the hostgroups that this host belongs to.
LASTHOSTCHECK	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the last host check was performed.
LASTHOSTSTATECHANGE	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time the host last changed state.
LASTHOSTUP	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the host last became UP.
LASTHOSTDOWN	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the host last became DOWN.
LASTHOSTUNREACHABLE	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the host last became UNREACHABLE.
HOSTOUTPUT	The first line of text output from the last host check (i.e. “Ping OK”).
LONGHOSTOUTPUT	The full text output (aside from the first line) from the last host check.
HOSTPERFDATA	This macro contains any <i>performance data</i> that may have been returned by the last host check.
HOSTCHECKCOMMAND	This macro contains the name of the command (along with any arguments passed to it) used to perform the last host check.
HOSTACKAUTHOR ⁸	A string containing the name of the user who acknowledged the host problem. This macro is only valid if the host is in a problem state.
HOSTACKAUTHORNAME ⁸	A string containing the short name of the contact (if applicable) who acknowledged the host problem. This macro is only valid if the host is in a problem state.
HOSTACKAUTHORALIAS ⁸	A string containing the alias of the contact (if applicable) who acknowledged the host problem. This macro is only valid if the host is in a problem state.
HOSTACKCOMMENT ⁸	A string containing the acknowledgement comment that was entered by the user who acknowledged the host problem. This macro is only valid if the host is in a problem state.
HOSTACTIONURL	Action URL for the host. This macro may contain other macros (e.g. \$HOSTNAME\$), which can be used to construct a URL.
HOSTNOTESURL	Notes URL for the host. This macro may contain other macros (e.g. \$HOSTNAME\$), which can be used to construct a URL.
HOSTNOTES	Notes for the host. This macro may contain other macros (e.g. \$HOSTNAME\$), which can be used to construct a URL.
TOTALHOSTSERVICES	The total number of services associated with the host.
TOTALHOSTSERVICESOK	The total number of services associated with the host that are in an OK state.
TOTALHOSTSERVICESWARNING	The total number of services associated with the host that are in a WARNING state.
TOTALHOSTSERVICESUNKNOWN	The total number of services associated with the host that are in an UNKNOWN state.
TOTALHOSTSERVICESCRITICAL	The total number of services associated with the host that are in a CRITICAL state.

Host Macros³

Host Group Macros

HOST- GROUPALIAS 5	The long name / alias of either 1) the hostgroup name passed as an on-demand macro argument or 2) the primary hostgroup associated with the current host (if not used in the context of an on-demand macro). This value is taken from the alias directive in the <i>hostgroup definition</i> .
HOSTGROUP- MEMBERS 5	A comma-separated list of all hosts that belong to either 1) the hostgroup name passed as an on-demand macro argument or 2) the primary hostgroup associated with the current host (if not used in the context of an on-demand macro).
HOSTGROUP- NOTES 5	The notes associated with either 1) the hostgroup name passed as an on-demand macro argument or 2) the primary hostgroup associated with the current host (if not used in the context of an on-demand macro). This value is taken from the notes directive in the <i>hostgroup definition</i> .
HOSTGROUP- NOTESURL 5	The notes URL associated with either 1) the hostgroup name passed as an on-demand macro argument or 2) the primary hostgroup associated with the current host (if not used in the context of an on-demand macro). This value is taken from the notes_url directive in the <i>hostgroup definition</i> .
HOST- GROUPACTIONURL 5	The action URL associated with either 1) the hostgroup name passed as an on-demand macro argument or 2) the primary hostgroup associated with the current host (if not used in the context of an on-demand macro). This value is taken from the action_url directive in the <i>hostgroup definition</i> .

SERVICEDESC	The long name/description of the service (i.e. "Main Website"). This value is taken from the service_desc directive in the <i>service definition</i> .
SERVICEDISPLAYNAME	An alternate display name for the service. This value is taken from the display_name directive in the <i>service definition</i> .
SERVICESTATE	A string indicating the current state of the service ("OK", "WARNING", "UNKNOWN", or "CRITICAL").
SERVICESTATEID	A number that corresponds to the current state of the service: 0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN.
LASTSERVICESTATE	A string indicating the last state of the service ("OK", "WARNING", "UNKNOWN", or "CRITICAL").
LASTSERVICESTATEID	A number that corresponds to the last state of the service: 0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN.
SERVICESTATETYPE	A string indicating the <i>state type</i> for the current service check ("HARD" or "SOFT"). Soft states occur when the service is in a warning or critical state but the check is not a hard check.
SERVICEATTEMPT	The number of the current service check retry. For instance, if this is the second time that the service check has failed, the value is 2.
MAXSERVICEATTEMPTS	The max check attempts as defined for the current service. Useful when writing host event handlers.
SERVICEISVOLATILE	Indicates whether the service is marked as being volatile or not: 0 = not volatile, 1 = volatile.
SERVICEEVENTID	A globally unique number associated with the service's current state. Every time a service (or host) changes state, the event ID is incremented.
LASTSERVICEEVENTID	The previous (globally unique) event number that was given to the service.
SERVICEPROBLEMID	A globally unique number associated with the service's current problem state. Every time a service (or host) changes state, the problem ID is incremented.
LASTSERVICEPROBLEMID	The previous (globally unique) problem number that was given to the service. Combined with event ID, this macro can be used to uniquely identify a service's state.
SERVICELATENCY	A (floating point) number indicating the number of seconds that a scheduled service check lagged behind the expected time.
SERVICEEXECUTIONTIME	A (floating point) number indicating the number of seconds that the service check took to execute (from the time the check was scheduled to the time the check returned).
SERVICEDURATION	A string indicating the amount of time that the service has spent in its current state. Format is "XX:YY:ZZ".
SERVICEDURATIONSEC	A number indicating the number of seconds that the service has spent in its current state.
SERVICEDOWNTIME	A number indicating the current "downtime depth" for the service. If this service is currently in a problem state, the value is the number of times the service has failed since the last time it was in a non-problem state.
SERVICEPERCENTCHANGE	A (floating point) number indicating the percent state change the service has undergone. Percent state change is calculated as follows: $\frac{\text{current_state_id} - \text{previous_state_id}}{\text{previous_state_id}} \times 100$.
SERVICEGROUPNAME	The short name of the servicegroup that this service belongs to. This value is taken from the service_group_name directive in the <i>service definition</i> .
SERVICEGROUPNAMES	A comma separated list of the short names of all the servicegroups that this service belongs to.
LASTSERVICECHECK	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the last service check was performed.
LASTSERVICESTATECHANGE	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time the service state last changed.
LASTSERVICEOK	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the service last reached the OK state.
LASTSERVICEWARNING	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the service last reached the WARNING state.
LASTSERVICEUNKNOWN	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the service last reached the UNKNOWN state.
LASTSERVICECRITICAL	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the service last reached the CRITICAL state.
SERVICEOUTPUT	The first line of text output from the last service check (i.e. "Ping OK").
LONGSERVICEOUTPUT	The full text output (aside from the first line) from the last service check.
SERVICEPERFDATA	This macro contains any <i>performance data</i> that may have been returned by the last service check.

Continued on next page

Table 1.6 – continued from previous page

SERVICECHECKCOMMAND	This macro contains the name of the command (along with any arguments passed to it) used to p
SERVICEACKAUTHOR ⁸	A string containing the name of the user who acknowledged the service problem. This macro is
SERVICEACKAUTHORNAME ⁸	A string containing the short name of the contact (if applicable) who acknowledged the service p
SERVICEACKAUTHORALIAS ⁸	A string containing the alias of the contact (if applicable) who acknowledged the service problem
SERVICEACKCOMMENT ⁸	A string containing the acknowledgement comment that was entered by the user who acknowledged
SERVICEACTIONURL	Action URL for the service. This macro may contain other macros (e.g. \$HOSTNAME\$ or \$SE
SERVICENOTESURL	Notes URL for the service. This macro may contain other macros (e.g. \$HOSTNAME\$ or \$SER
SERVICENOTES	Notes for the service. This macro may contain other macros (e.g. \$HOSTNAME\$ or \$SERVICE

Service Macros

Service Group Macros

SERVICE- GROUPALIAS 6	The long name / alias of either 1) the servicegroup name passed as an on-demand macro argument or 2) the primary servicegroup associated with the current service (if not used in the context of an on-demand macro). This value is taken from the alias directive in the <i>servicegroup</i> definition".
SERVICE- GROUPMEM- BERS 6	A comma-separated list of all services that belong to either 1) the servicegroup name passed as an on-demand macro argument or 2) the primary servicegroup associated with the current service (if not used in the context of an on-demand macro).
SERVICE- GROUPNOTES 6	The notes associated with either 1) the servicegroup name passed as an on-demand macro argument or 2) the primary servicegroup associated with the current service (if not used in the context of an on-demand macro). This value is taken from the notes directive in the <i>servicegroup</i> definition.
SERVICE- GROUP- NOTESURL 6	The notes URL associated with either 1) the servicegroup name passed as an on-demand macro argument or 2) the primary servicegroup associated with the current service (if not used in the context of an on-demand macro). This value is taken from the notes_url directive in the <i>servicegroup</i> definition.
SERVICE- GROUPACTIONURL 6	The action URL associated with either 1) the servicegroup name passed as an on-demand macro argument or 2) the primary servicegroup associated with the current service (if not used in the context of an on-demand macro). This value is taken from the action_url directive in the <i>servicegroup</i> definition..

Contact Macros

CONTACT-NAME	Short name for the contact (i.e. "jdoe") that is being notified of a host or service problem. This value is taken from the contact_name directive in the <i>contact definition</i> .
CONTACTALIAS	Long name/description for the contact (i.e. "John Doe") being notified. This value is taken from the alias directive in the <i>contact definition</i> .
CONTACTEMAIL	Email address of the contact being notified. This value is taken from the email directive in the <i>contact definition</i> .
CONTACT-PAGER	Pager number/address of the contact being notified. This value is taken from the pager directive in the <i>contact definition</i> .
CONTACTADDRESSn	Address of the contact being notified. Each contact can have six different addresses (in addition to email address and pager number). The macros for these addresses are \$CONTACTADDRESS1\$ - \$CONTACTADDRESS6\$. This value is taken from the addressx directive in the <i>contact definition</i> .
CONTACT-GROUPNAME	The short name of the contactgroup that this contact is a member of. This value is taken from the contactgroup_name directive in the <i>contactgroup definition</i> . If the contact belongs to more than one contactgroup this macro will contain the name of just one of them.
CONTACT-GROUP-NAMES	A comma separated list of the short names of all the contactgroups that this contact is a member of.

Contact Group Macros

CONTACT-GROUPALIAS ⁷	The long name / alias of either 1) the contactgroup name passed as an on-demand macro argument or 2) the primary contactgroup associated with the current contact (if not used in the context of an on-demand macro). This value is taken from the alias directive in the <i>contactgroup definition</i> .
CONTACT-GROUPMEMBERS ⁷	A comma-separated list of all contacts that belong to either 1) the contactgroup name passed as an on-demand macro argument or 2) the primary contactgroup associated with the current contact (if not used in the context of an on-demand macro).

Summary Macros

TOTALHOSTSUP	This macro reflects the total number of hosts that are currently in an UP state.
TOTALHOSTSDOWN	This macro reflects the total number of hosts that are currently in a DOWN state.
TOTALHOSTSUN-REACHABLE	This macro reflects the total number of hosts that are currently in an UNREACHABLE state.
TOTALHOSTSDOWNUNHANDLED	This macro reflects the total number of hosts that are currently in a DOWN state that are not currently being “handled”. Unhandled host problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
TOTALHOSTSUN-REACHABLEUNHANDLED	This macro reflects the total number of hosts that are currently in an UNREACHABLE state that are not currently being “handled”. Unhandled host problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
TOTALHOSTPROBLEMS	This macro reflects the total number of hosts that are currently either in a DOWN or an UNREACHABLE state.
TOTALHOSTPROBLEMSUNHANDLED	This macro reflects the total number of hosts that are currently either in a DOWN or an UNREACHABLE state that are not currently being “handled”. Unhandled host problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
TOTALSERVICESOK	This macro reflects the total number of services that are currently in an OK state.
TOTALSERVICESWARNING	This macro reflects the total number of services that are currently in a WARNING state.
TOTALSERVICES-CRITICAL	This macro reflects the total number of services that are currently in a CRITICAL state.
TOTALSERVICESUNKNOWN	This macro reflects the total number of services that are currently in an UNKNOWN state.
TOTALSERVICESWARNINGUNHANDLED	This macro reflects the total number of services that are currently in a WARNING state that are not currently being “handled”. Unhandled services problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
TOTALSERVICES-CRITICALUNHANDLED	This macro reflects the total number of services that are currently in a CRITICAL state that are not currently being “handled”. Unhandled services problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
TOTALSERVICESUNKNOWNUNHANDLED	This macro reflects the total number of services that are currently in an UNKNOWN state that are not currently being “handled”. Unhandled services problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
TOTALSERVICEPROBLEMS	This macro reflects the total number of services that are currently either in a WARNING, CRITICAL, or UNKNOWN state.
TOTALSERVICEPROBLEMSUNHANDLED	This macro reflects the total number of services that are currently either in a WARNING, CRITICAL, or UNKNOWN state that are not currently being “handled”. Unhandled services problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.

Notification Macros

NOTIFICATION-TYPE	A string identifying the type of notification that is being sent (“PROBLEM”, “RECOVERY”, “ACKNOWLEDGEMENT”, “FLAPPINGSTART”, “FLAPPINGSTOP”, “FLAPPINGDISABLED”, “DOWNTIMESTART”, “DOWNTIMEEND”, or “DOWNTIMECANCELLED”).
NOTIFICATION-RECIPIENTS	A comma-separated list of the short names of all contacts that are being notified about host or service.
NOTIFICATIONIS-ESCALATED	An integer indicating whether this was sent to normal contacts for the host or service or it was escalated. 0 = Normal (non-escalated) notification , 1 = Escalated notification.
NOTIFICATION-AUTHOR	A string containing the name of the user who authored the notification. If the \$NOTIFICATIONTYPE\$ macro is set to “DOWNTIMESTART” or “DOWNTIMEEND”, this will be the name of the user who scheduled downtime for the host or service. If the \$NOTIFICATIONTYPE\$ macro is “ACKNOWLEDGEMENT”, this will be the name of the user who acknowledged the host or service problem. If the \$NOTIFICATIONTYPE\$ macro is “CUSTOM”, this will be name of the user who initiated the custom host or service notification.
NOTIFICATION-AUTHORNAME	A string containing the short name of the contact (if applicable) specified in the \$NOTIFICATIONAUTHOR\$ macro.
NOTIFICATION-AUTHORALIAS	A string containing the alias of the contact (if applicable) specified in the \$NOTIFICATIONAUTHOR\$ macro.
NOTIFICATION-COMMENT	A string containing the comment that was entered by the notification author. If the \$NOTIFICATIONTYPE\$ macro is set to “DOWNTIMESTART” or “DOWNTIMEEND”, this will be the comment entered by the user who scheduled downtime for the host or service. If the \$NOTIFICATIONTYPE\$ macro is “ACKNOWLEDGEMENT”, this will be the comment entered by the user who acknowledged the host or service problem. If the \$NOTIFICATIONTYPE\$ macro is “CUSTOM”, this will be comment entered by the user who initiated the custom host or service notification.
HOSTNOTIFICATIONNUMBER	The current notification number for the host. The notification number increases by one (1) each time a new notification is sent out for the host (except for acknowledgements). The notification number is reset to 0 when the host recovers (after the recovery notification has gone out). Acknowledgements do not cause the notification number to increase, nor do notifications dealing with flap detection or scheduled downtime.
HOSTNOTIFICATIONID	A unique number identifying a host notification. Notification ID numbers are unique across both hosts and service notifications, so you could potentially use this unique number as a primary key in a notification database. Notification ID numbers should remain unique across restarts of the Centreon Engine process, so long as you have state retention enabled. The notification ID number is incremented by one (1) each time a new host notification is sent out, and regardless of how many contacts are notified.
SERVICENOTIFICATIONNUMBER	The current notification number for the service. The notification number increases by one (1) each time a new notification is sent out for the service (except for acknowledgements). The notification number is reset to 0 when the service recovers (after the recovery notification has gone out). Acknowledgements do not cause the notification number to increase, nor do notifications dealing with flap detection or scheduled downtime.
SERVICENOTIFICATIONID	A unique number identifying a service notification. Notification ID numbers are unique across both hosts and service notifications, so you could potentially use this unique number as a primary key in a notification database. Notification ID numbers should remain unique across restarts of the Centreon Engine process, so long as you have state retention enabled. The notification ID number is incremented by one (1) each time a new service notification is sent out, and regardless of how many contacts are notified.

Date/Time Macros

LONGDATETIME	Current date/time stamp (i.e. Fri Oct 13 00:30:28 C 2000). Format of date is determined by <i>date_format</i> directive.
SHORTDATETIME	Current date/time stamp (i.e. 10-13-2000 00:30:28). Format of date is determined by <i>date_format</i> directive.
DATE	Date stamp (i.e. 10-13-2000). Format of date is determined by <i>date_format</i> directive.
TIME	Current time stamp (i.e. 00:30:28).
TIMET	Current time stamp in time_t format (seconds since UNIX epoch).
ISVALIDTIME ⁹	<p>This is a special on-demand macro that returns a 1 or 0 depending on whether or not a particular time is valid within a specified timeperiod. There are two ways of using this macro:</p> <ul style="list-style-type: none"> • \$ISVALIDTIME:24x7\$ will be set to “1” if the current time is valid within the “24x7” timeperiod. If not, it will be set to “0”. • \$ISVALIDTIME:24x7:timestamp\$ will be set to “1” if the time specified by the “timestamp” argument (which must be in time_t format) is valid within the “24x7” timeperiod. If not, it will be set to “0”.
NEXTVALIDTIME ⁹	<p>This is a special on-demand macro that returns the next valid time (in time_t format) for a specified timeperiod. There are two ways of using this macro:</p> <ul style="list-style-type: none"> • \$NEXTVALIDTIME:24x7\$ will return the next valid time from and including the current time within the “24x7” timeperiod. • \$NEXTVALIDTIME:24x7:timestamp\$ will return the next valid time from and including the time specified by the “timestamp” argument (which must be specified in time_t format) in the “24x7” timeperiod. If a next valid time cannot be found in the specified timeperiod, the macro will be set to “0”.

File Macros	MAINCONFIGFILE	The location of the <i>main config file</i> .
	STATUSDATAFILE	The location of the <i>status data file</i> .
	COMMENTDATAFILE	The location of the comment data file.
	DOWNTIMEDATAFILE	The location of the downtime data file.
	RETENTIONDATAFILE	The location of the <i>retention data file</i> .
	OBJECTCACHEFILE	The location of the <i>object cache file</i> .
	TEMPFILE	The location of the <i>temp file</i> .
	TEMPPATH	The directory specified by the temp path variable.
	LOGFILE	The location of the <i>log file</i> .
	RESOURCEFILE	The location of the <i>resource file</i> .
	COMMANDFILE	The location of the <i>command file</i> .
	HOSTPERFDATAFILE	The location of the host performance data file (if defined).
	SERVICEPERFDATAFILE	The location of the service performance data file (if defined).

Misc Macros	PROCESSSTART-TIME	Time stamp in time_t format (seconds since the UNIX epoch) indicating when the Centreon Engine process was last (re)started. You can determine the number of seconds that Centreon Engine has been running (since it was last restarted) by subtracting \$PROCESSSTARTTIME\$ from TIMET.
	EVENTSTART-TIME	Time stamp in time_t format (seconds since the UNIX epoch) indicating when the Centreon Engine process starting process events (checks, etc.). You can determine the number of seconds that it took for Centreon Engine to startup by subtracting \$PROCESSSTARTTIME\$ from \$EVENTSTARTTIME\$.
	ADMIN-MAIL	Global administrative email address. This value is taken from the <i>admin_email</i> . directive.
	ADMIN-PAGER	Global administrative pager number/address. This value is taken from the <i>admin_pager</i> directive.
	ARGn	The nth argument passed to the command (notification, event handler, service check, etc.). Centreon Engine supports up to 32 argument macros (\$ARG1\$ through \$ARG32\$).
	USERn	The nth user-definable macro. User macros can be defined in one or more <i>resource files</i> . Centreon Engine supports up to 256 user macros (\$USER1\$ through \$USER256\$).

Notes

- ¹ These macros are not valid for the host they are associated with when that host is being checked (i.e. they make no sense, as they haven't been determined yet).
- ² These macros are not valid for the service they are associated with when that service is being checked (i.e. they make no sense, as they haven't been determined yet).
- ³ When host macros are used in service-related commands (i.e. service notifications, event handlers, etc) they refer to the host that the service is associated with.
- ⁴ When host and service summary macros are used in notification commands, the totals are filtered to reflect only those hosts and services for which the contact is authorized (i.e. hosts and services they are configured to receive notifications for).

- ⁵ These macros are normally associated with the first/primary hostgroup associated with the current host. They could therefore be considered host macros in many cases. However, these macros are not available as on-demand host macros. Instead, they can be used as on-demand hostgroup macros when you pass the name of a hostgroup to the macro. For example: `$HOSTGROUPMEMBERS:hg1$` would return a comma-delimited list of all (host) members of the hostgroup hg1.
- ⁶ These macros are normally associated with the first/primary servicegroup associated with the current service. They could therefore be considered service macros in many cases. However, these macros are not available as on-demand service macros. Instead, they can be used as on-demand servicegroup macros when you pass the name of a servicegroup to the macro. For example: `$SERVICEGROUPMEMBERS:sg1$` would return a comma-delimited list of all (service) members of the servicegroup sg1.
- ⁷ These macros are normally associated with the first/primary contactgroup associated with the current contact. They could therefore be considered contact macros in many cases. However, these macros are not available as on-demand contact macros. Instead, they can be used as on-demand contactgroup macros when you pass the name of a contactgroup to the macro. For example: `$CONTACTGROUPMEMBERS:cg1$` would return a comma-delimited list of all (contact) members of the contactgroup cg1.
- ⁸ These acknowledgement macros are deprecated. Use the more generic `$NOTIFICATIONAUTHOR$`, `$NOTIFICATIONAUTHORNAME$`, `$NOTIFICATIONAUTHORALIAS$` or `$NOTIFICATIONAUTHORCOMMENT$` macros instead.
- ⁹ These macro are only available as on-demand macros - e.g. you must supply an additional argument with them in order to use them. These macros are not available as environment variables.
- ¹⁰ Summary macros are not available as environment variables if the `use_large_installation_tweaks` option is enabled, as they are quite CPU-intensive to calculate.

Time Periods

Introduction *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

Precedence in Time Periods *Timeperiod definitions* may contain multiple types of directives, including weekdays, days of the month, and calendar dates. Different types of directives have different precedence levels and may override other directives in your timeperiod definitions. The order of precedence for different types of directives (in descending order) is as follows:

- Calendar date (2008-01-01)
- Specific month date (January 1st)
- Generic month date (Day 15)
- Offset weekday of specific month (2nd Tuesday in December)
- Offset weekday (3rd Monday)
- Normal weekday (Tuesday)

Examples of different timeperiod directives can be found *here*.

How Time Periods Work With Host and Service Checks Host and service definitions have an optional `check_period` directive that allows you to specify a timeperiod that should be used to restrict when regularly scheduled, active checks of the host or service can be made.

If you do not use the `check_period` directive to specify a timeperiod, Centreon Engine will be able to schedule active checks of the host or service anytime it needs to. This is essentially a 24x7 monitoring scenario.

Specifying a timeperiod in the `check_period` directive allows you to restrict the time that Centreon Engine perform regularly scheduled, active checks of the host or service. When Centreon Engine attempts to reschedule a host or service check, it will make sure that the next check falls within a valid time range within the defined timeperiod. If it doesn't, Centreon Engine will adjust the next check time to coincide with the next "valid" time in the specified timeperiod. This means that the host or service may not get checked again for another hour, day, or week, etc.

Note: On-demand checks and passive checks are not restricted by the timeperiod you specify in the `check_period` directive. Only regularly scheduled active checks are restricted.

Unless you have a good reason not to do so, I would recommend that you monitor all your hosts and services using timeperiods that cover a 24x7 time range. If you don't do this, you can run into some problems during "blackout" times (times that are not valid in the timeperiod definition):

1. The status of the host or service will appear unchanged during the blackout time.
2. Contacts will mostly likely not get re-notified of problems with a host or service during blackout times.
3. If a host or service recovers during a blackout time, contacts will not be immediately notified of the recovery.

How Time Periods Work With Contact Notifications By specifying a timeperiod in the `notification_period` directive of a host or service definition, you can control when Centreon Engine is allowed to send notifications out regarding problems or recoveries for that host or service. When a host notification is about to get sent out, Centreon Engine will make sure that the current time is within a valid range in the `notification_period` timeperiod. If it is a valid time, then Centreon Engine will attempt to notify each contact of the problem or recovery.

You can also use timeperiods to control when notifications can be sent out to individual contacts. By using the `service_notification_period` and `host_notification_period` directives in *contact definitions*, you're able to essentially define an "on call" period for each contact. Contacts will only receive host and service notifications during the times you specify in the notification period directives.

Examples of how to create timeperiod definitions for use for on-call rotations can be found [here](#).

How Time Periods Work With Notification Escalations Service and host notification escalations have an optional `escalation_period` directive that allows you to specify a timeperiod when the escalation is valid and can be used. If you do not use the `escalation_period` directive in an escalation definition, the escalation is considered valid at all times. If you specify a timeperiod in the `escalation_period` directive, Centreon Engine will only use the escalation definition during times that are valid in the timeperiod definition.

How Time Periods Work With Dependencies By specifying a timeperiod in the `notification_period` directive of a host or service definition, you can control when Centreon Engine is allowed to send notifications out regarding problems or recoveries for that host or service. When a host notification is about to get sent out, Centreon Engine will make sure that the current time is within a valid range in the `notification_period` timeperiod. If it is a valid time, then Centreon Engine will attempt to notify each contact of the problem or recovery.

You can also use timeperiods to control when notifications can be sent out to individual contacts. By using the `service_notification_period` and `host_notification_period` directives in contact definitions, you're able to essentially define an "on call" period for each contact. Contacts will only receive host and service notifications during the times you specify in the notification period directives.

Examples of how to create timeperiod definitions for use for on-call rotations can be found [here](#).

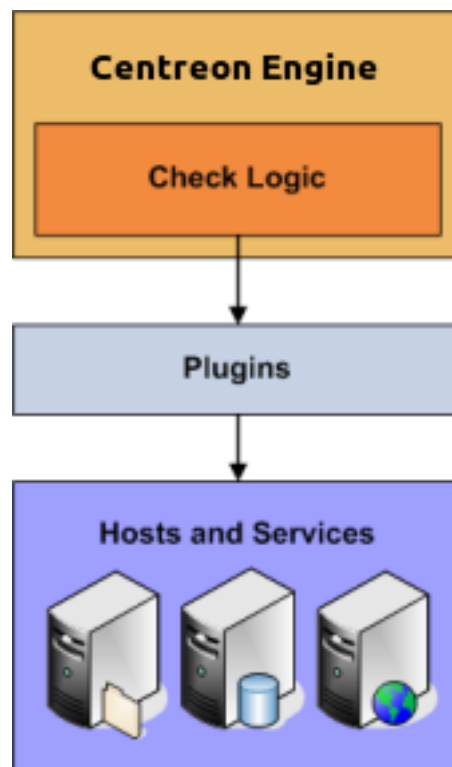
How Time Periods Work With Notification Escalations Service and host *notification escalations* have an optional `escalation_period` directive that allows you to specify a timeperiod when the escalation is valid and can be used. If you do not use the `escalation_period` directive in an escalation definition, the escalation is considered valid at all times. If you specify a timeperiod in the `escalation_period` directive, Centreon Engine will only use the escalation definition during times that are valid in the timeperiod definition.

How Time Periods Work With Dependencies Service and host *dependencies* have an optional `dependency_period` directive that allows you to specify a timeperiod when the dependencies are valid and can be used. If you do not use the `dependency_period` directive in a dependency definition, the dependency can be used at any time. If you specify a timeperiod in the `dependency_period` directive, Centreon Engine will only use the dependency definition during times that are valid in the timeperiod definition.

Active Checks

Introduction Centreon Engine is capable of monitoring hosts and services in two ways: actively and passively. Passive checks are described *elsewhere*, so we'll focus on active checks here. Active checks are the most common method for monitoring hosts and services. The main features of active checks are as follows:

- Active checks are initiated by the Centreon Engine 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 Centreon Engine daemon. When Centreon Engine 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 Centreon Engine daemon. Centreon Engine will process the results of the host or service check and take appropriate action as necessary (e.g. send notifications, run event handlers, etc).

More information on how plugins work can be found *here*.

When Are Active Checks Executed?

Active check are executed:

- At regular intervals, as defined by the `check_interval` and `retry_interval` options in your host and service definitions
- On-demand as needed

Regularly scheduled checks occur at intervals equaling either the `check_interval` or the `retry_interval` in your host or service definitions, depending on what *type of state* the host or service is in. If a host or service is in a **HARD** state, it will be actively checked at intervals equal to the `check_interval` option. If it is in a **SOFT** state, it will be checked at intervals equal to the `retry_interval` option.

On-demand checks are performed whenever Centreon Engine sees a need to obtain the latest status information about a particular host or service. For example, when Centreon Engine is determining the *reachability* of a host, it will often perform on-demand checks of parent and child hosts to accurately determine the status of a particular network segment. On-demand checks also occur in the *predictive dependency check* logic in order to ensure Centreon Engine has the most accurate status information.

Passive Checks

Introduction In most cases you'll use Centreon Engine to monitor your hosts and services using regularly scheduled *active checks*. Active checks can be used to "poll" a device or service for status information every so often. Centreon Engine also supports a way to monitor hosts and services passively instead of actively. The key features of passive checks are as follows:

- Passive checks are initiated and performed external applications/processes
- Passive check results are submitted to Centreon Engine for processing

The major difference between active and passive checks is that active checks are initiated and performed by Centreon Engine, while passive checks are performed by external applications.

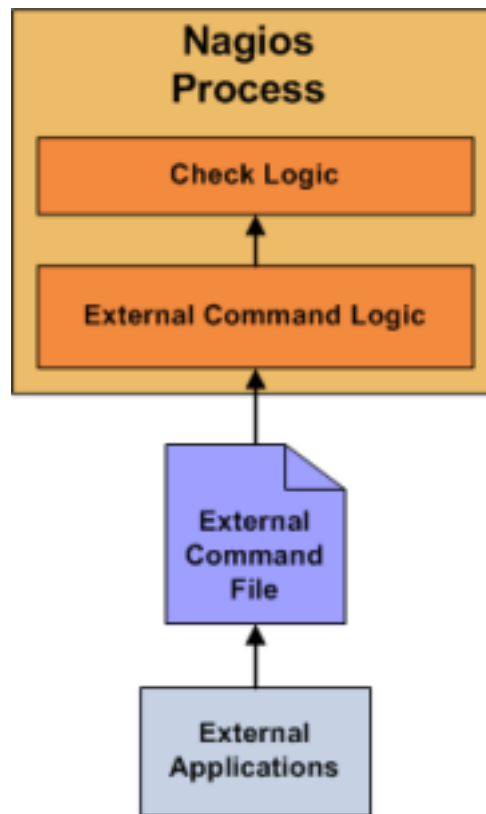
Uses For Passive Checks

Passive checks are useful for monitoring services that are:

- Asynchronous in nature and cannot be monitored effectively by polling their status on a regularly scheduled basis
- Located behind a firewall and cannot be checked actively from the monitoring host

Examples of asynchronous services that lend themselves to being monitored passively include SNMP traps and security alerts. You never know how many (if any) traps or alerts you'll receive in a given time frame, so it's not feasible to just monitor their status every few minutes.

Passive checks are also used when configuring *distributed* or *redundant* monitoring installations.



How Passive Checks Work

Here's how passive checks work in more detail...

- An external application checks the status of a host or service.
- The external application writes the results of the check to the *external command file*.
- The next time Centreon Engine reads the external command file it will place the results of all passive checks into a queue for later processing. The same queue that is used for storing results from active checks is also used to store the results from passive checks.
- Centreon Engine will periodically execute a *check result reaper event* and scan the check result queue. Each service check result that is found in the queue is processed in the same manner
- regardless of whether the check was active or passive. Centreon Engine may send out notifications, log alerts, etc. depending on the check result information.

The processing of active and passive check results is essentially identical. This allows for seamless integration of status information from external applications with Centreon Engine.

Enabling Passive Checks In order to enable passive checks in Centreon Engine, you'll need to do the following:

- Set *accept_passive_service_checks* directive to 1.
- Set the *passive_checks_enabled* directive in your host and service definitions to 1.

If you want to disable processing of passive checks on a global basis, set the *accept_passive_service_checks* directive to 0.

If you would like to disable passive checks for just a few hosts or services, use the *passive_checks_enabled* directive in the host and/or service definitions to do so.

Submitting Passive Service Check Results External applications can submit passive service check results to Centreon Engine by writing a `PROCESS_SERVICE_CHECK_RESULT` *external command* to the external command file.

The format of the command is as follows:

```
[<timestamp>] PROCESS_SERVICE_CHECK_RESULT;<host_name>;<svc_description>;<return_code>;<plugin_output>
```

where...

- `timestamp` is the time in `time_t` format (seconds since the UNIX epoch) that the service check was performed (or submitted). Please note the single space after the right bracket.
- `host_name` is the short name of the host associated with the service in the service definition
- `svc_description` is the description of the service as specified in the service definition
- `return_code` is the return code of the check (0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN)
- `plugin_output` is the text output of the service check (i.e. the plugin output)

Note: A service must be defined in Centreon Engine before you can submit passive check results for it! Centreon Engine will ignore all check results for services that had not been configured before it was last (re)started. An example shell script of how to submit passive service check results to Centreon Engine can be found in the documentation on *volatile services*.

Submitting Passive Host Check Results External applications can submit passive host check results to Centreon Engine by writing a `PROCESS_HOST_CHECK_RESULT` *external command* to the external command file.

The format of the command is as follows:

```
[<timestamp>] PROCESS_HOST_CHECK_RESULT;<host_name>;<host_status>;<plugin_output>
```

where...

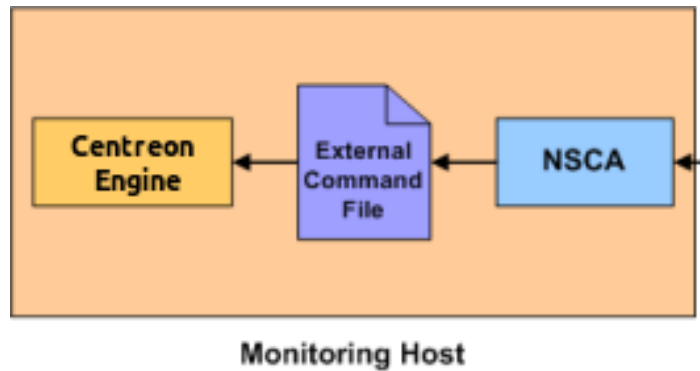
- `timestamp` is the time in `time_t` format (seconds since the UNIX epoch) that the host check was performed (or submitted). Please note the single space after the right bracket.
- `host_name` is the short name of the host (as defined in the host definition)
- `host_status` is the status of the host (0=UP, 1=DOWN, 2=UNREACHABLE)
- `plugin_output` is the text output of the host check

Note: A host must be defined in Centreon Engine before you can submit passive check results for it! Centreon Engine will ignore all check results for hosts that had not been configured before it was last (re)started.

Passive Checks and Host States Unlike with active host checks, Centreon Engine does not (by default) attempt to determine whether or host is DOWN or UNREACHABLE with passive checks. Rather, Centreon Engine takes the passive check result to be the actual state the host is in and doesn't try to determine the host's actual state using the *reachability logic*. This can cause problems if you are submitting passive checks from a remote host or you have a *distributed monitoring setup* where the parent/child host relationships are different.

You can tell Centreon Engine to translate DOWN/UNREACHABLE passive check result states to their “proper” state by using the `translate_passive_host_checks` variable. More information on how this works can be found [here](#).

Note: Passive host checks are normally treated as *HARD states*, unless the `passive_host_checks_are_soft` option is enabled.



Submitting Passive Check Results From Remote Hosts

If an application that resides on the same host as Centreon Engine is sending passive host or service check results, it can simply write the results directly to the external command file as outlined above. However, applications on remote hosts can't do this so easily.

In order to allow remote hosts to send passive check results to the monitoring host, I've developed the *NSCA* addon". The NSCA addon consists of a daemon that runs on the Centreon Engine hosts and a client that is executed from remote hosts. The daemon will listen for connections from remote clients, perform some basic validation on the results being submitted, and then write the check results directly into the external command file (as described above). More information on the NSCA addon can be found [here](#).

Host Checks

Introduction The basic workings of host checks are described here...

When Are Host Checks Performed? Hosts are checked by the Centreon Engine 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.

Regularly scheduled host checks are optional. If you set the `check_interval` option in your host definition to zero (0), Centreon Engine will not perform checks of the hosts on a regular basis. It will, however, still perform on-demand checks of the host as needed for other parts of the monitoring logic.

On-demand checks are made when a service associated with the host changes state because Centreon Engine needs to know whether the host has also changed state. Services that change state are often an indicator that the host may have also changed state. For example, if Centreon Engine detects that the HTTP service associated with a host just changed from a CRITICAL to an OK state, it may indicate that the host just recovered from a reboot and is now back up and running.

On-demand checks of hosts are also made as part of the *host reachability* logic. Centreon Engine is designed to detect network outages as quickly as possible, and distinguish between DOWN and UNREACHABLE host states. These are very different states and can help an admin quickly locate the cause of a network outage.

On-demand checks are also performed as part of the *predictive host dependency check* logic. These checks help ensure that the dependency logic is as accurate as possible.

Cached Host Checks The performance of on-demand host checks can be significantly improved by implementing the use of cached checks, which allow Centreon Engine to forgo executing a host check if it determines a relatively recent check result will do instead. More information on cached checks can be found *here*.

Dependencies and Checks You can define *host* execution dependencies” that prevent Centreon Engine from checking the status of a host depending on the state of one or more other hosts. More information on dependencies can be found *here*.

Parallelization of Host Checks

Scheduled host checks are run in parallel. When Centreon Engine needs to run a scheduled host check, it will initiate the host check and then return to doing other work (running service checks, etc). The host check runs in a child process that was fork()ed from the main Centreon Engine daemon. When the host check has completed, the child process will inform the main Centreon Engine process (its parent) of the check results. The main Centreon Engine process then handles the check results and takes appropriate action (running event handlers, sending notifications, etc.).

On-demand host checks are also run in parallel if needed. As mentioned earlier, Centreon Engine can forgo the actual execution of an on-demand host check if it can use the cached results from a relatively recent host check.

When Centreon Engine processes the results of scheduled and on-demand host checks, it may initiate (secondary) checks of other hosts. These checks can be initiated for two reasons: *predictive dependency checks* and to determining the status of the host using the *network reachability* logic. The secondary checks that are initiated are usually run in parallel. However, there is one big exception that you should be aware of, as it can have negative effect on performance...

Note: Hosts which have their `max_check_attempts` value set to 1 can cause serious performance problems. The reason? If Centreon Engine needs to determine their true state using the *network reachability* logic (to see if they’re DOWN or UNREACHABLE), it will have to launch serial checks of all of the host’s immediate parents. Just to reiterate, those checks are run serially, rather than in parallel, so it can cause a big performance hit. For this reason, I would recommend that you always use a value greater than 1 for the `max_check_attempts` directives in your host definitions.

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. How does Centreon Engine translate these plugin return codes into host states of UP, DOWN, or UNREACHABLE? Lets see...

The table below shows how plugin return codes correspond with preliminary host states. Some post-processing (which is described later) is done which may then alter the final host state.

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

Note: WARNING results usually means the host is UP. However, WARNING results are interpreted to mean the host is DOWN if the `use_aggressive_host_checking` option is enabled.

If the preliminary host state is DOWN, Centreon Engine will attempt to see if the host is really DOWN or if it is UNREACHABLE. The distinction between DOWN and UNREACHABLE host states is important, as it allows admins to determine root cause of network outages faster. The following table shows how Centreon Engine makes a final state determination based on the state of the hosts parent(s). A host's parents are defined in the parents directive in host definition.

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

More information on how Centreon Engine distinguishes between DOWN and UNREACHABLE states can be found [here](#).

Host State Changes As you are probably well aware, hosts don't always stay in one state. Things break, patches get applied, and servers need to be rebooted. When Centreon Engine checks the status of hosts, it will be able to detect when a host changes between UP, DOWN, and UNREACHABLE states and take appropriate action. These state changes result in different *state types* (HARD or SOFT), which can trigger *event handlers* to be run and *notifications* to be sent out. Detecting and dealing with state changes is what Centreon Engine is all about.

When hosts change state too frequently they are considered to be “flapping”. A good example of a flapping host would be server that keeps spontaneously rebooting as soon as the operating system loads. That's always a fun scenario to have to deal with. Centreon Engine can detect when hosts start flapping, and can suppress notifications until flapping stops and the host's state stabilizes. More information on the flap detection logic can be found [here](#).

Service Checks

Introduction The basic workings of service checks are described here...

When Are Service Checks Performed? Services are checked by the Centreon Engine 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*.

On-demand checks are performed as part of the *predictive service dependency check* logic. These checks help ensure that the dependency logic is as accurate as possible. If you don't make use of *service dependencies*, Centreon Engine won't perform any on-demand service checks.

Cached Service Checks The performance of on-demand service checks can be significantly improved by implementing the use of cached checks, which allow Centreon Engine to forgo executing a service check if it determines a relatively recent check result will do instead. Cached checks will only provide a performance increase if you are making use of *service dependencies*. More information on cached checks can be found [here](#).

Dependencies and Checks You can define *service execution dependencies* that prevent Centreon Engine from checking the status of a service depending on the state of one or more other services. More information on dependencies can be found [here](#).

Parallelization of Service Checks Scheduled service checks are run in parallel. When Centreon Engine needs to run a scheduled service check, it will initiate the service check and then return to doing other work (running host checks, etc). The service check runs in a child process that was fork(ed) from the main Centreon Engine daemon. When the service check has completed, the child process will inform the main Centreon Engine process (its parent)

of the check results. The main Centreon Engine process then handles the check results and takes appropriate action (running event handlers, sending notifications, etc.).

