WSGIDaemonProcess

Description:	Configure a distinct daemon process for running applications.
Syntax:	`WSGIDaemonProcess` name `[` options `]`
Context:	server config, virtual host

The `WSGIDaemonProcess` directive can be used to specify that distinct daemon processes should be created to which the running of WSGI applications can be delegated. Where Apache has been started as the `root` user, the daemon processes can be run as a user different to that which the Apache child processes would normally be run as.

When distinct daemon processes are enabled and used, the process is dedicated to mod_wsgi and the only thing that the processes do is run the WSGI applications assigned to that process group. Any other Apache modules such as PHP or activities such as serving up static files continue to be run in the standard Apache child processes.

Note that having denoted that daemon processes should be created by using the `WSGIDaemonProcess` directive, the `WSGIProcessGroup` directive, or the `process-group` option of `WSGIScriptAlias` still needs to be used to delegate specific WSGI applications to execute within those daemon processes.

Also note that the name of the daemon process group must be unique for the whole server. That is, it is not possible to use the same daemon process group name in different virtual hosts.

Options which can be supplied to the `WSGIDaemonProcess` directive are:

processes=num
Defines the number of daemon processes that should be started in this process group. If not defined then only one process will be run in this process group.

Note that if this option is defined as `processes=1`, then the WSGI environment attribute called `wsgi.multiprocess` will be set to be `True` whereas not providing the option at all will result in the attribute being set to be `False`. This distinction is to allow for where some form of load balancing is used across process groups in the same Apache instance, or separate Apache instances. If you need to ensure that `wsgi.multiprocess` is `False` so that interactive debuggers will work, simply do not specify the `processes` option and allow the default single daemon process to be created in the process group.

threads=num
Defines the number of threads to be created to handle requests in each daemon process within the process group.

If this option is not defined then the default will be to create 15 threads in each daemon process within the process group.

Do not get carried away and set this to a very large number in the belief that it will somehow magically enable you to handle many more concurrent users. Any sort of increased value would only be appropriate where your code is I/O bound. If you code is CPU bound, you are better of using at most 3 to 5 threads per process and using more processes.

display-name=value
Defines a different name to show for the daemon process when using the `ps` command to list processes. If the value is `%{GROUP}` then the name will be `(wsgi:group)` where `group` is replaced with the name of the daemon process group.

Note that only as many characters of the supplied value can be displayed as were originally taken up by `argv0` of the executing process. Anything in excess of this will be truncated.

This feature may not work as described on all platforms. Typically it also requires a `ps` program with BSD heritage. Thus on some versions of Solaris UNIX the `/usr/bin/ps` program doesn't work, but `/usr/ucb/ps` does. Other programs which can display this value include `htop`.

home=directory
Defines an absolute path of a directory which should be used as the initial current working directory of the daemon processes within the process group.

If this option is not defined the initial current working directory will be set to be the home directory of the user that the daemon process is configured to run as using the `user` option to the `WSGIDaemonProcess` directive. Otherwise the current working directory of Apache when started will be used, which if Apache is being started from system init scripts, would usually be the system root directory.

user=name | user=#uid
Defines the UNIX user name or numeric user uid of the user that the daemon processes should be run as. If this option is not supplied the daemon processes will be run as the same user that Apache would run child processes, as defined by the User directive, and it is not necessary to set this to the Apache user yourself.

Note that this option is ignored if Apache wasn't started as the root user, in which case no matter what the settings, the daemon processes will be run as the user that Apache was started as.

Also be aware that mod_wsgi will not allow you to run a daemon process group as the root user due to the security risk of running a web application as root.

group=name | group=#gid
Defines the UNIX group name or numeric group gid of the primary group that the daemon processes should be run as. If this option is not supplied the daemon processes will be run as the same group that Apache would run child processes, as defined by the Group directive, and it is not necessary to set this to the Apache group yourself.

Note that this option is ignored if Apache wasn't started as the root user, in which case no matter what the settings, the daemon processes will be run as the group that Apache was started as.

supplementary-groups=group1 | supplementary-groups=group1,group2
Defines a list of additional UNIX groups that the user the daemon process group runs as, should be added to, in addition to primary UNIX group associated with that user. When specifying more than one group, separate the names of the groups with a comma.

umask=0nnn
Defines a value to be used for the umask of the daemon processes within the process group. The value must be provided as an octal number.

