Directive	Format	Description and Values
accept_passive_host_checks	**accept_passive_host_checks=<0/1>**	This option determines whether or not Nagios will accept 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)
accept_passive_service_checks	**accept_passive_service_checks=<0/1>**	This option determines whether or not Nagios will accept 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)
additional_freshness_latency	**additional_freshness_latency=<#>**	This option determines the number of seconds Nagios will add to any host or services freshness threshold it automatically calculates (e.g. those not specified explicity by the user). More information on freshness checking can be found here.
admin_email	**admin_email=<email_address>**	This is the email address for the administrator of the local machine (i.e. the one that Nagios is running on). This value can be used in notification commands by using the $ADMINEMAIL$ macro.
admin_pager	**admin_pager=<pager_number_or_pager_email_gateway>**	This is the pager number (or pager email gateway) for the administrator of the local machine (i.e. the one that Nagios is running on). The pager number/address can be used in notification commands by using the $ADMINPAGER$ macro.
allow_empty_hostgroup_assignment	**allow_empty_hostgroup_assignment=[01]**	This boolean option determines whether services, service dependencies, or host dependencies assigned to empty host groups (host groups with no host members) will cause Nagios to exit with error on start up (or during a configuration check) or not. The default behavior if the option is not present in the main configuration file is for Nagios to exit with error if any of these objects are associated with host groups that have no hosts associated with them. Enabling this option can be useful when: Configuration files or pre-cached object files are distributed across many Nagios pollers. Automation is used to generate a Nagios configuration file tree. There is one set of services, service dependencies, or host dependencies maintained by administrators for which users may add hosts through host group membership for convenience (e.g. a suite of checks associated with a remote monitoring agent) but for which there is no guarantee that users will use the services associated with the host group or that the hosts will exists (e.g. a remote agent that is being deprecated over time or hosts that are being decomissioned).
auto_reschedule_checks	**auto_reschedule_checks=<0/1>**	This option determines whether or not Nagios 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.
auto_rescheduling_interval	**auto_rescheduling_interval=<seconds>**	This option determines how often (in seconds) Nagios will attempt to automatically reschedule checks. This option only has an effect if the auto_reschedule_checks option is enabled. Default is 30 seconds.
auto_rescheduling_window	**auto_rescheduling_window=<seconds>**	This option determines the "window" of time (in seconds) that Nagios 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).
bare_update_check	**bare_update_check=<0/1>**	This option deterines what data Nagios will send to api.nagios.org when it checks for updates. By default, Nagios will send information on the current version of Nagios you have installed, as well as an indicator as to whether this was a new installation or not. Nagios Enterprises uses this data to determine the number of users running specific version of Nagios. Enable this option if you do not wish for this information to be sent.
broker_module	**broker_module=<modulepath> [moduleargs]**	This directive is used to specify an event broker module that should by loaded by Nagios 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 seperated from the module path by a space. !!! WARNING !!! Do NOT overwrite modules while they are being used by Nagios or Nagios 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 Nagios... The correct/safe way of updating a module is by using one of these methods: Shutdown Nagios, replace the module file, restart Nagios While Nagios is running: delete the original module file, move the new module file into place, restart Nagios
cached_host_check_horizon	**cached_host_check_horizon=<seconds>**	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. A value for this option that is too high may result in (temporarily) inaccurate host states, while a value that is too low 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.
cached_service_check_horizon	**cached_service_check_horizon=<seconds>**	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. A value for this option that is too high 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.
cfg_dir	**cfg_dir=<directory_name>**	This directive is used to specify a directory which contains object configuration files that Nagios should use for monitoring. All files in the directory with a .cfg extension are processed as object config files. Additionally, Nagios will recursively process all config files in subdirectories of the directory you specify here. You can seperate your configuration files into different directories and specify multiple cfg_dir= statements to have all config files in each directory processed.
cfg_file	**cfg_file=<file_name>**	This directive is used to specify an object configuration file containing object definitions that Nagios should use for monitoring. Object configuration files contain definitions for hosts, host groups, contacts, contact groups, services, commands, etc. You can seperate your configuration information into several files and specify multiple cfg_file= statements to have each of them processed.
check_external_commands	**check_external_commands=<0/1>**	This option determines whether or not Nagios will check the here. 0 = Don't check external commands 1 = Check external commands (default)
check_for_orphaned_hosts	**check_for_orphaned_hosts=<0/1>**	This option allows you to enable or disable checks for orphaned hoste 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 Nagios 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)
check_for_orphaned_services	**check_for_orphaned_services=<0/1>**	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 Nagios 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)
check_for_updates	**check_for_updates=<0/1>**	This option determines whether Nagios will automatically check to see if new updates (releases) are available. It is recommend that you enable this option to ensure that you stay on top of the latest critical patches to Nagios. Nagios is critical to you - make sure you keep it in good shape. Nagios will check once a day for new updates. Data collected by Nagios Enterprises from the update check is processed in accordance with our privacy policy - see http://api.nagios.org for details.