On-demand service checks are also run in parallel if needed. As mentioned earlier, Centreon Engine can forgo the actual execution of an on-demand service check if it can use the cached results from a relatively recent service check.

Service States Services that are checked can be in one of four different states:

- OK
- WARNING
- UNKNOWN
- CRITICAL

Service State Determination Service checks are performed by *plugins*, which can return a state of OK, WARNING, UNKNOWN, or CRITICAL. These plugin states directly translate to service states. For example, a plugin which returns a WARNING state will cause a service to have a WARNING state.

Services State Changes When Centreon Engine checks the status of services, it will be able to detect when a service changes between OK, WARNING, UNKNOWN, and CRITICAL states and take appropriate action. These state changes result in different *state types* (HARD or SOFT), which can trigger *event handlers* to be run and *notifications* to be sent out. Service state changes can also trigger on-demand *host checks*. Detecting and dealing with state changes is what Centreon Engine is all about.

When services change state too frequently they are considered to be “flapping”. Centreon Engine can detect when services start flapping, and can suppress notifications until flapping stops and the service’s state stabilizes. More information on the flap detection logic can be found *here*.

Notifications



Introduction

I’ve had a lot of questions as to exactly how notifications work. This will attempt to explain exactly when and how host and service notifications are sent out, as well as who receives them.

Notification escalations are explained *here*.

When Do Notifications Occur? The decision to send out notifications is made in the service check and host check logic. Host and service notifications occur in the following instances...

- When a hard state change occurs. More information on state types and hard state changes can be found *here*.
- When a host or service remains in a hard non-OK state and the time specified by the <notification_interval> option in the host or service definition has passed since the last notification was sent out (for that specified host or service).

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.

When Centreon Engine sends out a host or service notification, it will notify each contact that is a member of any contact groups specified in the `<contactgroups>` option of the service definition. Centreon Engine realizes that a contact may be a member of more than one contact group, so it removes duplicate contact notifications before it does anything.

What Filters Must Be Passed In Order For Notifications To Be Sent? Just because there is a need to send out a host or service notification doesn't mean that any contacts are going to get notified. There are several filters that potential notifications must pass before they are deemed worthy enough to be sent out. Even then, specific contacts may not be notified if their notification filters do not allow for the notification to be sent to them. Let's go into the filters that have to be passed in more detail...

Program-Wide Filter: The first filter that notifications must pass is a test of whether or not notifications are enabled on a program-wide basis. This is initially determined by the `enable_notifications` directive in the main config file, but may be changed during runtime from the web interface. If notifications are disabled on a program-wide basis, no host or service notifications can be sent out - period. If they are enabled on a program-wide basis, there are still other tests that must be passed...

Service and Host Filters The first filter for host or service notifications is a check to see if the host or service is in a period of *scheduled downtime*. If it is in a scheduled downtime, no one gets notified. If it isn't in a period of downtime, it gets passed on to the next filter. As a side note, notifications for services are suppressed if the host they're associated with is in a period of scheduled downtime.

The second filter for host or service notification is a check to see if the host or service is *flapping* (if you enabled flap detection). If the service or host is currently flapping, no one gets notified. Otherwise it gets passed to the next filter.

The third host or service filter that must be passed is the host- or service-specific notification options. Each service definition contains options that determine whether or not notifications can be sent out for warning states, critical states, and recoveries. Similarly, each host definition contains options that determine whether or not notifications can be sent out when the host goes down, becomes unreachable, or recovers. If the host or service notification does not pass these options, no one gets notified. If it does pass these options, the notification gets passed to the next filter... Note: Notifications about host or service recoveries are only sent out if a notification was sent out for the original problem. It doesn't make sense to get a recovery notification for something you never knew was a problem.

The fourth host or service filter that must be passed is the time period test. Each host and service definition has a `<notification_period>` option that specifies which time period contains valid notification times for the host or service. If the time that the notification is being made does not fall within a valid time range in the specified time period, no one gets contacted. If it falls within a valid time range, the notification gets passed to the next filter... Note: If the time period filter is not passed, Centreon Engine will reschedule the next notification for the host or service (if its in a non-OK state) for the next valid time present in the time period. This helps ensure that contacts are notified of problems as soon as possible when the next valid time in time period arrives.

The last set of host or service filters is conditional upon two things:

(1) a notification was already sent out about a problem with the host or service at some point in the past and (2) the host or service has remained in the same non-OK state that it was when the last notification went out. If these two criteria are met, then Centreon Engine will check and make sure the time that has passed since the last notification went out either meets or exceeds the value specified by the `<notification_interval>` option in the host or service definition. If not enough time has passed since the last notification, no one gets contacted. If either enough time has passed since the last notification or the two criteria for this filter were not met, the notification will be sent out! Whether or not it actually is sent to individual contacts is up to another set of filters...

Contact Filters At this point the notification has passed the program mode filter and all host or service filters and Centreon Engine starts to notify *all the people it should*". Does this mean that each contact is going to receive the notification? No! Each contact has their own set of filters that the notification must pass before they receive it. Note: Contact filters are specific to each contact and do not affect whether or not other contacts receive notifications.

The first filter that must be passed for each contact are the notification options. Each contact definition contains options that determine whether or not service notifications can be sent out for warning states, critical states, and recoveries. Each contact definition also contains options that determine whether or not host notifications can be sent out when the host goes down, becomes unreachable, or recovers. If the host or service notification does not pass these options, the contact will not be notified. If it does pass these options, the notification gets passed to the next filter... Note: Notifications about host or service recoveries are only sent out if a notification was sent out for the original problem. It doesn't make sense to get a recovery notification for something you never knew was a problem...

The last filter that must be passed for each contact is the time period test. Each contact definition has a `<notification_period>` option that specifies which time period contains valid notification times for the contact. If the time that the notification is being made does not fall within a valid time range in the specified time period, the contact will not be notified. If it falls within a valid time range, the contact gets notified!

Notification Methods You can have Centreon Engine notify you of problems and recoveries pretty much anyway you want: pager, cellphone, email, instant message, audio alert, electric shocker, etc. How notifications are sent depend on the *notification commands* that are defined in your *object definition files*.

Note: If you install Centreon Engine according to the *quickstart guide*, it should be configured to send email notifications. You can see the email notification commands that are used by viewing the contents of the following file: `/etc/centreon-engine/objects/commands.cfg`.

Specific notification methods (paging, etc.) are not directly incorporated into the Centreon Engine code as it just doesn't make much sense. The "core" of Centreon Engine is not designed to be an all-in-one application. If service checks were embedded in Centreon Engine's core it would be very difficult for users to add new check methods, modify existing checks, etc. Notifications work in a similar manner. There are a thousand different ways to do notifications and there are already a lot of packages out there that handle the dirty work, so why re-invent the wheel and limit yourself to a bike tire? It's much easier to let an external entity (i.e. a simple script or a full-blown messaging system) do the messy stuff. Some messaging packages that can handle notifications for pagers and cellphones are listed below in the resource section.

Notification Type Macro When crafting your notification commands, you need to take into account what type of notification is occurring. The `NOTIFICATIONTYPE` macro contains a string that identifies exactly that. The table below lists the possible values for the macro and their respective descriptions:

Value	Description
PROBLEM	A service or host has just entered (or is still in) a problem state. If this is a service notification, it means the service is either in a WARNING, UNKNOWN or CRITICAL state. If this is a host notification, it means the host is in a DOWN or UNREACHABLE state.
RECOVERY	A service or host recovery has occurred. If this is a service notification, it means the service has just returned to an OK state. If it is a host notification, it means the host has just returned to an UP state.
ACKNOWLEDGE- MENT	This notification is an acknowledgement notification for a host or service problem. Acknowledgement notifications are initiated via the web interface by contacts for the particular host or service.
FLAP- PINGSTART	The host or service has just started <i>Detection and Handling of State Flapping</i>
FLAP- PINGSTOP	The host or service has just stopped <i>Detection and Handling of State Flapping</i>
FLAP- PINGDIS- ABLED	The host or service has just stopped <i>Detection and Handling of State Flapping</i>
DOWNTIME- START	The host or service has just entered a period of <i>Scheduled Downtime</i>
DOWN- TIMESTOP	The host or service has just exited from a period of <i>Scheduled Downtime</i>
DOWNTIME- CANCELLED	The period of <i>Scheduled Downtime</i>

Helpful Resources There are many ways you could configure Centreon Engine to send notifications out. Its up to you to decide which method(s) you want to use. Once you do that you'll have to install any necessary software and configure notification commands in your config files before you can use them. Here are just a few possible notification methods:

- Email
- Pager
- Phone (SMS)
- WinPopup message
- Yahoo, ICQ, or MSN instant message
- Audio alerts
- etc...

Basically anything you can do from a command line can be tailored for use as a notification command.

If you're looking for an alternative to using email for sending messages to your pager or cellphone, check out these packages. They could be used in conjunction with Centreon Engine to send out a notification via a modem when a problem

arises. That way you don't have to rely on email to send notifications out (remember, email may *not* work if

there are network problems). I haven't actually tried these packages myself, but others have reported success using them...

- [Gnokii](#) (SMS software for contacting Nokia phones via GSM network)
- [QuickPage](#) (alphanumeric pager software)
- [Sendpage](#) (paging software)
- [SMS Client](#) (command line utility for sending messages to pagers and mobile phones)

If you want to try out a non-traditional method of notification, you might want to mess around with audio alerts. If you want to have audio alerts played on the monitoring server (with synthesized speech), check out [Festival](#). If you'd rather leave the monitoring box alone and have audio alerts played on another box, check out the [Network Audio System \(NAS\)](#) and [rplay](#) projects.

State Types

Introduction The current state of monitored services and hosts is determined by two components:

- The status of the service or host (i.e. OK, WARNING, UP, DOWN, etc.)
- The type of state the service or host is in

There are two state types in Centreon Engine - SOFT states and HARD states. These state types are a crucial part of the monitoring logic, as they are used to determine when *event handlers* are executed and when *notifications* are initially sent out.

This document describes the difference between SOFT and HARD states, how they occur, and what happens when they occur.

Service and Host Check Retries In order to prevent false alarms from transient problems, Centreon Engine allows you to define how many times a service or host should be (re)checked before it is considered to have a “real” problem. This is controlled by the `max_check_attempts` option in the host and service definitions. Understanding how hosts and services are (re)checked in order to determine if a real problem exists is important in understanding how state types work.

Soft States Soft states occur in the following situations...

- 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.
- 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.

SOFT states are only logged if you enabled the `log_service_retries` or `log_host_retries` options in your main configuration file.

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.

The `HOSTSTATETYPE` or `SERVICESTATETYPE` macros will have a value of “SOFT” when event handlers are executed, which allows your event handler scripts to know when they should take corrective action. More information on event handlers can be found [here](#).

Hard States Hard states occur for hosts and services in the following situations:

- When a host or service check results in a non-UP or non-OK state and it has been (re)checked the number of times specified by the `max_check_attempts` option in the host or service definition. This is a hard error state.
- When a host or service transitions from one hard error state to another error state (e.g. WARNING to CRITICAL).
- When a service check results in a non-OK state and its corresponding host is either DOWN or UNREACHABLE.

- When a host or service recovers from a hard error state. This is considered to be a hard recovery.
- When a *passive host check* is received. Passive host checks are treated as HARD unless the *passive_host_checks_are_soft* option is enabled.

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

- The HARD state is logged.
- Event handlers are executed to handle the HARD state.
- Contacts are notified of the host or service problem or recovery.

The *HOSTSTATETYPE* or *SERVICESTATETYPE* macros will have a value of “HARD” when event handlers are executed, which allows your event handler scripts to know when they should take corrective action. More information on event handlers can be found *here*.

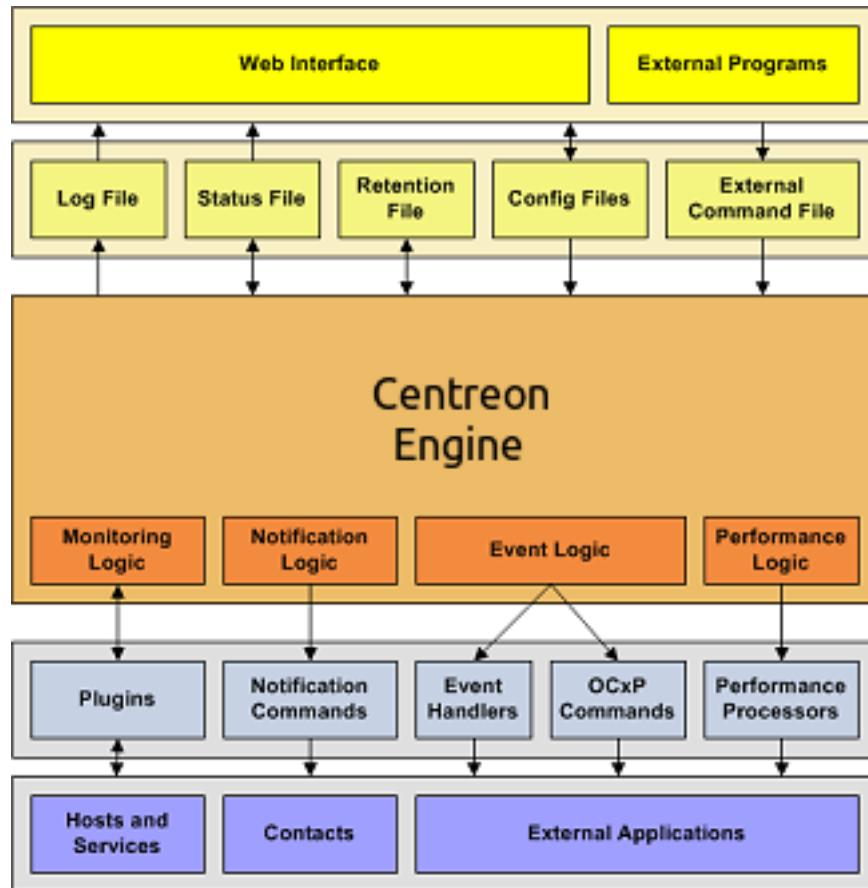
Example Here’s an example of how state types are determined, when state changes occur, and when event handlers and notifications are sent out. The table below shows consecutive checks of a service over time. The service has a *max_check_attempts* value of 3.

Time	Check	State	State Type	State Change	Notes
0	1	OK	HARD	No	Initial state of the service.
1	1	CRITICAL	SOFT	Yes	First detection of a non-OK state. Event handlers execute.
2	2	WARNING	SOFT	Yes	Service continues to be in a non-OK state. Event handlers execute.
3	3	CRITICAL	HARD	Yes	Max check attempts has been reached, so service goes into a HARD state. Event handlers execute and a problem notification is sent out. Check is reset to 1 immediately after this happens.
4	1	WARNING	HARD	Yes	Service changes to a HARD WARNING state. Event handlers execute and a problem notification is sent out.
5	1	WARNING	HARD	No	Service stabilizes in a HARD problem state. Depending on what the notification interval for the service is, another notification might be sent out.
6	1	OK	HARD	Yes	Service experiences a HARD recovery. Event handlers execute and a recovery notification is sent out.
7	1	OK	HARD	No	Service is still OK.
8	1	UNKNOWN	SOFT	Yes	Service is detected as changing to a SOFT non-OK state. Event handlers execute.
9	2	OK	SOFT	Yes	Service experiences a SOFT recovery. Event handlers execute, but notification are not sent, as this wasn’t a “real” problem. State type is set HARD and check is reset to 1 immediately after this happens.
10	1	OK	HARD	No	Service stabilizes in an OK state.

Integration Overview

Introduction One of the reasons that Centreon Engine is such a popular monitoring application is the fact that it can be easily integrated in your existing infrastructure. There are several methods of integrating Centreon Engine with the management software you’re already using and you can monitor almost any type of new or custom hardware, service, or application that you might have.

Integration Points



To monitor new hardware, services, or applications, check out the docs on:

- *Plugins*
- *Plugin API*
- *Passive Checks*
- *Event Handlers*

To get data into Centreon Engine from external applications, check out the docs on:

- *Passive Checks*
- *External Commands*

To send status, performance, or notification information from Centreon Engine to external applications, check out the docs on:

- *Event Handlers*
- *OCSP and OCHP Commands*
- *Performance Data*
- *Notifications*

Integration Examples

I've documented some examples on how to integrate Centreon Engine with external applications:

- *TCP Wrappers* (security alerts)

- *SNMP Traps* (Arcserve backup job status)

Addons

Introduction There are a lot of “addon” software packages that are available for Centreon Engine. Addons can be used to extend Centreon Engine’ functionality or integrate Centreon Engine with other applications.

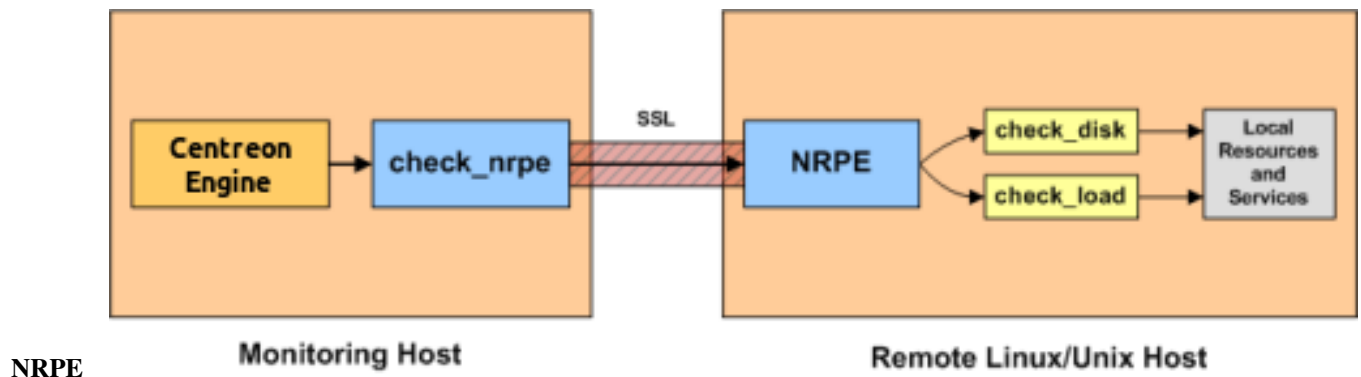
Addons are available for:

- Managing the config files through a web interface
- Monitoring remote hosts (NIX, Windows ...)
- Submitting passive checks from remote hosts
- Simplifying/extending the notification logic
- ...and much more

You can find many addons for Centreon Engine by visiting:

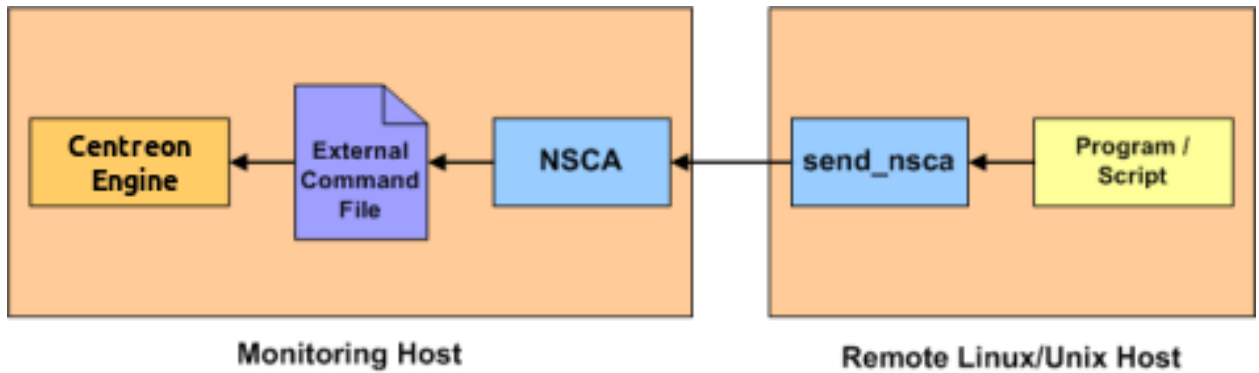
- nagios.org
- Nagios’ SourceForge.net page
- nagiosexchange.org

I’ll give a brief introduction to a few of the addons that I’ve developed for Centreon Engine...



NRPE is an addon that allows you to execute *plugins* on remote Linux/Unix hosts. This is useful if you need to monitor local resources/attributes like disk usage, CPU load, memory usage, etc. on a remote host. Similiar functionality can be accomplished by using the *check_by_ssh* plugin, although it can impose a higher CPU load on the monitoring machine - especially if you are monitoring hundreds or thousands of hosts.

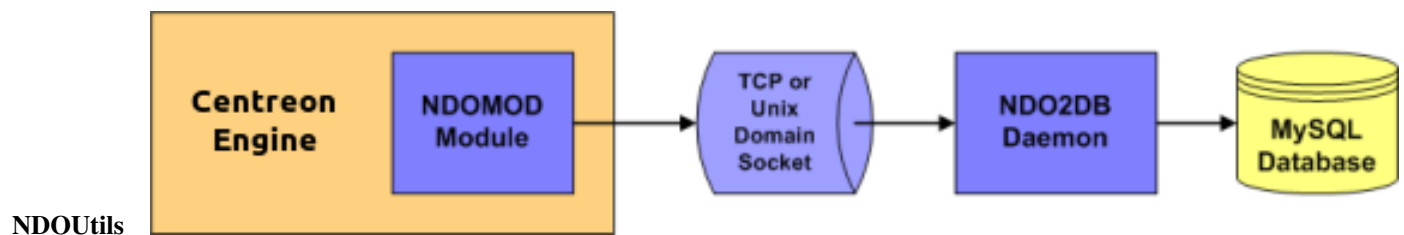
The NRPE addon and documentation can be found at <http://www.nagios.org/>.



NSCA

NSCA is an addon that allows you to send *passive check* results from remote Linux/Unix hosts to the Centreon Engine daemon running on the monitoring server. This is very useful in *distributed* and *redundant/failover* monitoring setups.

The NSCA addon can be found at <http://www.nagios.org/>.



NDOUtils

NDOUtils is an addon that allows you to store all status information from Centreon Engine in a MySQL database. Multiple instances of Centreon Engine can all store their information in a central database for centralized reporting. This will likely serve as the basis for a new PHP-based web interface for Centreon Engine in the future.

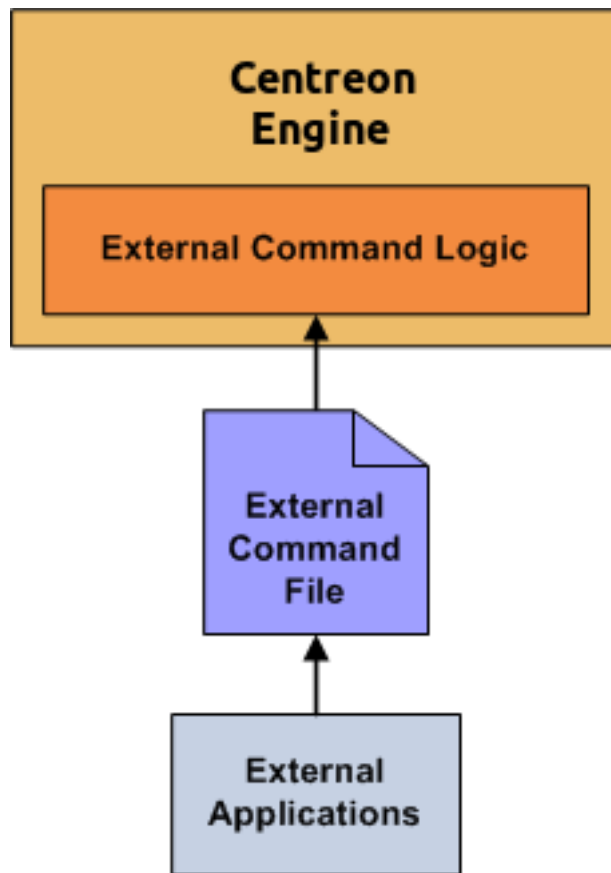
The NDOUtils addon and documentation can be found at <http://www.nagios.org/>.

Nagios Exchange - Hundreds of Other Addons Hundreds of community-developed Centreon Engine addons can be found on the Nagios Exchange website at exchange.nagios.org

Advanced

External Commands

Introduction Centreon Engine can process commands from external applications and alter various aspects of its monitoring functions based on the commands it receives. External applications can submit commands by writing to the *command file*, which is periodically processed by the Centreon Engine daemon.



Enabling External Commands In order to have Centreon Engine process external commands, make sure you do the following:

- enable external command checking with the *check_external_commands* option
- set the frequency of command checks with the *command_check_interval* option
- specify the location of the command file with the *command_file* option
- setup proper permissions on the directory containing the external command file, as described in the *quickstart guide*

When Does Centreon Engine Check For External Commands?

- At regular intervals specified by the *command_check_interval* option in the main configuration file
- Immediately after *event handlers* are executed. This is in addition to the regular cycle of external command checks and is done to provide immediate action if an event handler submits commands to Centreon Engine.

Using External Commands External commands can be used to accomplish a variety of things while Centreon Engine is running. Example of what can be done include temporarily disabling notifications for services and hosts, temporarily disabling service checks, forcing immediate service checks, adding comments to hosts and services, etc.

Command Format External commands that are written to the *command file* have the following format:

```
[time] command_id;command_arguments
```

...where time is the time (in time_t format) that the external application submitted the external command to the command file. The values for the command_id and command_arguments arguments will depend on what command is being submitted to Centreon Engine.

A full listing of external commands that can be used (along with examples of how to use them) can be found online at the following [URL](#)

Event Handlers



Introduction

Event handlers are optional system commands (scripts or executables) that are run whenever a host or service state change occurs.

An obvious use for event handlers is the ability for Centreon Engine to proactively fix problems before anyone is notified. Some other uses for event handlers include:

- Restarting a failed service
- Entering a trouble ticket into a helpdesk system
- Logging event information to a database
- Cycling power on a host*
- etc.
- Cycling power on a host that is experiencing problems with an automated script should not be implemented lightly. Consider the consequences of this carefully before implementing automatic reboots. :-)

When Are Event Handlers Executed? Event handlers are executed when a service or host:

- Is in a SOFT problem state
- Initially goes into a HARD problem state
- Initially recovers from a SOFT or HARD problem state

SOFT and HARD states are described in detail *here*.

Event Handler Types There are different types of optional event handlers that you can define to handle host and state changes:

- Global host event handler

- Global service event handler
- Host-specific event handlers
- Service-specific event handlers

Global host and service event handlers are run for every host or service state change that occurs, immediately prior to any host- or service-specific event handler that may be run. You can specify global event handler commands by using the *global_host_event_handler* and *global_service_event_handler* options in your main configuration file.

Individual hosts and services can have their own event handler command that should be run to handle state changes. You can specify an event handler that should be run by using the *event_handler* directive in your *host* and *service* definitions. These host- and service-specific event handlers are executed immediately after the (optional) global host or service event handler is executed.

Enabling Event Handlers Event handlers can be enabled or disabled on a program-wide basis by using the *enable_event_handlers* in your main configuration file.

Host-specific and service-specific event handlers can be enabled or disabled by using the *event_handler_enabled* directive in your *host* and *service* definitions. Host- and service-specific event handlers will not be executed if the global *enable_event_handlers* option is disabled.

Event Handler Execution Order As already mentioned, global host and service event handlers are executed immediately before host- or service-specific event handlers.

Event handlers are executed for HARD problem and recovery states immediately after notifications are sent out.

Writing Event Handler Commands Event handler commands will likely be shell or perl scripts, but they can be any type of executable that can run from a command prompt. At a minimum, the scripts should take the following *macros* as arguments:

- For Services: *SERVICESTATE SERVICESTATETYPE SERVICEATTEMPT*
- For Hosts: *HOSTSTATE HOSTSTATETYPE HOSTATTEMPT*

The scripts should examine the values of the arguments passed to it and take any necessary action based upon those values. The best way to understand how event handlers work is to see an example. Lucky for you, one is provided *below*.

Note: Additional sample event handler scripts can be found in the *contrib/event_handlers/* subdirectory of the Centreon Engine distribution. Some of these sample scripts demonstrate the use of *external commands* to implement a *redundant* and *distributed* monitoring environments.

Permissions For Event Handler Commands Event handler commands will normally execute with the same permissions as the user under which Centreon Engine is running on your machine. This can present a problem if you want to write an event handler that restarts system services, as root privileges are generally required to do these sorts of tasks.

Ideally you should evaluate the types of event handlers you will be implementing and grant just enough permissions to the Centreon Engine user for executing the necessary system commands. You might want to try using *sudo* to accomplish this.

Service Event Handler Example The example below assumes that you are monitoring the HTTP server on the local machine and have specified `restart-httpd` as the event handler command for the HTTP service definition. Also, I will be assuming that you have set the `max_check_attempts` option for the service to be a value of 4 or greater (i.e. the service is checked 4 times before it is considered to have a real problem). An abbreviated example service definition might look like this:

```
define service{
    host_name        somehost
    service_description HTTP
    max_check_attempts 4
    event_handler     restart-httpd
    ...
}
```

Once the service has been defined with an event handler, we must define that event handler as a command. An example command definition for `restart-httpd` is shown below. Notice the macros in the command line that I am passing to the event handler script - these are important:

```
define command{
    command_name restart-httpd
    command_line /usr/lib/nagios/plugins/event_handlers/restart-httpd $SERVICESTATE$ $SERVICESTATETYPE$
}
```

Now, let's actually write the event handler script (this is the `/usr/lib/nagios/plugins/event_handlers/restart-httpd` script):

```
#!/bin/sh
#
# Event handler script for restarting the web server on the local machine
#
# Note: This script will only restart the web server if the service is
# retried 3 times (in a "soft" state) or if the web service somehow
# manages to fall into a "hard" error state.
#
# What state is the HTTP service in?
case "$1" in
    OK)
        # The service just came back up, so don't do anything...
        ;;
    WARNING)
        # We don't really care about warning states, since the service is probably still running...
        ;;
    UNKNOWN)
        # We don't know what might be causing an unknown error, so don't do anything...
        ;;
    CRITICAL)
        # Aha! The HTTP service appears to have a problem - perhaps we should restart the server...
        # Is this a "soft" or a "hard" state?
        case "$2" in
            # We're in a "soft" state, meaning that Centreon Engine is in the middle of retrying the
            # check before it turns into a "hard" state and contacts get notified...
            SOFT)
                # What check attempt are we on? We don't want to restart the web server on the first
                # check, because it may just be a fluke!
                case "$3" in
                    # Wait until the check has been tried 3 times before restarting the web server.
                    # If the check fails on the 4th time (after we restart the web server), the state
                    # type will turn to "hard" and contacts will be notified of the problem.
                    # Hopefully this will restart the web server successfully, so the 4th check will
```

```

        # result in a "soft" recovery. If that happens no one gets notified because we
        # fixed the problem!
    3)
        echo -n "Restarting HTTP service (3rd soft critical state)..."
        # Call the init script to restart the HTTPD server
        /etc/rc.d/init.d/httpd restart
        ;;
    esac
;;
# The HTTP service somehow managed to turn into a hard error without getting fixed.
# It should have been restarted by the code above, but for some reason it didn't.
# Let's give it one last try, shall we?
# Note: Contacts have already been notified of a problem with the service at this
# point (unless you disabled notifications for this service)
HARD)
    echo -n "Restarting HTTP service..."
    # Call the init script to restart the HTTPD server
    /etc/rc.d/init.d/httpd restart
    ;;
esac
;;
esac
exit 0

```

The sample script provided above will attempt to restart the web server on the local machine in two different instances:

- After the service has been rechecked for the 3rd time and is in a SOFT CRITICAL state
- After the service first goes into a HARD CRITICAL state

The script should theoretically restart web server and fix the problem before the service goes into a HARD problem state, but we include a fallback case in the event it doesn't work the first time. It should be noted that the event handler will only be executed the first time that the service falls into a HARD problem state. This prevents Centreon Engine from continuously executing the script to restart the web server if the service remains in a HARD problem state. You don't want that. :-)

That's all there is to it! Event handlers are pretty simple to write and implement, so give it a try and see what you can do.

Volatile Services

Introduction Centreon Engine has the ability to distinguish between “normal” services and “volatile” services. The `is_volatile` option in each service definition allows you to specify whether a specific service is volatile or not. For most people, the majority of all monitored services will be non-volatile (i.e. “normal”). However, volatile services can be very useful when used properly...

What Are They Useful For? Volatile services are useful for monitoring...

- Things that automatically reset themselves to an “OK” state each time they are checked
- Events such as security alerts which require attention every time there is a problem (and not just the first time)

What's So Special About Volatile Services? Volatile services differ from “normal” services in three important ways. Each time they are checked when they are in a *hard* non-OK state, and the check returns a non-OK state (i.e. no state change has occurred)...

- The non-OK service state is logged

- Contacts are notified about the problem (if that's *what should be done*). .. note:

Notification intervals are ignored for volatile services.

- The *event handler* for the service is run (if one has been defined)

These events normally only occur for services when they are in a non-OK state and a hard state change has just occurred. In other words, they only happen the first time that a service goes into a non-OK state. If future checks of the service result in the same non-OK state, no hard state change occurs and none of the events mentioned take place again.

Note: If you are only interested in logging, consider using *stalking* options instead.

The Power Of Two If you combine the features of volatile services and *passive service checks*, you can do some very useful things. Examples of this include handling SNMP traps, security alerts, etc.

How about an example... Let's say you're running [PortSentry](#) to detect port scans on your machine and automatically firewall potential intruders. If you want to let Centreon Engine know about port scans, you could do the following...

Centreon Engine Configuration:

- Create a service definition called Port Scans and associate it with the host that PortSentry is running on.
- Set the `max_check_attempts` directive in the service definition to 1. This will tell Centreon Engine to immediately force the service into a *hard state* when a non-OK state is reported.
- Set the `active_checks_enabled` directive in the service definition to 0. This prevents Centreon Engine from actively checking the service.
- Set the `passive_checks_enabled` directive in the service definition to 1. This enables passive checks for the service.
- Set this `is_volatile` directive in the service definition to 1.

PortSentry Configuration Edit your PortSentry configuration file (`portsentry.conf`) and define a command for the `KILL_RUN_CMD` directive as follows:

```
KILL_RUN_CMD="/usr/local/Centreon Engine/libexec/event_handlers/submit_check_result host_name 'Port S"
```

Make sure to replace `host_name` with the short name of the host that the service is associated with.

Port Scan Script Create a shell script in the `/usr/lib/nagios/plugins/event_handlers` directory named `submit_check_result`. The contents of the shell script should be something similar to the following:

```
#!/bin/sh
# Write a command to the Centreon Engine command file to cause
# it to process a service check result
echocmd="/bin/echo"
CommandFile="/var/log/centreon-engine/rw/centengine.cmd"
# get the current date/time in seconds since UNIX epoch
datetime=`date +%s`
# create the command line to add to the command file
cmdline="[${datetime}] PROCESS_SERVICE_CHECK_RESULT;${1};${2};${3};${4}"
# append the command to the end of the command file
`$echocmd $cmdline >> $CommandFile`
```

What will happen when PortSentry detects a port scan on the machine in the future?

- PortSentry will firewall the host (this is a function of the PortSentry software)

- PortSentry will execute the `submit_check_result` shell script and send a passive check result to Centreon Engine
- Centreon Engine will read the external command file and see the passive service check submitted by PortSentry
- Centreon Engine will put the Port Scans service in a hard CRITICAL state and send notifications to contacts

Pretty neat, huh?

Service and Host Freshness Checks

Introduction Centreon Engine supports a feature that does “freshness” checking on the results of host and service checks. The purpose of freshness checking is to ensure that host and service checks are being provided passively by external applications on a regular basis.

Freshness checking is useful when you want to ensure that *passive checks* are being received as frequently as you want. This can be very useful in *distributed* and *failover* monitoring environments.



How Does Freshness Checking Work? Centreon Engine periodically checks the freshness of the results for all hosts services that have freshness checking enabled.

- A freshness threshold is calculated for each host or service.
- For each host/service, the age of its last check result is compared with the freshness threshold.
- If the age of the last check result is greater than the freshness threshold, the check result is considered “stale”.
- If the check results is found to be stale, Centreon Engine will force an *active check* of the host or service by executing the command specified by in the host or service definition.

Note: An active check is executed even if active checks are disabled on a program-wide or host- or service-specific basis.

For example, if you have a freshness threshold of 60 for one of your services, Centreon Engine will consider that service to be stale if its last check result is older than 60 seconds.

Enabling Freshness Checking Here’s what you need to do to enable freshness checking...

- Enable freshness checking on a program-wide basis with the *check_service_freshness* and *check_host_freshness* directives.
- Use *service_freshness_check_interval* and *host_freshness_check_interval* options to tell Centreon Engine how often in should check the freshness of service and host results.
- Enable freshness checking on a host- and service-specific basis by setting the *check_freshness* option in your host and service definitions to a value of 1.
- Configure freshness thresholds by setting the *freshness_threshold* option in your host and service definitions.

- Configure the `check_command` option in your host or service definitions to reflect a valid command that should be used to actively check the host or service when it is detected as stale.
- The `check_period` option in your host and service definitions is used when Centreon Engine determines when a host or service can be checked for freshness, so make sure it is set to a valid timeperiod.

Note: If you do not specify a host- or service-specific `freshness_threshold` value (or you set it to zero), Centreon Engine will automatically calculate a threshold automatically, based on how often you monitor that particular host or service. I would recommend that you explicitly specify a freshness threshold, rather than let Centreon Engine pick one for you.

Example An example of a service that might require freshness checking might be one that reports the status of your nightly backup jobs. Perhaps you have an external script that submits the results of the backup job to Centreon Engine once the backup is completed. In this case, all of the checks/results for the service are provided by an external application using passive checks. In order to ensure that the status of the backup job gets reported every day, you may want to enable freshness checking for the service. If the external script doesn't submit the results of the backup job, you can have Centreon Engine fake a critical result by doing something like this...

Here's what the definition for the service might look like (some required options are omitted):

```
define service{
    host_name          backup-server
    service_description ArcServe Backup Job
    active_checks_enabled 0           ; active checks are NOT enabled
    passive_checks_enabled 1          ; passive checks are enabled (this is how results are reported)
    check_freshness      1
    freshness_threshold   93600        ; 26 hour threshold, since backups may not always finish at 24h
    check_command          no-backup-report ; this command is run only if the service results are "stale"
    ...other options...
}
```

Notice that active checks are disabled for the service. This is because the results for the service are only made by an external application using passive checks. Freshness checking is enabled and the freshness threshold has been set to 26 hours. This is a bit longer than 24 hours because backup jobs sometimes run late from day to day (depending on how much data there is to backup, how much network traffic is present, etc.). The `no-backup-report` command is executed only if the results of the service are determined to be stale. The definition of the `no-backup-report` command might look like this:

```
define command{
    command_name no-backup-report
    command_line /usr/lib/nagios/plugins/check_dummy 2 "CRITICAL: Results of backup job were not reported"
}
```

If Centreon Engine detects that the service results are stale, it will run the `no-backup-report` command as an active service check. This causes the `check_dummy` plugin to be executed, which returns a critical state to Centreon Engine. The service will then go into a critical state (if it isn't already there) and someone will probably get notified of the problem.

Distributed Monitoring

Introduction Centreon Engine can be configured to support distributed monitoring of network services and resources. I'll try to briefly explain how this can be accomplished...

Goals The goal in the distributed monitoring environment that I will describe is to offload the overhead (CPU usage, etc.) of performing service checks from a "central" server onto one or more "distributed" servers. Most small to

medium sized shops will not have a real need for setting up such an environment. However, when you want to start monitoring hundreds or even thousands of hosts (and several times that many services) using Centreon Engine, this becomes quite important.

Reference Diagram The diagram below should help give you a general idea of how distributed monitoring works with Centreon Engine. I'll be referring to the items shown in the diagram as I explain things...

Central Server vs. Distributed Servers When setting up a distributed monitoring environment with Centreon Engine, there are differences in the way the central and distributed servers are configured. I'll show you how to configure both types of servers and explain what effects the changes being made have on the overall monitoring. For starters, let's describe the purpose of the different types of servers...

The function of a distributed server is to actively perform checks all the services you define for a "cluster" of hosts. I use the term "cluster" loosely - it basically just means an arbitrary group of hosts on your network. Depending on your network layout, you may have several clusters at one physical location, or each cluster may be separated by a WAN, its own firewall, etc. The important thing to remember is that for each cluster of hosts (however you define that), there is one distributed server that runs Centreon Engine and monitors the services on the hosts in the cluster. A distributed server is usually a bare-bones installation of Centreon Engine. It doesn't have to have the web interface installed, send out notifications, run event handler scripts, or do anything other than execute service checks if you don't want it to. More detailed information on configuring a distributed server comes later...

The purpose of the central server is to simply listen for service check results from one or more distributed servers. Even though services are occasionally actively checked from the central server, the active checks are only performed in dire circumstances, so let's just say that the central server only accepts passive check for now. Since the central server is obtaining *passive service check* results from one or more distributed servers, it serves as the focal point for all monitoring logic (i.e. it sends out notifications, runs event handler scripts, determines host states, has the web interface installed, etc).

Obtaining Service Check Information From Distributed Monitors Okay, before we go jumping into configuration detail we need to know how to send the service check results from the distributed servers to the central server. I've already discussed how to submit passive check results to Centreon Engine from same host that Centreon Engine is running on (as described in the documentation on *passive checks*), but I haven't given any info on how to submit passive check results from other hosts.

In order to facilitate the submission of passive check results to a remote host, I've written the *nsca addon*. The addon consists of two pieces. The first is a client program (*send_nsca*) which is run from a remote host and is used to send the service check results to another server. The second piece is the *nsca daemon* (*nsca*) which either runs as a standalone daemon or under *inetd* and listens for connections from client programs. Upon receiving service check information from a client, the daemon will submit the check information to Centreon Engine (on the central server) by inserting a *PROCESS_SVC_CHECK_RESULT* command into the *external command file*, along with the check results. The next time Centreon Engine checks for *external commands*, it will find the passive service check information that was sent from the distributed server and process it. Easy, huh?

Distributed Server Configuration So how exactly is Centreon Engine configured on a distributed server? Basically, it's just a bare-bones installation. You don't need to install the web interface or have notifications sent out from the server, as this will all be handled by the central server.

Key configuration changes:

- Only those services and hosts which are being monitored directly by the distributed server are defined in the *object configuration file*.
- The distributed server has its *enable_notifications* directive set to 0. This will prevent any notifications from being sent out by the server.

- The distributed server is configured to *obsess over services*.
- The distributed server has an *ocsp command* defined (as described below).