If this option is not defined then the umask of the user that Apache is initially started as will be inherited by the process. Typically the inherited umask would be '0022'.

lang=locale
Set the current language locale. This is the same as having set the `LANG` environment variable.

You will need to set this on many Linux systems where Apache when started up from system init scripts uses the default C locale, meaning that the default system encoding is ASCII. Unless you need a special language locale, set this to `en_US.UTF-8`.

Whether the `lang` or `locale` option works best can depend on the system being used. Set both if you aren't sure which is appropriate.

locale=locale
Set the current language locale. This is the same as having set the `LC_ALL` environment variable.

You will need to set this on many Linux systems where Apache when started up from system init scripts uses the default C locale, meaning that the default system encoding is ASCII. Unless you need a special language locale, set this to `en_US.UTF-8`.

Whether the `lang` or `locale` option works best can depend on the system being used. Set both if you aren't sure which is appropriate.

chroot=directory
Run the daemon process group process within a chroot jail. Use of a chroot jail is now deprecated due to the difficulty in setting up a chroot environment. It is recommended that you use more modern containerisation technologies such as Docker or runC.

script-user=name | script-user=#uid
Sets the user that must be the owner of any WSGI script file delegated to be run in the daemon process group. If the owner doesn't match a HTTP Forbidden response will be returned for any request.

Note that this doesn't change what user the daemon process group runs as at any time. If you want to set the user that the daemon process group runs as, use the `user` option.

Only one of `script-user` or `script-group` option can be used at the same time.

script-group=name | scrip-group=#gid
Sets the group that must be the group of any WSGI script file delegated to be run in the daemon process group. If the group doesn't match a HTTP Forbidden response will be returned for any request.

Note that this doesn't change what group the daemon process group runs as at any time. If you want to set the group that the daemon process group runs as, use the `group` option.

Only one of `script-user` or `script-group` option can be used at the same time.

python-home=directory
Set the location of the Python virtual environment to be used by the daemon processes. The directory to use is that which `sys.prefix` is set to for the Python virtual environment. The virtual environment can have been created by `virtualenv`, `pyvenv` or `python -m venv`.

Note that the Python virtual environment must have been created using the same base Python version as was used to compile the mod_wsgi module. You can't use this to force mod_wsgi to somehow use a different Python version than it was compiled for. If you want to use a different version of Python, you will need to reinstall mod_wsgi, compiling it for the version you want. It is not possible for the one mod_wsgi instance to run applications for both Python 2 and 3 at the same time.

python-path=directory | python-path=directory:directory
List of colon separated directories to add to the Python module search path, ie., `sys.path`.

Note that this is not strictly the same as having set the `PYTHONPATH` environment variable when running normal command line Python. When this option is used, the directories are added by calling `site.addsitedir()`. As well as adding the directory to `sys.path` this function has the effect of opening and interpreting any `.pth` files located in the specified directories.

If using a Python virtual environment, rather than use this option to refer to the `site-packages` directory of the Python virtual environment, you should use the `python-home` option to specify the root of the Python virtual environment instead.

In all cases, if the directory contains Python packages which have C extension components, those packages must have been installed using the same base Python version as was used to compile the mod_wsgi module. You should not mix packages from different Python versions or installations.

python-eggs=directory
Directory to be used as the Python egg cache directory. This is equivalent to having set the `PYTHON_EGG_CACHE` environment variable.

Note that the directory specified must exist and be writable by the user that the daemon process run as.

restart-interval=nnn
Defines a time limit on how long a daemon process should run before being restarted.

This might be use to periodically force restart the WSGI application processes when you have issues related to Python object reference count cycles, or incorrect use of in memory caching, which causes constant memory growth.

If this option is not defined, or is defined to be 0, then the daemon process will be persistent and will continue to service requests until Apache itself is restarted or shutdown.

Avoid setting this too low. This is because the constant restarting and reloading of your WSGI application may cause unecessary load on your system and affect performance.

You can use the `graceful-timeout` option in conjunction with this option to reduce the chances that an active request will be interrupted when a restart occurs due to the use of this option.

maximum-requests=nnn
Defines a limit on the number of requests a daemon process should process before it is shutdown and restarted.