Directive	Format	Description and Values
check_host_freshness	**check_host_freshness=<0/1>**	This option determines whether or not Nagios will periodically check the "freshness" of host checks. Enabling this option is useful for helping to ensure that here. 0 = Don't check host freshness 1 = Check host freshness (default)
check_result_path	**check_result_path=<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). Note: Make sure that only a single instance of Nagios has access to the check result path. If multiple instances of Nagios have their check result path set to the same directory, you will run into problems with check results being processed (incorrectly) by the wrong instance of Nagios! Also note that because of the new core worker architecture in Nagios Core 4, this path is now only used for passive check results.
check_result_reaper_frequency	**check_result_reaper_frequency=<frequency_in_seconds>**	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 consitute the core of the monitoring logic in Nagios. Beginning with Nagios Core 4, active check results are fed back to the main Nagios Core process by the core workers as soon as they are received. As a result, this variable has no effect on active check results. It still applies to passive check results.
check_service_freshness	**check_service_freshness=<0/1>**	This option determines whether or not Nagios will periodically check the "freshness" of service checks. Enabling this option is useful for helping to ensure that here. 0 = Don't check service freshness 1 = Check service freshness (default)
check_workers	**check_workers=<#>**	This setting specifies how many worker process should be started when Nagios Core starts. Worker processes are used to perform host and service checks. If the number of workers is not specified, a default number of workers is determined based on the number of CPU cores on the system (1.5 workers per core). If not specified, there is always a minimum of 4 workers.
child_processes_fork_twice	**child_processes_fork_twice=<0/1>**	Previously, this option determined whether or not Nagios would fork() child processes twice when it executed host and service checks. Because of the new core worker architecture, the main Nagios Core process forks workers when it starts and workers only fork themselves if the plugin command is not "simple". 0 = Fork() just once 1 = Fork() twice
command_check_interval	**command_check_interval=<xxx>[s]**	Prior to Nagios Core 4, this option specified the interval between checks for external commands. Beginning with Nagios Core 4, external commands are processed as soon as they are received and this variable has no effect.
command_file	**command_file=<file_name>**	This is the file that Nagios will check for external commands to process. The here.
date_format	**date_format=<option>**	This option allows you to specify what kind of date/time format Nagios 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
debug_file	**debug_file=<file_name>**	This option determines where Nagios should write debugging information. What (if any) information is written is determined by the max_debug_file_size option.
debug_level	**debug_verbosity=<#>**	This option determines how much debugging information Nagios should write to the debug_file. 0 = Basic information 1 = More detailed information (default) 2 = Highly detailed information
enable_environment_macros	**enable_environment_macros=<0/1>**	This option determines whether or not the Nagios daemon will make all standard macros available as environment variables to your check, notification, event hander, etc. commands. In large Nagios 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)
enable_event_handlers	**enable_event_handlers=<0/1>**	This option determines whether or not Nagios will run external command or change it via the web interface. Values are as follows: 0 = Disable event handlers 1 = Enable event handlers (default)
enable_flap_detection	**enable_flap_detection=<0/1>**	This option determines whether or not Nagios 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 Nagios 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 external command or change it via the web interface. 0 = Don't enable flap detection (default) 1 = Enable flap detection
enable_notifications	**enable_notifications=<0/1>**	This option determines whether or not Nagios will send out external command or change it via the web interface. Values are as follows: 0 = Disable notifications 1 = Enable notifications (default)
enable_predictive_host_dependency_checks	**enable_predictive_host_dependency_checks=<0/1>**	This option determines whether or not Nagios will execute predictive checks of hosts that are being depended upon (as defined in here. 0 = Disable predictive checks 1 = Enable predictive checks (default)
enable_predictive_service_dependency_checks	**enable_predictive_service_dependency_checks=<0/1>**	This option determines whether or not Nagios will execute predictive checks of services that are being depended upon as defined in here. 0 = Disable predictive checks 1 = Enable predictive checks (default)
event_broker_options	**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
event_handler_timeout	**event_handler_timeout=<seconds>**	This is the maximum number of seconds that Nagios 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, Nagios will kill it off thinking it is a runaway processes.