In order to make everything come together and work properly, we want the distributed server to report the results of all service checks to Centreon Engine. We could use *event handlers* to report changes in the state of a service, but that just doesn't cut it. In order to force the distributed server to report all service check results, you must enable the *obsess_over_services* option in the main configuration file and provide a *ocsp_command* to be run after every service check. We will use the *ocsp* command to send the results of all service checks to the central server, making use of the *send_nsca* client and *nsca* daemon (as described above) to handle the transmission.

In order to accomplish this, you'll need to define an *ocsp* command like this:

```
ocsp_command=submit_check_result
```

The command definition for the *submit_check_result* command looks something like this:

```
define command{
    command_name submit_check_result
    command_line /usr/lib/nagios/plugins/event_handlers/submit_check_result $HOSTNAME$ '$SERVICEDESC$'
}
```

The *submit_check_result* shell script looks something like this (replace *central_server* with the IP address of the central server):

```
#!/bin/sh
# Arguments:
# $1 = host_name (Short name of host that the service is
# associated with)
# $2 = svc_description (Description of the service)
# $3 = state_string (A string representing the status of
# the given service - "OK", "WARNING", "CRITICAL"
# or "UNKNOWN")
# $4 = plugin_output (A text string that should be used
# as the plugin output for the service checks)

# Convert the state string to the corresponding return code
return_code=-1

case "$3" in
    OK)
        return_code=0
        ;;
    WARNING)
        return_code=1
        ;;
    CRITICAL)
        return_code=2
        ;;
    UNKNOWN)
        return_code=-1
        ;;
esac

# pipe the service check info into the send_nsca program, which
# in turn transmits the data to the nsca daemon on the central
# monitoring server
/bin/printf "%s\t%s\t%s\t%s\n" "$1" "$2" "$return_code" "$4" | /usr/local/nsca/bin/send_nsca -H cent
```

The script above assumes that you have the *send_nsca* program and its configuration file (*send_nsca.cfg*) located

in the `/usr/local/nscd/bin/` and `/etc/centreon-engine/` directories, respectively.

That's it! We've successfully configured a remote host running Centreon Engine to act as a distributed monitoring server. Let's go over exactly what happens with the distributed server and how it sends service check results to Centreon Engine (the steps outlined below correspond to the numbers in the reference diagram above):

- After the distributed server finishes executing a service check, it executes the command you defined by the `ocsp_command` variable. In our example, this is the `/usr/lib/nagios/plugins/event_handlers/submit_check_result` script. Note that the definition for the `submit_check_result` command passed four pieces of information to the script: the name of the host the service is associated with, the service description, the return code from the service check, and the plugin output from the service check.
- The `submit_check_result` script pipes the service check information (host name, description, return code, and output) to the `send_nscd` client program.
- The `send_nscd` program transmits the service check information to the `nscd` daemon on the central monitoring server.
- The `nscd` daemon on the central server takes the service check information and writes it to the external command file for later pickup by Centreon Engine.
- The Centreon Engine process on the central server reads the external command file and processes the passive service check information that originated from the distributed monitoring server.

Central Server Configuration We've looked at how distributed monitoring servers should be configured, so let's turn to the central server. For all intensive purposes, the central is configured as you would normally configure a standalone server. It is setup as follows:

- The central server has the web interface installed (optional, but recommended)
- The central server has its `enable_notifications` directive set to 1. This will enable notifications. (optional, but recommended)
- The central server has `active service checks` disabled (optional, but recommended - see notes below)
- The central server has `external command checks` enabled (required)
- The central server has `passive service checks` enabled (required) There are three other very important things that you need to keep in mind when configuring the central server:
- The central server must have service definitions for all services that are being monitored by all the distributed servers. Centreon Engine will ignore passive check results if they do not correspond to a service that has been defined.
- If you're only using the central server to process services whose results are going to be provided by distributed hosts, you can simply disable all active service checks on a program-wide basis by setting the `execute_service_checks` directive to 0. If you're using the central server to actively monitor a few services on its own (without the aid of distributed servers), the `enable_active_checks` option of the definitions for service being monitored by distributed servers should be set to 0. This will prevent Centreon Engine from actively checking those services.

It is important that you either disable all service checks on a program-wide basis or disable the `enable_active_checks` option in the definitions for each service that is monitored by a distributed server. This will ensure that active service checks are never executed under normal circumstances. The services will keep getting rescheduled at their normal check intervals (3 minutes, 5 minutes, etc...), but they won't actually be executed. This rescheduling loop will just continue all the while Centreon Engine is running. I'll explain why this is done in a bit...

That's it! Easy, huh?

Problems With Passive Checks For all intensive purposes we can say that the central server is relying solely on passive checks for monitoring. The main problem with relying completely on passive checks for monitoring is the fact that Centreon Engine must rely on something else to provide the monitoring data. What if the remote host that is sending in passive check results goes down or becomes unreachable? If Centreon Engine isn't actively checking the services on the host, how will it know that there is a problem?

Fortunately, there is a way we can handle these types of problems...

Freshness Checking Centreon Engine supports a feature that does “freshness” checking on the results of service checks. More information freshness checking can be found [here](#). This feature gives some protection against situations where remote hosts may stop sending passive service checks into the central monitoring server. The purpose of “freshness” checking is to ensure that service checks are either being provided passively by distributed servers on a regular basis or performed actively by the central server if the need arises. If the service check results provided by the distributed servers get “stale”, Centreon Engine can be configured to force active checks of the service from the central monitoring host.

So how do you do this? On the central monitoring server you need to configure services that are being monitored by distributed servers as follows...

- The `check_freshness` option in the service definitions should be set to 1. This enables “freshness” checking for the services.
- The `freshness_threshold` option in the service definitions should be set to a value (in seconds) which reflects how “fresh” the results for the services (provided by the distributed servers) should be.
- The `check_command` option in the service definitions should reflect valid commands that can be used to actively check the service from the central monitoring server.

Centreon Engine periodically checks the “freshness” of the results for all services that have freshness checking enabled. The `freshness_threshold` option in each service definition is used to determine how “fresh” the results for each service should be. For example, if you set this value to 300 for one of your services, Centreon Engine will consider the service results to be “stale” if they're older than 5 minutes (300 seconds). If you do not specify a value for the `freshness_threshold` option, Centreon Engine will automatically calculate a “freshness” threshold by looking at either the `normal_check_interval` or `retry_check_interval` options (depending on what *type of state* the service is in). If the service results are found to be “stale”, Centreon Engine will run the service check command specified by the `check_command` option in the service definition, thereby actively checking the service.

Remember that you have to specify a `check_command` option in the service definitions that can be used to actively check the status of the service from the central monitoring server. Under normal circumstances, this check command is never executed (because active checks were disabled on a program-wide basis or for the specific services). When freshness checking is enabled, Centreon Engine will run this command to actively check the status of the service even if active checks are disabled on a program-wide or service-specific basis.

If you are unable to define commands to actively check a service from the central monitoring host (or if it turns out to be a major pain), you could simply define all your services with the `check_command` option set to run a dummy script that returns a critical status. Here's an example... Let's assume you define a command called 'service-is-stale' and use that command name in the `check_command` option of your services. Here's what the definition would look like:

```
define command{
    command_name service-is-stale
    command_line /usr/lib/nagios/plugins/check_dummy 2 "CRITICAL: Service results are stale"
}
```

When Centreon Engine detects that the service results are stale and runs the `service-is-stale` command, the `check_dummy` plugin is executed and the service will go into a critical state. This would likely cause notifications to be sent out, so you'll know that there's a problem.

Performing Host Checks At this point you know how to obtain service check results passively from distributed servers. This means that the central server is not actively checking services on its own. But what about host checks? You still need to do them, so how?

Since host checks usually compromise a small part of monitoring activity (they aren't done unless absolutely necessary), I'd recommend that you perform host checks actively from the central server. That means that you define host checks on the central server the same way that you do on the distributed servers (and the same way you would in a normal, non-distributed setup).

Passive host checks are available (read *here*), so you could use them in your distributed monitoring setup, but they suffer from a few problems. The biggest problem is that Centreon Engine does not translate passive host check problem states (DOWN and UNREACHABLE) when they are processed. This means that if your monitoring servers have a different parent/child host structure (and they will, if you monitoring servers are in different locations), the central monitoring server will have an inaccurate view of host states.

If you do want to send passive host checks to a central server in your distributed monitoring setup, make sure:

- The central server has *passive host checks* enabled (required)
- The distributed server is configured to *obsess over hosts*.
- The distributed server has an *ochp command* defined.

The ochp command, which is used for processing host check results, works in a similar manner to the ocsd command, which is used for processing service check results (see documentation above). In order to make sure passive host check results are up to date, you'll want to enable *freshness checking* for hosts (similar to what is described above for services).

Redundant and Failover Network Monitoring

Introduction This section describes a few scenarios for implementing redundant monitoring hosts in various types of network layouts. With redundant hosts, you can maintain the ability to monitor your network when the primary host that runs Centreon Engine fails or when portions of your network become unreachable.

Note: If you are just learning how to use Centreon Engine, I would suggest not trying to implement redundancy until you have becoming familiar with the *prerequisites* I've laid out. Redundancy is a relatively complicated issue to understand, and even more difficult to implement properly.

Prerequisites Before you can even think about implementing redundancy with Centreon Engine, you need to be familiar with the following...

- Implementing *event handlers* for hosts and services
- Issuing *external commands* to Centreon Engine via shell scripts
- Executing plugins on remote hosts using either the *NRPE add-on* or some other method
- Checking the status of the Centreon Engine process with the *check_centengine* plugin

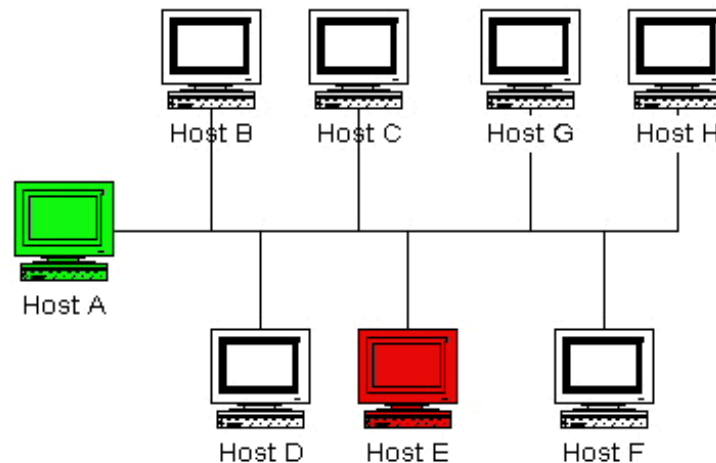
Sample Scripts All of the sample scripts that I use in this documentation can be found in the *event_handlers/* subdirectory of the Centreon Engine distribution. You'll probably need to modify them to work on your system...

Scenario 1 - Redundant Monitoring This is an easy (and naive) method of implementing redundant monitoring hosts on your network and it will only protect against a limited number of failures. More complex setups are necessary in order to provide smarter redundancy, better redundancy across different network segments, etc.

Goals The goal of this type of redundancy implementation is simple. Both the “master” and “slave” hosts monitor the same hosts and service on the network. Under normal circumstances only the “master” host will be sending out notifications to contacts about problems. We want the “slave” host running Centreon Engine to take over the job of notifying contacts about problems if:

- The “master” host that runs Centreon Engine is down or..
- The Centreon Engine process on the “master” host stops running for some reason

Network Layout Diagram The diagram below shows a very simple network setup. For this scenario I will be assuming that hosts A and E are both running Centreon Engine and are monitoring all the hosts shown. Host A will be considered the “master” host and host E will be considered the “slave” host.



Initial Program Settings The slave host (host E) has its initial *enable_notifications* directive disabled, thereby preventing it from sending out any host or service notifications. You also want to make sure that the slave host has its *check_external_commands* directive enabled. That was easy enough...

Initial Configuration Next we need to consider the differences between the *object configuration file(s)* on the master and slave hosts...

I will assume that you have the master host (host A) setup to monitor services on all hosts shown in the diagram above. The slave host (host E) should be setup to monitor the same services and hosts, with the following additions in the configuration file...

- The host definition for host A (in the host E configuration file) should have a host *event handler* defined. Lets say the name of the host event handler is *handle-master-host-event*.
- The configuration file on host E should have a service defined to check the status of the Centreon Engine process on host A. Lets assume that you define this service check to run the *check_centengine* plugin on host A. This can be done by using one of the methods described in this FAQ (update this!).
- The service definition for the Centreon Engine process check on host A should have an *event handler* defined. Lets say the name of the service event handler is *handle-master-proc-event*.

It is important to note that host A (the master host) has no knowledge of host E (the slave host). In this scenario it simply doesn't need to. Of course you may be monitoring services on host E from host A, but that has nothing to do with the implementation of redundancy...

Event Handler Command Definitions We need to stop for a minute and describe what the command definitions for the event handlers on the slave host look like. Here is an example:

```
define command{
    command_name handle-master-host-event
    command_line /usr/lib/nagios/plugins/event_handlers/handle-master-host-event $HOSTSTATE$ $HOSTSTATES$
}

define command{
    command_name handle-master-proc-event
    command_line /usr/lib/nagios/plugins/event_handlers/handle-master-proc-event $SERVICESTATE$ $SERVICESTATES$
}
```

This assumes that you have placed the event handler scripts in the `/usr/lib/nagios/plugins/event_handlers` directory. You may place them anywhere you wish, but you'll need to modify the examples I've given here.

Event Handler Scripts Okay, now lets take a look at what the event handler scripts look like...

Host Event Handler (handle-master-host-event):

```
#!/bin/sh
# Only take action on hard host states...

case "$2" in
    HARD)
        case "$1" in
            DOWN)
                # The master host has gone down!
                # We should now become the master host and take
                # over the responsibilities of monitoring the
                # network, so enable notifications...
                /usr/lib/nagios/plugins/event_handlers/enable_notifications
                ;;
            UP)
                # The master host has recovered!
                # We should go back to being the slave host and
                # let the master host do the monitoring, so
                # disable notifications...
                /usr/lib/nagios/plugins/event_handlers/disable_notifications
                ;;
        esac
    esac
    ;;
esac
exit 0
```

Service Event Handler (handle-master-proc-event):

```
#!/bin/sh
# Only take action on hard service states...

case "$2" in
    HARD)
        case "$1" in
            CRITICAL)
                # The master Centreon Engine process is not running!
                # We should now become the master host and
                # take over the responsibility of monitoring
                # the network, so enable notifications...
                /usr/lib/nagios/plugins/event_handlers/enable_notifications
            ;;
        esac
    esac
    ;;
esac
exit 0
```

```

;;
WARNING)
UNKNOWN)
    # The master Centreon Engine process may or may not
    # be running.. We won't do anything here, but
    # to be on the safe side you may decide you
    # want the slave host to become the master in
    # these situations...
;;
OK)
    # The master Centreon Engine process running again!
    # We should go back to being the slave host,
    # so disable notifications...
    /usr/lib/nagios/plugins/event_handlers/disable_notifications
;;
esac
;;
esac
exit 0

```

What This Does For Us The slave host (host E) initially has notifications disabled, so it won't send out any host or service notifications while the Centreon Engine process on the master host (host A) is still running.

The Centreon Engine process on the slave host (host E) becomes the master host when...

- The master host (host A) goes down and the handle-master-host-event host event handler is executed.
- The Centreon Engine process on the master host (host A) stops running and the handle-master-proc-event service event handler is executed.

When the Centreon Engine process on the slave host (host E) has notifications enabled, it will be able to send out notifications about any service or host problems or recoveries. At this point host E has effectively taken over the responsibility of notifying contacts of host and service problems!

The Centreon Engine process on host E returns to being the slave host when...

- Host A recovers and the handle-master-host-event host event handler is executed.
- The Centreon Engine process on host A recovers and the handle-master-proc-event service event handler is executed.

When the Centreon Engine process on host E has notifications disabled, it will not send out notifications about any service or host problems or recoveries. At this point host E has handed over the responsibilities of notifying contacts of problems to the Centreon Engine process on host A. Everything is now as it was when we first started!

Time Lags Redundancy in Centreon Engine is by no means perfect. One of the more obvious problems is the lag time between the master host failing and the slave host taking over. This is affected by the following...

- The time between a failure of the master host and the first time the slave host detects a problem
- The time needed to verify that the master host really does have a problem (using service or host check retries on the slave host)
- The time between the execution of the event handler and the next time that Centreon Engine checks for external commands

You can minimize this lag by...

- Ensuring that the Centreon Engine process on host E (re)checks one or more services at a high frequency. This is done by using the check_interval and retry_interval arguments in each service definition.

- Ensuring that the number of host rechecks for host A (on host E) allow for fast detection of host problems. This is done by using the `max_check_attempts` argument in the host definition.
- Increase the frequency of *external command* checks on host E. This is done by modifying the *command_check_interval* option in the main configuration file.

When Centreon Engine recovers on the host A, there is also some lag time before host E returns to being a slave host. This is affected by the following...

- The time between a recovery of host A and the time the Centreon Engine process on host E detects the recovery
- The time between the execution of the event handler on host B and the next time the Centreon Engine process on host E checks for external commands

The exact lag times between the transfer of monitoring responsibilities will vary depending on how many services you have defined, the interval at which services are checked, and a lot of pure chance. At any rate, it's definitely better than nothing.

Special Cases Here is one thing you should be aware of... If host A goes down, host E will have notifications enabled and take over the responsibilities of notifying contacts of problems. When host A recovers, host E will have notifications disabled. If - when host A recovers - the Centreon Engine process on host A does not start up properly, there will be a period of time when neither host is notifying contacts of problems! Fortunately, the service check logic in Centreon Engine accounts for this. The next time the Centreon Engine process on host E checks the status of the Centreon Engine process on host A, it will find that it is not running. Host E will then have notifications enabled again and take over all responsibilities of notifying contacts of problems.

The exact amount of time that neither host is monitoring the network is hard to determine. Obviously, this period can be minimized by increasing the frequency of service checks (on host E) of the Centreon Engine process on host A. The rest is up to pure chance, but the total "blackout" time shouldn't be too bad.

Scenario 2 - FailoverMonitoring

Introduction Failover monitoring is similar to, but slightly different than redundant monitoring (as discussed above in *scenario 1*).

Goals The basic goal of failover monitoring is to have the Centreon Engine process on the slave host sit idle while the Centreon Engine process on the master host is running. If the process on the master host stops running (or if the host goes down), the Centreon Engine process on the slave host starts monitoring everything.

While the method described in *scenario 1* will allow you to continue receive notifications if the master monitoring hosts goes down, it does have some pitfalls. The biggest problem is that the slave host is monitoring the same hosts and servers as the master at the same time as the master! This can cause problems with excessive traffic and load on the machines being monitored if you have a lot of services defined. Here's how you can get around that problem...

Initial Program Settings Disable active service checks and notifications on the slave host using the *execute_service_checks* and *enable_notifications* directives. This will prevent the slave host from monitoring hosts and services and sending out notifications while the Centreon Engine process on the master host is still up and running. Make sure you also have the *check_external_commands* directive enabled on the slave host.

Master Process Check Set up a cron job on the slave host that periodically (say every minute) runs a script that checks the status of the Centreon Engine process on the master host (using the *check_nrpe* plugin on the slave host and the *nrpe daemon* and *check_centengine* plugin on the master host). The script should check the return code of the *check_nrpe* plugin. If it returns a non-OK state, the script should send the appropriate commands to the *external*

command file to enable both notifications and active service checks. If the plugin returns an OK state, the script should send commands to the external command file to disable both notifications and active checks.

By doing this you end up with only one process monitoring hosts and services at a time, which is much more efficient than monitoring everything twice.

Also of note, you don't need to define host and service handlers as mentioned in *scenario 1* because things are handled differently.

Additional Issues At this point, you have implemented a very basic failover monitoring setup. However, there is one more thing you should consider doing to make things work smoother.

The big problem with the way things have been setup thus far is the fact that the slave host doesn't have the current status of any services or hosts at the time it takes over the job of monitoring. One way to solve this problem is to enable the *ocsp command* on the master host and have it send all service check results to the slave host using the *nsca* add-on". The slave host will then have up-to-date status information for all services at the time it takes over the job of monitoring things. Since active service checks are not enabled on the slave host, it will not actively run any service checks. However, it will execute host checks if necessary. This means that both the master and slave hosts will be executing host checks as needed, which is not really a big deal since the majority of monitoring deals with service checks.

That's pretty much it as far as setup goes.

Detection and Handling of State Flapping

Introduction Centreon Engine supports optional detection of hosts and services that are "flapping". Flapping occurs when a service or host changes state too frequently, resulting in a storm of problem and recovery notifications. Flapping can be indicative of configuration problems (i.e. thresholds set too low), troublesome services, or real network problems.

How Flap Detection Works Before I get into this, let me say that flapping detection has been a little difficult to implement. How exactly does one determine what "too frequently" means in regards to state changes for a particular host or service? When I first started thinking about implementing flap detection I tried to find some information on how flapping could/should be detected. I couldn't find any information about what others were using (where they using any?), so I decided to settle with what seemed to me to be a reasonable solution...

Whenever Centreon Engine checks the status of a host or service, it will check to see if it has started or stopped flapping. It does this by:

- Storing the results of the last 21 checks of the host or service
- Analyzing the historical check results and determine where state changes/transitions occur
- Using the state transitions to determine a percent state change value (a measure of change) for the host or service
- Comparing the percent state change value against low and high flapping thresholds

A host or service is determined to have started flapping when its percent state change first exceeds a high flapping threshold.

A host or service is determined to have stopped flapping when its percent state goes below a low flapping threshold (assuming that it was previously flapping).

Example Let's describe in more detail how flap detection works with services...

The image below shows a chronological history of service states from the most recent 21 service checks. OK states are shown in green, WARNING states in yellow, CRITICAL states in red, and UNKNOWN states in orange.

The historical service check results are examined to determine where state changes/transitions occur. State changes occur when an archived state is different from the archived state that immediately precedes it chronologically. Since we keep the results of the last 21 service checks in the array, there is a possibility of having at most 20 state changes. In this example there are 7 state changes, indicated by blue arrows in the image above.

The flap detection logic uses the state changes to determine an overall percent state change for the service. This is a measure of volatility/change for the service. Services that never change state will have a 0% state change value, while services that change state each time they're checked will have 100% state change. Most services will have a percent state change somewhere in between.

When calculating the percent state change for the service, the flap detection algorithm will give more weight to new state changes compare to older ones. Specifically, the flap detection routines are currently designed to make the newest possible state change carry 50% more weight than the oldest possible state change. The image below shows how recent state changes are given more weight than older state changes when calculating the overall or total percent state change for a particular service.

Using the images above, lets do a calculation of percent state change for the service. You will notice that there are a total of 7 state changes (at t_3 , t_4 , t_5 , t_9 , t_{12} , t_{16} and t_{19}). Without any weighting of the state changes over time, this would give us a total state change of 35%:

$$(7 \text{ observed state changes} / \text{possible } 20 \text{ state changes}) * 100 = 35 \%$$

Since the flap detection logic will give newer state changes a higher rate than older state changes, the actual calculated percent state change will be slightly less than 35% in this example. Let's say that the weighted percent of state change turned out to be 31%...

The calculated percent state change for the service (31%) will then be compared against flapping thresholds to see what should happen:

- If the service was not previously flapping and 31% is equal to or greater than the high flap threshold, Centreon Engine considers the service to have just started flapping.
- If the service was previously flapping and 31% is less than the low flap threshold, Centreon Engine considers the service to have just stopped flapping.

If neither of those two conditions are met, the flap detection logic won't do anything else with the service, since it is either not currently flapping or it is still flapping.

Flap Detection for Services Centreon Engine checks to see if a service is flapping whenever the service is checked (either actively or passively).

The flap detection logic for services works as described in the example above.

Flap Detection for Hosts Host flap detection works in a similar manner to service flap detection, with one important difference: Centreon Engine will attempt to check to see if a host is flapping whenever:

- The host is checked (actively or passively)
- Sometimes when a service associated with that host is checked. More specifically, when at least x amount of time has passed since the flap detection was last performed, where x is equal to the average check interval of all services associated with the host.

Why is this done? With services we know that the minimum amount of time between consecutive flap detection routines is going to be equal to the service check interval. However, you might not be monitoring hosts on a regular basis, so there might not be a host check interval that can be used in the flap detection logic. Also, it makes sense that checking a service should count towards the detection of host flapping. Services are attributes of or things associated with host after all... At any rate, that's the best method I could come up with for determining how often flap detection could be performed on a host, so there you have it.

Flap Detection Thresholds Centreon Engine uses several variables to determine the percent state change thresholds it uses for flap detection. For both hosts and services, there are global high and low thresholds and host- or service-specific thresholds that you can configure. Centreon Engine will use the global thresholds for flap detection if you do not specify host- or service-specific thresholds.

The table below shows the global and host- or service-specific variables that control the various thresholds used in flap detection.

Object Type	Global Variables	Object-Specific Variables
Host	low_host_flap_threshold, high_host_flap_threshold	low_flap_threshold, high_flap_threshold
Service	low_service_flap_threshold, high_service_flap_threshold	low_flap_threshold, high_flap_threshold

States Used For Flap Detection Normally Centreon Engine will track the results of the last 21 checks of a host or service, regardless of the check result (host/service state), for use in the flap detection logic.

Note: You can exclude certain host or service states from use in flap detection logic by using the `flap_detection_options` directive in your host or service definitions. This directive allows you to specify what host or service states (i.e. “UP”, “DOWN”, “OK”, “CRITICAL”) you want to use for flap detection. If you don’t use this directive, all host or service states are used in flap detection.

Flap Handling When a service or host is first detected as flapping, Centreon Engine will:

- Log a message indicating that the service or host is flapping.
- Add a non-persistent comment to the host or service indicating that it is flapping.
- Send a “flapping start” notification for the host or service to appropriate contacts.
- Suppress other notifications for the service or host (this is one of the filters in the *notification logic*).

When a service or host stops flapping, Centreon Engine will:

- Log a message indicating that the service or host has stopped flapping.
- Delete the comment that was originally added to the service or host when it started flapping.
- Send a “flapping stop” notification for the host or service to appropriate contacts.
- Remove the block on notifications for the service or host (notifications will still be bound to the normal *notification logic*).

Enabling Flap Detection In order to enable the flap detection features in Centreon Engine, you’ll need to:

- Set `enable_flap_detection` directive is set to 1.
- Set the `flap_detection_enabled` directive in your host and service definitions is set to 1.

If you want to disable flap detection on a global basis, set the `enable_flap_detection` directive to 0.

If you would like to disable flap detection for just a few hosts or services, use the `flap_detection_enabled` directive in the host and/or service definitions to do so.

Notification Escalations



Introduction

Centreon Engine supports optional escalation of contact notifications for hosts and services. Escalation of host and service notifications is accomplished by defining *host escalations* and *service escalations* in your *object configuration file(s)*".

Note: The examples I provide below all make use of service escalation definitions, but host escalations work the same way. Except, of course, that they're for hosts instead of services. :-)

When Are Notifications Escalated? Notifications are escalated if and only if one or more escalation definitions matches the current notification that is being sent out. If a host or service notification does not have any valid escalation definitions that applies to it, the contact group(s) specified in either the host group or service definition will be used for the notification. Look at the example below:

```
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
    service_description HTTP
    first_notification  6
    last_notification   10
    notification_interval 60
    contact_groups      nt-admins,managers,everyone
}
```

Notice that there are "holes" in the notification escalation definitions. In particular, notifications 1 and 2 are not handled by the escalations, nor are any notifications beyond 10. For the first and second notification, as well as all notifications beyond the tenth one, the default contact groups specified in the service definition are used. For all the examples I'll be using, I'll be assuming that the default contact groups for the service definition is called nt-admins.

Contact Groups When defining notification escalations, it is important to keep in mind that any contact groups that were members of "lower" escalations (i.e. those with lower notification number ranges) should also be included in "higher" escalation definitions. This should be done to ensure that anyone who gets notified of a problem continues to get notified as the problem is escalated.

Example:

```
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
    service_description HTTP
    first_notification  6
    last_notification   0
    notification_interval 60
    contact_groups      nt-admins,managers,everyone
}

```

The first (or “lowest”) escalation level includes both the nt-admins and managers contact groups. The last (or “highest”) escalation level includes the nt-admins, managers, and everyone contact groups. Notice that the nt-admins contact group is included in both escalation definitions. This is done so that they continue to get paged if there are still problems after the first two service notifications are sent out. The managers contact group first appears in the “lower” escalation definition - they are first notified when the third problem notification gets sent out. We want the managers group to continue to be notified if the problem continues past five notifications, so they are also included in the “higher” escalation definition.

Overlapping Escalation Ranges Notification escalation definitions can have notification ranges that overlap. Take the following example:

```

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification  3
    last_notification   5
    notification_interval 20
    contact_groups      nt-admins,managers
}

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification  4
    last_notification   0
    notification_interval 30
    contact_groups      on-call-support
}

```

In the example above:

- The nt-admins and managers contact groups get notified on the third notification
- All three contact groups get notified on the fourth and fifth notifications
- Only the on-call-support contact group gets notified on the sixth (or higher) notification

Recovery Notifications Recovery notifications are slightly different than problem notifications when it comes to escalations. Take the following example:

```

define serviceescalation{
    host_name          webserver

```



```

    service_description    HTTP
    first_notification     3
    last_notification      5
    notification_interval  20
    contact_groups         nt-admins,managers
}

define serviceescalation{
    host_name              webserver
    service_description    HTTP
    first_notification     4
    last_notification      0
    notification_interval  30
    contact_groups         on-call-support
}

```

If, after three problem notifications, a recovery notification is sent out for the service, who gets notified? The recovery is actually the fourth notification that gets sent out. However, the escalation code is smart enough to realize that only those people who were notified about the problem on the third notification should be notified about the recovery. In this case, the nt-admins and managers contact groups would be notified of the recovery.

Notification Intervals You can change the frequency at which escalated notifications are sent out for a particular host or service by using the `notification_interval` option of the hostgroup or service escalation definition.

Example:

```

define serviceescalation{
    host_name              webserver
    service_description    HTTP
    first_notification     3
    last_notification      5
    notification_interval  45
    contact_groups         nt-admins,managers
}

define serviceescalation{
    host_name              webserver
    service_description    HTTP
    first_notification     6
    last_notification      0
    notification_interval  60
    contact_groups         nt-admins,managers,everyone
}

```

In this example we see that the default notification interval for the services is 240 minutes (this is the value in the service definition). When the service notification is escalated on the 3rd, 4th, and 5th notifications, an interval of 45 minutes will be used between notifications. On the 6th and subsequent notifications, the notification interval will be 60 minutes, as specified in the second escalation definition.

Since it is possible to have overlapping escalation definitions for a particular hostgroup or service, and the fact that a host can be a member of multiple hostgroups, Centreon Engine has to make a decision on what to do as far as the notification interval is concerned when escalation definitions overlap. In any case where there are multiple valid escalation definitions for a particular notification, Centreon Engine will choose the smallest notification interval. Take the following example:

```

define serviceescalation{
    host_name              webserver
    service_description    HTTP

```

```

first_notification    3
last_notification     5
notification_interval 45
contact_groups        nt-admins,managers
}

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification  4
    last_notification   0
    notification_interval 60
    contact_groups      nt-admins,managers,everyone
}

```

We see that the two escalation definitions overlap on the 4th and 5th notifications. For these notifications, Centreon Engine will use a notification interval of 45 minutes, since it is the smallest interval present in any valid escalation definitions for those notifications.

One last note about notification intervals deals with intervals of 0. An interval of 0 means that Centreon Engine should only sent a notification out for the first valid notification during that escalation definition. All subsequent notifications for the hostgroup or service will be suppressed. Take this example:

```

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification  3
    last_notification   5
    notification_interval 45
    contact_groups      nt-admins,managers
}

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification  4
    last_notification   6
    notification_interval 0
    contact_groups      nt-admins,managers,everyone
}

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification  7
    last_notification   0
    notification_interval 30
    contact_groups      nt-admins,managers
}

```

In the example above, the maximum number of problem notifications that could be sent out about the service would be four. This is because the notification interval of 0 in the second escalation definition indicates that only one notification should be sent out (starting with and including the 4th notification) and all subsequent notifications should be repressed. Because of this, the third service escalation definition has no effect whatsoever, as there will never be more than four notifications.

Time Period Restrictions Under normal circumstances, escalations can be used at any time that a notification could normally be sent out for the host or service. This “notification time window” is determined by the `notification_period`

directive in the *host* or *service* definition.

You can optionally restrict escalations so that they are only used during specific time periods by using the `escalation_period` directive in the host or service escalation definition. If you use the `escalation_period` directive to specify a *timeperiod* during which the escalation can be used, the escalation will only be used during that time. If you do not specify any `escalation_period` directive, the escalation can be used at any time within the “notification time window” for the host or service.

Note: Escalated notifications are still subject to the normal time restrictions imposed by the `notification_period` directive in a host or service definition, so the *timeperiod* you specify in an escalation definition should be a subset of that larger “notification time window”.

State Restrictions If you would like to restrict the escalation definition so that it is only used when the host or service is in a particular state, you can use the `escalation_options` directive in the host or service escalation definition. If you do not use the `escalation_options` directive, the escalation can be used when the host or service is in any state.

On-Call Rotations



Introduction

Admins often have to shoulder the burden of answering pagers, cell phone calls, etc. when they least desire them. No one likes to be woken up at 4 am to fix a problem. But it's often better to fix the problem in the middle of the night, rather than face the wrath of an unhappy boss when you stroll in at 9 am the next morning.

For those lucky admins who have a team of gurus who can help share the responsibility of answering alerts, on-call rotations are often setup. Multiple admins will often alternate taking notifications on weekends, weeknights, holidays, etc.

I'll show you how you can create *timeperiod* definitions in a way that can facilitate most on-call notification rotations. These definitions won't handle human issues that will inevitably crop up (admins calling in sick, swapping shifts, or throwing their pagers into the river), but they will allow you to setup a basic structure that should work the majority of the time.

Scenario 1: Holidays and Weekends Two admins - John and Bob - are responsible for responding to Centreon Engine alerts. John receives all notifications for weekdays (with 24 hour days), excluding holidays; Bob handles notifications during the weekends and holidays. Lucky Bob. Here's how you can define this type of rotation using *timeperiods*...

First, define 3 *timeperiods* that contains time ranges for holidays, weekdays, and weekends:

```
define timeperiod{
    name                weekdays
    timeperiod_name      weekdays
    monday               00:00-24:00
    tuesday              00:00-24:00
    wednesday            00:00-24:00
```

```

    thursday      00:00-24:00
    friday         00:00-24:00
}

define timeperiod{
    name           weekends
    timeperiod_name weekends
    saturday       00:00-24:00
    sunday         00:00-24:00
}

define timeperiod{
    name           holidays
    timeperiod_name holidays
    january 1      00:00-24:00 ; New Year's Day
    2008-03-23     00:00-24:00 ; Easter (2008)
    2009-04-12     00:00-24:00 ; Easter (2009)
    monday -1 may  00:00-24:00 ; Memorial Day (Last Monday in May)
    july 4         00:00-24:00 ; Independence Day
    monday 1 september 00:00-24:00 ; Labor Day (1st Monday in September)
    thursday 4 november 00:00-24:00 ; Thanksgiving (4th Thursday in November)
    december 25    00:00-24:00 ; Christmas
    december 31    17:00-24:00 ; New Year's Eve (5pm onwards)
}

```

Next, define a timeperiod for John's on-call times that include weekdays, but excludes the dates/times defined in the holidays timeperiod above:

```

define timeperiod{
    timeperiod_name john-oncall
    use             weekdays ; Include weekdays
    exclude         holidays ; Exclude holiday dates/times defined elsewhere
}

```

You can now reference this timeperiod in John's contact definition:

```

define contact{
    contact_name      john
    ...
    host_notification_period  john-oncall
    service_notification_period john-oncall
}

```

Define a new timeperiod for Bob's on-call times that include weekends and the dates/times defined in the holidays timeperiod above:

```

define timeperiod{
    timeperiod_name bob-oncall
    use             weekends,holidays ; Include weekend and holiday date/times defined elsewhere
}

```

You can now reference this timeperiod in Bob's contact definition:

```

define contact{
    contact_name      bob
    ...
    host_notification_period  bob-oncall
    service_notification_period bob-oncall
}

```

Scenario 2: Alternating Days In this scenario John and Bob alternate handling alerts every other day - regardless of whether its a weekend, weekday, or holiday.

Define a timeperiod for when John should receive notifications. Assuming today's date is August 1st, 2007 and John is handling notifications starting today, the definition would look like this:

```
define timeperiod{
    timeperiod_name john-oncall
    2007-08-01 / 2 00:00-24:00 ; Every two days, starting August 1st,
    2007
}
```

Now define a timeperiod for when Bob should receive notifications. Bob gets notifications on the days that John doesn't, so his first on-call day starts tomorrow (August 2nd, 2007):

```
define timeperiod{
    timeperiod_name bob-oncall
    2007-08-02 / 2 00:00-24:00 ; Every two days, starting August 2nd, 2007
}
```

Now you need to reference these timeperiod definitions in the contact definitions for John and Bob:

```
define contact{
    contact_name          john
    ...
    host_notification_period  john-oncall
    service_notification_period john-oncall
}

define contact{
    contact_name          bob
    ...
    host_notification_period  bob-oncall
    service_notification_period bob-oncall
}
```

Scenario 3: Alternating Weeks In this scenario John and Bob alternate handling alerts every other week. John handles alerts Sunday through Saturday one week, and Bob handles alerts for the following seven days. This continues in perpetuity.

Define a timeperiod for when John should receive notifications. Assuming today's date is Sunday, July 29th, 2007 and John is handling notifications this week (starting today), the definition would look like this:

```
define timeperiod{
    timeperiod_name john-oncall
    2007-07-29 / 14 00:00-24:00 ; Every 14 days (two weeks), starting Sunday, July 29th, 2007
    2007-07-30 / 14 00:00-24:00 ; Every other Monday starting July 30th, 2007
    2007-07-31 / 14 00:00-24:00 ; Every other Tuesday starting July 31st, 2007
    2007-08-01 / 14 00:00-24:00 ; Every other Wednesday starting August 1st, 2007
    2007-08-02 / 14 00:00-24:00 ; Every other Thursday starting August 2nd, 2007
    2007-08-03 / 14 00:00-24:00 ; Every other Friday starting August 3rd, 2007
    2007-08-04 / 14 00:00-24:00 ; Every other Saturday starting August 4th, 2007
}
```

Now define a timeperiod for when Bob should receive notifications. Bob gets notifications on the weeks that John doesn't, so his first on-call day starts next Sunday (August 5th, 2007):

```
define timeperiod{
    timeperiod_name bob-oncall
```

```

2007-08-05 / 14 00:00-24:00 ; Every 14 days (two weeks), starting Sunday, August 5th, 2007
2007-08-06 / 14 00:00-24:00 ; Every other Monday starting August 6th, 2007
2007-08-07 / 14 00:00-24:00 ; Every other Tuesday starting August 7th, 2007
2007-08-08 / 14 00:00-24:00 ; Every other Wednesday starting August 8th, 2007
2007-08-09 / 14 00:00-24:00 ; Every other Thursday starting August 9th, 2007
2007-08-10 / 14 00:00-24:00 ; Every other Friday starting August 10th, 2007
2007-08-11 / 14 00:00-24:00 ; Every other Saturday starting August 11th, 2007
}

```

Now you need to reference these timeperiod definitions in the contact definitions for John and Bob:

```

define contact{
    contact_name          john
    ...
    host_notification_period  john-oncall
    service_notification_period john-oncall
}

define contact{
    contact_name          bob
    ...
    host_notification_period  bob-oncall
    service_notification_period bob-oncall
}

```

Scenario 4: Vacation Days In this scenarios, John handles notifications for all days except those he has off. He has several standing days off each month, as well as some planned vacations. Bob handles notifications when John is on vacation or out of the office.

First, define a timeperiod that contains time ranges for John's vacation days and days off:

```

define timeperiod{
    name                  john-out-of-office
    timeperiod_name       john-out-of-office
    day 15                00:00-24:00 ; 15th day of each month
    day -1               00:00-24:00 ; Last day of each month (28th, 29th, 30th, or 31st)
    day -2               00:00-24:00 ; 2nd to last day of each month (27th, 28th, 29th, or 30th)
    january 2            00:00-24:00 ; January 2nd each year
    june 1 - july 5      00:00-24:00 ; Yearly camping trip (June 1st - July 5th)
    2007-11-01 - 2007-11-10 00:00-24:00 ; Vacation to the US Virgin Islands (November 1st-10th, 2007)
}

```

Next, define a timeperiod for John's on-call times that excludes the dates/times defined in the timeperiod above:

```

define timeperiod{
    timeperiod_name       john-oncall
    monday               00:00-24:00
    tuesday              00:00-24:00
    wednesday            00:00-24:00
    thursday             00:00-24:00
    friday               00:00-24:00
    exclude              john-out-of-office ; Exclude dates/times John is out
}

```

You can now reference this timeperiod in John's contact definition:

```

define contact{
    contact_name          john

```

```
...
host_notification_period    john-oncall
service_notification_period john-oncall
}
```

Define a new timeperiod for Bob's on-call times that include the dates/times that John is out of the office:

```
define timeperiod{
    timeperiod_name bob-oncall
    use                john-out-of-office ; Include holiday date/times that John is out
}
```

You can now reference this timeperiod in Bob's contact definition:

```
define contact{
    contact_name          bob
    ...
    host_notification_period    bob-oncall
    service_notification_period bob-oncall
}
```

Other Scenarios There are a lot of other on-call notification rotation scenarios that you might have. The date exception directive in *timeperiod definitions* is capable of handling most dates and date ranges that you might need to use, so check out the different formats that you can use. If you make a mistake when creating timeperiod definitions, always err on the side of giving someone else more on-call duty time. :-)

Monitoring Service and Host Clusters

Introduction Several people have asked how to go about monitoring clusters of hosts or services, so I decided to write up a little documentation on how to do this. It's fairly straightforward, so hopefully you find things easy to understand...

First off, we need to define what we mean by a "cluster". The simplest way to understand this is with an example. Let's say that your organization has five hosts which provide redundant DNS services to your organization. If one of them fails, it's not a major catastrophe because the remaining servers will continue to provide name resolution services. If you're concerned with monitoring the availability of DNS service to your organization, you will want to monitor five DNS servers. This is what I consider to be a service cluster. The service cluster consists of five separate DNS services that you are monitoring. Although you do want to monitor each individual service, your main concern is with the overall status of the DNS service cluster, rather than the availability of any one particular service.