This might be use to periodically force restart the WSGI application processes when you have issues related to Python object reference count cycles, or incorrect use of in memory caching, which causes constant memory growth.

If this option is not defined, or is defined to be 0, then the daemon process will be persistent and will continue to service requests until Apache itself is restarted or shutdown.

Avoid setting this to a low number of requests on a site which handles a lot of traffic. This is because the constant restarting and reloading of your WSGI application may cause unecessary load on your system and affect performance. Only use this option if you have no other choice due to a memory usage issue. Stop using it as soon as any memory issue has been resolved.

You can use the `graceful-timeout` option in conjunction with this option to reduce the chances that an active request will be interrupted when a restart occurs due to the use of this option.

inactivity-timeout=sss
Defines the maximum number of seconds allowed to pass before the daemon process is shutdown and restarted when the daemon process has entered an idle state. For the purposes of this option, being idle means there are no currently active requests and no new requests are being received.

This option exists to allow infrequently used applications running in a daemon process to be restarted, thus allowing memory being used to be reclaimed, with process size dropping back to the initial startup size before any application had been loaded or requests processed.

Note that after any restart of the WSGI application process, the WSGI application will need to be reloaded. This can mean that the first request received by a process after the process was restarted can be slower. If you WSGI application has a very high startup cost on CPU and time, it may not be a good idea to use the option.

See also the `request-timeout` option for forcing a process restart when requests block for a specified period of time.

Note that similar functionality to that of the `request-timeout` option, for forcing a restart when requests blocked, was part of what was implemented by the `inactivity-timeout` option. The request timeout was broken out into a separate feature in version 4.1.0 of mod_wsgi.

request-timeout=sss
Defines the maximum number of seconds that a request is allowed to run before the daemon process is restarted. This can be used to recover from a scenario where a request blocks indefinitely, and where if all request threads were consumed in this way, would result in the whole WSGI application process being blocked.

How this option is seen to behave is different depending on whether a daemon process uses only one thread, or more than one thread for handling requests, as set by the `threads` option.

If there is only a single thread, and so the process can only handle one request at a time, as soon as the timeout has passed, a restart of the process will be initiated.

If there is more than one thread, the request timeout is applied to the average running time for any requests, across all threads. This means that a request can run longer than the request timeout. This is done to reduce the possibility of interupting other running requests, and causing a user to see a failure. So where there is still capacity to handle more requests, restarting of the process will be delayed if possible.

deadlock-timeout=sss
Defines the maximum number of seconds allowed to pass before the daemon process is shutdown and restarted after a potential deadlock on the Python GIL has been detected. The default is 300 seconds.

This option exists to combat the problem of a daemon process freezing as the result of a rouge Python C extension module which doesn't properly release the Python GIL when entering into a blocking or long running operation.

startup-timeout=sss
Defines the maximum number of seconds allowed to pass waiting to see if a WSGI script file can be loaded successfully by a daemon process. When the timeout is passed, the process will be restarted.

This can be used to force the reloading of a process when a transient issue occurs on the first attempt to load the WSGI script file, but subsequent attempts still fail because a Python package that was loaded has retained state that prevents attempts to run initialisation a second time within the same process. The Django package can cause this scenario as the initialisation of Django itself can no longer be attempted more than once in the same process.

graceful-timeout=sss
When `maximum-requests` is used and the maximum has been reached, or `cpu-time-limit` is used and the CPU limit reached, or `restart-interval` is used and the time limit reached, if `graceful-timeout` is set, then the process will continue to run for the number of second specified by this option, while still accepting new requests, to see if the process reaches an idle state. If the process reaches an idle state, it will then be resarted immediately. If the process doesn't reach an idle state and the graceful restart timeout expires, the process will be restarted, even if it means that requests may be interrupted.

eviction-timeout=sss
When a daemon process is sent the graceful restart signal, usually `SIGUSR1`, to restart a process, this timeout controls how many seconds the process will wait, while still accepting new requests, before it reaches an idle state with no active requests and shutdown.