Directive	Format	Description and Values
execute_host_checks	**execute_host_checks=<0/1>**	This option determines whether or not Nagios will execute on-demand and regularly scheduled host checks when it initially (re)starts. If this option is disabled, Nagios will not actively execute any host checks, although it can still accept external command or change it via the web interface. Values are as follows: 0 = Don't execute host checks 1 = Execute host checks (default)
execute_service_checks	**execute_service_checks=<0/1>**	This option determines whether or not Nagios will execute service checks when it initially (re)starts. If this option is disabled, Nagios will not actively execute any service checks and will remain in a sort of "sleep" mode (it can still accept external command or change it via the web interface. Values are as follows: 0 = Don't execute service checks 1 = Execute service checks (default)
external_command_buffer_slots	**external_command_buffer_slots=<#>**	Note: This is an advanced feature. Prior to Nagios Core 4, this option determined how many buffer slots Nagios would reserve for caching external commands that had been read from the external command file by a worker thread, but had not yet been processed by the main thread of the Nagios deamon. Each slot could hold one external command, so this option essentially determined how many commands could be buffered. For installations where you process a large number of passive checks (e.g. distributed setups), you may have neeeded to increase this number. Beginning with Nagios Core 4, external commands are processed as soon as they are received by a worker process and this variable has no effect.
free_child_process_memory	**free_child_process_memory=<0/1>**	This option determines whether or not Nagios will free memory in child processes when they are fork()ed off from the main process. By default, Nagios frees memory. However, if the use_large_installation_tweaks option is enabled, it will not. By defining this option in your configuration file, you are able to override things to get the behavior you want. 0 = Don't free memory 1 = Free memory
global_host_event_handler	**global_host_event_handler=<command>**	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 here.
global_service_event_handler	**global_service_event_handler=<command>**	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 here.
high_host_flap_threshold	**high_host_flap_threshold=<percent>**	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.
high_service_flap_threshold	**high_service_flap_threshold=<percent>**	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.
host_check_timeout	**host_check_timeout=<seconds>**	This is the maximum number of seconds that Nagios 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, Nagios will kill it off thinking it is a runaway processes.
host_down_disable_service_checks	**host_down_disable_service_checks=<0/1>**	This option will disable all service checks if the host is not in an UP state. While desirable in some environments, enabling this value can distort report values as the expected quantity of checks will not have been performed
host_freshness_check_interval	**host_freshness_check_interval=<seconds>**	This setting determines how often (in seconds) Nagios will periodically check the "freshness" of host check results. If you have disabled host freshness checking (with the here.
host_inter_check_delay_method	**host_inter_check_delay_method=<n/d/s/x.xx>**	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 Nagios 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
host_perfdata_command	**host_perfdata_command=<command>**	This option allows you to specify a command to be run after every host check to process host host definition is enabled.
host_perfdata_file	**host_perfdata_file=<file_name>**	This option allows you to specify a file to which host host definition is enabled.
host_perfdata_file_mode	**host_perfdata_file_mode=<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)
host_perfdata_file_processing_command	**host_perfdata_file_processing_command=<command>**	This option allows you to specify the command that should be executed to process the host_perfdata_file_processing_interval directive.
host_perfdata_file_processing_interval	**host_perfdata_file_processing_interval=<seconds>**	This option allows you to specify the interval (in seconds) at which the host performance data file processing command. A value of 0 indicates that the performance data file should not be processed at regular intervals.
host_perfdata_file_template	**host_perfdata_file_template=<template>**	This option determines what (and how) data is written to the 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.
illegal_macro_output_chars	**illegal_macro_output_chars=<chars...>**	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: $HOSTOUTPUT$ $HOSTPERFDATA$ $HOSTACKAUTHOR$ $HOSTACKCOMMENT$ $SERVICEOUTPUT$ $SERVICEPERFDATA$ $SERVICEACKAUTHOR$ $SERVICEACKCOMMENT$
illegal_object_name_chars	**illegal_object_name_chars=<chars...>**	This option allows you to specify illegal characters that cannot be used in host names, service descriptions, or names of other object types. Nagios 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.