If your organization has a group of hosts that provide a high-availability (clustering) solution, I would consider those to be a host cluster. If one particular host fails, another will step in to take over all the duties of the failed server. As a side note, check out the [High-Availability Linux Project](#) for information on providing host and service redundancy with Linux.

Plan of Attack There are several ways you could potentially monitor service or host clusters. I'll describe the method that I believe to be the easiest. Monitoring service or host clusters involves two things:

- Monitoring individual cluster elements
- Monitoring the cluster as a collective entity

Monitoring individual host or service cluster elements is easier than you think. In fact, you're probably already doing it. For service clusters, just make sure that you are monitoring each service element of the cluster. If you've got a cluster of five DNS servers, make sure you have five separate service definitions (probably using the `check_dns` plugin). For host clusters, make sure you have configured appropriate host definitions for each member of the cluster.

(you'll also have to define at least one service to be monitored for each of the hosts). Important: You're going to want to disable notifications for the individual cluster elements (host or service definitions).

Monitoring the overall cluster can be done by using the previously cached results of cluster elements. Although you could re-check all elements of the cluster to determine the cluster's status, why waste bandwidth and resources when you already have the results cached? Where are the results cached? Cached results for cluster elements can be found in the *status file* (assuming you are monitoring each element). The `check_cluster` plugin is designed specifically for checking cached host and service states in the status file. Important: Although you didn't enable notifications for individual elements of the cluster, you will want them enabled for the overall cluster status check.

Using the `check_cluster` Plugin The `check_cluster` plugin is designed to report the overall status of a host or service cluster by checking the status information of each individual host or service cluster elements.

More to come... The `check_cluster` plugin can be found in the contrib directory of the Centreon Engine Plugins release at <http://sourceforge.net/projects/nagiosplug/>.

Monitoring Service Clusters Let's say you have three DNS servers that provide redundant services on your network. First off, you need to be monitoring each of these DNS servers separately before you can monitor them as a cluster. I'll assume that you already have three separate services (all called "DNS Service") associated with your DNS hosts (called "host1", "host2" and "host3").

In order to monitor the services as a cluster, you'll need to create a new "cluster" service. However, before you do that, make sure you have a service cluster check command configured. Let's assume that you have a command called `check_service_cluster` defined as follows:

```
define command{
    command_name check_service_cluster
    command_line /usr/lib/nagios/plugins/check_cluster --service -l $ARG1$ -w $ARG2$ -c $ARG3$ -d $ARG4$
}
```

Now you'll need to create the "cluster" service and use the `check_service_cluster` command you just created as the cluster's check command. The example below gives an example of how to do this. The example below will generate a **CRITICAL** alert if 2 or more services in the cluster are in a non-OK state, and a **WARNING** alert if only 1 of the services is in a non-OK state. If all the individual service members of the cluster are OK, the cluster check will return an OK state as well:

```
define service{
    ...
    check_command check_service_cluster!"DNS Cluster"!1!2!$SERVICESTATEID:host1:DNS Service$, $SERVICESTATEID:host2:DNS Service$, $SERVICESTATEID:host3:DNS Service$
    ...
}
```

It is important to notice that we are passing a comma-delimited list of on-demand service state *macros* to the `$ARG4$` macro in the cluster check command. That's important! Centreon Engine will fill those on-demand macros in with the current service state IDs (numerical values, rather than text strings) of the individual members of the cluster.

Monitoring Host Clusters Monitoring host clusters is very similar to monitoring service clusters. Obviously, the main difference is that the cluster members are hosts and not services. In order to monitor the status of a host cluster, you must define a service that uses the `check_cluster` plugin. The service should not be associated with any of the hosts in the cluster, as this will cause problems with notifications for the cluster if that host goes down. A good idea might be to associate the service with the host that Centreon Engine is running on. After all, if the host that Centreon Engine is running on goes down, then Centreon Engine isn't running anymore, so there isn't anything you can do as far as monitoring (unless you've setup *redundant monitoring hosts*)

Anyway, let's assume that you have a `check_host_cluster` command defined as follows:


```
define command{
    command_name check_host_cluster
    command_line /usr/lib/nagios/plugins/check_cluster --host -l $ARG1$ -w $ARG2$ -c $ARG3$ -d $ARG4$
}
```

Let's say you have three hosts (named "host1", "host2" and "host3") in the host cluster. If you want Centreon Engine to generate a warning alert if one host in the cluster is not UP or a critical alert if two or more hosts are not UP, the service you define to monitor the host cluster might look something like this:

```
define service{
    ...
    check_command check_host_cluster!"Super Host Cluster"!1!2!$HOSTSTATEID:host1,$HOSTSTATEID:host2,$
    ...
}
```

It is important to notice that we are passing a comma-delimited list of on-demand host state *macros* to the \$ARG4\$ macro in the cluster check command. That's important! Centreon Engine will fill those on-demand macros in with the current host state IDs (numerical values, rather than text strings) of the individual members of the cluster.

That's it! Centreon Engine will periodically check the status of the host cluster and send notifications to you when its status is degraded (assuming you've enabled notification for the service). Note that for the host definitions of each cluster member, you will most likely want to disable notifications when the host goes down. Remember that you don't care as much about the status of any individual host as you do the overall status of the cluster. Depending on your network layout and what you're trying to accomplish, you may wish to leave notifications for unreachable states enabled for the host definitions.

Host and Service Dependencies

Introduction Service and host dependencies are an advanced feature of Centreon Engine that allow you to control the behavior of hosts and services based on the status of one or more other hosts or services. I'll explain how dependencies work, along with the differences between host and service dependencies.

Service Dependencies Overview There are a few things you should know about service dependencies:

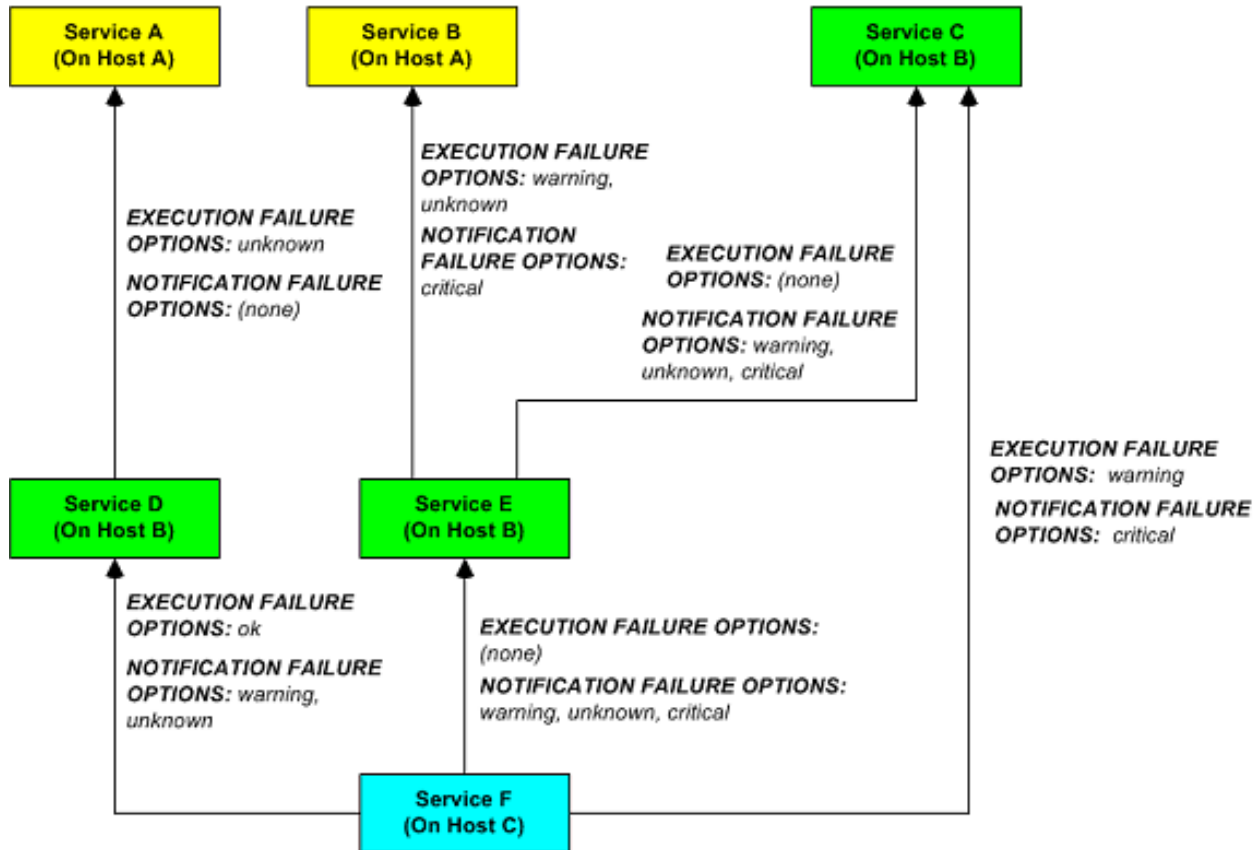
- A service can be dependent on one or more other services
- A service can be dependent on services which are not associated with the same host
- Service dependencies are not inherited (unless specifically configured to)
- Service dependencies can be used to cause service check execution and service notifications to be suppressed under different circumstances (OK, WARNING, UNKNOWN, and/or CRITICAL states)
- Service dependencies might only be valid during specific *timeperiods*

Defining Service Dependencies First, the basics. You create service dependencies by adding *service dependency* definitions in your *object config file(s)*. In each definition you specify the dependent service, the service you are depending on, and the criteria (if any) that cause the execution and notification dependencies to fail (these are described later).

You can create several dependencies for a given service, but you must add a separate service dependency definition for each dependency you create.

Example Service Dependencies The image below shows an example logical layout of service notification and execution dependencies. Different services are dependent on other services for notifications and check execution.

Service Dependencies



In this example, the dependency definitions for Service F on Host C would be defined as follows:

```

define servicedependency{
  host_name           Host B
  service_description Service D
  dependent_host_name Host C
  dependent_service_description Service F
  execution_failure_criteria o
  notification_failure_criteria w,u
}

define servicedependency{
  host_name           Host B
  service_description Service E
  dependent_host_name Host C
  dependent_service_description Service F
  execution_failure_criteria n
  notification_failure_criteria w,u,c
}

define servicedependency{
  host_name           Host B

```

```

service_description      Service C
dependent_host_name      Host C
dependent_service_description Service F
execution_failure_criteria w
notification_failure_criteria c
}

```

The other dependency definitions shown in the image above would be defined as follows:

```

define servicedependency{
    host_name              Host A
    service_description     Service A
    dependent_host_name     Host B
    dependent_service_description Service D
    execution_failure_criteria u
    notification_failure_criteria n
}

```

```

define servicedependency{
    host_name              Host A
    service_description     Service B
    dependent_host_name     Host B
    dependent_service_description Service E
    execution_failure_criteria w,u
    notification_failure_criteria c
}

```

```

define servicedependency{
    host_name              Host B
    service_description     Service C
    dependent_host_name     Host B
    dependent_service_description Service E
    execution_failure_criteria n
    notification_failure_criteria w,u,c
}

```

How Service Dependencies Are Tested Before Centreon Engine executes a service check or sends notifications out for a service, it will check to see if the service has any dependencies. If it doesn't have any dependencies, the check is executed or the notification is sent out as it normally would be. If the service does have one or more dependencies, Centreon Engine will check each dependency entry as follows:

- * Centreon Engine gets the current status of the service that is being depended upon.
- * Centreon Engine compares the current status of the service that is being depended upon against either the execution or notification failure options in the dependency definition (whichever one is relevant at the time).
- * If the current status of the service that is being depended upon matches one of the failure options, the dependency is said to have failed and Centreon Engine will break out of the dependency check loop.
- * If the current state of the service that is being depended upon does not match any of the failure options for the dependency entry, the dependency is said to have passed and Centreon Engine will go on and check the next dependency entry.

This cycle continues until either all dependencies for the service have been checked or until one dependency check fails.

Note: One important thing to note is that by default, Centreon Engine will use the most current *hard state* of the service(s) that is/are being depended upon when it does the dependency checks. If you want Centreon Engine to use the most current state of the services (regardless of whether its a soft or hard state), enable the *soft_state_dependencies* option.

Execution Dependencies Execution dependencies are used to restrict when *active checks* of a service can be performed. *Passive checks* are not restricted by execution dependencies.

If all of the execution dependency tests for the service passed, Centreon Engine will execute the check of the service as it normally would. If even just one of the execution dependencies for a service fails, Centreon Engine will temporarily prevent the execution of checks for that (dependent) service. At some point in the future the execution dependency tests for the service may all pass. If this happens, Centreon Engine will start checking the service again as it normally would. More information on the check scheduling logic can be found *here*.

In the example above, Service E would have failed execution dependencies if Service B is in a WARNING or UNKNOWN state. If this was the case, the service check would not be performed and the check would be scheduled for (potential) execution at a later time.

Notification Dependencies If all of the notification dependency tests for the service passed, Centreon Engine will send notifications out for the service as it normally would. If even just one of the notification dependencies for a service fails, Centreon Engine will temporarily repress notifications for that (dependent) service. At some point in the future the notification dependency tests for the service may all pass. If this happens, Centreon Engine will start sending out notifications again as it normally would for the service. More information on the notification logic can be found *here*.

In the example above, Service F would have failed notification dependencies if Service C is in a CRITICAL state, and/or Service D is in a WARNING or UNKNOWN state, and/or if Service E is in a WARNING, UNKNOWN, or CRITICAL state. If this were the case, notifications for the service would not be sent out.

Dependency Inheritance As mentioned before, service dependencies are not inherited by default. In the example above you can see that Service F is dependent on Service E. However, it does not automatically inherit Service E's dependencies on Service B and Service C. In order to make Service F dependent on Service C we had to add another service dependency definition. There is no dependency definition for Service B, so Service F is not dependent on Service B.

If you do wish to make service dependencies inheritable, you must use the *inherits_parent* directive in the *service dependency* definition". When this directive is enabled, it indicates that the dependency inherits dependencies of the service that is being depended upon (also referred to as the master service). In other words, if the master service is dependent upon other services and any one of those dependencies fail, this dependency will also fail.

In the example above, imagine that you want to add a new dependency for service F to make it dependent on service A. You could create a new dependency definition that specified service F as the dependent service and service A as being the master service (i.e. the service that is being dependend on). You could alternatively modify the dependency definition for services D and F to look like this:

```
define servicedependency{
    host_name           Host B
    service_description Service D
    dependent_host_name Host C
    dependent_service_description Service F
    execution_failure_criteria o
    notification_failure_criteria n
    inherits_parent     1
}
```

Since the `inherits_parent` directive is enabled, the dependency between services A and D will be tested when the dependency between services F and D are being tested.

Dependencies can have multiple levels of inheritance. If the dependency definition between A and D had its `inherits_parent` directive enable and service A was dependent on some other service (let's call it service G), the service F would be dependent on services D, A, and G (each with potentially different criteria).

Host Dependencies As you'd probably expect, host dependencies work in a similar fashion to service dependencies. The difference is that they're for hosts, not services.

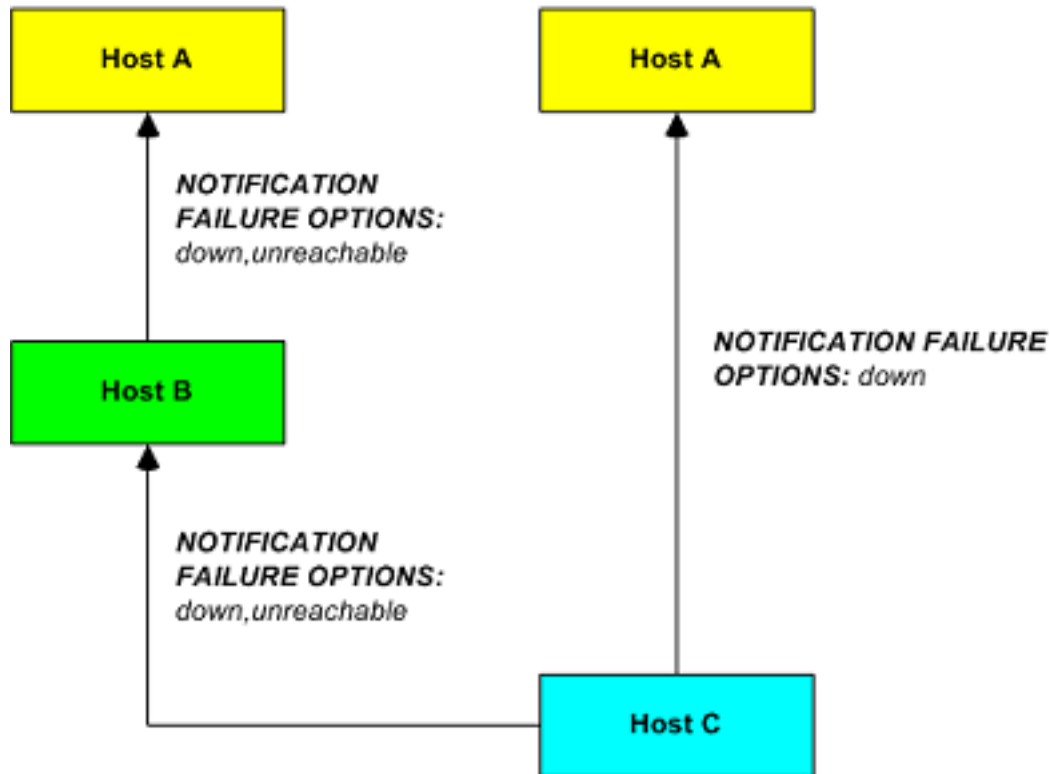
Note: Do not confuse host dependencies with parent/child host relationships. You should be using parent/child host relationships (defined with the `parents` directive in *host* definitions) for most cases, rather than host dependencies. A description of how parent/child host relationships work can be found in the documentation on *network reachability*.

Here are the basics about host dependencies:

- A host can be dependent on one or more other host
- Host dependencies are not inherited (unless specifically configured to)
- Host dependencies can be used to cause host check execution and host notifications to be suppressed under different circumstances (UP, DOWN, and/or UNREACHABLE states)
- Host dependencies might only be valid during specific *timeperiods*

Example Host Dependencies The image below shows an example of the logical layout of host notification dependencies. Different hosts are dependent on other hosts for notifications.

Host Dependencies



In the example above, the dependency definitions for Host C would be defined as follows:

```
define hostdependency{
    host_name           Host A
    dependent_host_name Host C
    notification_failure_criteria d
}

define hostdependency{
    host_name           Host B
    dependent_host_name Host C
    notification_failure_criteria d,u
}
```

As with service dependencies, host dependencies are not inherited. In the example image you can see that Host C does not inherit the host dependencies of Host B. In order for Host C to be dependent on Host A, a new host dependency definition must be defined.

Host notification dependencies work in a similar manner to service notification dependencies. If all of the notification dependency tests for the host pass, Centreon Engine will send notifications out for the host as it normally would. If even just one of the notification dependencies for a host fails, Centreon Engine will temporarily repress notifications for that (dependent) host. At some point in the future the notification dependency tests for the host may all pass. If this happens, Centreon Engine will start sending out notifications again as it normally would for the host. More information on the notification logic can be found [here](#).

State Stalking

Introduction State “stalking” is a feature which is probably not going to be used by most users. When enabled, it allows you to log changes in the output service and host checks even if the state of the host or service does not change. When stalking is enabled for a particular host or service, Centreon Engine will watch that host or service very carefully and log any changes it sees in the output of check results. As you’ll see, it can be very helpful to you in later analysis of the log files.

How Does It Work? Under normal circumstances, the result of a host or service check is only logged if the host or service has changed state since it was last checked. There are a few exceptions to this, but for the most part, that’s the rule.

If you enable stalking for one or more states of a particular host or service, Centreon Engine will log the results of the host or service check if the output from the check differs from the output from the previous check. Take the following example of eight consecutive checks of a service:

Service Check	Service State	Service Check Output	Logged Normally	Logged With Stalking
x	OK	RAID array optimal	.	.
x+1	OK	RAID array optimal	.	.
x+2	WARNING	RAID array degraded (1 drive bad, 1 hot spare rebuilding)	Yes	Yes
x+3	CRITICAL	RAID array degraded (2 drives bad, 1 hot spare online, 1 hot spare rebuilding)	Yes	Yes
x+4	CRITICAL	RAID array degraded (3 drives bad, 2 hot spares online)	.	Yes
x+5	CRITICAL	RAID array failed	.	Yes
x+6	CRITICAL	RAID array failed	.	.
x+7	CRITICAL	RAID array failed	.	.

Given this sequence of checks, you would normally only see two log entries for this catastrophe. The first one would occur at service check x+2 when the service changed from an OK state to a WARNING state. The second log entry would occur at service check x+3 when the service changed from a WARNING state to a CRITICAL state.

For whatever reason, you may like to have the complete history of this catastrophe in your log files. Perhaps to help explain to your manager how quickly the situation got out of control, perhaps just to laugh at it over a couple of drinks at the local pub...

Well, if you had enabled stalking of this service for CRITICAL states, you would have events at x+4 and x+5 logged in addition to the events at x+2 and x+3. Why is this? With state stalking enabled, Centreon Engine would have examined the output from each service check to see if it differed from the output of the previous check. If the output differed and the state of the service didn’t change between the two checks, the result of the newer service check would get logged.

A similar example of stalking might be on a service that checks your web server. If the check_http plugin first returns

a WARNING state because of a 404 error and on subsequent checks returns a WARNING state because of a particular pattern not being found, you might want to know that. If you didn't enable state stalking for WARNING states of the service, only the first WARNING state event (the 404 error) would be logged and you wouldn't have any idea (looking back in the archived logs) that future WARNING states were not due to a 404, but rather some text pattern that could not be found in the returned web page.

Should I Enable Stalking? First, you must decide if you have a real need to analyze archived log data to find the exact cause of a problem. You may decide you need this feature for some hosts or services, but not for all. You may also find that you only have a need to enable stalking for some host or service states, rather than all of them. For example, you may decide to enable stalking for WARNING and CRITICAL states of a service, but not for OK and UNKNOWN states.

The decision to to enable state stalking for a particular host or service will also depend on the plugin that you use to check that host or service. If the plugin always returns the same text output for a particular state, there is no reason to enable stalking for that state.

How Do I Enable Stalking? You can enable state stalking for hosts and services by using the `stalking_options` directive in *host and service definitions*.

How Does Stalking Differ From Volatile Services? *Volatile services* are similar, but will cause notifications and event handlers to run. Stalking is purely for logging purposes.

Caveats You should be aware that there are some potential pitfalls with enabling stalking. These all relate to the reporting functions (histogram, alert summary, etc.). Because state stalking will cause additional alert entries to be logged, the data produced by the reports will show evidence of inflated numbers of alerts.

As a general rule, I would suggest that you not enable stalking for hosts and services without thinking things through. Still, it's there if you need and want it.

Performance Data

Introduction Centreon Engine is designed to allow *plugins* to return optional performance data in addition to normal status data, as well as allow you to pass that performance data to external applications for processing. A description of the different types of performance data, as well as information on how to go about processing that data is described below...

Types of Performance Data There are two basic categories of performance data that can be obtained from Centreon Engine:

- Check performance data
- Plugin performance data

Check performance data is internal data that relates to the actual execution of a host or service check. This might include things like service check latency (i.e. how "late" was the service check from its scheduled execution time) and the number of seconds a host or service check took to execute. This type of performance data is available for all checks that are performed. The *HOSTEXECUTIONTIME* and *SERVICEEXECUTIONTIME* macros can be used to determine the number of seconds a host or service check was running and the *HOSTLATENCY* and *SERVICELATENCY* macros can be used to determine how "late" a regularly-scheduled host or service check was.

Plugin performance data is external data specific to the plugin used to perform the host or service check. Plugin-specific data can include things like percent packet loss, free disk space, processor load, number of current users, etc. - basically any type of metric that the plugin is measuring when it executes. Plugin-specific performance data

is optional and may not be supported by all plugins. Plugin-specific performance data (if available) can be obtained by using the *HOSTPERFDATA* and *SERVICEPERFDATA* macros. Read on for more information on how plugins can return performance data to Centreon Engine for inclusion in the *HOSTPERFDATA* and *SERVICEPERFDATA* macros.

Plugin Performance Data At a minimum, Centreon Engine plugins must return a single line of human-readable text that indicates the status of some type of measurable data. For example, the *check_ping* plugin might return a line of text like the following:

```
PING ok - Packet loss = 0%, RTA = 0.80 ms
```

With this simple type of output, the entire line of text is available in the *\$HOSTOUTPUT\$* or *\$SERVICEOUTPUT\$* macros (depending on whether this plugin was used as a host check or service check).

Plugins can return optional performance data in their output by sending the normal, human-readable text string that they usually would, followed by a pipe character (*|*), and then a string containing one or more performance data metrics. Let's take the *check_ping* plugin as an example and assume that it has been enhanced to return percent packet loss and average round trip time as performance data metrics. Sample output from the plugin might look like this:

```
PING ok - Packet loss = 0%, RTA = 0.80 ms | percent_packet_loss=0, rta=0.80
```

When Centreon Engine sees this plugin output format it will split the output into two parts:

- Everything before the pipe character is considered to be the “normal” plugin output and will be stored in either the *\$HOSTOUTPUT\$* or *\$SERVICEOUTPUT\$* macro
- Everything after the pipe character is considered to be the plugin-specific performance data and will be stored in the *\$HOSTPERFDATA\$* or *\$SERVICEPERFDATA\$* macro

In the example above, the *\$HOSTOUTPUT\$* or *\$SERVICEOUTPUT\$* macro would contain “PING ok - Packet loss = 0%, RTA = 0.80 ms” (without quotes) and the *\$HOSTPERFDATA\$* or *\$SERVICEPERFDATA\$* macro would contain “percent_packet_loss=0, rta=0.80” (without quotes).

Multiple lines of performance data (as well as normal text output) can be obtained from plugins, as described in the *plugin API documentation*.

Note: The Centreon Engine daemon doesn't directly process plugin performance data, so it doesn't really care what the performance data looks like. There aren't really any inherent limitations on the format or content of the performance data. However, if you are using an external addon to process the performance data (i.e. *PerfParse*), the addon may be expecting that the plugin returns performance data in a specific format. Check the documentation that comes with the addon for more information.

Processing Performance Data If you want to process the performance data that is available from Centreon Engine and the plugins, you'll need to do the following:

- Enable the *process_performance_data* option.
- Configure Centreon Engine so that performance data is either written to files and/or processed by executing commands.

Read on for information on how to process performance data by writing to files or executing commands.

Processing Performance Data Using Commands The most flexible way to process performance data is by having Centreon Engine execute commands (that you specify) to process or redirect the data for later processing by external applications. The commands that Centreon Engine executes to process host and service performance data are determined by the *host_perfdata_command* and *service_perfdata_command* options, respectively.

An example command definition that redirects service check performance data to a text file for later processing by another application is shown below:

```
define command{
    command_name store-service-perfdata
    command_line /bin/echo -e "$LASTSERVICECHECK$\t$HOSTNAME$\t$SERVICEDESC$\t$SERVICESTATE$\t$SERVICEEXECUTED"
}
```

Note: This method, while flexible, comes with a relatively high CPU overhead. If you're processing performance data for a large number of hosts and services, you'll probably want Centreon Engine to write performance data to files instead. This method is described in the next section.

Writing Performance Data To Files You can have Centreon Engine write all host and service performance data directly to text files using the *host_perfdata_file* and *service_perfdata_file* options. The format in which host and service performance data is written to those files is determined by the *template host_perfdata_file_template* and *service_perfdata_file_template* options.

An example file format template for service performance data might look like this:

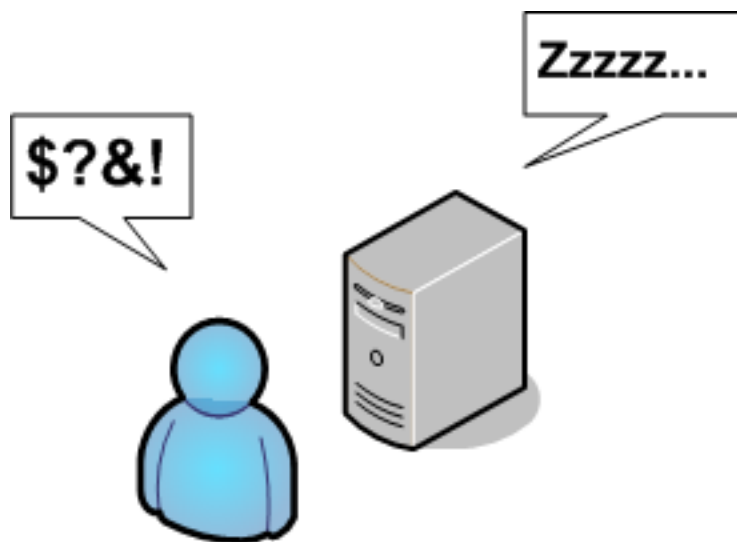
```
service_perfdata_file_template=[SERVICEPERFDATA]\t$TIMET$\t$HOSTNAME$\t$SERVICEDESC$\t$SERVICEEXECUTED
```

By default, the text files will be opened in “append” mode. If you need to change the modes to “write” or “non-blocking read/write” (useful when writing to pipes), you can use the *host_perfdata_file_mode* and *service_perfdata_file_mode* options.

Additionally, you can have Centreon Engine periodically execute commands to periodically process the performance data files (e.g. rotate them) using the *host_perfdata_file_processing_command* and *service_perfdata_file_processing_command* options. The interval at which these commands are executed are governed by the *host_perfdata_file_processing_interval* and *service_perfdata_file_processing_interval* options, respectively.

Scheduled Downtime

Introduction Centreon Engine allows you to schedule periods of planned downtime for hosts and service that you're monitoring. This is useful in the event that you actually know you're going to be taking a server down for an upgrade, etc.



Scheduling Downtime Once you schedule downtime for a host or service, Centreon Engine will add a comment to that host/service indicating that it is scheduled for downtime during the period of time you indicated. When that period of downtime passes, Centreon Engine will automatically delete the comment that it added. Nice, huh?

Fixed vs. Flexible Downtime When you schedule downtime for a host or service through the web interface you'll be asked if the downtime is fixed or flexible. Here's an explanation of how "fixed" and "flexible" downtime differs:

"Fixed" downtime starts and stops at the exact start and end times that you specify when you schedule it. Okay, that was easy enough...

"Flexible" downtime is intended for times when you know that a host or service is going to be down for X minutes (or hours), but you don't know exactly when that'll start. When you schedule flexible downtime, Centreon Engine will start the scheduled downtime sometime between the start and end times you specified. The downtime will last for as long as the duration you specified when you scheduled the downtime. This assumes that the host or service for which you scheduled flexible downtime either goes down (or becomes unreachable) or goes into a non-OK state sometime between the start and end times you specified. The time at which a host or service transitions to a problem state determines the time at which Centreon Engine actually starts the downtime. The downtime will then last for the duration you specified, even if the host or service recovers before the downtime expires. This is done for a very good reason. As we all know, you might think you've got a problem fixed, but then have to restart a server ten times before it actually works right. Smart, eh?

Triggered Downtime When scheduling host or service downtime you have the option of making it "triggered" downtime. What is triggered downtime, you ask? With triggered downtime the start of the downtime is triggered by the start of some other scheduled host or service downtime. This is extremely useful if you're scheduling downtime for a large number of hosts or services and the start time of the downtime period depends on the start time of another downtime entry. For instance, if you schedule flexible downtime for a particular host (because its going down for maintenance), you might want to schedule triggered downtime for all of that hosts's "children".

How Scheduled Downtime Affects Notifications When a host or service is in a period of scheduled downtime, Centreon Engine will not allow normal notifications to be sent out for the host or service. However, a "DOWN-TIMESTART" notification will get sent out for the host or service, which will serve to put any admins on notice that they won't receive upcoming problem alerts.

When the scheduled downtime is over, Centreon Engine will allow normal notifications to be sent out for the host or service again. A "DOWNTIMEEND" notification will get sent out notifying admins that the scheduled downtime is over, and they will start receiving normal alerts again.

If the scheduled downtime is cancelled prematurely (before it expires), a "DOWNTIMECANCELLED" notification will get sent out to the appropriate admins.

Overlapping Scheduled Downtime I like to refer to this as the "Oh crap, its not working" syndrome. You know what I'm talking about. You take a server down to perform a "routine" hardware upgrade, only to later realize that the OS drivers aren't working, the RAID array blew up, or the drive imaging failed and left your original disks useless to the world. Moral of the story is that any routine work on a server is quite likely to take three or four times as long as you had originally planned...

Let's take the following scenario:

- You schedule downtime for host A from 7:30pm-9:30pm on a Monday
- You bring the server down about 7:45pm Monday evening to start a hard drive upgrade
- After wasting an hour and a half battling with SCSI errors and driver incompatibilities, you finally get the machine to boot up
- At 9:15 you realize that one of your partitions is either hosed or doesn't seem to exist anywhere on the drive

- Knowing you're in for a long night, you go back and schedule additional downtime for host A from 9:20pm Monday evening to 1:30am Tuesday Morning.

If you schedule overlapping periods of downtime for a host or service (in this case the periods were 7:40pm-9:30pm and 9:20pm-1:30am), Centreon Engine will wait until the last period of scheduled downtime is over before it allows notifications to be sent out for that host or service. In this example notifications would be suppressed for host A until 1:30am Tuesday morning.

Adaptive Monitoring

Introduction Centreon Engine allows you to change certain commands and host and service check attributes during runtime. I'll refer to this feature as "adaptive monitoring". Please note that the adaptive monitoring features found in Centreon Engine will probably not be of much use to 99% of users, but they do allow you to do some neat things.

What Can Be Changed? The following service check attributes can be changed during runtime:

- Check command (and command arguments)
- Check interval
- Max check attempts
- Check timeperiod
- Event handler command (and command arguments)

The following host check attributes can be changed during runtime:

- Check command (and command arguments)
- Check interval
- Max check attempts
- Check timeperiod
- Event handler command (and command arguments)

The following global attributes can be changed during runtime:

- Global host event handler command (and command arguments)
- Global service event handler command (and command arguments)

External Commands For Adaptive Monitoring In order to change global or host- or service-specific attributes during runtime, you must submit the appropriate *external command* to Centreon Engine via the *external command file*. The table below lists the different attributes that may be changed during runtime, along with the external command to accomplish the job.

A full listing of external commands that can be used for adaptive monitoring (along with examples of how to use them) can be found online at the following [URL](#)

Note: When changing check commands, check timeperiods, or event handler commands, it is important to note that the new values for these options must have been defined before Centreon Engine was started. Any request to change a command or timeperiod to one which had not been defined when Centreon Engine was started is ignored. You can specify command arguments along with the actual command name - just separate individual arguments from the command name (and from each other) using bang (!) characters. More information on how arguments in command definitions are processed during runtime can be found in the documentation on *macros*.

Predictive Dependency Checks

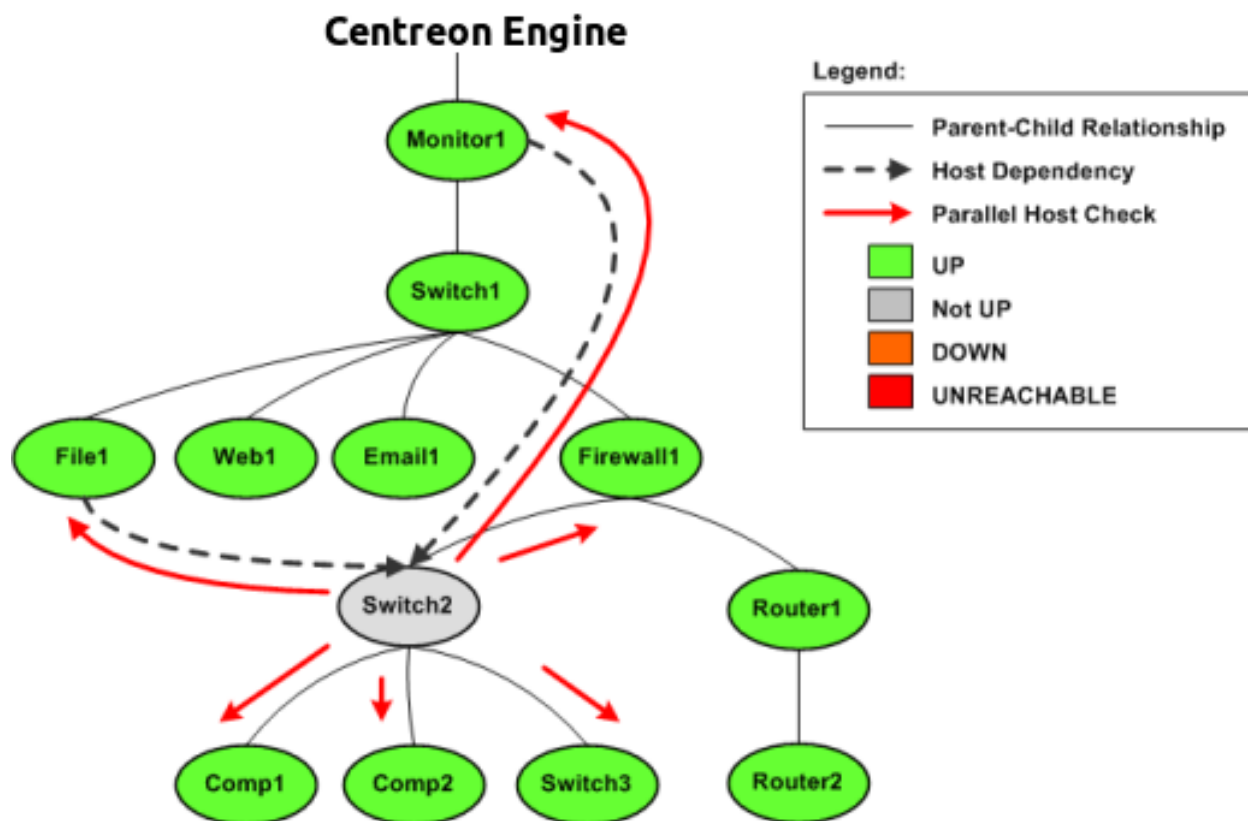
Introduction Host and service *dependencies* can be defined to allow you greater control over when checks are executed and when notifications are sent out. As dependencies are used to control basic aspects of the monitoring process, it is crucial to ensure that status information used in the dependency logic is as up to date as possible.

Centreon Engine allows you to enable predictive dependency checks for hosts and services to ensure that the dependency logic will have the most up-to-date status information when it comes to making decisions about whether to send out notifications or allow active checks of a host or service.

How Do Predictive Checks Work? The image below shows a basic diagram of hosts that are being monitored by Centreon Engine, along with their parent/child relationships and dependencies.

The Switch2 host in this example has just changed state from an UP state to a problem state. Centreon Engine needs to determine whether the host is DOWN or UNREACHABLE, so it will launch parallel checks of Switch2's immediate parents (Firewall1) and children (Comp1, Comp2, and Switch3). This is a normal function of the *host reachability* logic.

You will also notice that Switch2 is depending on Monitor1 and File1 for either notifications or check execution (which one is unimportant in this example). If predictive host dependency checks are enabled, Centreon Engine will launch parallel checks of Monitor1 and File1 at the same time it launches checks of Switch2's immediate parents and children. Centreon Engine does this because it knows that it will have to test the dependency logic in the near future (e.g. for purposes of notification) and it wants to make sure it has the most current status information for the hosts that take part in the dependency.



That's how predictive dependency checks work. Simple, eh?

Note: Predictive service dependency checks work in a similar manner to what is described above. Except, of course,

they deal with services instead of hosts.

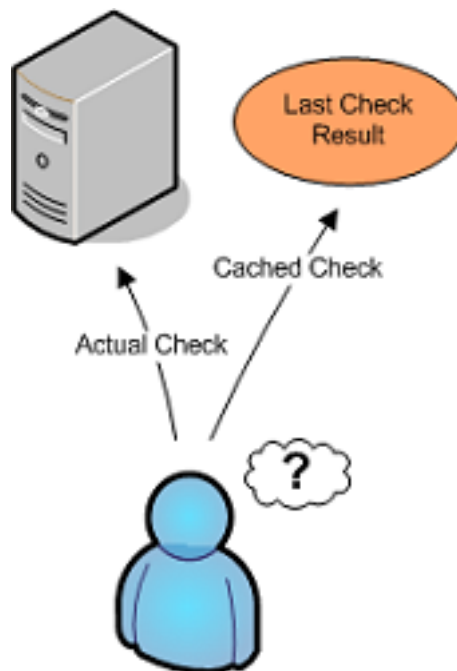
Enabling Predictive Checks Predictive dependency checks involve rather little overhead, so I would recommend that you enable them. In most cases, the benefits of having accurate information for the dependency logic outweighs the extra overhead imposed by these checks.

Enabling predictive dependency checks is easy:

- Predictive host dependency checks are controlled by the `enable_predictive_host_dependency_checks` option.
- Predictive service dependency checks are controlled by the `enable_predictive_service_dependency_checks` option.

Cached Checks Predictive dependency checks are on-demand checks and are therefore subject to the rules of *cached checks*. Cached checks can provide you with performance improvements by allowing Centreon Engine to forgo running an actual host or service check if it can use a relatively recent check result instead. More information on cached checks can be found [here](#).

Cached Checks



Introduction

The performance of Centreon Engine' monitoring logic can be significantly improved by implementing the use of cached checks. Cached checks allow Centreon Engine to forgo executing a host or service check command if it determines a relatively recent check result will do instead.

For On-Demand Checks Only Regularly scheduled host and service checks will not see a performance improvement with use of cached checks. Cached checks are only useful for improving the performance of on-demand host and service checks. Scheduled checks help to ensure that host and service states are updated regularly, which may result in a greater possibility their results can be used as cached checks in the future.