If this timeout is not specified, then the value of the `graceful-timeout` will instead be used. If the `graceful-timeout` is not specified, then the restart when sent the graceful restart signal will instead happen immediately, with the process being forcibly killed, if necessary, when the shutdown timeout has expired.

shutdown-timeout=sss
Defines the maximum number of seconds allowed to pass when waiting for a daemon process to shutdown. When this timeout has been reached the daemon process will be forced to exited even if there are still active requests or it is still running Python exit functions. The shutdown timeout is applied after any graceful restart timeout or eviction timeout if they have been specified. No new requests are accepted during the shutdown timeout is being applied.

If this option is not defined, then the shutdown timeout will be set to 5 seconds. Note that this option does not change the shutdown timeout applied to daemon processes when Apache itself is being stopped or restarted. That timeout value is defined internally to Apache as 3 seconds and cannot be overridden.

connect-timeout=sss
Defines the maximum amount of time for an Apache child process to wait trying to get a successful connection to the mod_wsgi daemon processes. This defaults to 15 seconds.

socket-timeout=sss
Defines the timeout on individual reads/writes on the socket connection between the Apache child processes and the mod_wsgi daemon processes. If this is not specified, the number of seconds specified by the Apache Timeout directive will be used instead.

queue-timeout=sss
Defines the timeout on how long to wait for a mod_wsgi daemon process to accept a request for processing.

This option is to allow one to control what to do when backlogging of requests occurs. If the daemon process is overloaded and getting behind, then it is more than likely that a user will have given up on the request anyway if they have to wait too long. This option allows you to specify that a request that was queued up waiting for too long is discarded, allowing any transient backlog to be quickly discarded and not simply cause the daemon process to become even more backlogged. When this occurs the user will recieve a 504 Gateway Time Out response.

listen-backlog=nnn
Defines the depth of the daemon process socket listener queue. By default the limit is 100, although this is actually a hint, as different operating systems can have different limits on the maximum value or otherwise treat it in special ways.a

This option can be set, along with `queue-timeout` to try and better handle back logging when the WGSI application gets overloaded.

socket-user=name | socket-user=#uid
Set the owner of the UNIX listener socket for the daemon process group.

This can be used when using the Apache PrivilegesMode directive with value of `SECURE` to change the owner of the socket from the default Apache user, to the user under which the Apache child process which is attempting to connect to the daemon process group, will run when handling requests. This is necessary otherwise the Apache child worker process will not be able to connect to the listener socket for the mod_wsgi daemon process to proxy the request to the WSGI application.

This option can also be used when using third party Apache modules such as mod_ruid, mod_ruid2, mod_suid as well as the ITK MPM for Apache.

cpu-time-limit=sss
Define the maximum amount of CPU time a daemon process is allowed to consume before a shutdown is triggered and the daemon process restarted. The point of this is to provide some means of controlling potentially run away processes due to bad code that gets stuck in heavy processing loops.

Note that CPU time used is recorded from when the daemon process is first created. This means that a process will eventually reach the limit in normal use and would be restarted. You can use the `graceful-timeout` option to reduce the chances that an active request will be interrupted.

cpu-priority=num
Sets the scheduling priority set to the daemon processes. This can be a number of the range -20 to 20. The default priority is 0. A lower priority gives more favourable scheduling.

memory-limit=num
Sets the maximum amount of memory a daemon process can use. This will have no affect on some platforms as `RLIMIT_AS`/`RLIMIT_DATA` with `setrlimit()` isn't always implemented. For example MacOS X and older Linux kernel versions do not implement this feature. You will need to test whether this feature works or not before depending on it.

virtual-memory-limit=num
Sets the maximum amount of virtual memory a daemon process can use. This will have no affect on some platforms as `RLIMIT_VMEM` with `setrlimit()` isn't always implemented. You will need to test whether this feature works or not before depending on it.

stack-size=nnn
The amount of virtual memory in bytes to be allocated for the stack corresponding to each thread created by mod_wsgi in a daemon process.

This option would be used when running Linux in a VPS system which has been configured with a quite low 'Memory Limit' in relation to the 'Context RSS' and 'Max RSS Memory' limits. In particular, the default stack size for threads under Linux is 8MB is quite excessive and could for such a VPS result in the 'Memory Limit' being exceeded before the RSS limits were exceeded. In this situation, the stack size should be dropped down to be in the region of 512KB (524288 bytes).

receive-buffer-size=nnn
Defines the UNIX socket buffer size for data being received by the daemon process from the Apache child process.