Directive	Format	Description and Values
interval_length	**interval_length=<seconds>**	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. Important: 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).
lock_file	**lock_file=<file_name>**	This option specifies the location of the lock file that Nagios should create when it runs as a daemon (when started with the -d command line argument). This file contains the process id (PID) number of the running Nagios process.
log_archive_path	**log_archive_path=<path>**	This is the directory where Nagios should place log files that have been rotated. This option is ignored if you choose to not use the log rotation functionality.
log_current_states	**log_current_states=<0/1>**	This option determines whether or not Nagios will log host and service current states at the beginning of a newly created log file after log rotation occurs. In Nagios Core 3, current states were always logged after a log rotation. In Nagios Core 4, the default behavior is to log current states after log rotation, but it can be disabled by setting log_current_states=0. In a large installation, disabling the logging of current states after log rotation can save considerable amounts of disk space, especially if the logs are rotated frequently. This risk is that, if logs are aged off and deleted, you may not have sufficient state information to calculate things like availability. 0 = Disable logging current state after log rotation 1 = Enable logging current state after log rotation (default)
log_event_handlers	**log_event_handlers=<0/1>**	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 Nagios or first trying out your event handler scripts. 0 = Don't log event handlers 1 = Log event handlers
log_external_commands	**log_external_commands=<0/1>**	This variable determines whether or not Nagios will log log_passive_checks option. 0 = Don't log external commands 1 = Log external commands (default)
log_file	**log_file=<file_name>**	This variable specifies where Nagios should create its main log file. This should be the first variable that you define in your configuration file, as Nagios will try to write errors that it finds in the rest of your configuration data to this file. If you have log rotation enabled, this file will automatically be rotated every hour, day, week, or month.
log_host_retries	**log_host_retries=<0/1>**	This variable determines whether or not host check retries are logged. Logging host check retries is mostly useful when attempting to debug Nagios or test out host event handlers. 0 = Don't log host check retries 1 = Log host check retries
log_initial_states	**log_initial_states=<0/1>**	This variable determines whether or not Nagios 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
log_notifications	**log_notifications=<0/1>**	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
log_passive_checks	**log_passive_checks=<0/1>**	This variable determines whether or not Nagios will log 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)
log_rotation_method	**log_rotation_method=<n/h/d/w/m>**	This is the rotation method that you would like Nagios to use for your log file. Values are as follows: n = None (don't rotate the log - this is the default) h = Hourly (rotate the log at the top of each hour) d = Daily (rotate the log at midnight each day) w = Weekly (rotate the log at midnight on Saturday) m = Monthly (rotate the log at midnight on the last day of the month)
log_service_retries	**log_service_retries=<0/1>**	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 Nagios 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 Nagios or test out service event handlers. 0 = Don't log service check retries 1 = Log service check retries
low_host_flap_threshold	**low_host_flap_threshold=<percent>**	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.
low_service_flap_threshold	**low_service_flap_threshold=<percent>**	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.
max_check_result_file_age	**max_check_result_file_age=<seconds>**	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 :-).
max_check_result_reaper_time	**max_check_result_reaper_time=<seconds>**	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 Nagios for other portions of the monitoring logic. Beginning with Nagios Core 4, active check results are fed back to the main Nagios Core process by the core workers as soon as they are received. As a result, this variable has no effect on active check results. It still applies to passive check results.
max_concurrent_checks	**max_concurrent_checks=<max_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 Nagios, 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.

Directive	Format	Description and Values
max_debug_file_size	**max_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 Nagios.
max_host_check_spread	**max_host_check_spread=<minutes>**	This option determines the maximum number of minutes from when Nagios starts that all hosts (that are scheduled to be regularly checked) are checked. This option will automatically adjust the use_retained_scheduling_info option. Default value is 30 (minutes).
max_service_check_spread	**max_service_check_spread=<minutes>**	This option determines the maximum number of minutes from when Nagios starts that all services (that are scheduled to be regularly checked) are checked. This option will automatically adjust the use_retained_scheduling_info option. Default value is 30 (minutes).
nagios_group	**nagios_group=<groupname/GID>**	This is used to set the effective group that the Nagios process should run as. After initial program startup and before starting to monitor anything, Nagios will drop its effective privileges and run as this group. You may specify either a groupname or a GID.
nagios_user	**nagios_user=<username/UID>**	This is used to set the effective user that the Nagios process should run as. After initial program startup and before starting to monitor anything, Nagios will drop its effective privileges and run as this user. You may specify either a username or a UID.
notification_timeout	**notification_timeout=<seconds>**	This is the maximum number of seconds that Nagios 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, Nagios will kill it off thinking it is a runaway processes.