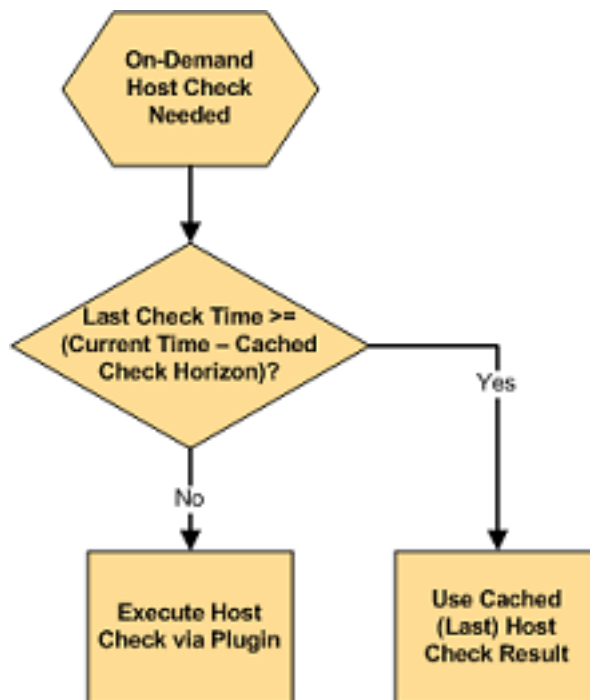
For reference, on-demand host checks occur...

- When a service associated with the host changes state.
- As needed as part of the *host reachability* logic.
- As needed for *predictive host dependency checks*.

And on-demand service checks occur...

- As needed for *predictive service dependency checks*.

Note: Unless you make use of service dependencies, Centreon Engine will not be able to use cached check results to improve the performance of service checks. Don't worry about that - its normal. Cached host checks are where the big performance improvements lie, and everyone should see a benefit there.



How Caching Works

When Centreon Engine needs to perform an on-demand host or service check, it will make a determination as to whether it can use a cached check result or if it needs to perform an actual check by executing a plugin. It does this by checking to see if the last check of the host or service occurred within the last X minutes, where X is the cached host or service check horizon.

If the last check was performed within the timeframe specified by the cached check horizon variable, Centreon Engine will use the result of the last host or service check and will not execute a new check. If the host or service has not yet been checked, or if the last check falls outside of the cached check horizon timeframe, Centreon Engine will execute a new host or service check by running a plugin.

What This Really Means Centreon Engine performs on-demand checks because it needs to know the current state of a host or service at that exact moment in time. Utilizing cached checks allows you to make Centreon Engine think that recent check results are “good enough” for determining the current state of hosts, and that it doesn't need to go out and actually re-check the status of that host or service.

The cached check horizon tells Centreon Engine how recent check results must be in order to reliably reflect the current state of a host or service. For example, with a cached check horizon of 30 seconds, you are telling Centreon Engine that if a host's state was checked sometime in the last 30 seconds, the result of that check should still be considered the current state of the host.

The number of cached check results that Centreon Engine can use versus the number of on-demand checks it has to actually execute can be considered the cached check “hit” rate. By increasing the cached check horizon to equal the regular check interval of a host, you could theoretically achieve a cache hit rate of 100%. In that case all on-demand checks of that host would use cached check results. What a performance improvement! But is it really? Probably not.

The reliability of cached check result information decreases over time. Higher cache hit rates require that previous check results are considered “valid” for longer periods of time. Things can change quickly in any network scenario, and there’s no guarantee that a server that was functioning properly 30 seconds ago isn’t on fire right now. There’s the tradeoff - reliability versus speed. If you have a large cached check horizon, you risk having unreliable check result values being used in the monitoring logic.

Centreon Engine will eventually determine the correct state of all hosts and services, so even if cached check results prove to unreliably represent their true value, Centreon Engine will only work with incorrect information for a short period of time. Even short periods of unreliable status information can prove to be a nuisance for admins, as they may receive notifications about problems which no longer exist.

There is no standard cached check horizon or cache hit rate that will be acceptable to every Centreon Engine users. Some people will want a short horizon timeframe and a low cache hit rate, while others will want a larger horizon timeframe and a larger cache hit rate (with a low reliability rate). Some users may even want to disable cached checks altogether to obtain a 100% reliability rate. Testing different horizon timeframes, and their effect on the reliability of status information, is the only way that an individual user will find the “right” value for their situation. More information on this is discussed below.

Configuration Variables The following variables determine the timeframes in which a previous host or service check result may be used as a cached host or service check result:

- The *cached_host_check_horizon* variable controls cached host checks.
- The *cached_service_check_horizon* variable controls cached service checks.

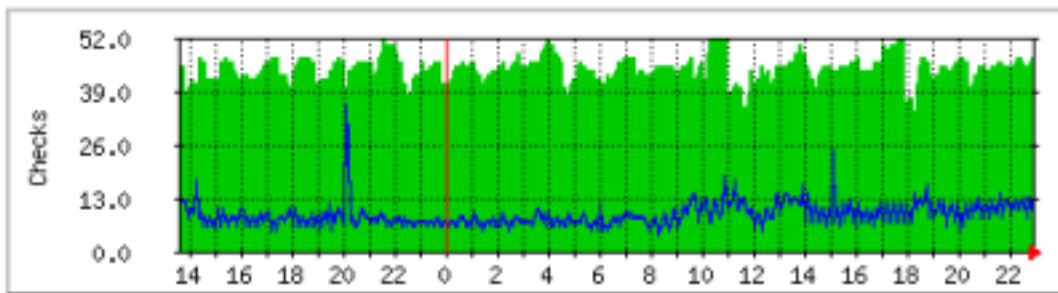
Optimizing Cache Effectiveness In order to make the most effective use of cached checks, you should:

- Schedule regular checks of your hosts
- Use MRTG to graph statistics for 1) on-demand checks and 2) cached checks
- Adjust cached check horizon variables to fit your needs

You can schedule regular checks of your hosts by specifying a value greater than 0 for *check_interval* option in your *host definitions*. If you do this, make sure that you set the *max_check_attempts* option to a value greater than 1, or it will cause a big performance hit. This potential performance hit is describe in detail [here](#).

Active Host Checks

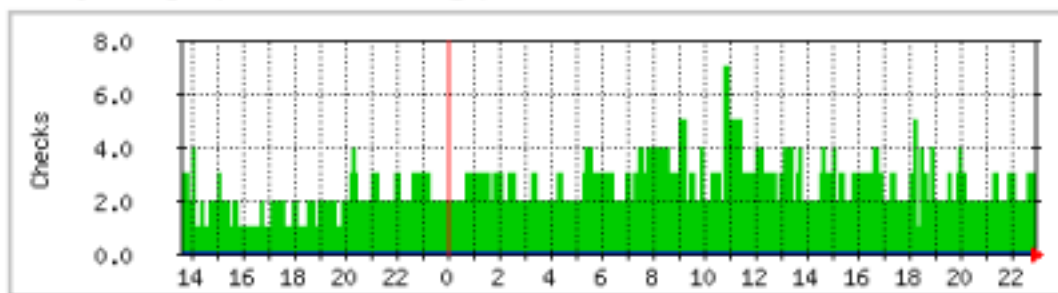
'Daily' Graph (5 Minute Average)



Max Scheduled Checks: 52.0 Average Scheduled Checks: 44.0 Current Scheduled Checks: 47.0
Max On-Demand Checks: 35.0 Average On-Demand Checks: 9.0 Current On-Demand Checks: 14.0

Cached Host Checks

'Daily' Graph (5 Minute Average)



Max Host Checks: 7.0 Average Host Checks: 2.0 Current Host Checks: 3.0

A good way to determine the proper value for the cached check horizon options is to compare how many on-demand checks Centreon Engine has to actually run versus how many it can use cached values for. The *centenginets* utility can produce information on cached checks.

The monitoring installation which produced the graphs above had:

- A total of 44 hosts, all of which were checked at regular intervals
- An average (regularly scheduled) host check interval of 5 minutes
- A *cached_host_check_horizon* of 15 seconds

The first MRTG graph shows how many regularly scheduled host checks compared to how many cached host checks have occurred. In this example, an average of 53 host checks occur every five minutes. 9 of these (17%) are on-demand checks.

The second MRTG graph shows how many cached host checks have occurred over time. In this example an average of 2 cached host checks occurs every five minutes.

Remember, cached checks are only available for on-demand checks. Based on the 5 minute averages from the graphs, we see that Centreon Engine is able to use cached host check results every 2 out of 9 times an on-demand check has to be run. That may not seem much, but these graphs represent a small monitoring environment. Consider that 2 out of 9 is 22% and you can start to see how this could significantly help improve host check performance in large environments. That percentage could be higher if the cached host check horizon variable value was increased, but that

would reduce the reliability of the cached host state information.

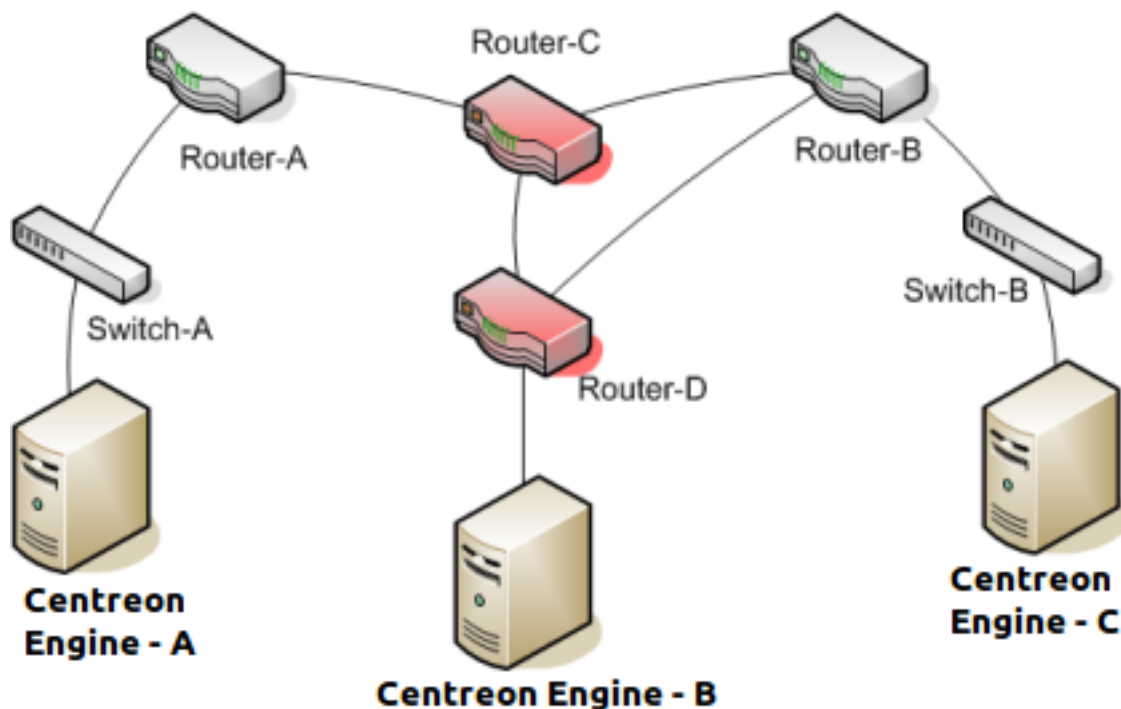
Once you've had a few hours or days worth of MRTG graphs, you should see how many host and service checks were done by executing plugins versus those that used cached check results. Use that information to adjust the cached check horizon variables appropriately for your situation. Continue to monitor the MRTG graphs over time to see how changing the horizon variables affected cached check statistics. Rinse and repeat as necessary.

Passive Host State Translation

Introduction When Centreon Engine receives passive host checks from remote sources (i.e other Centreon Engine instances in distributed or failover setups), the host state reported by the remote source may not accurately reflect the state of the host from Centreon Engine' view. As distributed and failover monitoring installations are fairly common, it is important to provide a mechanism for ensuring accurate host states between different instances of Centreon Engine.

Different World Views The image below shows a simplified view of a failover monitoring setup.

- Centreon Engine-A is the primary monitoring server, and is actively monitoring all switches and routers.
- Centreon Engine-B and Centreon Engine-C are backup monitoring servers, and are receiving passive check results from Centreon Engine-A
- Both Router-C and Router-D have suffered failures and are offline.



What states are Router-C and Router-D currently in? The answer depends on which Centreon Engine instance you ask.

- Centreon Engine-A sees Router-D as DOWN and Router-C as UNREACHABLE
- Centreon Engine-B should see Router-C as DOWN and Router-D as UNREACHABLE
- Centreon Engine-C should see both routers as being DOWN.

Each Centreon Engine instance has a different view of the network. The backup monitoring servers should not blindly accept passive host states from the primary monitoring server, or they will have incorrect information on the current state of the network.

Without translating passive host check results from the primary monitoring server (Centreon Engine-A), Centreon Engine-C would see Router-D as UNREACHABLE, when it is really DOWN based on its viewpoint. Similarly, the DOWN/UNREACHABLE states (from the viewpoint of Centreon Engine-A) for Router-C and Router-D should be flipped from the viewpoint of Centreon Engine-B.

Note: There may be some situations where you do not want Centreon Engine to translate DOWN/UNREACHABLE states from remote sources to their “correct” state from the viewpoint of the local Centreon Engine instance. For example, in distributed monitoring environments you may want the central Centreon Engine instance to know how distributed instances see their respective portions of the network.

Enabling State Translation By default, Centreon Engine will not automatically translate DOWN/UNREACHABLE states from passive check results. You will need to enable this feature if you need and want it.

The automatic translation of passive host check states is controlled by the *translate_passive_host_checks* variable. Enable it and Centreon Engine will automatically translate DOWN and UNREACHABLE states from remote sources to their correct state for the local instance of Centreon Engine.

Service and Host Check Scheduling

Introduction There were a lot of questions regarding how service checks are scheduled in certain situations, along with how the scheduling differs from when the checks are actually executed and their results are processed. We’ll try to go into a little more detail on how this all works ...

Configuration Options Before we begin, there are several configuration options that affect how service checks are scheduled, executed, and processed. For starters, each *service definition* contains three options that determine when and how each specific service check is scheduled and executed. Those three options are:

- *normal_check_interval*
- *retry_check_interval*
- *check_period*

There are also four configuration options in the *main configuration file* that affect service checks. These include:

- *service_inter_check_delay_method*
- *service_interleave_factor*
- *max_concurrent_checks*
- *check_result_reaper_frequency*

Note: The last directive affects host checks as well.

We’ll go into more detail on how all these options affect service check scheduling as we progress. First off, let’s see how services are initially scheduled when Centreon Engine first starts or restarts ...

Initial Scheduling When Centreon Engine (re)starts, it will attempt to schedule the initial check of all services in a manner that will minimize the load imposed on the local and remote hosts. This is done by spacing the initial service checks out, as well as interleaving them. The spacing of service checks (also known as the inter-check delay) is used to minimize/equalize the load on the local host running Centreon Engine and the interleaving is used to minimize/equalize load imposed on remote hosts. Both the inter-check delay and interleave functions are discussed below.

Even though service checks are initially scheduled to balance the load on both the local and remote hosts, things will eventually give in to the ensuing chaos and be a bit random. Reasons for this include the fact that services are not all checked at the same interval, some services take longer to execute than others, host and/or service problems can alter the timing of one or more service checks, etc. At least we try to get things off to a good start. Hopefully the initial scheduling will keep the load on the local and remote hosts fairly balanced as time goes by ...

Note: If you want to view the initial service check scheduling information, start Centreon Engine using the `-s` command line option. Doing so will display basic scheduling information (inter-check delay, interleave factor, first and last service check time, etc) and will create a new status log that shows the exact time that all services are initially scheduled. Because this option will overwrite the status log, you should not use it when another copy of Centreon Engine is running. Centreon Engine does not start monitoring anything when this argument is used.

Inter-Check Delay As mentioned before, Centreon Engine attempts to equalize the load placed on the machine that is running Centreon Engine by equally spacing out initial service checks. The spacing between consecutive service checks is called the inter-check delay. By giving a value to the `service_inter_check_delay_method` variable in the main config file, you can modify how this delay is calculated. We will discuss how the “smart” calculation works, as this is the setting you will want to use for normal operation.

When using the “smart” setting of the `service_inter_check_delay_method` variable, Centreon Engine will calculate an inter-check delay value by using the following calculation:

$$\text{inter-check delay} = (\text{average check interval for all services}) / (\text{total number of services})$$

Let’s take an example. Say you have 1,000 services that each have a normal check interval of 5 minutes (obviously some services are going to be checked at different intervals, but let’s look at an easy case...). The total check interval time for all services is 5,000 (1,000 * 5). That means that the average check interval for each service is 5 minutes (5,000 / 1,000). Give that information, we realize that (on average) we need to re-check 1,000 services every 5 minutes. This means that we should use an inter-check delay of 0.005 minutes (0.3 seconds) when spacing out the initial service checks. By spacing each service check out by 0.3 seconds, we can somewhat guarantee that Centreon Engine is scheduling and/or executing 3 new service checks every second. By spacing the checks out evenly over time like this, we can hope that the load on the local server that is running Centreon Engine remains somewhat balanced.

Service Interleaving As discussed above, the inter-check delay helps to equalize the load that Centreon Engine imposes on the local host. What about remote hosts? Is it necessary to equalize load on remote hosts? Why? Yes, it is important and yes, Centreon Engine can help out with this. If you monitor a large number of services on a remote host and the checks were not spread out, the remote host might think that it was the victim of a SYN attack if there were a lot of open connections on the same port. Plus, attempting to equalize the load on hosts is just a nice thing to do ...

By giving a value to the `service_interleave_factor` variable in the main config file, you can modify how the interleave factor is calculated. We will discuss how the “smart” calculation works, as this will probably be the setting you will want to use for normal operation. You can, however, use a pre-set interleave factor instead of having Centreon Engine calculate one for you. Also of note, if you use an interleave factor of 1, service check interleaving is basically disabled.

When using the “smart” setting of the `service_interleave_factor` variable, Centreon Engine will calculate an interleave factor by using the following calculation:

$$\text{interleave factor} = \text{ceil}(\text{total number of services} / \text{total number of hosts})$$

Let’s take an example. Say you have a total of 1,000 services and 150 hosts that you monitor. Centreon Engine would calculate the interleave factor to be 7. This means that when Centreon Engine schedules initial service checks it will

schedule the first one it finds, skip the next 6, schedule the next one, and so on ... This process will keep repeating until all service checks have been scheduled. Since services are sorted (and thus scheduled) by the name of the host they are associated with, this will help with minimizing/equalizing the load placed upon remote hosts.

The images below depict how service checks are scheduled when they are not interleaved (service_interleave_factor=1) and when they are interleaved with the service_interleave_factor variable equal to 4.

Host ↑	Service ↑	Status ↑	Last Check ↑	Duration ↑	Attempt ↑	Service Information
nt1	Average Processor Load	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:25 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:27 2001
	Paged Memory Use	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:29 2001
	Physical Memory Use	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:31 2001
	PING	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:33 2001
nt2	PING	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:35 2001
	Physical Memory Use	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:37 2001
	Paged Memory Use	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:39 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:41 2001
	Average Processor Load	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:43 2001
nt3	PING	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:45 2001
	Physical Memory Use	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:46 2001
	Paged Memory Use	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:48 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:50 2001
	Average Processor Load	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:52 2001
nt4	PING	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:54 2001
	Physical Memory Use	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:56 2001
	Paged Memory Use	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:04:58 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:05:00 2001
	Average Processor Load	PENDING	N/A	0d 0h 0m 13s*	0/3	Service check scheduled for Wed Aug 1 23:05:02 2001

Notice how checks are scheduled consecutively, in the order they appear in the list. This is a result of setting the service_interleave_factor to 1.

Non-Interleaved Checks

Host ↑	Service ↑	Status ↑	Last Check ↑	Duration ↑	Attempt ↑	Service Information
nt1	Average Processor Load	OK	08-01-2001 23:14:41	0d 0h 0m 26s	1/3	Load ok - 1-min avg=0.2%, 5-min avg=1.1%, 15-min avg=0.8%
	Drive C: Free Space	OK	08-01-2001 23:14:43	0d 0h 0m 24s	1/3	Disk space ok - 717 MB (23%) of 2996 MB used
	Paged Memory Use	OK	08-01-2001 23:14:45	0d 0h 0m 22s	1/3	Paged memory ok - 203.4 MB (38%) of 524.8 MB used
	Physical Memory Use	OK	08-01-2001 23:14:47	0d 0h 0m 20s	1/3	Physical memory ok - 135.8 MB (47%) of 287.4 MB used
	PING	OK	08-01-2001 23:14:49	0d 0h 0m 18s	1/3	PING ok - Packet loss = 0%, RTA = 0.30 ms
nt2	PING	OK	08-01-2001 23:14:51	0d 0h 0m 16s	1/3	PING ok - Packet loss = 0%, RTA = 0.20 ms
	Physical Memory Use	OK	08-01-2001 23:14:53	0d 0h 0m 14s	1/3	Physical memory ok - 74.8 MB (29%) of 255.4 MB used
	Paged Memory Use	OK	08-01-2001 23:14:55	0d 0h 0m 12s	1/3	Paged memory ok - 55.9 MB (11%) of 507.7 MB used
	Drive C: Free Space	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:14:57 2001
	Average Processor Load	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:14:59 2001
nt3	PING	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:15:01 2001
	Physical Memory Use	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:15:02 2001
	Paged Memory Use	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:15:04 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:15:06 2001
	Average Processor Load	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:15:08 2001
nt4	PING	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:15:10 2001
	Physical Memory Use	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:15:12 2001
	Paged Memory Use	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:15:14 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:15:16 2001
	Average Processor Load	PENDING	N/A	0d 0h 3m 20s*	0/3	Service check scheduled for Wed Aug 1 23:15:18 2001

Notice how services are checked consecutively, in the order they appear in the list. Compare this to the manner in which services are checked with a service_interleave_factor greater than 1.

Host ↑	Service ↑	Status ↑	Last Check ↑	Duration ↑	Attempt ↑	Service Information
nt1	Average Processor Load	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:33:06 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:34:14 2001
	Paged Memory Use	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:35:23 2001
	Physical Memory Use	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:36:31 2001
	PING	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:33:08 2001
nt2	PING	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:34:16 2001
	Physical Memory Use	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:35:25 2001
	Paged Memory Use	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:36:33 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:33:09 2001
	Average Processor Load	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:34:18 2001
nt3	PING	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:35:27 2001
	Physical Memory Use	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:36:35 2001
	Paged Memory Use	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:33:11 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:34:20 2001
	Average Processor Load	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:35:29 2001
nt4	PING	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:36:37 2001
	Physical Memory Use	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:33:13 2001
	Paged Memory Use	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:34:22 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:35:31 2001
	Average Processor Load	PENDING	N/A	0d 0h 0m 3s+	0/3	Service check scheduled for Wed Aug 1 22:36:39 2001

Notice how every 4th check in the list is PENDING. This is the fact that the automatically calculated service interleave factor in this case was 4.

Interleaved Checks

Host ↑	Service ↑	Status ↑	Last Check ↑	Duration ↑	Attempt ↑	Service Information
nt1	Average Processor Load	OK	08-01-2001 22:36:05	0d 0h 1m 2s	1/3	Load ok - 1-min avg=3.8%, 5-min avg=14.2%, 15-min avg=14.9%
	Drive C: Free Space	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:35:04 2001
	Paged Memory Use	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:36:13 2001
	Physical Memory Use	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:37:21 2001
	PING	OK	08-01-2001 22:36:05	0d 0h 1m 2s	1/3	PING ok - Packet loss = 0%, RTA = 0.30 ms
nt2	PING	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:35:06 2001
	Physical Memory Use	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:36:15 2001
	Paged Memory Use	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:37:23 2001
	Drive C: Free Space	OK	08-01-2001 22:36:05	0d 0h 1m 2s	1/3	Disk space ok - 2356 MB (78%) of 2996 MB used
	Average Processor Load	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:35:08 2001
nt3	PING	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:36:17 2001
	Physical Memory Use	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:37:25 2001
	Paged Memory Use	OK	08-01-2001 22:36:05	0d 0h 1m 2s	1/3	Paged memory ok - 531.1 MB (66%) of 804.2 MB used
	Drive C: Free Space	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:35:10 2001
	Average Processor Load	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:36:19 2001
nt4	PING	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:37:27 2001
	Physical Memory Use	OK	08-01-2001 22:36:06	0d 0h 1m 1s	1/3	Physical memory ok - 350.6 MB (68%) of 511.4 MB used
	Paged Memory Use	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:35:12 2001
	Drive C: Free Space	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:36:21 2001
	Average Processor Load	PENDING	N/A	0d 0h 3m 56s+	0/3	Service check scheduled for Wed Aug 1 22:37:29 2001

Since the service checks are interleaved, the load imposed on remote servers by the checks at any given time are somewhat balanced.

Host ↑	Service ↑	Status ↑	Last Check ↑	Duration ↑	Attempt ↑	Service Information
nt1	Average Processor Load	OK	08-01-2001 22:36:05	0d 0h 4m 19s	1/3	Load ok - 1-min avg=3.8%, 5-min avg=14.2%, 15-min avg=14.9%
	Drive C: Free Space	OK	08-01-2001 22:37:04	0d 0h 3m 20s	1/3	Disk space ok - 717 MB (23%) of 2996 MB used
	Paged Memory Use	OK	08-01-2001 22:37:12	0d 0h 3m 12s	1/3	Paged memory ok - 221.4 MB (42%) of 524.8 MB used
	Physical Memory Use	PENDING	N/A	0d 0h 7m 13s+	0/3	Service check scheduled for Wed Aug 1 22:37:21 2001
	PING	OK	08-01-2001 22:36:05	0d 0h 4m 19s	1/3	PING ok - Packet loss = 0%, RTA = 0.30 ms
nt2	PING	OK	08-01-2001 22:37:04	0d 0h 3m 20s	1/3	PING ok - Packet loss = 0%, RTA = 0.20 ms
	Physical Memory Use	OK	08-01-2001 22:37:12	0d 0h 3m 12s	1/3	Physical memory ok - 74.8 MB (29%) of 255.4 MB used
	Paged Memory Use	PENDING	N/A	0d 0h 7m 13s+	0/3	Service check scheduled for Wed Aug 1 22:37:23 2001
	Drive C: Free Space	OK	08-01-2001 22:36:05	0d 0h 4m 19s	1/3	Disk space ok - 2356 MB (78%) of 2996 MB used
	Average Processor Load	OK	08-01-2001 22:37:04	0d 0h 3m 20s	1/3	Load ok - 1-min avg=2.6%, 5-min avg=2.0%, 15-min avg=1.6%
nt3	PING	OK	08-01-2001 22:37:12	0d 0h 3m 12s	1/3	PING ok - Packet loss = 0%, RTA = 0.40 ms
	Physical Memory Use	PENDING	N/A	0d 0h 7m 13s+	0/3	Service check scheduled for Wed Aug 1 22:37:25 2001
	Paged Memory Use	OK	08-01-2001 22:36:05	0d 0h 4m 19s	1/3	Paged memory ok - 531.1 MB (66%) of 804.2 MB used
	Drive C: Free Space	OK	08-01-2001 22:37:04	0d 0h 3m 20s	1/3	Disk space ok - 792 MB (26%) of 2996 MB used
	Average Processor Load	OK	08-01-2001 22:37:12	0d 0h 3m 12s	1/3	Load ok - 1-min avg=5.0%, 5-min avg=4.9%, 15-min avg=5.3%
nt4	PING	PENDING	N/A	0d 0h 7m 13s+	0/3	Service check scheduled for Wed Aug 1 22:37:27 2001
	Physical Memory Use	OK	08-01-2001 22:36:06	0d 0h 4m 18s	1/3	Physical memory ok - 350.6 MB (68%) of 511.4 MB used
	Paged Memory Use	OK	08-01-2001 22:37:04	0d 0h 3m 20s	1/3	Paged memory ok - 448.8 MB (60%) of 740.1 MB used
	Drive C: Free Space	OK	08-01-2001 22:37:12	0d 0h 3m 12s	1/3	Disk space ok - 907 MB (30%) of 2996 MB used
	Average Processor Load	PENDING	N/A	0d 0h 7m 13s+	0/3	Service check scheduled for Wed Aug 1 22:37:29 2001

This shows how service checks have been interleaved after three "passes".

FIRST PASS

SECOND PASS

THIRD PASS

Maximum Concurrent Service Checks In order to prevent Centreon Engine from consuming all of your CPU resources, you can restrict the maximum number of concurrent service checks that can be running at any given time. This is controlled by using the *max_concurrent_checks* option in the main config file.

The good thing about this setting is that you can regulate Centreon Engine' CPU usage. The down side is that service checks may fall behind if this value is set too low. When it comes time to execute a service check, Centreon Engine will make sure that no more than x service checks are either being executed or waiting to have their results processed (where x is the number of checks you specified for the *max_concurrent_checks* option). If that limit has been reached, Centreon Engine will postpone the execution of any pending checks until some of the previous checks have completed. So how does one determine a reasonable value for the *max_concurrent_checks* option ?

First off, you need to know the following things ...

- the inter-check delay that Centreon Engine uses to initially schedule service checks (use the -s command line argument to check this)
- the frequency (in seconds) of reaper events, as specified by the *check_result_reaper_frequency* variable in the main config file.
- a general idea of the average time that service checks actually take to execute (most plugins timeout after 10 seconds, so the average is probably going to be lower)

Next, use the following calculation to determine a reasonable value for the maximum number of concurrent checks that are allowed:

```
max. concurrent checks = ceil(max(check result reaper frequency,  
average check execution time) / inter-check delay)
```

The calculated number should provide a reasonable starting point for the *max_concurrent_checks* variable. You may have to increase this value a bit if service checks are still falling behind schedule or decrease it if Centreon Engine is hogging too much CPU time.

Let's say you are monitoring 875 services, each with an average check interval of 2 minutes. That means that your inter-check delay is going to be 0.137 seconds. If you set the check result reaper frequency to be 10 seconds, you can calculate a rough value for the max. number of concurrent checks as follows (we'll assume that the average execution time for service checks is less than 10 seconds):

```
max. concurrent checks = ceil(10 / 0.137)
```

In this case, the calculated value is going to be 73. This makes sense because (on average) Centreon Engine is going to be executing just over 7 new service checks per second and it only processes service check results every 10 seconds. That means at given time there will be a just over 70 service checks that are either being executed or waiting to have their results processed. In this case, we would probably recommend bumping the max. concurrent checks value up to 80, since there will be delays when Centreon Engine processes service check results and does its other work. Obviously, you're going to have test and tweak things a bit to get everything running smoothly on your system, but hopefully this provided some general guidelines ...

Time Constraints The *check_period* option determines the *time period* during which Centreon Engine can run checks of the service. Regardless of what status a particular service is in, if the time that it is actually executed is not a valid time within the time period that has been specified, the check will not be executed. Instead, Centreon Engine will reschedule the service check for the next valid time in the time period. If the check can be run (e.g. the time is valid within the time period), the service check is executed.

Note: Even though a service check may not be able to be executed at a given time, Centreon Engine may still schedule it to be run at that time. This is most likely to happen during the initial scheduling of services, although it may happen in other instances as well. This does not mean that Centreon Engine will execute the check ! When it comes time to actually execute a service check, Centreon Engine will verify that the check can be run at the current time. If it cannot, Centreon Engine will not execute the service check, but will instead just reschedule it for a later time. Don't let this

one throw you confuse you ! The scheduling and execution of service checks are two distinctly different (although related) things.

Normal Scheduling In an ideal world you wouldn't have network problems. But if that were the case, you wouldn't need a network monitoring tool. Anyway, when things are running smoothly and a service is in an OK state, we'll call that "normal". Service checks are normally scheduled at the frequency specified by the *check_interval* option. That's it. Simple, huh ?

Scheduling During Problems So what happens when there are problems with a service ? Well, one of the things that happens is the service check scheduling changes. If you've configured the *max_attempts* option of the service definition to be something greater than 1, Centreon Engine will recheck the service before deciding that a real problem exists. While the service is being rechecked (up to *max_attempts* times) it is considered to be in a "soft" state (as described here) and the service checks are rescheduled at a frequency determined by the *retry_interval* option.

If Centreon Engine rechecks the service *max_attempts* times and it is still in a non-OK state, Centreon Engine will put the service into a "hard" state, send out notifications to contacts (if applicable), and start rescheduling future checks of the service at a frequency determined by the *check_interval* option.

As always, there are exceptions to the rules. When a service check results in a non-OK state, Centreon Engine will check the host that the service is associated with to determine whether or not is up (see the note below for info on how this is done). If the host is not up (i.e. it is either down or unreachable), Centreon Engine will immediately put the service into a hard non-OK state and it will reset the current attempt number to 1. Since the service is in a hard non-OK state, the service check will be rescheduled at the normal frequency specified by the *check_interval* option instead of the *retry_interval* option.

Host Checks Unlike service checks, host checks are not scheduled on a regular basis. Instead they are run on demand, as Centreon Engine sees a need. This is a common question asked by users, so it needs to be clarified.

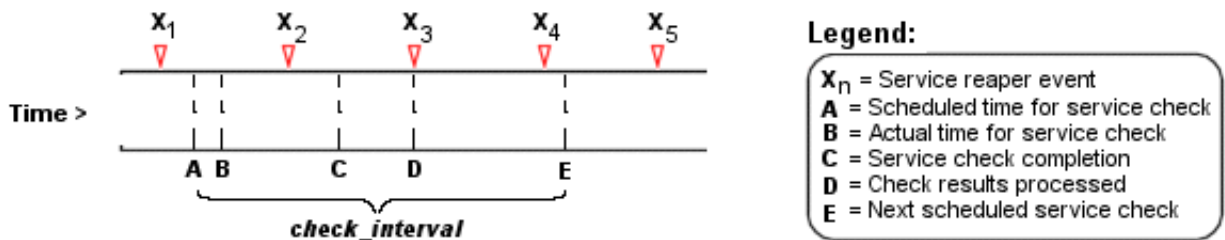
One instance where Centreon Engine checks the status of a host is when a service check results in a non-OK status. Centreon Engine checks the host to decide whether or not the host is up, down, or unreachable. If the first host check returns a non-OK state, Centreon Engine will keep pounding out checks of the host until either (a) the maximum number of host checks (specified by the *max_attempts* option in the host definition) is reached or (b) a host check results in an OK state.

Also of note - when Centreon Engine is check the status of a host, it holds off on doing anything else (executing new service checks, processing other service check results, etc). This can slow things down a bit and cause pending service checks to be delayed for a while, but it is necessary to determine the status of the host before Centreon Engine can take any further action on the service(s) that are having problems.

Scheduling Delays It should be noted that service check scheduling and execution is done on a best effort basis. Individual service checks are considered to be low priority events in Centreon Engine, so they can get delayed if high priority events need to be executed. Examples of high priority events include log file rotations, external command checks, and check results reaper events. Additionally, host checks will slow down the execution and processing of service checks.

Scheduling Example The scheduling of service checks, their execution, and the processing of their results can be a bit difficult to understand, so let's look at a simple example. Look at the diagram below - we'll refer to it as we explain how things are done.

Service Check Timing



First off, the **X :sub:'n'** events are check result reaper events that are scheduled at a frequency specified by the *check_result_reaper_frequency* option in the main config file. Check result reaper events do the work of gathering and processing service check results. They serve as the core logic for Centreon Engine, kicking off host checks, event handlers and notifications as necessary.

For the example here, a service has been scheduled to be executed at time **A**. However, Centreon Engine got behind in its event queue, so the check was not actually executed until time **B**. The service check finished executing at time **C**, so the difference between points **C** and **B** is the actual amount of time that the check was running.

The results of the service check are not processed immediately after the check is done executing. Instead, the results are saved for later processing by a check result reaper event. The next check result reaper event occurs at time **D**, so that is approximately the time that the results are processed (the actual time may be later than **D** since other service check results may be processed before this one).

At the time that the check result reaper event processes the service check results, it will reschedule the next service check and place it into Centreon Engine's event queue. We'll assume that the service check resulted in an OK status, so the next check at time **E** is scheduled after the originally scheduled check time by a length of time specified by the *check_interval* option. Note that the service is *not* rescheduled based off the time that it was actually executed ! There is one exception to this (isn't there always ?) - if the time that the service check is actually executed (point **B**) occurs after the next service check time (point **E**), Centreon Engine will compensate by adjusting the next check time. This is done to ensure that Centreon Engine doesn't go nuts trying to keep up with service checks if it comes under heavy load. Besides, what's the point of scheduling something in the past ... ?

Service Definition Options That Affect Scheduling Each service definition contains a *normal_check_interval* and *retry_check_interval* option. Hopefully this will clarify what these two options do, how they relate to the *max_check_attempts* option in the service definition, and how they affect the scheduling of the service.

First off, the *normal_check_interval* option is the interval at which the service is checked under "normal" circumstances. "Normal" circumstances mean whenever the service is in an OK state or when its in a *hard* non-OK state.

When a service first changes from an OK state to a non-OK state, Centreon Engine gives you the ability to temporarily slow down or speed up the interval at which subsequent checks of that service will occur. When the service first changes state, Centreon Engine will perform up to *max_check_attempts-1* retries of the service check before it decides its a real problem. While the service is being retried, it is scheduled according to the *retry_check_interval* option, which might be faster or slower than the normal *normal_check_interval* option. While the service is being rechecked (up to *max_check_attempts-1* times), the service is in a *soft* state. If the service is rechecked *max_check_attempts-1* times and it is still in a non-OK state, the service turns into a *hard* state and is subsequently rescheduled at the normal rate specified by the *check_interval* option.

On a side note, if you specify a value of 1 for the *max_check_attempts* option, the service will not ever be checked at the interval specified by the *retry_check_interval* option. Instead, it immediately turns into a *hard* state and is subsequently rescheduled at the rate specified by the *normal_check_interval* option.

Host Check Directives Most of the above applies to host checks as well.

Time-Saving Tricks For Object Definitions

Introduction This documentation attempts to explain how you can exploit the (somewhat) hidden features of *template-based object definitions* to save your sanity. How so, you ask? Several types of objects allow you to specify multiple host names and/or hostgroup names in definitions, allowing you to “copy” the object definition to multiple hosts or services. I’ll cover each type of object that supports these features separately. For starters, the object types which support this time-saving feature are as follows:

- *Services*
- *Service escalations*
- *Service dependencies*
- *Host escalations*
- *Host dependencies*
- *Hostgroups*

Object types that are not listed above (i.e. timeperiods, commands, etc.) do not support the features I’m about to describe.

Regular Expression Matching The examples I give below use “standard” matching of object names and *require use_regexp_matching* to be *disabled*.

If you wish, you can enable regular expression matching for object names by using the *use_regexp_matching* config option. By default, regular expression matching will only be used in object names that contain -, ?, +, or .. If you want regular expression matching to be used on all object names, enable the *use_true_regexp_matching* config option. Regular expressions can be used in any of the fields used in the examples below (host names, hostgroup names, service names, and servicegroup names).

Note: Be careful when enabling regular expression matching - you may have to change your config file, since some directives that you might not want to be interpreted as a regular expression just might be! Any problems should become evident once you verify your configuration.