This option may need to be used to override small default values set by certain operating systems and would help avoid possibility of deadlock between Apache child process and daemon process when the WSGI application generates large responses but doesn't consume request content. In general such deadlock problems would not arise with well behaved WSGI applications, but some spam bots attempting to post data to web sites are known to trigger the problem.

The maximum possible value that can be set for the buffer size is operating system dependent and will need to be calculated through trial and error.

send-buffer-size=nnn
Defines the UNIX socket buffer size for data being sent in the direction daemon process back to Apache child process.

This option may need to be used to override small default values set by certain operating systems and would help avoid possibility of deadlock between Apache child process and daemon process when the WSGI application generates large responses but doesn't consume request content. In general such deadlock problems would not arise with well behaved WSGI applications, but some spam bots attempting to post data to web sites are known to trigger the problem.

The maximum possible value that can be set for the buffer size is operating system dependent and will need to be calculated through trial and error.

header-buffer-size=nnn
Defines the maximum size that a response header/value can be that is returned from a WSGI application. The default size is 32768 bytes. This might need to be overridden where excessively large response headers are returned, such as in custom authentication challenge schemes which use the `WWW-Authenticate` header.

response-buffer-size=nnn
Defines the maximum number of bytes that will be buffered for a response in the Apache child processes when proxying the response body from the WSGI application. The default size is 65536 bytes. Be careful increasing this to provide extra buffering of responses as it contributes to the runtime memory size of the Apache child processes.

response-socket-timeout=nnn
Defines the maximum number of seconds allowed to pass before timing out on a write operation back to the HTTP client when the response buffer has filled and data is being forcibly flushed. Defaults to 0 seconds indicating that it will default to the value of the `socket-timeout` option.

To delegate a particular WSGI application to run in a named set of daemon processes, the `WSGIProcessGroup` directive should be specified in appropriate context for that application, or the `process-group` option used on the `WSGIScriptAlias` directive. If neither is used to delegate the WSGI application to run in a daemon process group, the application will be run within the standard Apache child processes.

If the `WSGIDaemonProcess` directive is specified outside of all virtual host containers, any WSGI application can be delegated to be run within that daemon process group. If the `WSGIDaemonProcess` directive is specified within a virtual host container, only WSGI applications associated with virtual hosts with the same server name as that virtual host can be delegated to that set of daemon processes.

In the case where you have two separate `VirtualHost` definitions for the same `ServerName`, but where one is for port 80 and the other for port 443, specify the `WSGIDaemonProcess` directive in the first `VirtualHost`. You can then refer to that daemon process group by name from the second `VirtualHost`. Using one daemon process group across the two virtual hosts in this case is preferred as then you do not have two whole separate instances of your application for port 80 and 443.

```
<VirtualHost *:80>
ServerName www.site1.com

WSGIDaemonProcess www.site1.com user=joe group=joe processes=2 threads=25
WSGIProcessGroup www.site1.com

...
</VirtualHost>

<VirtualHost *:443>
ServerName www.site1.com

WSGIProcessGroup www.site1.com

...
</VirtualHost>
```

When `WSGIDaemonProcess` is associated with a virtual host, the error log associated with that virtual host will be used for all Apache error log output from mod_wsgi rather than it appear in the main Apache error log.

For example, if a server is hosting two virtual hosts and it is desired that the WSGI applications related to each virtual host run in distinct processes of their own and as a user which is the owner of that virtual host, the following could be used:

```
<VirtualHost *:80>
ServerName www.site1.com
CustomLog logs/www.site1.com-access_log common
ErrorLog logs/ww.site1.com-error_log

WSGIDaemonProcess www.site1.com user=joe group=joe processes=2 threads=25
WSGIProcessGroup www.site1.com

...
</VirtualHost>

<VirtualHost *:80>
ServerName www.site2.com
CustomLog logs/www.site2.com-access_log common
ErrorLog logs/www.site2.com-error_log

WSGIDaemonProcess www.site2.com user=bob group=bob processes=2 threads=25
WSGIProcessGroup www.site2.com

...
</VirtualHost>
```

Note that the `WSGIDaemonProcess` directive and corresponding features are not available on Windows.