object_cache_file	**object_cache_file=<file_name>**	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 Nagios is (re)started and is used by the CGIs. It is intended to speed up config file caching in the CGIs and allow you to edit the source object config files while Nagios is running without affecting the output displayed in the CGIs. In Nagios Core 4, setting the path of the object_cache_file to '/dev/null' will cause Nagios Core not to cache object information. This can be done to speed up operations, but should not be done if the CGIs will be used.
obsess_over_hosts	**obsess_over_hosts=<0/1>**	This value determines whether or not Nagios will "obsess" over host checks results and run the 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
obsess_over_services	**obsess_over_services=<0/1>**	This value determines whether or not Nagios will "obsess" over service checks results and run the 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
ochp_command	**ochp_command=<command>**	This option allows you to specify a command to be run after every host check, which can be useful in host definition is enabled.
ochp_timeout	**ochp_timeout=<seconds>**	This is the maximum number of seconds that Nagios 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.
ocsp_command	**ocsp_command=<command>**	This option allows you to specify a command to be run after every service check, which can be useful in service definition is enabled.
ocsp_timeout	**ocsp_timeout=<seconds>**	This is the maximum number of seconds that Nagios 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.
passive_host_checks_are_soft	**passive_host_checks_are_soft=<0/1>**	This option determines whether or not Nagios will treat 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
perfdata_timeout	**perfdata_timeout=<seconds>**	This is the maximum number of seconds that Nagios will allow a 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.
precached_object_file	**precached_object_file=<file_name>**	This directive is used to specify a file in which a pre-processed, pre-cached copy of here.
process_performance_data	**process_performance_data=<0/1>**	This value determines whether or not Nagios will process host and service check performance data. 0 = Don't process performance data (default) 1 = Process performance data
query_socket	**query_socket=<path>**	This is the path to the Unix-domain socket used by the query handler interface. The default value is '/usr/local/nagios/var/rw/nagios.qh'.
resource_file	**resource_file=<file_name>**	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). The CGIs will not attempt to read resource files, so you can set restrictive permissions (600 or 660) on them to protect sensitive information. You can include multiple resource files by adding multiple resource_file statements to the main config file - Nagios will process them all. See the sample resource.cfg file in the sample-config/ subdirectory of the Nagios distribution for an example of how to define $USERn$ macros.
retain_state_information	**retain_state_information=<0/1>**	This option determines whether or not Nagios 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, Nagios will save all state information for hosts and services 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)
retained_contact_host_attribute_mask		
retained_contact_service_attribute_mask		WARNING: This is an advanced feature. You'll need to read the Nagios source code to use this option effectively. 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.
retained_host_attribute_mask		
retained_process_host_attribute_mask		
retained_process_service_attribute_mask		WARNING: This is an advanced feature. You'll need to read the Nagios source code to use this option effectively. These options determine which process attributes are NOT retained across program restarts. There are two masks because there are often separate host and service process attributes that can be changed. For example, host checks can be disabled at the program level, while service checks are still enabled. 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.

Directive	Format	Description and Values
retained_service_attribute_mask		WARNING: This is an advanced feature. You'll need to read the Nagios source code to use this option effectively. These options determine which host or service attributes are NOT retained across program restarts. 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 host and service attributes are retained.
retention_update_interval	**retention_update_interval=<minutes>**	This setting determines how often (in minutes) that Nagios will automatically save retention data during normal operation. If you set this value to 0, Nagios 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.
service_check_timeout	**service_check_timeout=<seconds>**	This is the maximum number of seconds that Nagios 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, Nagios will kill it off thinking it is a runaway processes.
service_check_timeout_state	**service_check_timeout_state=<state>**	This setting determines the state Nagios will report when a service check times out - that is does not respond within service_check_timeout seconds. This can be useful if a machine is running at too high a load and you do not want to consider a failed service check to be critical (the default). Valid settings are: c - Critical (default) u - Unknown w - Warning o - OK
service_freshness_check_interval	**service_freshness_check_interval=<seconds>**	This setting determines how often (in seconds) Nagios will periodically check the "freshness" of service check results. If you have disabled service freshness checking (with the here.
service_inter_check_delay_method	**service_inter_check_delay_method=<n/d/s/x.xx>**	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 Nagios 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
service_interleave_factor	**service_interleave_factor=<s|x>**	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 Nagios 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. The best way to understand how interleaving works is to watch the 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)
service_perfdata_command	**service_perfdata_command=<command>**	This option allows you to specify a command to be run after every service check to process service service definition is enabled.