Service Definitions **Multiple Hosts:** If you want to create identical *services* that are assigned to multiple hosts, you can specify multiple hosts in the *host_name* directive. The definition below would create a service called SOMESERVICE on hosts HOST1 through HOSTN. All the instances of the SOMESERVICE service would be identical (i.e. have the same check command, max check attempts, notification period, etc.):

```
define service{
    host_name          HOST1,HOST2,HOST3,...,HOSTN
    service_description SOMESERVICE
    other service directives ...
}
```

All Hosts In Multiple Hostgroups: If you want to create identical services that are assigned to all hosts in one or more hostgroups, you can do so by creating a single service definition. How? The *hostgroup_name* directive allows you to specify the name of one or more hostgroups that the service should be created for. The definition below would create a service called SOMESERVICE on all hosts that are members of hostgroups HOSTGROUP1 through HOSTGROUPN. All the instances of the SOMESERVICE service would be identical (i.e. have the same check command, max check attempts, notification period, etc.):

```
define service{
    hostgroup_name      HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
    service_description SOMESERVICE
}
```

```

    other service directives ...
}

```

All Hosts: If you want to create identical services that are assigned to all hosts that are defined in your configuration files, you can use a wildcard in the `host_name` directive. The definition below would create a service called `SOMESERVICE` on all hosts that are defined in your configuration files. All the instances of the `SOMESERVICE` service would be identical (i.e. have the same check command, max check attempts, notification period, etc.):

```

define service{
    host_name          *
    service_description SOMESERVICE
    other service directives ...
}

```

Excluding Hosts: If you want to create identical services on numerous hosts or hostgroups, but would like to exclude some hosts from the definition, this can be accomplished by preceding the host or hostgroup with a `!` symbol:

```

define service{
    host_name          HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
    hostgroup_name      HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
    service_description SOMESERVICE
    other service directives ...
}

```

Service Escalation Definitions Multiple Hosts: If you want to create *service escalations* for services of the same name/description that are assigned to multiple hosts, you can specify multiple hosts in the `host_name` directive. The definition below would create a service escalation for services called `SOMESERVICE` on hosts `HOST1` through `HOSTN`. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.):

```

define serviceescalation{
    host_name          HOST1,HOST2,HOST3,...,HOSTN
    service_description SOMESERVICE
    other escalation directives ...
}

```

All Hosts In Multiple Hostgroups: If you want to create service escalations for services of the same name/description that are assigned to all hosts in in one or more hostgroups, you can do use the `hostgroup_name` directive. The definition below would create a service escalation for services called `SOMESERVICE` on all hosts that are members of hostgroups `HOSTGROUP1` through `HOSTGROUPN`. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.):

```

define serviceescalation{
    hostgroup_name      HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
    service_description SOMESERVICE
    other escalation directives ...
}

```

All Hosts: If you want to create identical service escalations for services of the same name/description that are assigned to all hosts that are defined in your configuration files, you can use a wildcard in the `host_name` directive. The definition below would create a service escalation for all services called `SOMESERVICE` on all hosts that are defined in your configuration files. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.):

```

define serviceescalation{
    host_name          *
    service_description SOMESERVICE
}

```

```

    other escalation directives ...
}

```

Excluding Hosts: If you want to create identical services escalations for services on numerous hosts or hostgroups, but would like to exclude some hosts from the definition, this can be accomplished by preceding the host or hostgroup with a ! symbol:

```

define serviceescalation{
    host_name          HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
    hostgroup_name     HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
    service_description SOMEERVICE
    other escalation directives ...
}

```

All Services On Same Host: If you want to create *service escalations* for all services assigned to a particular host, you can use a wildcard in the service_description directive. The definition below would create a service escalation for all services on host HOST1. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

If you feel like being particularly adventurous, you can specify a wildcard in both the host_name and service_description directives. Doing so would create a service escalation for all services that you've defined in your configuration files:

```

define serviceescalation{
    host_name          HOST1
    service_description *
    other escalation directives ...
}

```

Multiple Services On Same Host: If you want to create *service escalations* for all multiple services assigned to a particular host, you can use a specify more than one service description in the service_description directive. The definition below would create a service escalation for services SERVICE1 through SERVICEN on host HOST1. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.):

```

define serviceescalation{
    host_name          HOST1
    service_description SERVICE1,SERVICE2,...,SERVICEN
    other escalation directives ...
}

```

All Services In Multiple Servicegroups: If you want to create service escalations for all services that belong in one or more servicegroups, you can do use the servicegroup_name directive. The definition below would create service escalations for all services that are members of servicegroups SERVICEGROUP1 through SERVICEGROUPN. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.):

```

define serviceescalation{
    servicegroup_name SERVICEGROUP1,SERVICEGROUP2,...,SERVICEGROUPN
    other escalation directives ...
}

```

Service Dependency Definitions **Multiple Hosts:** If you want to create *service dependencies* for services of the same name/description that are assigned to multiple hosts, you can specify multiple hosts in the host_name and or dependent_host_name directives. In the example below, service SERVICE2 on hosts HOST3 and HOST4 would be dependent on service SERVICE1 on hosts HOST1 and HOST2. All the instances of the service dependencies would be identical except for the host names (i.e. have the same notification failure criteria, etc.):

```
define servicedependency{
    host_name                HOST1,HOST2
    service_description      SERVICE1
    dependent_host_name      HOST3,HOST4
    dependent_service_description SERVICE2
    other dependency directives ...
}
```

All Hosts In Multiple Hostgroups: If you want to create service dependencies for services of the same name/description that are assigned to all hosts in one or more hostgroups, you can do use the `hostgroup_name` and/or `dependent_hostgroup_name` directives. In the example below, service `SERVICE2` on all hosts in hostgroups `HOSTGROUP3` and `HOSTGROUP4` would be dependent on service `SERVICE1` on all hosts in hostgroups `HOSTGROUP1` and `HOSTGROUP2`. Assuming there were five hosts in each of the hostgroups, this definition would be equivalent to creating 100 single service dependency definitions! All the instances of the service dependency would be identical except for the host names (i.e. have the same notification failure criteria, etc.):

```
define servicedependency{
    hostgroup_name           HOSTGROUP1,HOSTGROUP2
    service_description      SERVICE1
    dependent_hostgroup_name HOSTGROUP3,HOSTGROUP4
    dependent_service_description SERVICE2
    other dependency directives ...
}
```

All Services On A Host: If you want to create service dependencies for all services assigned to a particular host, you can use a wildcard in the `service_description` and/or `dependent_service_description` directives. In the example below, all services on host `HOST2` would be dependent on all services on host `HOST1`. All the instances of the service dependencies would be identical (i.e. have the same notification failure criteria, etc.):

```
define servicedependency{
    host_name                HOST1
    service_description      *
    dependent_host_name      HOST2
    dependent_service_description *
    other dependency directives ...
}
```

Multiple Services On A Host: If you want to create service dependencies for multiple services assigned to a particular host, you can specify more than one service description in the `service_description` and/or `dependent_service_description` directives as follows:

```
define servicedependency{
    host_name                HOST1
    service_description      SERVICE1,SERVICE2,...,SERVICEN
    dependent_host_name      HOST2
    dependent_service_description SERVICE1,SERVICE2,...,SERVICEN
    other dependency directives ...
}
```

All Services In Multiple Servicegroups: If you want to create service dependencies for all services that belong in one or more servicegroups, you can do use the `servicegroup_name` and/or `dependent_servicegroup_name` directive as follows:

```
define servicedependency{
    servicegroup_name        SERVICEGROUP1,SERVICEGROUP2,...,SERVICEGROUPN
    dependent_servicegroup_name SERVICEGROUP3,SERVICEGROUP4,...SERVICEGROUPN
    other dependency directives ...
}
```

Same Host Dependencies: If you want to create service dependencies for multiple services that are dependent on services on the same host, leave the `dependent_host_name` and `dependent_hostgroup_name` directives empty. The example below assumes that hosts HOST1 and HOST2 have at least the following four services associated with them: SERVICE1, SERVICE2, SERVICE3, and SERVICE4. In this example, SERVICE3 and SERVICE4 on HOST1 will be dependent on both SERVICE1 and SERVICE2 on HOST1. Similarly, SERVICE3 and SERVICE4 on HOST2 will be dependent on both SERVICE1 and SERVICE2 on HOST2:

```
define servicedependency{
    host_name                HOST1,HOST2
    service_description       SERVICE1,SERVICE2
    dependent_service_description SERVICE3,SERVICE4
    other dependency directives ...
}
```

Host Escalation Definitions Multiple Hosts: If you want to create *host escalations* for multiple hosts, you can specify multiple hosts in the `host_name` directive. The definition below would create a host escalation for hosts HOST1 through HOSTN. All the instances of the host escalation would be identical (i.e. have the same contact groups, notification interval, etc.):

```
define hostescalation{
    host_name HOST1,HOST2,HOST3,...,HOSTN
    other escalation directives ...
}
```

All Hosts In Multiple Hostgroups: If you want to create host escalations for all hosts in one or more hostgroups, you can use the `hostgroup_name` directive. The definition below would create a host escalation on all hosts that are members of hostgroups HOSTGROUP1 through HOSTGROUPN. All the instances of the host escalation would be identical (i.e. have the same contact groups, notification interval, etc.):

```
define hostescalation{
    hostgroup_name HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
    other escalation directives ...
}
```

All Hosts: If you want to create identical host escalations for all hosts that are defined in your configuration files, you can use a wildcard in the `host_name` directive. The definition below would create a hosts escalation for all hosts that are defined in your configuration files. All the instances of the host escalation would be identical (i.e. have the same contact groups, notification interval, etc.):

```
define hostescalation{
    host_name *
    other escalation directives ...
}
```

Excluding Hosts: If you want to create identical host escalations on numerous hosts or hostgroups, but would like to exclude some hosts from the definition, this can be accomplished by preceding the host or hostgroup with a ! symbol:

```
define hostescalation{
    host_name        HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
    hostgroup_name    HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
    other escalation directives ...
}
```

Host Dependency Definitions Multiple Hosts: If you want to create *host dependencies* for multiple hosts, you can specify multiple hosts in the `host_name` and/or `dependent_host_name` directives. The definition below would be equivalent to creating six separate host dependencies. In the example above, hosts HOST3, HOST4 and HOST5 would

be dependent upon both HOST1 and HOST2. All the instances of the host dependencies would be identical except for the host names (i.e. have the same notification failure criteria, etc.):

```
define hostdependency{
    host_name          HOST1,HOST2
    dependent_host_name HOST3,HOST4,HOST5
    other dependency directives ...
}
```

All Hosts In Multiple Hostgroups: If you want to create host escalations for all hosts in in one or more hostgroups, you can do use the `hostgroup_name` and `/or dependent_hostgroup_name` directives. In the example below, all hosts in hostgroups HOSTGROUP3 and HOSTGROUP4 would be dependent on all hosts in hostgroups HOSTGROUP1 and HOSTGROUP2. All the instances of the host dependencies would be identical except for host names (i.e. have the same notification failure criteria, etc.):

```
define hostdependency{
    hostgroup_name      HOSTGROUP1,HOSTGROUP2
    dependent_hostgroup_name HOSTGROUP3,HOSTGROUP4
    other dependency directives ...
}
```

Hostgroups All Hosts: If you want to create a hostgroup that has all hosts that are defined in your configuration files as members, you can use a wildcard in the members directive. The definition below would create a hostgroup called HOSTGROUP1 that has all hosts that are defined in your configuration files as members:

```
define hostgroup{
    hostgroup_name HOSTGROUP1
    members        *
    other hostgroup directives ...
}
```

Tuning Centreon Engine For Maximum Performance

Introduction So you've finally got Centreon Engine up and running and you want to know how you can tweak it a bit. Tuning Centreon Engine to increase performance can be necessary when you start monitoring a large number (> 1,000) of hosts and services. Here are a few things to look at for optimizing Centreon Engine...

Optimization Tips

- Use large installation tweaks. Enabling the `use_large_installation_tweaks` option may provide you with better performance. Read more about what this option does [here](#).
- Disable environment macros. Macros are normally made available to check, notification, event handler, etc. commands as environment variables. This can be a problem in a large Centreon Engine installation, as it consumes some additional memory and (more importantly) more CPU. If your scripts don't need to access the macros as environment variables (e.g. you pass all necessary macros on the command line), you don't need this feature. You can prevent macros from being made available as environment variables by using the `enable_environment_macros` option.
- Check Result Reaper Frequency. The `check_result_reaper_frequency` variable determines how often Centreon Engine should check for host and service check results that need to be processed. The maximum amount of time it can spend processing those results is determined by the max reaper time (see below). If your reaper frequency is too high (too infrequent), you might see high latencies for host and service checks.
- Max Reaper Time. The `max_check_result_reaper_time` variables determines the maximum amount of time the Centreon Engine daemon can spend processing the results of host and service checks before moving on to other

things - like executing new host and service checks. Too high of a value can result in large latencies for your host and service checks. Too low of a value can have the same effect. If you're experiencing high latencies, adjust this variable and see what effect it has.

- Adjust buffer slots. You may need to adjust the value of the *external_command_buffer_slots* option.
- Check service latencies to determine best value for maximum concurrent checks. Centreon Engine can restrict the number of maximum concurrently executing service checks to the value you specify with the *max_concurrent_checks* option. This is good because it gives you some control over how much load Centreon Engine will impose on your monitoring host, but it can also slow things down. If you are seeing high latency values (> 10 or 15 seconds) for the majority of your service checks, you are probably starving Centreon Engine of the checks it needs. That's not Centreon Engine's fault - its yours. Under ideal conditions, all service checks would have a latency of 0, meaning they were executed at the exact time that they were scheduled to be executed. However, it is normal for some checks to have small latency values. I would recommend taking the minimum number of maximum concurrent checks reported when running Centreon Engine with the *-s* command line argument and doubling it. Keep increasing it until the average check latency for your services is fairly low. More information on service check scheduling can be found [here](#).
- Use passive checks when possible. The overhead needed to process the results of *passive service checks* is much lower than that of "normal" active checks, so make use of that piece of info if you're monitoring a slew of services. It should be noted that passive service checks are only really useful if you have some external application doing some type of monitoring or reporting, so if you're having Centreon Engine do all the work, this won't help things.
- Avoid using interpreted plugins. One thing that will significantly reduce the load on your monitoring host is the use of compiled (C/C++, etc.) plugins rather than interpreted script (Perl, etc) plugins. While Perl scripts and such are easy to write and work well, the fact that they are compiled/interpreted at every execution instance can significantly increase the load on your monitoring host if you have a lot of service checks. If you want to use Perl plugins, consider compiling them into true executables using *perlcc*(1) (a utility which is part of the standard Perl distribution).
- Optimize host check commands. If you're checking host states using the *check_ping* plugin you'll find that host checks will be performed much faster if you break up the checks. Instead of specifying a *max_attempts* value of 1 in the host definition and having the *check_ping* plugin send 10 ICMP packets to the host, it would be much faster to set the *max_attempts* value to 10 and only send out 1 ICMP packet each time. This is due to the fact that Centreon Engine can often determine the status of a host after executing the plugin once, so you want to make the first check as fast as possible. This method does have its pitfalls in some situations (i.e. hosts that are slow to respond may be assumed to be down), but you'll see faster host checks if you use it. Another option would be to use a faster plugin (i.e. *check_fping*) as the *host_check_command* instead of *check_ping*.
- Schedule regular host checks. Scheduling regular checks of hosts can actually help performance in Centreon Engine. This is due to the way the *cached check logic* works (see below). Prior to Centreon Engine 3, regularly scheduled host checks used to result in a big performance hit. This is no longer the case, as host checks are run in parallel - just like service checks. To schedule regular checks of a host, set the *check_interval* directive in the *host definition* to something greater than 0.
- Enable cached host checks. Beginning in Centreon Engine 1, on-demand host checks can benefit from caching. On-demand host checks are performed whenever Centreon Engine detects a service state change. These on-demand checks are executed because Centreon Engine wants to know if the host associated with the service changed state. By enabling cached host checks, you can optimize performance. In some cases, Centreon Engine may be able to use the old/cached state of the host, rather than actually executing a host check command. This can speed things up and reduce load on monitoring server. In order for cached checks to be effective, you need to schedule regular checks of your hosts (see above). More information on cached checks can be found [here](#).
- Don't use aggressive host checking. Unless you're having problems with Centreon Engine recognizing host recoveries, I would recommend not enabling the *use_aggressive_host_checking* option. With this option turned off host checks will execute much faster, resulting in speedier processing of service check results. However, host recoveries can be missed under certain circumstances when this is turned off. For example, if a host recovers

and all of the services associated with that host stay in non-OK states (and don't "wobble" between different non-OK states), Centreon Engine may miss the fact that the host has recovered. A few people may need to enable this option, but the majority don't and I would recommend not using it unless you find it necessary...

- External command optimizations. If you're processing a lot of external commands (i.e. passive checks in a *distributed setup*, you'll probably want to set the *command_check_interval* variable to -1. This will cause Centreon Engine to check for external commands as often as possible. You should also consider increasing the number of available *external command buffer slots*. Buffer slots are used to hold external commands that have been read from the *external command file* (by a separate thread) before they are processed by the Centreon Engine daemon. If your Centreon Engine daemon is receiving a lot of passive checks or external commands, you could end up in a situation where the buffers are always full. This results in child processes (external scripts, NSCA daemon, etc.) blocking when they attempt to write to the external command file.
- Disable use setpgid. When you enable setpgid, we force Centreon Engine to use low performance create process. See *this* documentation.
- Optimize hardware for maximum performance.

Note: Hardware performance shouldn't be an issue unless: 1) you're monitoring thousands of services, 2) you're doing a lot of post-processing of performance data, etc. Your system configuration and your hardware setup are going to directly affect how your operating system performs, so they'll affect how Centreon Engine performs. The most common hardware optimization you can make is with your hard drives. CPU and memory speed are obviously factors that affect performance, but disk access is going to be your biggest bottleneck. Don't store plugins, the status log, etc on slow drives (i.e. old IDE drives or NFS mounts). If you've got them, use UltraSCSI drives or fast IDE drives. An important note for IDE/Linux users is that many Linux installations do not attempt to optimize disk access. If you don't change the disk access parameters (by using a utility like *hdparam*), you'll lose out on a lot of the speedy features of the new IDE drives.

Custom Object Variables

Introduction Users often request that new variables be added to host, service, and contact definitions. These include variables for SNMP community, MAC address, AIM username, Skype number, and street address. The list is endless. The problem that I see with doing this is that it makes Centreon Engine less generic and more infrastructure-specific. Centreon Engine was intended to be flexible, which meant things needed to be designed in a generic manner. Host definitions in Centreon Engine, for example, have a generic "address" variable that can contain anything from an IP address to human-readable driving directions - whatever is appropriate for the user's setup.

Still, there needs to be a method for admins to store information about their infrastructure components in their Centreon Engine configuration without imposing a set of specific variables on others. Centreon Engine attempts to solve this problem by allowing users to define custom variables in their object definitions. Custom variables allow users to define additional properties in their host, service, and contact definitions, and use their values in notifications, event handlers, and host and service checks.

Custom Variable Basics There are a few important things that you should note about custom variables:

- Custom variable names must begin with an underscore (`_`) to prevent name collision with standard variables
- Custom variable names are converted to all uppercase before use
- Custom variables are *inherited* from object templates like normal variables
- Scripts can reference custom variable values with *macros and environment variables*

Examples Here's an example of how custom variables can be defined in different types of object definitions:

```

define host{
    host_name      linuxserver
    _mac_address 00:06:5B:A6:AD:AA ; <-- Custom MAC_ADDRESS variable
    _rack_number R32                ; <-- Custom RACK_NUMBER variable
    ...
}

define service{
    host_name      linuxserver
    description    Memory Usage
    _SNMP_community public ; <-- Custom SNMP_COMMUNITY variable
    _TechContact   Jane Doe ; <-- Custom TECHCONTACT variable
    ....
}

define contact{
    contact_name   john
    _AIM_username  john16 ; <-- Custom AIM_USERNAME variable
    _YahooID       john32 ; <-- Custom YAHOOID variable
    ...
}

```

Custom Variables As Macros Custom variable values can be referenced in scripts and executables that Centreon Engine runs for checks, notifications, etc. by using *macros* or environment variables. Custom variable macros are trusted (because you define them) and therefore not cleaned/sanitized before they are made available to scripts.

In order to prevent name collision among custom variables from different object types, Centreon Engine prepends “_HOST”, “_SERVICE”, or “_CONTACT” to the beginning of custom host, service, or contact variables, respectively, in macro and environment variable names. The table below shows the corresponding macro and environment variable names for the custom variables that were defined in the example above.

Object Type	Variable Name	Macro Name	Environment Variable
Host	MAC_ADDRESS	\$_HOSTMAC_ADDRESS\$	NAGIOS__HOSTMAC_ADDRESS
Host	RACK_NUMBER	\$_HOSTRACK_NUMBER\$	NAGIOS__HOSTRACK_NUMBER
Service	SNMP_COMMUNITY	\$_SERVICES- NMP_COMMUNITY\$	NA- GIOS__SERVICESNMP_COMMUNITY
Service	TECHCONTACT	\$_SERVICETECHCONTACT\$	NA- GIOS__SERVICETECHCONTACT
Contact	AIM_USERNAME	\$_CONTACT- TAIM_USERNAME\$	NA- GIOS__CONTACTAIM_USERNAME
Contact	YAHOOID	\$_CONTACTYAHOOID\$	NAGIOS__CONTACTYAHOOID

Custom Variables And Inheritance Custom object variables are *inherited* just like standard host, service, or contact variables.

Determining Status and Reachability of Network Hosts

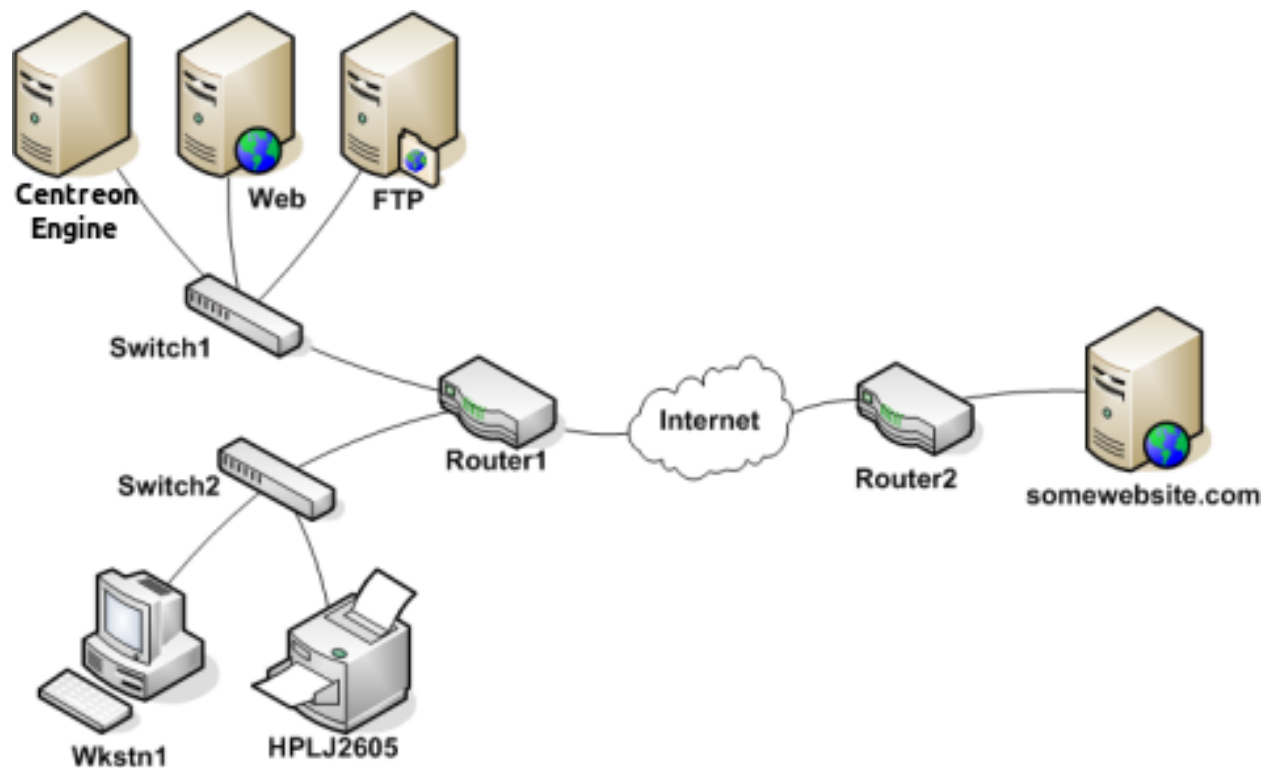
Introduction If you’ve ever work in tech support, you’ve undoubtedly had users tell you “the Internet is down”. As a techie, you’re pretty sure that no one pulled the power cord from the Internet. Something must be going wrong somewhere between the user’s chair and the Internet.

Assuming its a technical problem, you begin to search for the problem. Perhaps the user’s computer is turned off, maybe their network cable is unplugged, or perhaps your organization’s core router just took a dive. Whatever the

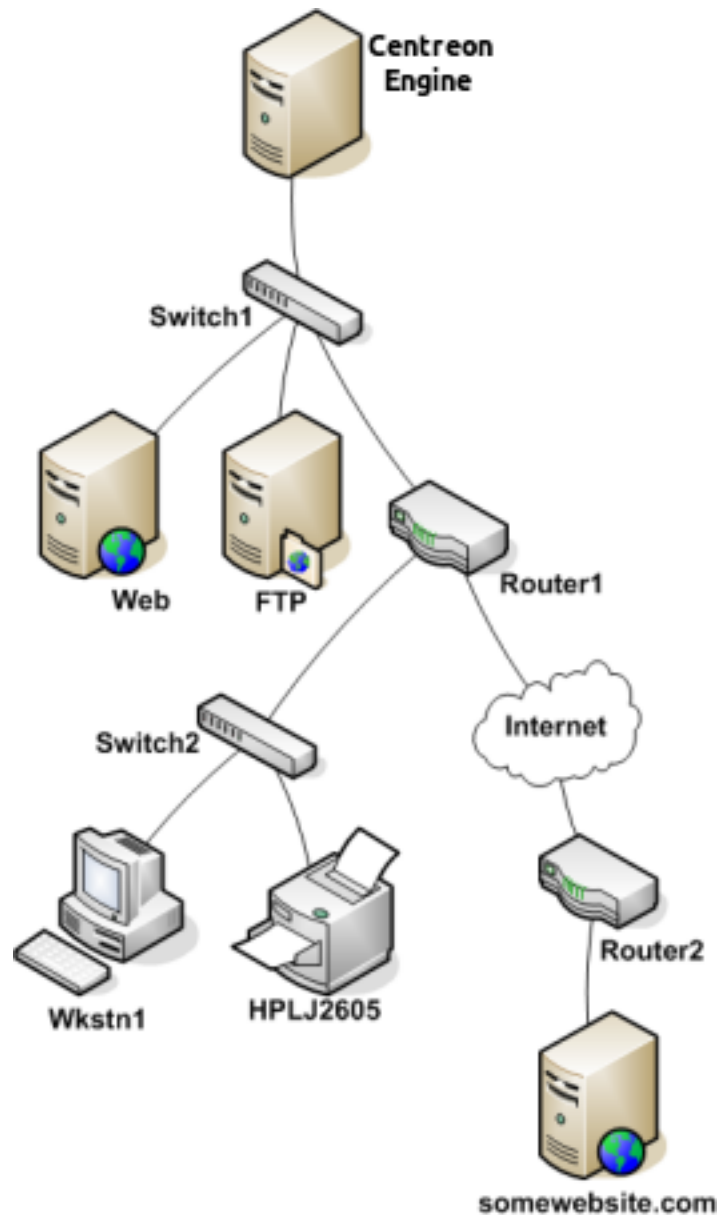
problem might be, one thing is most certain - the Internet isn't down. It just happens to be unreachable for that user.

Centreon Engine 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...

Example Network Take a look at the simple network diagram below. For this example, let's assume you're monitoring all the hosts (server, routers, switches, etc) that are pictured. Centreon Engine is installed and running on the Centreon Engine host.



Defining Parent/Child Relationships In order for Centreon Engine to be able to distinguish between DOWN and UNREACHABLE states for the hosts that are being monitored, you'll need to tell Centreon Engine how those hosts are connected to each other - from the standpoint of the Centreon Engine daemon. To do this, trace the path that a data packet would take from the Centreon Engine daemon to each individual host. Each switch, router, and server the packet encounters or passes through is considered a "hop" and will require that you define a parent/child host relationship in Centreon Engine. Here's what the host parent/child relationships look like from the viewpoint of Centreon Engine:



Now that you know what the parent/child relationships look like for hosts that are being monitored, how do you configure Centreon Engine to reflect them? The `parents` directive in your *host definitions* allows you to do this. Here's what the (abbreviated) host definitions with parent/child relationships would look like for this example:

```

define host{
    host_name Centreon Engine ; <-- The local host has no parent - it is the topmost host
}

define host{
    host_name Switch1
    parents    Centreon Engine
}

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
}

define host{
    host_name Wkstn1
    parents    Switch2
}

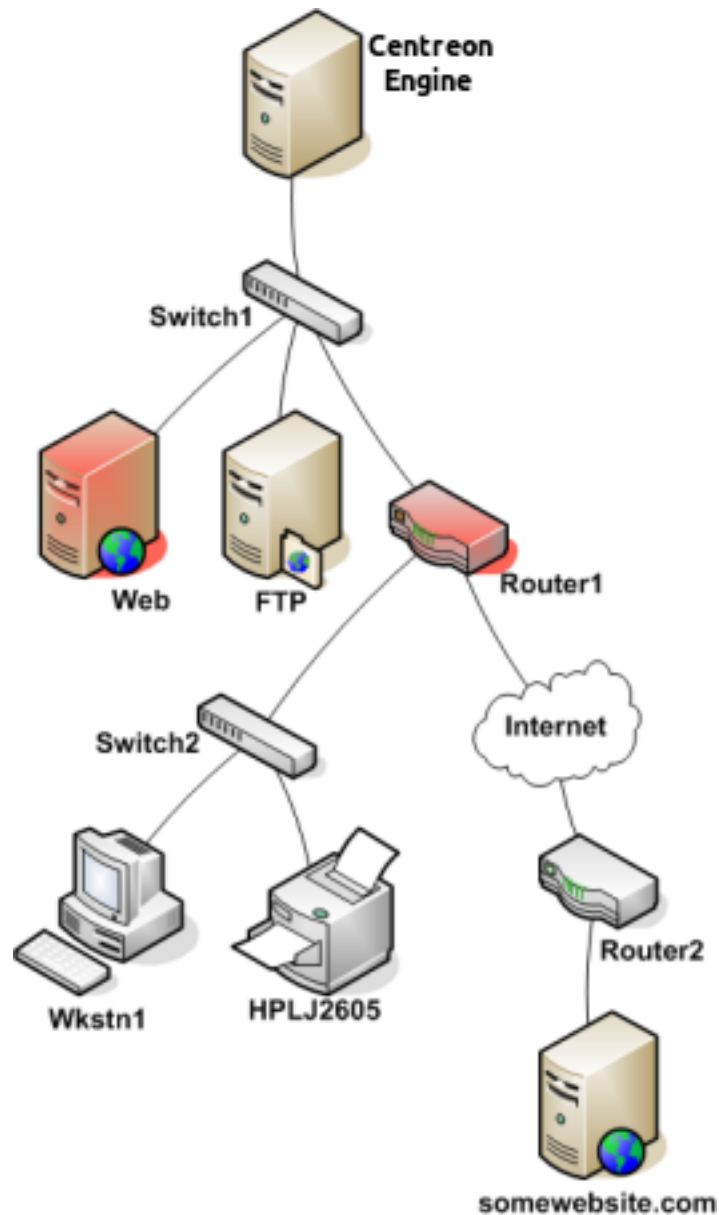
define host{
    host_name HPLJ2605
    parents    Switch2
}

define host{
    host_name Router2
    parents    Router1
}

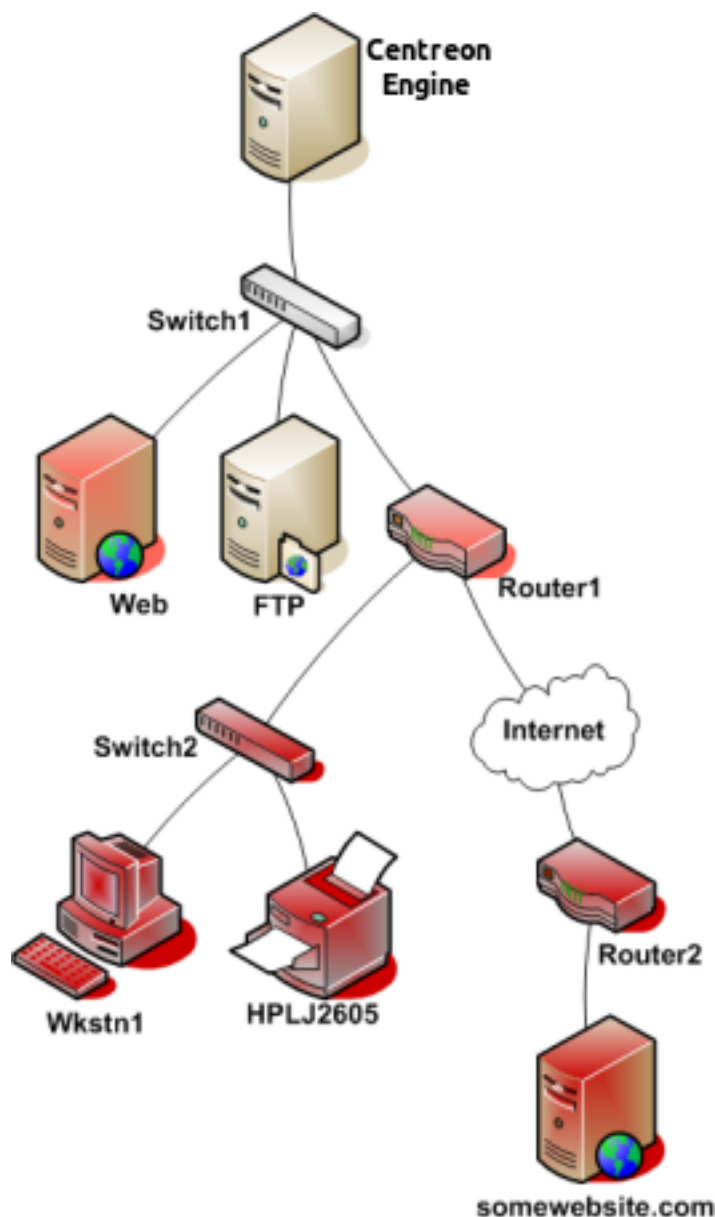
define host{
    host_name somewebsite.com
    parents    Router2
}

```

Reachability Logic in Action Now that you're configured Centreon Engine 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...



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



In this example, Centreon Engine will determine that Web and Router1 are both in DOWN states because the “path” to those hosts is not being blocked.

Centreon Engine will determine that all the hosts “beneath” Router1 are all in an UNREACHABLE state because Centreon Engine 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 - Centreon Engine doesn’t know because it can’t reach them. Hence Centreon Engine considers them to be UNREACHABLE instead of DOWN.

Unreachable States and Notifications By default, Centreon Engine will notify contacts about both DOWN and UNREACHABLE host states. As an admin/tech, you might not want to get notifications about hosts that are UNREACHABLE. You know your network structure, and if Centreon Engine notifies you that your router/firewall is down, you know that everything behind it is unreachable.

If you want to spare yourself from a flood of UNREACHABLE notifications during network outages, you can exclude the unreachable (u) option from the notification_options directive in your *host* definitions and/or the host_notification_options directive in your *contact* definitions.

Fast Startup Options

Introduction There are a few things you can do that can decrease the amount of time it take Centreon Engine to startup (or restart). These speedups involve easing some of the burden involved in processing your configuration files.

Using these techniques is particularly useful when you have one or more of the following:

- Large configurations
- Complex configurations (heavy use of template features)
- Installations where frequent restarts are necessary

Background Whenever Centreon Engine starts/restarts it has to process your configuration files before it can get down to the business of monitoring. This configuration startup process involves a number of steps:

- Reading the config files
- Resolving template definitions
- “Recombobulating” your objects (my term for the various types of work that occurs)
- Duplicating object definitions
- Inheriting object properties
- Sorting your object definitions
- Verifying object relationship integrity
- Checking for circular paths
- and more...

Some of these steps can be quite time-consuming when you have large or complex configurations. Is there a way to speed any of these steps up? Yes!

Evaluating Startup Times Before we get on to making things faster, we need to see what’s possible and whether or not we should even bother with the whole thing. This is easy to do - simply start centengine with the -s command line switch to get timing and scheduling information.

An example of the output (abbreviated to only show relevant portions) is shown below. For this example, I’m using a Centreon Engine config that has 25 hosts defined and just over 10,000 services:

```
$ /usr/sbin/centengine -s /etc/centreon-engine/centengine.cfg
Timing information on object configuration processing is listed
below. You can use this information to see if precaching your
object configuration would be useful.
```

Object Config Source: Config files (uncached)

OBJECT CONFIG PROCESSING TIMES (* = Potential for precache savings
with -u option)

```
-----
Read: 0.486780 sec
Resolve: 0.004106 sec *
Recomb Contactgroups: 0.000077 sec *
Recomb Hostgroups: 0.000172 sec *
Dup Services: 0.028801 sec *
```



```
Recomb Servicegroups: 0.010358 sec *
Duplicate: 5.666932 sec *
Inherit: 0.003770 sec *
Recomb Contacts: 0.030085 sec *
Sort: 2.648863 sec *
Register: 2.654628 sec
Free: 0.021347 sec
```

=====

```
TOTAL: 11.555925 sec * = 8.393170 sec (72.63%) estimated savings
Timing information on configuration verification is listed below.
CONFIG VERIFICATION TIMES (* = Potential for speedup with -x
option)
```

```
Object Relationships: 1.400807 sec
Circular Paths: 54.676622 sec *
Misc: 0.006924 sec
```

=====

```
TOTAL: 56.084353 sec * = 54.676622 sec (97.5%) estimated savings
```

Okay, lets see what happened. Looking at the totals, it took roughly 11.6 seconds to process the configuration files and another 56 seconds to verify the config. That means that every time I start or restart Centreon Engine with this configuration, it will take nearly 68 seconds of startup work before it can monitor anything! That's not acceptable if I have to restart Centreon Engine on a semi-regular basis.

What can I do about this? Take another look at the output and you'll see that Centreon Engine estimates that I could save about 8.4 seconds off the configuration processing time and another 54.7 off the verification times. In total, Centreon Engine thinks I could save 63 seconds of the normal startup time if some optimizations were taken.

Whoa! From 68 seconds to just 5 seconds? Yep, read on for how to do it.

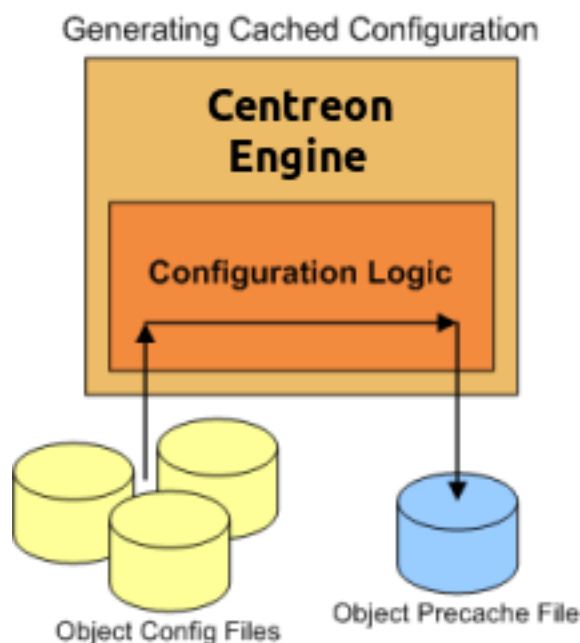
Pre-Caching Object Configuration Centreon Engine can spend quite a bit of time parsing your config files, especially if you make use of the template features such as inheritance, etc. In order to reduce the time it takes to parse your config, you can have Centreon Engine pre-process and pre-cache your config files for future use.

When you run Centreon Engine with the `-p` command line option, Centreon Engine will read your config files in, process them, and save them to a pre-cached object config file (specified by the `precached_object_file` directive). This pre-cached config file will contain pre-processed configuration entries that are easier/faster for Centreon Engine to process in the future.

You must use the `-p` command line option along with either the `-v` or `-s` command line options, as shown below. This ensures that your configuration is verified before the precached file is created:

```
$ /usr/sbin/centengine -pv /etc/centreon-engine/centengine.cfg
```

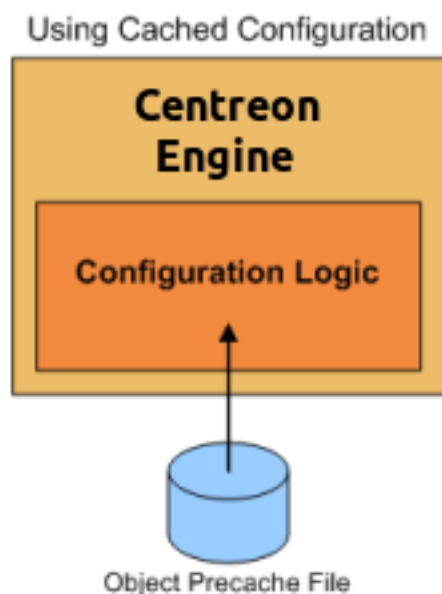
The size of your precached config file will most likely be significantly larger than the sum of the sizes of your object config files. This is normal and by design.



Once the precached object configuration file have been created, you can start Centreon Engine and tell it to use the precached config file instead of your object config file(s) by using the `-u` command line option:

```
$ /usr/sbin/centengine -ud /etc/centreon-engine/centengine.cfg
```

Note: If you modify your configuration files, you will need to re-verify and re-cache your configuration files before restarting Centreon Engine. If you don't re-generate the precached object file, Centreon Engine will continue to use your old configuration because it is now reading from the precached file, rather than your source configuration files.



Skipping Circular Path Tests The second (and most time-intensive) portion of the configuration startup phase is the circular path check. In the example above, it took nearly a minute to perform this step of the configuration verification.

What is the circular path check and why does it take so long? The circular patch check is designed to ensure that

you don't define any circular paths in your host, host dependency, or service dependency definitions. If a circular path existed in your config files, Centreon Engine could end up in a deadlock situation. The most likely reason for the check taking so long is that I'm not using an efficient algorithm. A much more efficient algorithm for detecting circular paths would be most welcomed. Hint: That means all you CompSci graduate students who have been emailing me about doing your thesis on Centreon Engine can contribute some code back. :-)

If you want to skip the circular path check when Centreon Engine starts, you can add the `-x` command line option like this:

```
$ /usr/sbin/centengine -xd /etc/centreon-engine/centengine.cfg
```

Note: It is of utmost importance that you verify your configuration before starting/restarting Centreon Engine when skipping circular path checks. Failure to do so could lead to deadlocks in the Centreon Engine logic. You have been warned.

Putting It All Together Follow these steps if you want to make use of potential speedups from pre-caching your configuration and skipping circular path checks.

1. Verify your configuration and create the precache file with the following command:

```
$ /usr/sbin/centengine -vp /etc/centreon-engine/centengine.cfg
```

2. Stop Centreon Engine if it is currently running.

3. Start Centreon Engine like so to use the precached config file and skip circular path checks:

```
$ /usr/sbin/centengine -uxd /etc/centreon-engine/centengine.cfg
```

4. When you modify your original configuration files in the future and need to restart Centreon Engine to make those changes take place, repeat step 1 to re-verify your config and regenerate your cached config file. Once that is done you can restart Centreon Engine through the web interface or by sending a SIGHUP signal. If you don't re-generate the precached object file, Centreon Engine will continue to use your old configuration because it is now reading from the precached file, rather than your source configuration files.

5. That's it! Enjoy the increased startup speed.

Known Issues

Timeperiods Exclusions and Host/Service Checks - There is a bug in the service/host check scheduling logic that rears its head when you use timeperiod definitions that use the exclude directive. The problem occurs when Centreon Engine Core tries to re-schedule the next check. In this case, the scheduling logic may incorrectly schedule the next check further out in the future than it should. In essence, it skips over the (missing) logic where it could determine an earlier possible time using the exception times. Imperfect Solution: Don't use timeperiod definitions that exclude other timeperiods for your host/service check periods. A fix is being worked on, and will hopefully make it into a 3.4.x release.

Large Installation Tweaks

Introduction Users with large Centreon Engine installations may benefit from the `use_large_installation_tweaks` configuration option. Enabling this option allows the Centreon Engine daemon to take certain shortcuts which result in lower system load and better performance.

Effects When you enable the *use_large_installation_tweaks* option in your main Centreon Engine config file, several changes are made to the way the Centreon Engine daemon operates:

- No Summary Macros In Environment Variables - The *summary macros* will not be available to you as environment variables. Calculating the values of these macros can be quite time-intensive in large configurations, so they are not available as environment variables when use this option. Summary macros will still be available as regular macros if you pass them to your scripts as arguments.
- Different Memory Cleanup - Normally Centreon Engine will free all allocated memory in child processes before they exit. This is probably best practice, but is likely unnecessary in most installations, as most OSes will take care of freeing allocated memory when processes exit. The OS tends to free allocated memory faster than can be done within Centreon Engine itself, so Centreon Engine won't attempt to free memory in child processes if you enable this option.
- Checks fork() Less - Normally Centreon Engine will fork() twice when it executes host and service checks. This is done to (1) ensure a high level of resistance against plugins that go awry and segfault and (2) make the OS deal with cleaning up the grandchild process once it exits. The extra fork() is not really necessary, so it is skipped when you enable this option. As a result, Centreon Engine will itself clean up child processes that exit (instead of leaving that job to the OS). This feature should result in significant load savings on your Centreon Engine installation.

Security Considerations



Introduction

This is intended to be a brief overview of some things you should keep in mind when installing Centreon Engine, so as set it up in a secure manner.

Your monitoring box should be viewed as a backdoor into your other systems. In many cases, the Centreon Engine server might be allowed access through firewalls in order to monitor remote servers. In most all cases, it is allowed to query those remote servers for various information. Monitoring servers are always given a certain level of trust in order to query remote systems. This presents a potential attacker with an attractive backdoor to your systems. An attacker might have an easier time getting into your other systems if they compromise the monitoring server first. This is particularly true if you are making use of shared SSH keys in order to monitor remote systems.

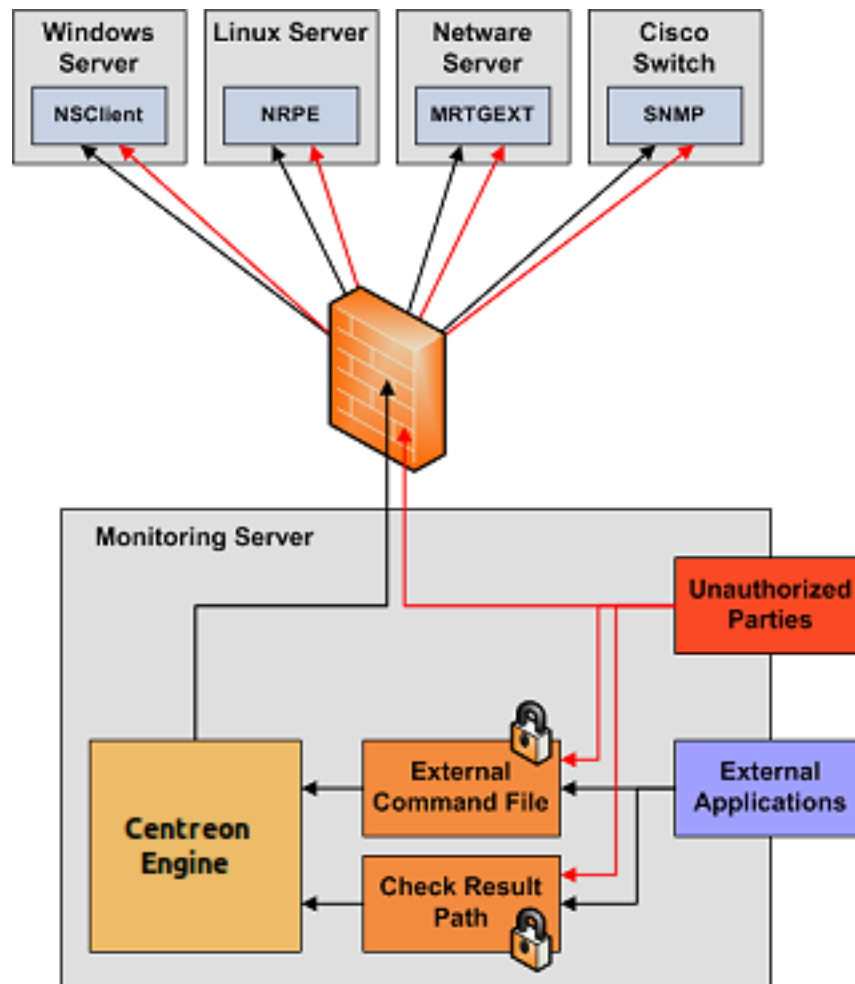
If an intruder has the ability to submit check results or external commands to the Centreon Engine daemon, they have the potential to submit bogus monitoring data, drive you nuts you with bogus notifications, or cause event handler scripts to be triggered. If you have event handler scripts that restart services, cycle power, etc. this could be particularly problematic.

Another area of concern is the ability for intruders to sniff monitoring data (status information) as it comes across the wire. If communication channels are not encrypted, attackers can gain valuable information by watching your monitoring information. Take as an example the following situation: An attacker captures monitoring data on the wire over a period of time and analyzes the typical CPU and disk load usage of your systems, along with the number of users that are typically logged into them. The attacker is then able to determine the best time to compromise a system and use its resources (CPU, etc.) without being noticed.

Here are some tips to help ensure that you keep your systems secure when implementing a Centreon Engine-based monitoring solution...

Best Practices

Use I would recommend that you install Centreon Engine on a server that is dedicated to monitoring (and possibly other admin tasks). Protect your monitoring server as if it were one of the most important servers on your network. Keep running services to a minimum and lock down access to it via TCP wrappers, firewalls, etc. Since the Centreon Engine server is allowed to talk to your servers and may be able to poke through your firewalls, allowing users access to your monitoring server can be a security risk. Remember, its always easier to gain root access through a system security hole if you have a local account on a box.



Don't Run Centreon Engine As Root Centreon Engine doesn't need to run as root, so don't do it. If you need to execute event handlers or plugins which require root access, you might want to try using `sudo`.

Down The Check Result Directory Make sure that only the centengine user is able to read/write in the check result path. If users other than centengine (or root) are able to write to this directory, they could send fake host/service check results to the Centreon Engine daemon. This could result in annoyances (bogus notifications) or security problems (event handlers being kicked off).

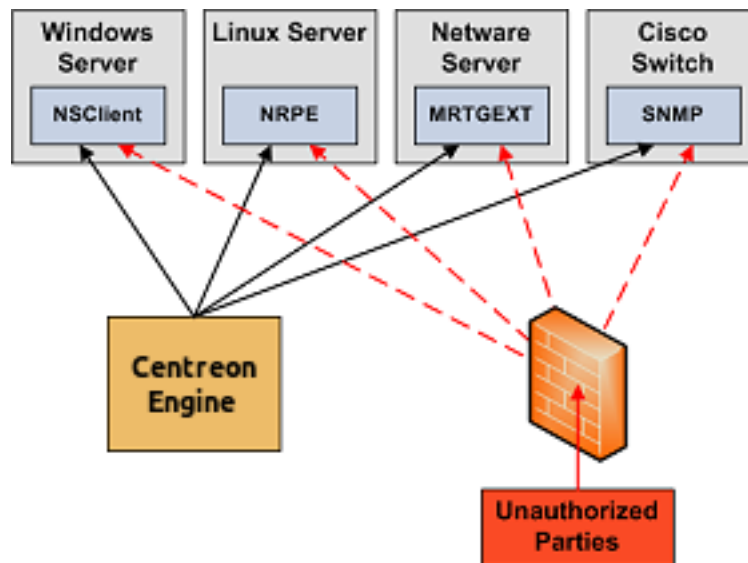
Lock Down The External Command File If you enable *external commands*, make sure you set proper permissions on the `/var/log/centreon-engine/rw` directory. You only want the Centreon Engine user (usually centengine) and the web server user (usually nobody, httpd, apache2, or www-data) to have permissions to write to

the command file. If you've installed Centreon Engine on a machine that is dedicated to monitoring and admin tasks and is not used for public accounts, that should be fine. If you've installed it on a public or multi-user machine (not recommended), allowing the web server user to have write access to the command file can be a security problem. After all, you don't want just any user on your system controlling Centreon Engine through the external command file.

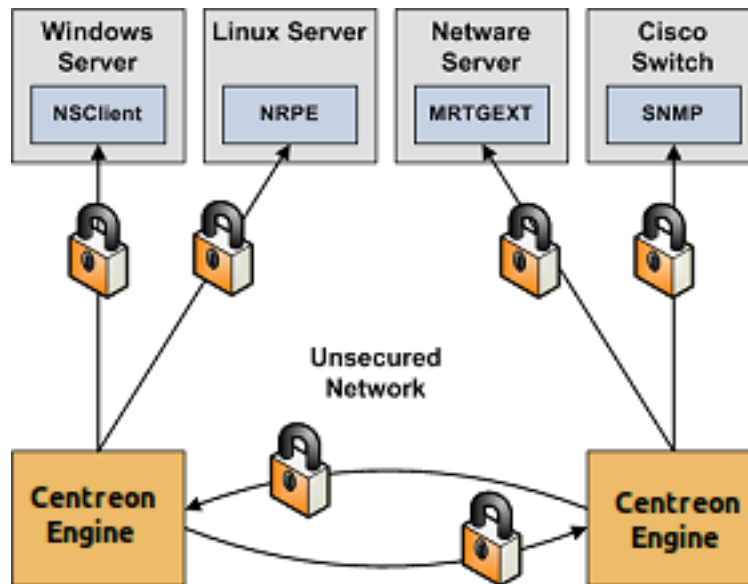
Use Full Paths In Command Definitions When you define commands, make sure you specify the full path (not a relative one) to any scripts or binaries you're executing.

Strip Dangerous Characters From Macros Use the *illegal_macro_output_chars* directive to strip dangerous characters from the \$HOSTOUTPUT\$, \$SERVICEOUTPUT\$, \$HOSTPERFDATA\$, and \$SERVICEPERFDATA\$ macros before they're used in notifications, etc. Dangerous characters can be anything that might be interpreted by the shell, thereby opening a security hole. An example of this is the presence of backtick (') characters in the \$HOSTOUTPUT\$, \$SERVICEOUTPUT\$, \$HOSTPERFDATA\$, and/or \$SERVICEPERFDATA\$ macros, which could allow an attacker to execute an arbitrary command as the centengine user (one good reason not to run Centreon Engine as the root user).

Secure Access to Remote Agents Make sure you lock down access to agents (NRPE, NSClient, SNMP, etc.) on remote systems using firewalls, access lists, etc. You don't want everyone to be able to query your systems for status information. This information could be used by an attacker to execute remote event handler scripts or to determine the best times to go unnoticed.



Secure Communication Channels Make sure you encrypt communication channels between different Centreon Engine installations and between your Centreon Engine servers and your monitoring agents whenever possible. You don't want someone to be able to sniff status information going across your network. This information could be used by an attacker to determine the best times to go unnoticed.



Centreon Engine Plugin API

Other Resources If you're looking at writing your own plugins for Centreon Engine, please make sure to visit these other resources:

- The official [Nagios plugin project website](#)

Plugin Overview Scripts and executables must do two things (at a minimum) in order to function as Centreon Engine plugins:

- Exit with one of several possible return values
- Return at least one line of text output to STDOUT

The inner workings of your plugin are unimportant to Centreon Engine. Your plugin could check the status of a TCP port, run a database query, check disk free space, or do whatever else it needs to check something. The details will depend on what needs to be checked - that's up to you.

Return Code Centreon Engine determines the status of a host or service by evaluating the return code from plugins. The following tables shows a list of valid return codes, along with their corresponding service or host states.

Plugin Return Code	Service State	Host State
0	OK	UP
1	WARNING	UP or DOWN/UNREACHABLE
2	CRITICAL	DOWN/UNREACHABLE
3	UNKNOWN	DOWN/UNREACHABLE

Note: If the `use_aggressive_host_checking` option is enabled, return codes of 1 will result in a host state of DOWN or UNREACHABLE. Otherwise return codes of 1 will result in a host state of UP. The process by which Centreon Engine determines whether or not a host is DOWN or UNREACHABLE is discussed *here toto* .

Plugin Output Spec At a minimum, plugins should return at least one of text output. Beginning with Centreon Engine 3, plugins can optionally return multiple lines of output. Plugins may also return optional performance data that can be processed by external applications. The basic format for plugin output is shown below:

TEXT OUTPUT | OPTIONAL PERFDATA LONG TEXT LINE 1 LONG TEXT LINE 2 ... LONG TEXT LINE N | PERFDATA LI

The performance data (shown in orange) is optional. If a plugin returns performance data in its output, it must separate the performance data from the other text output using a pipe (|) symbol. Additional lines of long text output (shown in blue) are also optional.

Plugin Output Examples

Let's see some examples of possible plugin output...

- Case 1: One line of output (text only) Assume we have a plugin that returns one line of output that looks like this:

```
DISK OK - free space: / 3326 MB (56%);
```

If this plugin was used to perform a service check, the entire line of output will be stored in the *SERVICEOUTPUT* macro.

- Case 2: One line of output (text and perfddata) A plugin can return optional performance data for use by external applications. To do this, the performance data must be separated from the text output with a pipe | symbol like such:

```
DISK OK - free space: / 3326 MB (56%);|/=2643MB;5948;5958;0;5968
```

If this plugin was used to perform a service check, the first portion of output (left of the pipe separator) will be stored in the *SERVICEOUTPUT* macro and the second portion of output (right of the pipe separator) will be stored in the *SERVICEPERFDATA* macro.

- Case 3: Multiple lines of output (text and perfddata) A plugin optionally return multiple lines of both text output and perfddata, like such:

```
DISK OK - free space: / 3326 MB (56%);|/=2643MB;5948;5958;0;5968
/ 15272 MB (77%);
/boot 68 MB (69%);
/home 69357 MB (27%);
/var/log 819 MB (84%);|/boot=68MB;88;93;0;98
/home=69357MB;253404;253409;0;253414
/var/log=818MB;970;975;0;980
```

If this plugin was used to perform a service check, the red portion of first line of output (left of the pipe separator) will be stored in the *SERVICEOUTPUT* macro.

The orange portions of the first and subsequent lines are concatenated (with spaces) are stored in the *SERVICEPERFDATA* macro. The blue portions of the 2nd - 5th lines of output will be concatenated (with escaped newlines) and stored in *LONGSERVICEOUTPUT* the macro.

The final contents of each macro are listed below:

Macro	Value
\$SERVICEOUTPUT\$	DISK OK - free space: / 3326 MB (56%);
\$SERVICEPERFDATA\$	/=2643MB;5948;5958;0;5968 /boot=68MB;88;93;0;98
\$LONGSERVICEOUTPUT\$	/home=69357MB;253404;253409;0;253414 /var/log=818MB;970;975;0;980 / 15272 MB (77%);\ /boot 68 MB (69%);\ /var/log 819 MB (84%);

With regards to multiple lines of output, you have the following options for returning performance data:

- You can choose to return no performance data whatsoever
- You can return performance data on the first line only

- You can return performance data only in subsequent lines (after the first)
- You can return performance data in both the first line and subsequent lines (as shown above)

Plugin Output Length Restrictions Centreon Engine will only read the first 4 KB of data that a plugin returns. This is done in order to prevent runaway plugins from dumping megs or gigs of data back to Centreon Engine. This 4 KB output limit is fairly easy to change if you need. Simply edit the value of the `MAX_PLUGIN_OUTPUT_LENGTH` definition in the `include/centengine.h.in` file of the source code distribution and recompile Centreon Engine. There's nothing else you need to change!

Examples If you're looking for some example plugins to study, I would recommend that you download the official Centreon Engine plugins and look through the code for various C, Perl, and shell script plugins. Information on obtaining the official Centreon Engine plugins can be found *here*.

Migrating from Nagios

While Nagios and Centreon Engine are mostly compatible (they share the same configuration file format, internal logic, ...) some slight differences occur.

Bug Report and Feature Request

Centreon Engine is based Nagios, which has shown over the years its extreme robustness. However as any other piece of software Centreon Engine might from time to time suffer from bugs. In those hopefully rare cases, Merethis would be glad to receive bug reports from users which are necessary for fast bug resolution.

Merethis also welcomes feature requests and code patches. Do not hesitate to write to us !

Privilege Drop

With Nagios, a user **had to** define a `nagios_user` variable and a `nagios_group` variable in the main configuration file. These variables were used by the program itself to drop privileges when the process was started. Centreon Engine delegates this feature to the system administrator. However he is not left alone, as Centreon Engine provide some startup scripts for various platforms (please refer to *Running Centreon Engine* section for more information). User and group used by the init script can be configured when compiling Centreon Engine as explained in the *Compiling Centreon Engine* section.

Daemonization

Nagios provides a `-d` flag that allows process to daemonize and run in background. In Centreon Engine, this feature must be provided by the user which runs Centreon Engine. However, init scripts handle the daemonization of Centreon Engine on supported platforms.

As a result, the `daemon_dumps_core` variable is no longer supported. If a user wish to coredump Centreon Engine, it has to be done as per his operating system's manual.

Command Execution

When executing commands, Nagios used to feed them to a shell for command-line like parsing. This feature does not seem to be used a lot and add an extra load on supervision servers that do not rely on it. As a consequence, the shell execution has been removed from the execution sequence. Users that wish to restore previous behavior must write a

shell script that will call their original command. They will then replace the command in Centreon Engine with the shell script.

Best practice

This documentation provide best practice for Centreon-Engine configuration. Most parameters described in this page ensure that you get an optimal experience with Centreon Engine. The remaining defined parameters are configured to make sure that do not fall in the following cases :

- high check latency
- check stop or failure in notifications
- high monitoring server load
- “holes” in graphs generated by Centreon Broker
- Centreon Engine logs show multiple lines with “Max concurrent service checks (x) has been reached”

Warning: System time must be perfectly synchronized among all pollers (NTP).

Check options

Disable check options if these options are not necessary.

- enable_event_handlers=0
- enable_predictive_host_dependency_checks=0
- enable_predictive_service_dependency_checks=0
- check_service_freshness=0
- check_host_freshness=0
- enable_flap_detection=0
- obsess_over_services=0
- obsess_over_hosts=0
- soft_state_dependencies=0
- use_aggressive_host_checking=0

Logging

- use_syslog=0
- log_notifications=1
- log_host_retries=1
- log_service_retries=1
- log_event_handlers=0
- log_initial_states=1
- log_external_commands=1

- log_passive_checks=0
- host_check_timeout=12
- retain_state_information=1
- retention_update_interval=60
- use_retained_program_state=1

Broker modules

Enable external command.

- broker_module=/path/externalcmd.so

Enable Centreon-Broker.

- broker_module=/path/cbmod.so

Check the event broker options.

- event_broker_options=-1

Perfdata

With Centreon-Broker the perfdata processing need to disable because this options is not necessary for Centreon-Broker and make some latency.

- process_performance_data=0

Tuning

Enable cached check to increased performance.

- host_inter_check_delay_method=s
- max_host_check_spread=5
- sleep_time=0.2
- service_inter_check_delay_method=s
- max_service_check_spread=5
- service_interleave_factor_method=s
- max_concurrent_checks=400
- auto_reschedule_checks=0
- use_large_installation_tweaks=1
- enable_environment_macros=0
- cached_service_check_horizon=60
- cached_host_check_horizon=60

Optimization

Large installation enable some optimization enable it.

- `use_large_installation_tweaks=1`

Environment macros have a poor performance disable it.

- `enable_environment_macros=0`

Centreon-Engine allow to improve performance without using `setpgid` (see the documentation!).

- `use_setpgid=0`

1.3.2 Modules

This is a way of extending functionality in Centreon Engine without having to alter main code. Modules are based on shared code libraries. Modules are hooked into the Centreon Engine process. Modules uses callback routines which have to be served by the modules. These routines are executed when special events occur in the Centreon Engine server process.

External Command

The external commands system was moved on a module. To use it, you need to add line on Centreon Engine configuration:

```
broker_module=/usr/lib/centreon-engine/externalcmd.so
```

1.4 Exploit

1.4.1 Quickstart

This section will show you the basic configuration you need for your monitoring system.

Advice for Beginners

Congratulations on choosing Centreon Engine! Centreon Engine is quite powerful and flexible, but it can take a lot of work to get it configured just the way you'd like. Once you become familiar with how it works and what it can do for you, you'll never want to be without it. Here are some important things to keep in mind for first-time Centreon Engine users:

- Centreon Engine relies on configuration files that might be difficult to write and maintain by hand. You might want to use a frontend to configure Centreon Engine like [Centreon](#). Centreon can help you get your IT network monitored in very few time and provide much more feature to Centreon Engine.
- Relax - it's going to take some time. Don't expect to be able to get things working exactly the way you want them right off the bat. It's not that easy. Setting up Centreon Engine can involve a bit of work, partly because of the options that Centreon Engine offers, partly because you need to know what to monitor on your network (and how best to do it).
- The *quickstart installation guide* is designed to get most new users up and running with a basic Centreon Engine setup fairly quickly. Within 20 minutes you can have Centreon Engine installed and monitoring your local system. Once that's complete, you can move on to learning how to configure Centreon Engine to do more.

- Centreon Engine can be tricky to configure when you've got a good grasp of what's going on, and nearly impossible if you don't. Make sure you read the documentation (particularly the sections on *Configuring Centreon Engine*). Save the advanced topics for when you've got a good understanding of the basics.
- If you've read the documentation, reviewed the sample config files, and are still having problems, send an email message describing your problems to the Centreon Engine mailing list. If you've done some background reading and you provide a good problem description, odds are that someone will give you some pointers on getting things working properly.

Post-Installation Modifications

Once you get Centreon Engine installed and running properly, you'll no doubt want to start monitoring more than just your local machine. Check out the following docs for how to go about monitoring other things:

- *Monitoring Windows machines*
- *Monitoring Linux/Unix machines*
- *Monitoring routers/switches*
- *Monitoring network printers*
- *Monitoring publicly available services (HTTP, FTP, SSH, etc.)*

What You'll End Up With

- Centreon Engine will be installed underneath `/usr/sbin/`.
- Centreon Engine will be configured to monitor a few aspects of your local system (CPU load, disk usage, etc.).

Customize Configuration

Sample configuration files have now been installed in the `/etc/centreon-engine/` directory. These sample files should work fine for getting started with Centreon Engine. You'll need to make just one change before you proceed ...

Edit the `/etc/centreon-engine/objects/contacts.cfg` config file with your favorite editor and change the email address associated with the `centengineadmin` contact definition to the address you'd like to use for receiving alerts:

```
$ vi /etc/centreon-engine/objects/contacts.cfg
```

Other Modifications

Configuring email notifications is out of the scope of this documentation. While Centreon Engine is currently configured to send you email notifications, your system may not yet have a mail program properly installed or configured. Refer to your system documentation, search the web for specific instructions on configuring your system to send email messages to external addresses.

1.4.2 Monitoring

Monitoring Windows machines

Introduction

This document describes how you can monitor “private” services and attributes of Windows machines, such as:

- Memory usage
- CPU load
- Disk usage
- Service states
- Running processes
- etc.

Publicly available services that are provided by Windows machines (HTTP, FTP, POP3, etc.) can be monitored easily by following the documentation on monitoring publicly available services.

Note: These instructions assume that you’ve installed Centreon Engine according to the *quickstart guide*. The sample configuration entries below reference objects that are defined in the sample config files (commands.cfg, templates.cfg, etc.) that are installed if you follow the quickstart.

Overview

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 Centreon Engine plugin that does the monitoring and the actual service or attribute of the Windows machine. Without installing an agent on the Windows box, Centreon Engine would be unable to monitor private services or attributes of the Windows box.

For this example, we will be installing the NSClient++ addon on the Windows machine and using the check_nt plugin to communicate with the NSClient++ addon. The check_nt plugin should already be installed on the Centreon Engine server if you followed the *quickstart guide*.

Other Windows agents (like NC_Net) could be used instead of NSClient++ if you wish - provided you change command and service definitions, etc. a bit. For the sake of simplicity I will only cover using the NSClient++ addon in these instructions.

Steps

There are several steps you’ll need to follow in order to monitor a new Windows machine. They are: 1. Perform first-time prerequisites 2. Install a monitoring agent on the Windows machine 3. Create new host and service definitions for monitoring the Windows machine 4. Restart the Centreon Engine daemon.

What’s Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A check_nt command definition has been added to the commands.cfg file. This allows you to use the check_nt plugin to monitor Window services.

- A Windows server host template (called windows-server) has already been created in the templates.cfg file. This allows you to add new Windows host definitions in a simple manner.

The above-mentioned config files can be found in the `/etc/centreon-engine/objects/` directory. You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Centreon Engine before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your Windows boxes in no time.

Prerequisites

The first time you configure Centreon Engine to monitor a Windows machine, you'll need to do a bit of extra work. Remember, you only need to do this for the *first* Windows machine you monitor.

Edit the main Centreon Engine config file:

```
$ vi /etc/centreon-engine/centengine.cfg
```

Remove the leading pound (#) sign from the following line in the main configuration file:

```
#cfg_file=/etc/centreon-engine/objects/windows.cfg
```

Save the file and exit.

What did you just do? You told Centreon Engine to look to the `/etc/centreon-engine/objects/windows.cfg` to find additional object definitions. That's where you'll be adding Windows host and service definitions. That configuration file already contains some sample host, hostgroup, and service definitions. For the *first* Windows machine you monitor, you can simply modify the sample host and service definitions in that file, rather than creating new ones.

Installing the Windows Agent

Before you can begin monitoring private services and attributes of Windows machines, you'll need to install an agent on those machines. I recommend using the NSClient++ addon, which can be found at <http://sourceforge.net/projects/nscplus>. These instructions will take you through a basic installation of the NSClient++ addon, as well as the configuration of Centreon Engine for monitoring the Windows machine.

1. Download the latest stable version of the NSClient++ addon from <http://sourceforge.net/projects/nscplus>
2. Unzip the NSClient++ files into a new C:\NSClient++ directory
3. Open a command prompt and change to the C:\NSClient++ directory
4. Register the NSClient++ system service with the following command: `nsclient++ /install`
5. Install the NSClient++ systray with the following command ('SysTray' is case-sensitive): `nsclient++ SysTray`
6. Open the services manager and make sure the NSClientpp service is allowed to interact with the desktop (see the 'Log On' tab of the services manager). If it isn't already allowed to interact with the desktop, check the box to allow it to.
7. Edit the NSC.INI file (located in the C:\NSClient++ directory) and make the following changes:
 - Uncomment all the modules listed in the [modules] section, except for CheckWMI.dll and RemoteConfiguration.dll
 - Optionally require a password for clients by changing the 'password' option in the [Settings] section.
 - Uncomment the 'allowed_hosts' option in the [Settings] section. Add the IP address of the Centreon Engine server to this line, or leave it blank to allow all hosts to connect.

- Make sure the 'port' option in the [NSClient] section is uncommented and set to '12489' (the default port).
8. Start the NSClient++ service with the following command: `nsclient++ /start`
 9. If installed properly, a new icon should appear in your system tray. It will be a yellow circle with a black 'M' inside.
 10. Success! The Windows server can now be added to the Centreon Engine monitoring configuration...

Configuring Centreon Engine

Now it's time to define some object definitions in your Centreon Engine configuration files in order to monitor the new Windows machine.

Open the `windows.cfg` file for editing:

```
$ vi /etc/centreon-engine/objects/windows.cfg
```

Add a new host definition for the Windows machine that you're going to monitor. If this is the *first* Windows machine you're monitoring, you can simply modify the sample host definition in `windows.cfg`. Change the `host_name`, `alias`, and `address` fields to appropriate values for the Windows box:

```
define host{
    use          windows-server      ; Inherit default values from a Windows server template (make sure you
    host_name    winserver
    alias        My Windows Server
    address      192.168.1.2
}
```

Good. Now you can add some service definitions (to the same configuration file) in order to tell Centreon Engine to monitor different aspects of the Windows machine. If this is the *first* Windows machine you're monitoring, you can simply modify the sample service definitions in `windows.cfg`.

Note: Replace "winserver" in the example definitions below with the name you specified in the `host_name` directive of the host definition you just added.

Add the following service definition to monitor the version of the NSClient++ addon that is running on the Windows server. This is useful when it comes time to upgrade your Windows servers to a newer version of the addon, as you'll be able to tell which Windows machines still need to be upgraded to the latest version of NSClient++:

```
define service{
    use          generic-service
    host_name    winserver
    service_description NSClient++ Version
    check_command check_nt!CLIENTVERSION
}
```

Add the following service definition to monitor the uptime of the Windows server:

```
define service{
    use          generic-service
    host_name    winserver
    service_description Uptime
    check_command check_nt!UPTIME
}
```

Add the following service definition to monitor the CPU utilization on the Windows server and generate a CRITICAL alert if the 5-minute CPU load is 90% or more or a WARNING alert if the 5-minute load is 80% or greater:


```
define service{
    use                generic-service
    host_name          winserver
    service_description CPU Load
    check_command      check_nt!CPULOAD!-l 5,80,90
}
```

Add the following service definition to monitor memory usage on the Windows server and generate a **CRITICAL** alert if memory usage is 90% or more or a **WARNING** alert if memory usage is 80% or greater:

```
define service{
    use                generic-service
    host_name          winserver
    service_description Memory Usage
    check_command      check_nt!MEMUSE!-w 80 -c 90
}
```

Add the following service definition to monitor usage of the C:\drive on the Windows server and generate a **CRITICAL** alert if disk usage is 90% or more or a **WARNING** alert if disk usage is 80% or greater:

```
define service{
    use                generic-service
    host_name          winserver
    service_description C:\ Drive Space
    check_command      check_nt!USEDISKSPACE!-l c -w 80 -c 90
}
```

Add the following service definition to monitor the W3SVC service state on the Windows machine and generate a **CRITICAL** alert if the service is stopped:

```
define service{
    use                generic-service
    host_name          winserver
    service_description W3SVC
    check_command      check_nt!SERVICESTATE!-d SHOWALL -l W3SVC
}
```

Add the following service definition to monitor the Explorer.exe process on the Windows machine and generate a **CRITICAL** alert if the process is not running:

```
define service{
    use                generic-service
    host_name          winserver
    service_description Explorer
    check_command      check_nt!PROCSTATE!-d SHOWALL -l Explorer.exe
}
```

That's it for now. You've added some basic services that should be monitored on the Windows box. Save the configuration file.

Password Protection

If you specified a password in the NSClient++ configuration file on the Windows machine, you'll need to modify the `check_nt` command definition to include the password. Open the `commands.cfg` file for editing:

```
$ vi /etc/centreon-engine/objects/commands.cfg
```

Change the definition of the `check_nt` command to include the “-s <PASSWORD>” argument (where `PASSWORD` is the password you specified on the Windows machine) like this:

```
define command{
    command_name check_nt
    command_line $USER1$/check_nt -H $HOSTADDRESS$ -p 12489 -s PASSWORD -v $ARG1$ $ARG2$
}
```

Save the file.

Restarting Centreon Engine

You're done with modifying the Centreon Engine configuration, so you'll need to verify your configuration files and restart Centreon Engine.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Centreon Engine until the verification process completes without any errors!

Monitoring Linux/Unix machines

Introduction

This document describes how you can monitor “private” services and attributes of Linux/UNIX servers, such as:

- CPU load
- Memory usage
- Disk usage
- Logged in users
- Running processes
- etc.

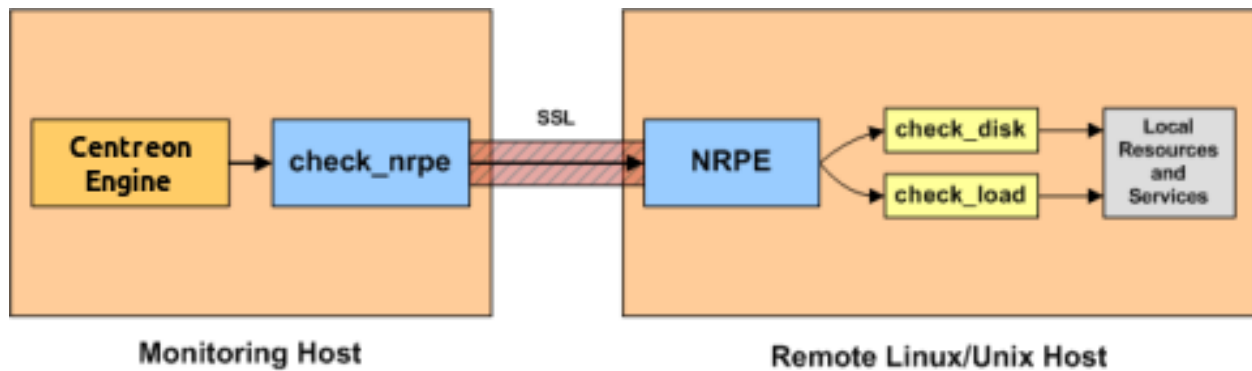
Publicly available services that are provided by Linux servers (HTTP, FTP, SSH, SMTP, etc.) can be monitored easily by following the documentation on monitoring publicly available services.

Note: These instructions assume that you've installed Centreon Engine according to the *quickstart guide*. The sample configuration entries below reference objects that are defined in the sample config files (`commands.cfg`, `templates.cfg` ...) that are installed if you follow the quickstart.

Overview

Note: This document has not been completed. I would recommend you read the documentation on the NRPE add-on for instructions on how to monitor a remote Linux/Unix server.

There are several different ways to monitor attributes or remote Linux/Unix servers. One is by using shared SSH keys and the `check_by_ssh` plugin to execute plugins on remote servers. This method will not be covered here, but can result in high load on your monitoring server if you are monitoring hundreds or thousands of services. The overhead of setting up/destroying SSH connections is the cause of this.



Another common method of monitoring remote Linux/Unix hosts is to use the NRPE add-on. NRPE allows you to execute plugins on remote Linux/Unix hosts. This is useful if you need to monitor local resources/attributes like disk usage, CPU load, memory usage, etc. on a remote host.

Monitoring routers/switches

Introduction

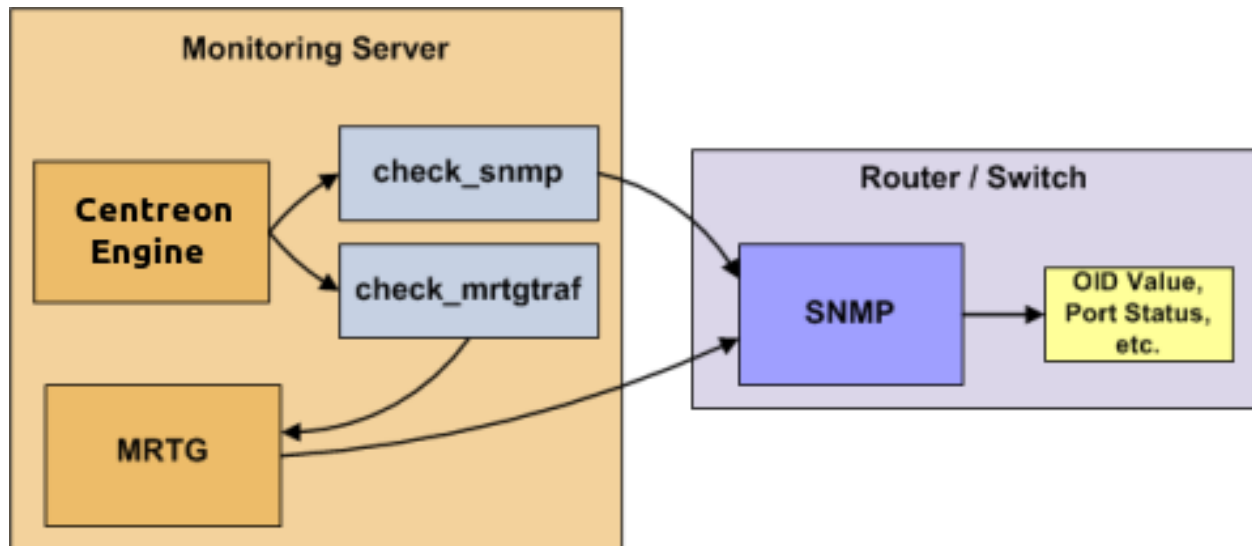
This document describes how you can monitor the status of network switches and routers. Some cheaper “unmanaged” switches and hubs don’t have IP addresses and are essentially invisible on your network, so there’s not any way to monitor them. More expensive switches and routers have addresses assigned to them and can be monitored by pinging them or using SNMP to query status information.

I’ll describe how you can monitor the following things on managed switches, hubs, and routers:

- Packet loss, round trip average
- SNMP status information
- Bandwidth / traffic rate

Note: These instructions assume that you’ve installed Centreon Engine according to the *quickstart guide*. The sample configuration entries below reference objects that are defined in the sample config files (commands.cfg, templates.cfg ...) that are installed when you follow the quickstart.

Overview



Monitoring switches and routers can either be easy or more involved

- depending on what equipment you have and what you want to monitor. As they are critical infrastructure components, you'll no doubt want to monitor them in at least some basic manner.

Switches and routers can be monitored easily by “pinging” them to determine packet loss, RTA, etc. If your switch supports SNMP, you can monitor port status, etc. with the `check_snmp` plugin and bandwidth (if you're using MRTG) with the `check_mrtgtraf` plugin.

The `check_snmp` plugin will only get compiled and installed if you have the `net-snmp` and `net-snmp-utils` packages installed on your system. Make sure the plugin exists in `/libexec` before you continue. If it doesn't, install `net-snmp` and `net-snmp-utils` and recompile/reinstall the Centreon Engine plugins.

Steps

There are several steps you'll need to follow in order to monitor a new router or switch. They are:

- Perform first-time prerequisites
- Create new host and service definitions for monitoring the device
- Restart the Centreon Engine daemon

What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- Two command definitions (`check_snmp` and `check_local_mrtgtraf`) have been added to the `commands.cfg` file. These allows you to use the `check_snmp` and `check_mrtgtraf` plugins to monitor network routers.
- A switch host template (called `generic-switch`) has already been created in the `templates.cfg` file. This allows you to add new router/switch host definitions in a simple manner.

The above-mentioned config files can be found in the `/etc/centreon-engine/objects/` directory. You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend

waiting until you're more familiar with configuring Centreon Engine before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your network routers/switches in no time.

Prerequisites

The first time you configure Centreon Engine to monitor a network switch, you'll need to do a bit of extra work. Remember, you only need to do this for the *first* switch you monitor.

Edit the main Centreon Engine config file:

```
$ vi /etc/centreon-engine/centengine.cfg
```

Remove the leading pound (#) sign from the following line in the main configuration file:

```
#cfg_file=/etc/centreon-engine/objects/switch.cfg
```

Save the file and exit.

What did you just do? You told Centreon Engine to look to the `/etc/centreon-engine/objects/switch.cfg` to find additional object definitions. That's where you'll be adding host and service definitions for routers and switches. That configuration file already contains some sample host, hostgroup, and service definitions. For the *first* router/switch you monitor, you can simply modify the sample host and service definitions in that file, rather than creating new ones.

Configuring Centreon Engine

You'll need to create some *object definitions* in order to monitor a new router/switch.

Open the switch.cfg file for editing:

```
$ vi /etc/centreon-engine/objects/switch.cfg
```

Add a new *host* definition for the switch that you're going to monitor. If this is the *first* switch you're monitoring, you can simply modify the sample host definition in switch.cfg. Change the `host_name`, `alias`, and `address` fields to appropriate values for the switch:

```
define host{
    use         generic-switch          ; Inherit default values from a template
    host_name   linksys-srw224          ; The name we're giving to this switch
    alias       Linksys SRW224P Switch ; A longer name associated with the switch
    address     192.168.1.253           ; IP address of the switch
    hostgroups  allhosts,switches       ; Host groups this switch is associated with
}
```

Monitoring Services

Now you can add some service definitions (to the same configuration file) to monitor different aspects of the switch. If this is the *first* switch you're monitoring, you can simply modify the sample service definition in switch.cfg.

Note: Replace “linksys-srw224p” in the example definitions below with the name you specified in the `host_name` directive of the host definition you just added.

Monitoring Packet Loss and RTA

Add the following service definition in order to monitor packet loss and round trip average between the Centreon Engine host and the switch every 5 minutes under normal conditions:

```
define service{
    use                generic-service                ; Inherit values from a template
    host_name          linksys-srw224p                ; The name of the host the service is associat
    service_description PING                          ; The service description
    check_command       check_ping!200.0,20%!600.0,60% ; The command used to monitor the service
    normal_check_interval 5                          ; Check the service every 5 minutes under norm
    retry_check_interval 1                          ; Re-check the service every minute until its
}
```

This service will be:

- CRITICAL if the round trip average (RTA) is greater than 600 milliseconds or the packet loss is 60% or more.
- WARNING if the RTA is greater than 200 ms or the packet loss is 20% or more.
- OK if the RTA is less than 200 ms and the packet loss is less than 20%.

Monitoring SNMP Status Information

If your switch or router supports SNMP, you can monitor a lot of information by using the check_snmp plugin. If it doesn't, skip this section.

Add the following service definition to monitor the uptime of the switch:

```
define service{
    use                generic-service ; Inherit values from a template
    host_name          linksys-srw224p
    service_description Uptime
    check_command       check_snmp!-C public -o sysUpTime.0
}
```

In the check_command directive of the service definition above, the “-C public” tells the plugin that the SNMP community name to be used is “public” and the “-o sysUpTime.0” indicates which OID should be checked.

If you want to ensure that a specific port/interface on the switch is in an up state, you could add a service definition like this:

```
define service{
    use                generic-service ; Inherit values from a template
    host_name          linksys-srw224p
    service_description Port 1 Link Status
    check_command       check_snmp!-C public -o ifOperStatus.1 -r 1 -m RFC1213-MIB
}
```

In the example above, the “-o ifOperStatus.1” refers to the OID for the operational status of port 1 on the switch. The “-r 1” option tells the check_snmp plugin to return an OK state if “1” is found in the SNMP result (1 indicates an “up” state on the port) and CRITICAL if it isn't found. The “-m RFC1213-MIB” is optional and tells the check_snmp plugin to only load the “RFC1213-MIB” instead of every single MIB that's installed on your system, which can help speed things up.

That's it for the SNMP monitoring example. There are a million things that can be monitored via SNMP, so it's up to you to decide what you need and want to monitor. Good luck!

Note: You can usually find the OIDs that can be monitored on a switch by running the following command (replace

192.168.1.253 with the IP address of the switch):

```
$ snmpwalk -v1 -c public 192.168.1.253 -m ALL .1
```

Monitoring Bandwidth / Traffic Rate

If you're monitoring bandwidth usage on your switches or routers using [MRTG](#), you can have Centreon Engine alert you when traffic rates exceed thresholds you specify. The `check_mrtgtraf` plugin (which is included in the Centreon Engine plugins distribution) allows you to do this.

You'll need to let the `check_mrtgtraf` plugin know what log file the MRTG data is being stored in, along with thresholds, etc. In my example, I'm monitoring one of the ports on a Linksys switch. The MRTG log file is stored in `/var/lib/mrtg/192.168.1.253_1.log`. Here's the service definition I use to monitor the bandwidth data that's stored in the log file:

```
define service{
    use                generic-service ; Inherit values from a template
    host_name          linksys-srw224p
    service_description Port 1 Bandwidth Usage
    check_command       check_local_mrtgtraf!/var/lib/mrtg/192.168.1.253_1.log!AVG!1000000,2000000!5000000,5000000!10
}
```

In the example above, the `/var/lib/mrtg/192.168.1.253_1.log` option that gets passed to the `check_local_mrtgtraf` command tells the plugin which MRTG log file to read from. The "AVG" option tells it that it should use average bandwidth statistics. The "1000000,2000000" options are the warning thresholds (in bytes) for incoming traffic rates. The "5000000,5000000" are critical thresholds (in bytes) for outgoing traffic rates. The "10" option causes the plugin to return a CRITICAL state if the MRTG log file is older than 10 minutes (it should be updated every 5 minutes).