service_perfdata_file	**service_perfdata_file=<file_name>**	This option allows you to specify a file to which service service definition is enabled.
service_perfdata_file_mode	**service_perfdata_file_mode=<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)
service_perfdata_file_processing_command	**service_perfdata_file_processing_command=<command>**	This option allows you to specify the command that should be executed to process the service_perfdata_file_processing_interval directive.
service_perfdata_file_processing_interval	**service_perfdata_file_processing_interval=<seconds>**	This option allows you to specify the interval (in seconds) at which the service performance data file processing command. A value of 0 indicates that the performance data file should not be processed at regular intervals.
service_perfdata_file_template	**service_perfdata_file_template=<template>**	This option determines what (and how) data is written to the 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.
sleep_time	**sleep_time=<seconds>**	Prior to Nagios Core 4, this was the number of seconds that Nagios would sleep before checking to see if the next service or host check in the scheduling queue should be executed. Note that Nagios would only sleep after it "caught up" with queued service checks that have fallen behind. Starting with Nagios Core 4, this variable has no effect.
soft_state_dependencies	**soft_state_dependencies=<0/1>**	This option determines whether or not Nagios will use soft state information when checking state type), enable this option. 0 = Don't use soft state dependencies (default) 1 = Use soft state dependencies
state_retention_file	**state_retention_file=<file_name>**	This is the file that Nagios will use for storing status, downtime, and comment information before it shuts down. When Nagios 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 Nagios retain state information between program restarts, you must enable the retain_state_information option.
status_file	**status_file=<file_name>**	This is the file that Nagios uses to store the current status, comment, and downtime information. This file is used by the CGIs so that current monitoring status can be reported via a web interface. The CGIs must have read access to this file in order to function properly. This file is deleted every time Nagios stops and recreated when it starts. In Nagios Core 4, setting the path of the status_flie to '/dev/null' will cause Nagios Core not to store status information. This can be done to speed up operations, but should not be done if the CGIs will be used.
status_update_interval	**status_update_interval=<seconds>**	This setting determines how often (in seconds) that Nagios will update status data in the status file. The minimum update interval is 1 second.
temp_file	**temp_file=<file_name>**	This is a temporary file that Nagios periodically creates to use when updating comment data, status data, etc. The file is deleted when it is no longer needed.
temp_path	**temp_path=<dir_name>**	This is a directory that Nagios can use as scratch space for creating temporary files used during the monitoring process. You should run tmpwatch, or a similiar utility, on this directory occassionally to delete files older than 24 hours.
translate_passive_host_checks	**translate_passive_host_checks=<0/1>**	This option determines whether or not Nagios will translate DOWN/UNREACHABLE passive host check results to their "correct" state from the viewpoint of the local Nagios 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
use_aggressive_host_checking	**use_aggressive_host_checking=<0/1>**	Nagios tries to be smart about how and when it checks the status of hosts. In general, disabling this option will allow Nagios 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 Nagios 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

Directive	Format	Description and Values
use_large_installation_tweaks	**use_large_installation_tweaks=<0/1>**	This option determines whether or not the Nagios 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. More information on what optimizations are taken when you enable this option can be found here. 0 = Don't use tweaks (default) 1 = Use tweaks
use_regexp_matching	**use_regexp_matching=<0/1>**	This option determines whether or not various directives in your here. 0 = Don't use regular expression matching (default) 1 = Use regular expression matching
use_retained_program_state	**use_retained_program_state=<0/1>**	This setting determines whether or not Nagios 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 state retention enabled, this option has no effect. 0 = Don't use retained program state 1 = Use retained program state (default)
use_retained_scheduling_info	**use_retained_scheduling_info=<0/1>**	This setting determines whether or not Nagios 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 Nagios, 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)
use_syslog	**use_syslog=<0/1>**	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
use_timezone	**use_timezone=<tz>**	This option allows you to override the default timezone that this instance of Nagios runs in. Useful if you have multiple instances of Nagios that need to run from the same server, but have different local times associated with them. If not specified, Nagios will use the system configured timezone. Note: If you use this option to specify a custom timezone, you will also need to alter the Apache configuration directives for the CGIs to specify the timezone you want.
use_true_regexp_matching	**use_true_regexp_matching=<0/1>**	If you've enabled regular expression matching of various object directives using the here. 0 = Don't use true regular expression matching (default) 1 = Use true regular expression matching