Save the file.

Restarting Centreon Engine

Once you've added the new host and service definitions to the `switch.cfg` file, you're ready to start monitoring the router/switch. To do this, you'll need to *verify your configuration* and *restart Centreon Engine*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Centreon Engine until the verification process completes without any errors!

Monitoring Network Printers

Introduction

This document describes how you can monitor the status of networked printers. Specifically, HP printers that have internal/external JetDirect cards/devices, or other print servers (like the Troy PocketPro 100S or the Netgear PS101) that support the JetDirect protocol.

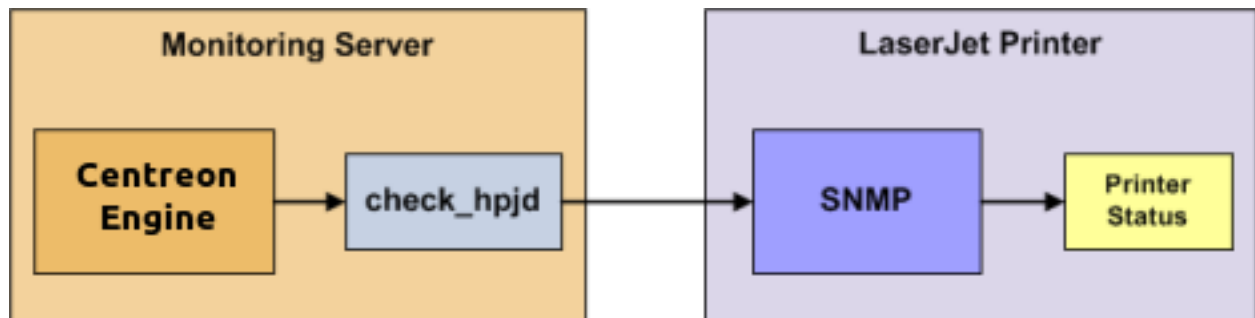
The `check_hpjd` plugin (which is part of the standard Centreon Engine plugins distribution) allows you to monitor the status of JetDirect-capable printers which have SNMP enabled. The plugin is capable of detecting the following printer states:

- Paper Jam
- Out of Paper

- Printer Offline
- Intervention Required
- Toner Low
- Insufficient Memory
- Open Door
- Output Tray is Full
- and more...

Note: These instructions assume that you've installed Centreon Engine according to the *quickstart guide*. The sample configuration entries below reference objects that are defined in the sample config files (`commands.cfg`, `templates.cfg`, etc.) that are installed if you follow the quickstart.

Overview



Monitoring the status of a networked printer is pretty simple. JetDirect-enabled printers usually have SNMP enabled, which allows Centreon Engine to monitor their status using the `check_hpjd` plugin.

The `check_hpjd` plugin will only get compiled and installed if you have the `net-snmp` and `net-snmp-utils` packages installed on your system. Make sure the plugin exists in `/libexec` before you continue. If it doesn't, install `net-snmp` and `net-snmp-utils` and recompile/reinstall the Centreon Engine plugins.

Steps

There are several steps you'll need to follow in order to monitor a new network printer. They are:

1. Perform first-time prerequisites
2. Create new host and service definitions for monitoring the printer
3. Restart the Centreon Engine daemon.

What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A `check_hpjd` command definition has been added to the `commands.cfg` file. This allows you to use the `check_hpjd` plugin to monitor network printers.
- A printer host template (called `generic-printer`) has already been created in the `templates.cfg` file. This allows you to add new printer host definitions in a simple manner.

The above-mentioned config files can be found in the `/etc/centreon-engine/objects/` directory. You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Centreon Engine before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your network printers in no time.

Prerequisites

The first time you configure Centreon Engine to monitor a network printer, you'll need to do a bit of extra work. Remember, you only need to do this for the *first* printer you monitor.

Edit the main Centreon Engine config file:

```
$ vi /etc/centreon-engine/centengine.cfg
```

Remove the leading pound (#) sign from the following line in the main configuration file:

```
#cfg_file=/etc/centreon-engine/objects/printer.cfg
```

Save the file and exit.

What did you just do? You told Centreon Engine to look to the `/etc/centreon-engine/objects/printer.cfg` to find additional object definitions. That's where you'll be adding host and service definitions for the printer. That configuration file already contains some sample host, hostgroup, and service definitions. For the *first* printer you monitor, you can simply modify the sample host and service definitions in that file, rather than creating new ones.

Configuring Centreon Engine

You'll need to create some *object definitions* in order to monitor a new printer.

Open the `printer.cfg` file for editing:

```
$ vi /etc/centreon-engine/objects/printer.cfg
```

Add a new *host* definition for the networked printer that you're going to monitor. If this is the *first* printer you're monitoring, you can simply modify the sample host definition in `printer.cfg`. Change the `host_name`, `alias`, and `address` fields to appropriate values for the printer:

```
define host{
    use         generic-printer      ; Inherit default values from a template
    host_name    hplj2605dn          ; The name we're giving to this printer
    alias        HP LaserJet 2605dn  ; A longer name associated with the printer
    address      192.168.1.30        ; IP address of the printer
    hostgroups   allhosts            ; Host groups this printer is associated with
}
```

Now you can add some service definitions (to the same configuration file) to monitor different aspects of the printer. If this is the *first* printer you're monitoring, you can simply modify the sample service definition in `printer.cfg`.

Note: Replace "hplj2605dn" in the example definitions below with the name you specified in the `host_name` directive of the host definition you just added.

Add the following service definition to check the status of the printer. The service uses the `check_hpjd` plugin to check the status of the printer every 10 minutes by default. The SNMP community string used to query the printer is "public" in this example:

```

define service{
    use                generic-service        ; Inherit values from a template
    host_name          hplj2605dn            ; The name of the host the service is associated with
    service_description Printer Status        ; The service description
    check_command       check_hpjd!-C public  ; The command used to monitor the service
    normal_check_interval 10                  ; Check the service every 10 minutes under normal conditions
    retry_check_interval 1                    ; Re-check the service every minute until its final/hard
}

```

Add the following service definition to ping the printer every 10 minutes by default. This is useful for monitoring RTA, packet loss, and general network connectivity:

```

define service{
    use                generic-service
    host_name          hplj2605dn
    service_description PING
    check_command       check_ping!3000.0,80%!5000.0,100%
    normal_check_interval 10
    retry_check_interval 1
}

```

Save the file.

Restarting Centreon Engine

Once you've added the new host and service definitions to the `printer.cfg` file, you're ready to start monitoring the printer. To do this, you'll need to *verify your configuration* and *restart Centreon Engine*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Centreon Engine until the verification process completes without any errors!

Monitoring publicly available services (HTTP, FTP, SSH, etc.)

Introduction

This document describes how you can monitor publicly available services, applications and protocols. By "public" I mean services that are accessible across the network - either the local network or the greater Internet. Examples of public services include HTTP, POP3, IMAP, FTP, and SSH. There are many more public services that you probably use on a daily basis. These services and applications, as well as their underlying protocols, can usually be monitored by Centreon Engine without any special access requirements.

Private services, in contrast, cannot be monitored with Centreon Engine without an intermediary agent of some kind. Examples of private services associated with hosts are things like CPU load, memory usage, disk usage, current user count, process information, etc. These private services or attributes of hosts are not usually exposed to external clients. This situation requires that an intermediary monitoring agent be installed on any host that you need to monitor such information on. More information on monitoring private services on different types of hosts can be found in the documentation on:

- *Monitoring Windows machines*
- *Monitoring Linux/Unix machines*

Note: Occasionally you will find that information on private services and applications can be monitored with SNMP. The SNMP agent allows you to remotely monitor otherwise private (and inaccessible) information about the host. For more information about monitoring services using SNMP, check out the documentation on *monitoring routers/switches*. These instructions assume that you've installed Centreon Engine according to the *quickstart*

guide. The sample configuration entries below reference objects that are defined in the sample `commands.cfg` and `localhost.cfg` config files.

Plugins For Monitoring Services

When you find yourself needing to monitor a particular application, service, or protocol, chances are good that a *plugin* exists to monitor it. The official Centreon Engine plugins distribution comes with plugins that can be used to monitor a variety of services and protocols. There are also a large number of contributed plugins that can be found in the `contrib/` subdirectory of the plugin distribution. The `nagios.org` website hosts a number of additional plugins that have been written by users, so check it out when you have a chance.

If you don't happen to find an appropriate plugin for monitoring what you need, you can always write your own. Plugins are easy to write, so don't let this thought scare you off. Read the documentation on *developing plugins* for more information.

I'll walk you through monitoring some basic services that you'll probably use sooner or later. Each of these services can be monitored using one of the plugins that gets installed as part of the Centreon Engine plugins distribution. Let's get started...

Creating A Host Definition

Before you can monitor a service, you first need to define a *host* that is associated with the service. You can place host definitions in any object configuration file specified by a `cfg_file` directive or placed in a directory specified by a `cfg_dir` directive. If you have already created a host definition, you can skip this step.

For this example, let's say you want to monitor a variety of services on a remote host. Let's call that host `remotehost`. The host definition can be placed in its own file or added to an already existing object configuration file. Here's what the host definition for `remotehost` might look like:

```
define host{
    use generic-host ; Inherit default values from a template
    host_name remotehost ; The name we're giving to this host
    alias Some Remote Host ; A longer name associated with the host
    address 192.168.1.50 ; IP address of the host
    hostgroups allhosts ; Host groups this host is associated with
}
```

Now that a definition has been added for the host that will be monitored, we can start defining services that should be monitored. As with host definitions, service definitions can be placed in any object configuration file.

Creating Service Definitions

For each service you want to monitor, you need to define a *service* in Centreon Engine that is associated with the host definition you just created. You can place service definitions in any object configuration file specified by a `cfg_file` directive or placed in a directory specified by a `cfg_dir` directive.

Some example service definitions for monitoring common public service (HTTP, FTP, etc.) are given below.

Monitoring HTTP

Chances are you're going to want to monitor web servers at some point, either yours or someone else's. The `check_http` plugin is designed to do just that. It understands the HTTP protocol and can monitor response time, error codes, strings in the returned HTML, server certificates, and much more.

The `commands.cfg` file contains a command definition for using the `check_http` plugin. It looks like this:

```
define command{
    name          check_http
    command_name  check_http
    command_line  $USER1$/check_http -I $HOSTADDRESS$ $ARG1$
}
```

A simple service definition for monitoring the HTTP service on the remotehost machine might look like this:

```
define service{
    use                generic-service ; Inherit default values from a template
    host_name          remotehost
    service_description HTTP
    check_command       check_http
}
```

This simple service definition will monitor the HTTP service running on remotehost. It will produce alerts if the web server doesn't respond within 10 seconds or if it returns HTTP errors codes (403, 404, etc.). That's all you need for basic monitoring. Pretty simple, huh?

Note: For more advanced monitoring, run the `check_http` plugin manually with `-help` as a command-line argument to see all the options you can give the plugin. This `-help` syntax works with all of the plugins I'll cover in this document.

A more advanced definition for monitoring the HTTP service is shown below. This service definition will check to see if the `/download/index.php` URI contains the string `latest-version.tar.gz`. It will produce an error if the string isn't found, the URI isn't valid, or the web server takes longer than 5 seconds to respond:

```
define service{
    use                generic-service ; Inherit default values from a template
    host_name          remotehost
    service_description Product Download Link
    check_command       check_http!-u /download/index.php -t 5 -s "latest-version.tar.gz"
}
```

Monitoring FTP

When you need to monitor FTP servers, you can use the `check_ftp` plugin. The `commands.cfg` file contains a command definition for using the `check_ftp` plugin, which looks like this:

```
define command{
    command_name  check_ftp
    command_line  $USER1$/check_ftp -H $HOSTADDRESS$ $ARG1$
}
```

A simple service definition for monitoring the FTP server on remotehost would look like this:

```
define service{
    use                generic-service ; Inherit default values from a template
    host_name          remotehost
    service_description FTP
    check_command       check_ftp
}
```

This service definition will monitor the FTP service and generate alerts if the FTP server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the FTP server running on port 1023 on remotehost. It will generate an alert if the server doesn't respond within 5 seconds or if the server response doesn't contain the string "Pure-FTPd [TLS]":

```
define service{
    use                generic-service ; Inherit default values from a template
    host_name          remotehost
    service_description Special FTP
    check_command       check_ftp!-p 1023 -t 5 -e "Pure-FTPd [TLS]"
}
```

Monitoring SSH

When you need to monitor SSH servers, you can use the check_ssh plugin. The `commands.cfg` file contains a command definition for using the check_ssh plugin, which looks like this:

```
define command{
    command_name check_ssh
    command_line $USER1$/check_ssh $ARG1$ $HOSTADDRESS$
}
```

A simple service definition for monitoring the SSH server on remotehost would look like this:

```
define service{
    use                generic-service ; Inherit default values from a template
    host_name          remotehost
    service_description SSH
    check_command       check_ssh
}
```

This service definition will monitor the SSH service and generate alerts if the SSH server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the SSH server and generate an alert if the server doesn't respond within 5 seconds or if the server version string doesn't match "OpenSSH_4.2":

```
define service{
    use                generic-service ; Inherit default values from a template
    host_name          remotehost
    service_description SSH Version Check
    check_command       check_ssh!-t 5 -r "OpenSSH_4.2"
}
```

Monitoring SMTP

The check_smtp plugin can be using for monitoring your email servers. The `commands.cfg` file contains a command definition for using the check_smtp plugin, which looks like this:

```
define command{
    command_name check_smtp
    command_line $USER1$/check_smtp -H $HOSTADDRESS$ $ARG1$
}
```

A simple service definition for monitoring the SMTP server on remotehost would look like this:

```
define service{
    use                generic-service ; Inherit default values from a template
    host_name          remotehost
    service_description SMTP
    check_command       check_smtp
}
```

This service definition will monitor the SMTP service and generate alerts if the SMTP server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the SMTP server and generate an alert if the server doesn't respond within 5 seconds or if the response from the server doesn't contain "mygreat-mailserver.com":

```
define service{
    use                generic-service ; Inherit default values from a template
    host_name          remotehost
    service_description SMTP Response Check
    check_command       check_smtp!-t 5 -e "mygreatmailserver.com"
}
```

Monitoring POP3

The `check_pop` plugin can be using for monitoring the POP3 service on your email servers. The `commands.cfg` file contains a command definition for using the `check_pop` plugin, which looks like this:

```
define command{
    command_name check_pop
    command_line $USER1$/check_pop -H $HOSTADDRESS$ $ARG1$
}
```

A simple service definition for monitoring the POP3 service on remotehost would look like this:

```
define service{
    use                generic-service ; Inherit default values from a template
    host_name          remotehost
    service_description POP3
    check_command       check_pop
}
```

This service definition will monitor the POP3 service and generate alerts if the POP3 server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the POP3 service and generate an alert if the server doesn't respond within 5 seconds or if the response from the server doesn't contain "mygreat-mailserver.com":

```
define service{
    use                generic-service ; Inherit default values from a template
    host_name          remotehost
    service_description POP3 Response Check
    check_command       check_pop!-t 5 -e "mygreatmailserver.com"
}
```

Monitoring IMAP

The `check_imap` plugin can be used for monitoring IMAP4 service on your email servers. The `commands.cfg` file contains a command definition for using the `check_imap` plugin, which looks like this:

```
define command{
    command_name check_imap
    command_line $USER1$/check_imap -H $HOSTADDRESS$ $ARG1$
}
```

A simple service definition for monitoring the IMAP4 service on `remotehost` would look like this:

```
define service{
    use generic-service ; Inherit default values from a template
    host_name remotehost
    service_description IMAP
    check_command check_imap
}
```

This service definition will monitor the IMAP4 service and generate alerts if the IMAP server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the IMAP4 service and generate an alert if the server doesn't respond within 5 seconds or if the response from the server doesn't contain "mygreat-mailserver.com":

```
define service{
    use generic-service ; Inherit default values from a template
    host_name remotehost
    service_description IMAP4 Response Check
    check_command check_imap!-t 5 -e "mygreatmailserver.com"
}
```

Restarting Centreon Engine

Once you've added the new host and service definitions to your object configuration file(s), you're ready to start monitoring them. To do this, you'll need to *verify your configuration* and *restart Centreon Engine*.

If the verification process produces any error messages, fix your configuration file before continuing. Make sure that you don't (re)start Centreon Engine until the verification process completes without any errors!

1.4.3 Plugins

Introduction

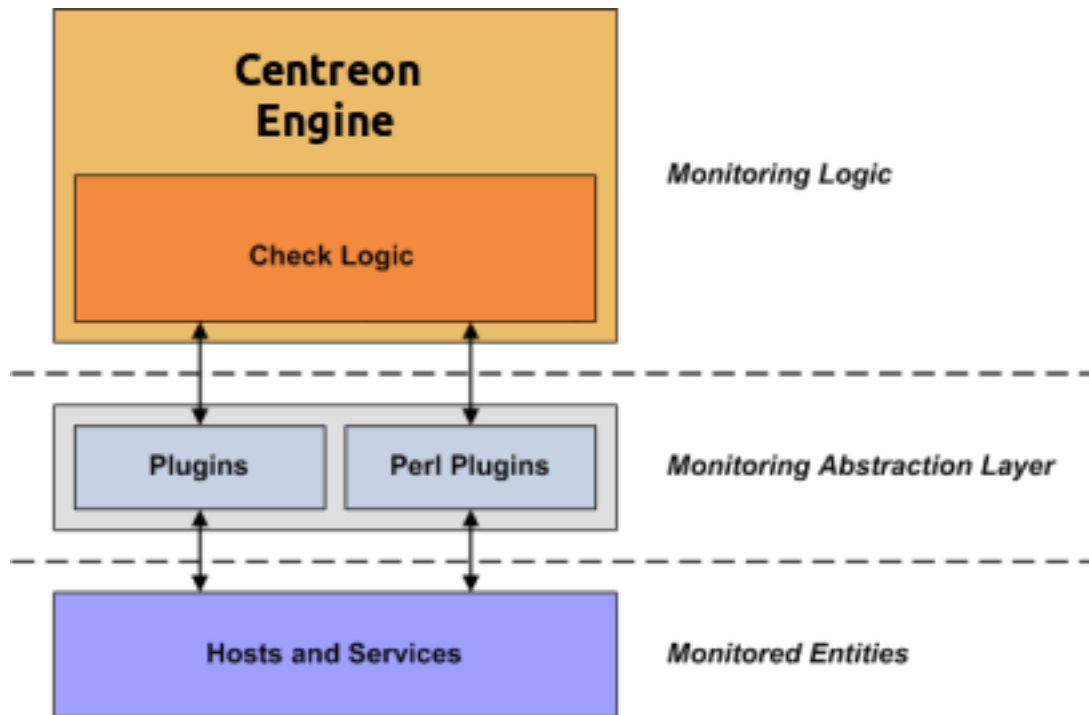
Unlike many other monitoring tools, Centreon Engine does not include any internal mechanisms for checking the status of hosts and services on your network. Instead, Centreon Engine relies on external programs (called plugins) to do all the dirty work.

What Are Plugins?

Plugins are compiled executables or scripts (Perl scripts, shell scripts, etc.) that can be run from a command line to check the status of a host or service. Centreon Engine uses the results from plugins to determine the current status of hosts and services on your network.

Centreon Engine will execute a plugin whenever there is a need to check the status of a service or host. The plugin does something (notice the very general term) to perform the check and then simply returns the results to Centreon Engine. Centreon Engine will process the results that it receives from the plugin and take any necessary actions (running *event handlers*, sending out *notifications*, etc).

Plugins As An Abstraction Layer



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

The upside of this type of plugin architecture is 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 Centreon Engine. There are already a lot of plugins that have been created in order to monitor basic resources such as processor load, disk usage, ping rates, etc. If you want to monitor something else, take a look at the documentation on *writing plugins* and roll your own. Its simple!

The downside to this type of plugin architecture is the fact that Centreon Engine has absolutely no idea what it is that you're monitoring. You could be monitoring network traffic statistics, data error rates, room temperature, CPU voltage, fan speed, processor load, disk space, or the ability of your super-fantastic toaster to properly brown your bread in the morning... Centreon Engine doesn't understand the specifics of what's being monitored - it just tracks changes in the state of those resources. Only the plugins themselves know exactly what they're monitoring and how to perform the actual checks.

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 ...

Obtaining Plugins

Plugins are not distributed with Centreon Engine, but you can download the official Centreon Engine plugins and many additional plugins created and maintained by Centreon Engine users from the following locations:

- [Centreon Engine Plugins Project](#)
- [Centreon Engine Downloads Page](#)
- nagiosexchange.org

How Do I Use Plugin X?

Most all plugins will display basic usage information when you execute them using ‘-h’ or ‘-help’ on the command line. For example, if you want to know how the check_http plugin works or what options it accepts, you should try executing the following command:

```
$ ./check_http --help
```

Plugin API

You can find information on the technical aspects of plugins, as well as how to go about creating your own custom plugins *here*.

1.4.4 Running Centreon Engine

Verifying Your Configuration

Every time you modify your configuration files, you should run a sanity check on them. It is important to do this before you (re)start Centreon Engine, as Centreon Engine will shut down if your configuration contains errors.

In order to verify your configuration, run Centreon Engine with the **-v** command line option like so:

```
$ /usr/sbin/centengine -v /etc/centreon-engine/engine.cfg
```

If you’ve forgotten to enter some critical data or misconfigured things, Centreon Engine will spit out a warning or error message that should point you to the location of the problem. Error messages generally print out the line in the configuration file that seems to be the source of the problem. On errors, Centreon Engine will often exit the pre-flight check and return to the command prompt after printing only the first error that it has encountered. This is done so that one error does not cascade into multiple errors as the remainder of the configuration data is verified. If you get any error messages you’ll need to go and edit your configuration files to remedy the problem. Warning messages can generally be safely ignored, since they are only recommendations and not requirements.

Once you’ve verified your configuration files and fixed any errors you can go ahead and (re)start Centreon Engine.

Starting and Stopping Centreon Engine

There’s more than one way to start, stop, and restart Centreon Engine. Here are some of the more common ones ...

Note: Always make sure you *verify your configuration* before you (re)start Centreon Engine.

On CentOS

On CentOS, Centreon Engine provides an init script called `centengine` that should have been installed in `/etc/init.d/`. The script can be used like this as a superuser (**root**):

```
$ /etc/init.d/centengine <operation>
```

Where operation is one of:

Operation	Description
start	Start Centreon Engine if it is not already running.
stop	Stop Centreon Engine if it is running.
restart	Stop and restart Centreon Engine if it is already running, otherwise start it.
try-restart	Restart Centreon Engine if it is already running.
reload	Cause the configuration to be reloaded without actually stopping and restarting the service. Warning : the use of this feature is highly discouraged.
force-reload	Similar to try-restart.
status	Print the current status of the service.

On Ubuntu

On Ubuntu, Centreon Engine provides an upstart configuration file called `centengine.conf` that should have been installed in the `/etc/init.d/` directory. Starting and stopping the daemon should be as simple as:

```
$ start centengine # Start Centreon Engine
$ stop centengine  # Stop Centreon Engine
```

Please refer to the *initctl* manpage for the full list of operations available.

1.4.5 Nagios migration

Centreon Engine is compatible with Nagios. This means that it will recognize Nagios's configuration files and behave just like Nagios would. However some subtle differences arise when migrating from an existing Nagios setup. This document will explain what this differences are and how to modify them.

All example path in this documentation is based on CentOS i386. You probably need to change these path with you own configuration.

Prerequisites

Centreon Engine is assumed to have been properly installed in your nagios monitoring server.

Startup file

SysV init script (default) By default, Centreon Engine do not usually use the same files as Nagios does. This can cause issues when interfacing with Centreon. Modify `/etc/init.d/centengine`, replace the following variables, so that Centreon Engine use the exact same files and directories as Nagios.

```
var_dir=/var/log/nagios
config_file=/etc/nagios/nagios.cfg
command_file=$var_dir/rw/nagios.cmd
user=nagios
group=nagios
```

Upstart configuration file On systems with Upstart installed (such as recent Ubuntu versions), Centreon Engine provides a configuration file to control it. Modify it like below.

```
env HOME=/etc/nagios
env CFG_FILE="$HOME/nagios.cfg"
```

Nagios configuration file

Be sure you have defined all file paths in the Nagios configuration file. All these variables can be found in Centreon 2.3.5 and above. You need to modify these variables, because Centreon Engine (just like Nagios) use default values for variables that are not defined. This can cause issues when interfacing with Centreon.

```
log_file=/var/log/nagios/nagios.log
temp_file=/var/log/nagios/nagios.tmp
pl_file=/usr/sbin/pl.pl
log_archive_path=/var/log/nagios/archives
command_file=/var/log/nagios/rw/nagios.cmd
lock_file=/var/log/nagios/nagios.lock
state_retention_file=/var/log/nagios/status.sav
object_cache_file=/var/log/nagios/objects.cache
precached_object_file=/var/log/nagios/objects.precache
debug_file=/var/log/nagios/nagios.debug
```

Commands and notification

Centreon Engine is very fast compared to Nagios. The downside is, some rarely used features were deactivated. For example, commands are not processed by a shell before execution (Nagios did that). So if some of your commands use shell syntax (pipes, globbing, ...) you have to modify them.

For check or notification commands with multiple commands like this:

```
$ cat /etc/nagios/commands.cfg
define command{
    command_name host-notify-by-email
    command_line printf "Message..." | mail -s "Subject..." $CONTACTEMAILS$
}
```

The solution is to add 'sh -c' on the command line, so that the shell (sh) will execute and parse your command:

```
$ cat /etc/nagios/commands.cfg
define command{
    command_name host-notify-by-email
    command_line sh -c 'printf "Message..." | mail -s "Subject..." $CONTACTEMAILS$'
}
```

Warning: the new command obeys Nagios syntax. So if your old command contains single quotes ('), they need to be escaped.

External commands

Centreon Engine by itself does not need any system to control it. This is why we decided to extract the external commands system from the core. For Centreon Engine, this is now a broker module, like NDOUtils or Centreon Broker. However, Centreon uses it for some tasks (scheduled downtimes, comments, acknowledgement, ...) and is therefore required in such setups.

In Centreon, go to *Configuration -> Monitoring Engine -> main.cfg*. Choose your poller and go to the Data tab. Add a new Broker Module with the following configuration line:

```
/usr/lib/centreon-engine/externalcmd.so
```

The module does not need any configuration as Centreon Engine handle all the parsing process.

Centreon

Centreon needs to control Centreon Engine to perform tasks such as configuration checks. At time of writing, Centreon Engine it not a specific monitoring engine but will work as “Nagios”.

In Centreon go to *Configuration -> Centreon -> Pollers*. Choose your poller and modify Scheduler Information.

SysV init script (default)

```
Engine: Centreon-Engine (with centreon >= 2.3.5)
Nagios Init Script: /etc/init.d/centengine
Scheduler Binary: /usr/sbin/centengine
Nagios Statistics Binary: /usr/sbin/centenginestats
```

Upstart configuration file

```
Engine: Centreon-Engine (with centreon >= 2.3.5)
Nagios Init Script: service centengine
Scheduler Binary: /usr/sbin/centengine
Nagios Statistics Binary: /usr/sbin/centenginestats
```

Sudoers

SysV init script (default) Add these lines into */etc/sudoers*:

```
# Centengine Restart
CENTREON  ALL = NOPASSWD: /etc/init.d/centengine restart
# Centengine stop
CENTREON  ALL = NOPASSWD: /etc/init.d/centengine start
# Centengine stop
CENTREON  ALL = NOPASSWD: /etc/init.d/centengine stop
# Centengine reload
CENTREON  ALL = NOPASSWD: /etc/init.d/centengine reload
# Centengine test config
CENTREON  ALL = NOPASSWD: /usr/sbin/centengine -v *
# Centengine test for optim config
CENTREON  ALL = NOPASSWD: /usr/sbin/centengine -s *
```

Upstart configuration file Add these lines into /etc/sudoers:

```
# Centengine Restart
CENTREON ALL = NOPASSWD: service centengine restart
# Centengine stop
CENTREON ALL = NOPASSWD: service centengine start
# Centengine stop
CENTREON ALL = NOPASSWD: service centengine stop
# Centengine reload
CENTREON ALL = NOPASSWD: service centengine reload
# Centengine test config
CENTREON ALL = NOPASSWD: /usr/sbin/centengine -v *
# Centengine test for optim config
CENTREON ALL = NOPASSWD: /usr/sbin/centengine -s *
```

Stop/Start

To finish the migration you need to stop nagios and start centreon-engine.

1.4.6 Centreon Engine Stats Utility

Introduction

A utility called centenginestat is included in the Centreon Engine distribution. It is compiled and installed along with the main Centreon Engine daemon. The centenginestat utility allows you to obtain various information about a running Centreon Engine process that can be very helpful in *tuning performance*.

Usage Information

You can run the centenginestat utility with the `-help` option to get usage information.

Human-Readable Output

To obtain human-readable information on the performance of a running Centreon Engine process, run the centenginestat utility with the `-c` command line argument to specify your main configuration file location like such:

```
$ /usr/sbin/centenginestat -c /etc/centreon-engine/centengine.cfg
```

```
Centreon Engine Stats
Copyright (c) 2003-2007 Ethan Galstad (www.centengine.org)
License: GPL
```

```
CURRENT STATUS DATA
```

```
-----

Status File: /var/log/centreon-engine/status.dat
Status File Age: 0d 0h 0m 9s
Status File Version: 3.0prealpha-05202006
Program Running Time: 0d 5h 20m 39s
Centreon Engine PID: 10119
Used/High/Total Command Buffers: 0 / 0 / 64
Used/High/Total Check Result Buffers: 0 / 7 / 512
```

```

Total Services: 95
Services Checked: 94
Services Scheduled: 91
Services Actively Checked: 94
Services Passively Checked: 1
Total Service State Change: 0.000 / 78.950 / 1.026 %
Active Service Latency: 0.000 / 4.272 / 0.561 sec
Active Service Execution Time: 0.000 / 60.007 / 2.066 sec
Active Service State Change: 0.000 / 78.950 / 1.037 %
Active Services Last 1/5/15/60 min: 4 / 68 / 91 / 91
Passive Service State Change: 0.000 / 0.000 / 0.000 %
Passive Services Last 1/5/15/60 min: 0 / 0 / 0 / 0
Services Ok/Warn/Unk/Crit: 58 / 16 / 0 / 21
Services Flapping: 1
Services In Downtime: 0
Total Hosts: 24
Hosts Checked: 24
Hosts Scheduled: 24
Hosts Actively Checked: 24
Host Passively Checked: 0
Total Host State Change: 0.000 / 9.210 / 0.384 %
Active Host Latency: 0.000 / 0.446 / 0.219 sec
Active Host Execution Time: 1.019 / 10.034 / 2.764 sec
Active Host State Change: 0.000 / 9.210 / 0.384 %
Active Hosts Last 1/5/15/60 min: 5 / 22 / 24 / 24
Passive Host State Change: 0.000 / 0.000 / 0.000 %
Passive Hosts Last 1/5/15/60 min: 0 / 0 / 0 / 0
Hosts Up/Down/Unreach: 18 / 4 / 2
Hosts Flapping: 0
Hosts In Downtime: 0
Active Host Checks Last 1/5/15 min: 9 / 52 / 164
Scheduled: 4 / 23 / 75
On-demand: 3 / 23 / 69
Cached: 2 / 6 / 20
Passive Host Checks Last 1/5/15 min: 0 / 0 / 0
Active Service Checks Last 1/5/15 min: 9 / 80 / 244
Scheduled: 9 / 80 / 244
On-demand: 0 / 0 / 0
Cached: 0 / 0 / 0
Passive Service Checks Last 1/5/15 min: 0 / 0 / 0
External Commands Last 1/5/15 min: 0 / 0 / 0

```

As you can see, the utility displays a number of different metrics pertaining to the Centreon Engine process. Metrics which have multiple values are (unless otherwise specified) min, max and average values for that particular metric.

1.4.7 Centreon Engine Benchmarking Tools

Introduction

Centreon Engine provides a benchmarking tool designed to measure the speed of passive check result processing. The utility is named *centengine_bench_passive*.

Usage

- **-? -help** print tool help

- **-M --mode** specify the mode in which the tool should run (see below)
- **-h --activehosts** number of active hosts in the configuration (default is 0)
- **-s --activeservices** number of active services in the configuration (default is 0)
- **-H --passivehosts** number of passive hosts in the configuration (default is 1)
- **-S --passiveservices** number of passive services in the configuration (default is 100)
- **-c --count** number of passive check results to send (default is 1000)
- **-e --engine** path to Centreon Engine binary (default is `/usr/sbin/centengine`)
- **-m --module** Centreon Engine external command module (default is `/usr/lib64/centreon-engine/externalcmd.so`)

Modes

The benchmarking tool has three modes :

- **benchmark** perform a full benchmark test. It generates a Centreon Engine configuration and mesure time needed to process the expected number of passive check results
- **configuration** generate a configuration compatible with the later *command* mode. Note that when generating the configuration, all number of hosts and services must be equal to the numbers provided to the *command* mode.
- **command** generate external commands of passive check results. Note that the number of hosts and services provided in this mode must be equal to the numbers provided when generating the benchmarking configuration.

Use over a network

If you wish to test passive check processing over a network the best way to do it is to perform the following steps :

- generate a bencharking configuration using the *configuration* mode on the target server
- run Centreon Engine on the server using this configuration
- run the *netcat* tool listening on some port. Redirect its output to Centreon Engine external command FIFO (`nc -l -k -p 6000 > /tmp/centengine.cmd`)
- run the bencharking tool on one or multiple remote servers and redirect their output to a *netcat* that will connect to the target server (`centengine_bench_passive <params> | nc remoteserver 6000`)

1.4.8 SNMP Wrapper Integration

Introduction

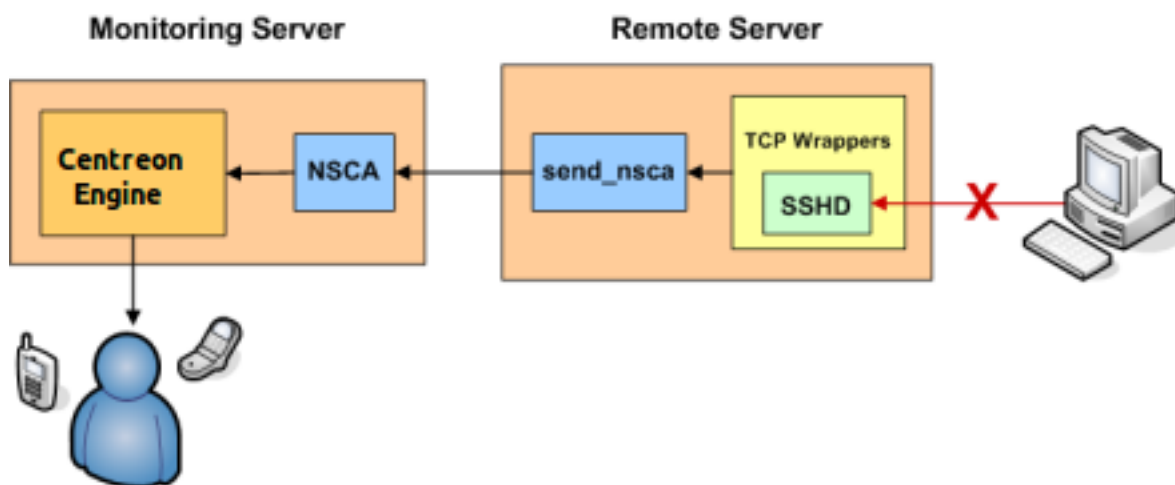
Note: Centreon Engine is not designed to be a replacement for a full-blown SNMP management application like HP OpenView or OpenNMS. However, you can set things up so that SNMP traps received by a host on your network can generate alerts in Centreon Engine.

As if designed to make the Gods of Hypocrisy die of laughter, SNMP is anything but simple. Translating SNMP traps and getting them into Centreon Engine (as passive check results) can be a bit tedious. To make this task easier, I suggest you check out Alex Burger's SNMP Trap Translator project located at <http://www.snmpptt.org>. When combined with Net-SNMP, SNMPPTT provides an enhanced trap handling system that can be integrated with Centreon Engine.

Yep, that's all.

1.4.9 TCP Wrapper Integration

Introduction



This document explains how to easily generate alerts in Centreon Engine for connection attempts that are rejected by TCP wrappers. For example, if an unauthorized host attempts to connect to your SSH server, you can receive an alert in Centreon Engine that contains the name of the host that was rejected. If you implement this on your Linux/Unix boxes, you'll be surprised how many port scans you can detect across your network.

These directions assume:

- You are already familiar with *passive checks* and how they work.
- You are already familiar with *volatile services* and how they work.
- The host which you are generating alerts for (i.e. the host you are using TCP wrappers on) is a remote host (called *firestorm* in this example). If you want to generate alerts on the same host that Centreon Engine is running you will need to make a few modifications to the examples I provide.
- You have installed the *NSCA daemon* on your monitoring server and the NSCA client (*send_nsca*) on the remote machine that you are generating TCP wrapper alerts from.

Defining A Service

If you haven't done so already, create a *host definition* for the remote host (*firestorm*).

Next, define a service in one of your *object configuration files* for the TCP wrapper alerts on host *firestorm*. The service definition might look something like this:

```
define service{
  host_name          firestorm
  service_description TCP Wrappers
  is_volatile         1
  active_checks_enabled 0
  passive_checks_enabled 1
  max_check_attempts  1
  check_command       check_none
  ...
}
```

There are some important things to note about the above service definition:

- The volatile option enabled. We want this option enabled because we want a notification to be generated for every alert that comes in.
- Active checks of the service as disabled, while passive checks are enabled. This means that the service will never be actively checked by Centreon Engine - all alert information will have to be received passively from an external source.
- The max_check_attempts value is set to 1. This guarantees you will get a notification when the first alert is generated.

Configuring TCP Wrappers

Now you're going to have to modify the `/etc/hosts.deny` file on firestorm. In order to have the TCP wrappers send an alert to the monitoring host whenever a connection attempt is denied, you'll have to add a line similar to the following:

```
ALL: ALL: RFC931: twist (/usr/lib/nagios/plugins/event_handlers/handle_tcp_wrapper %h %d) &
```

This line assumes that there is a script called `handle_tcp_wrapper` in the `/usr/lib/nagios/plugins/event_handlers/` directory on firestorm. We'll write that script next.

Writing The Script

The last thing you need to do is write the `handle_tcp_wrapper` script on firestorm that will send the alert back to the Centreon Engine server. It might look something like this:

```
#!/bin/sh
/usr/lib/nagios/plugins/event_handlers/submit_check_result firestorm "TCP Wrappers" 2 "Denied $2-$1"
```

Notice that the `handle_tcp_wrapper` script calls the `submit_check_result` script to actually send the alert back to the monitoring host. Assuming your Centreon Engine server is called monitor, the `submit_check_result` script might look like this:

```
#!/bin/sh
# Arguments
#   $1 = name of host in service definition
#   $2 = name/description of service in service definition
#   $3 = return code
#   $4 = output

/bin/echo -e "$1\t$2\t$3\t$4\n" | /usr/local/centengine/bin/send_nscd monitor -c /etc/centreon-engine
```

Finishing Up

You've now configured everything you need to, so all you have to do is restart the `inetd` process on firestorm and restart Centreon Engine on your monitoring server. That's it! When the TCP wrappers on firestorm deny a connection attempt, you should be getting alerts in Centreon Engine. The plugin output for the alert will look something like the following:

```
Denied sshd2-sdn-ar-002mnminnP321.dialsprint.net
```

1.4.10 MK Livestatus

The MK Livestatus module works with Centreon-Engine 1.3 but you can have some issues.

Undefined symbole

The first one is the undefined symbole

```
$ cat /var/log/centreon-engine/centengine.log | grep 'Error'
[1358499328] Error: Could not load module '/usr/lib64/centreon-engine/livestatus.o' -> load library f
```

To resolve this probleme, you need to load the Centreon External Command module before the MK Livestatus module. This is necessary to provide some symbole use by MK Livestatus. You just need to edit Centreon-Engine configuration file like that

```
$ cat /etc/centreon-engine/centengine.cfg | grep 'broker_module'
broker_module=/usr/lib64/centreon-engine/externalcmd.so
broker_module=/usr/lib64/centreon-engine/livestatus.so
```

Unable to bind address

The second one is the unable to bind address

```
$ cat /var/log/centreon-engine/centengine.log | grep 'livestatus:'
[1358499695] livestatus: Livestatus 1.2.0p3 by Mathias Kettner. Socket: '/usr/local/nagios/var/rw/live'
[1358499695] livestatus: Please visit us at http://mathias-kettner.de/
[1358499695] livestatus: Hint: please try out OMD - the Open Monitoring Distribution
[1358499695] livestatus: Please visit OMD at http://omdistro.org
[1358499695] livestatus: Unable to bind adress /usr/local/nagios/var/rw/live to UNIX socket: Aucun f
[1358499695] livestatus: deinitializing
```

MK Livestatus use nagios path by default. You need to set Centreon-Engine path. For that you need to edit Centreon-Engine configuration file like that

```
$ cat /etc/centreon-engine/centengine.cfg | grep 'livestatus'
broker_module=/usr/lib64/centreon-engine/livestatus.o /var/lib/centreon-engine/rw/livestatus.sock
```

1.5 Licensing

Centreon Engine is licensed under the terms of the [GNU General Public License Version 2](#) as published by the [Free Software Foundation](#). This gives you legal permission to copy, distribute and/or modify Centreon Engine under certain conditions. Read the `license.txt` file in the Centreon Engine distribution or read the [online version of the license](#) for more details.

Centreon Engine has started as a fork of the Nagios Core project. Nagios and the Nagios logo are trademarks, service-marks, registered trademarks or registered servicemarks owned by Nagios Enterprises, LLC. Centreon, the Centreon Logo, Merethis and the Merethis logo are trademarks, servicemarks, registered trademarks or registered servicemarks owned by Merethis. All other trademarks, servicemarks, registered trademarks, and registered servicemarks are the property of their respective owner(s).

All source code, binaries, documentation, information, and other files contained in this distribution are provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Several people have contributed to Centreon Engine or Nagios Core by either reporting bugs, suggesting improvements, writing plugins, etc. A list of some of the many contributors to the development of Centreon Engine or Nagios Core can be found in the `thanks.txt` file in the root of the Centreon Engine distribution.