Gunicorn Documentation

Release 0.17.2

Benoit Chesneau

April 03, 2013

CONTENTS

Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX. It's a pre-fork worker model ported from Ruby's Unicorn project. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

CHAPTER

ONE

FEATURES

- Natively supports WSGI, Django, and Paster
- Automatic worker process management
- Simple Python configuration
- Multiple worker configurations
- Various server hooks for extensibility
- Compatible with Python 2.x >= 2.6

Note: main Gunicorn website on http://gunicorn.org

Note: gunicorn source code is hosted on Github

CHAPTER

TWO

CONTENTS

2.1 Installation

Follow the following steps to install Gunicorn.

2.1.1 Requirements

- **Python 2.x >= 2.6** (Python 3.x will be supported soon)
- setuptools >= 0.6c6
- nosetests (for the test suite only)

2.1.2 With easy_install

If you don't already have `easy_install` available you'll want to download and run the `ez_setup.py` script:

```
$ curl -O http://peak.telecommunity.com/dist/ez_setup.py
$ sudo python ez_setup.py -U setuptools
```

To install or upgrade to the latest released version of Gunicorn:

```
$ sudo easy_install -U gunicorn
```

Note: There is a limited support version of Gunicorn that is compatible with Python 2.4. This fork is managed by Randall Leeds and can be found here on github. To install this version you must specify the full url to something like `pip`. This hasn't been tested wtih `easy_install` just yet:

```
$ pip install -f http://github.com/tilgovi/gunicorn/tarball/py24 gunicorn
```

2.1.3 From Source

You can install Gunicorn from source just as you would install any other Python package. Gunicorn uses setuptools which will automatically fetch all dependencies (including setuptools itself).

You can download a tarball of the latest sources from GitHub Downloads or fetch them with git:

```
# Using git:
$ git clone git://github.com/benoitc/gunicorn.git
$ cd gunicorn

# Or using a tarball:
$ wget http://github.com/benoitc/gunicorn/tarball/master -o gunicorn.tar.gz
$ tar -xvzf gunicorn.tar.gz
$ cd gunicorn-$HASH/

# Install
$ sudo python setup.py install
```

If you've cloned the git repository, its highly recommended that you use the `develop` command which will allow you to use Gunicorn from the source directory. This will allow you to keep up to date with development on GitHub as well as make changes to the source:

```
$ python setup.py develop
```

2.1.4 Async Workers

You may also want to install Eventlet or Gevent if you expect that your application code may need to pause for extended periods of time during request processing. Check out the design docs for more information on when you'll want to consider one of the alternate worker types.

```
$ easy_install -U greenlet  # Required for both
$ easy_install -U eventlet  # For eventlet workers
$ easy_install -U gevent    # For gevent workers
```

Note: If installing `greenlet` fails you probably need to install the Python headers. These headers are available in most package managers. On Ubuntu the package name for `apt-get` is `python-dev`.

Gevent also requires that `libevent` 1.4.x or 2.0.4 is installed. This could be a more recent version than what is available in your package manager. If Gevent fails to build even with libevent installed, this is the most likely reason.

2.1.5 Debian GNU/Linux

If you are using Debian GNU/Linux and it is recommended that you use system packages to install Gunicorn except maybe when you want to use different versions of gunicorn with virtualenv. This has a number of advantages:

- Zero-effort installation: Automatically starts multiple Gunicorn instances based on configurations defined in `/etc/gunicorn.d`.
- Sensible default locations for logs (`/var/log/gunicorn`). Logs can be automatically rotated and compressed using `logrotate`.
- Improved security: Can easily run each Gunicorn instance with a dedicated UNIX user/group.
- Sensible upgrade path: Upgrades to newer versions result in less downtime, handle conflicting changes in configuration options, and can be quickly rolled back in case of incompatibility. The package can also be purged entirely from the system in seconds.

Stable ("squeeze")

The version of Gunicorn in the Debian "stable" distribution is 0.10.0 (July 2010). It is not recommended that you use this version.

However, you can use the most recent version by using Debian Backports. First, copy the following line to your `/etc/apt/sources.list`:

```
deb http://backports.debian.org/debian-backports squeeze-backports main
```

Then, update your local package lists:

```
$ sudo apt-get update
```

You can then install the latest version using:

```
$ sudo apt-get -t squeeze-backports install gunicorn
```

Testing ("wheezy") / Unstable ("sid")

"wheezy" and "sid" contain the latest released version of Gunicorn. You can install it in the usual way:

```
$ sudo apt-get install gunicorn
```

2.1.6 Ubuntu

If you use Ubuntu, you can update your system with packages from our PPA by adding `ppa:gunicorn/ppa` to your system's Software Sources. Use the `apt-add-repository` command from the `python-software-properties` package to add the Gunicorn software source.

```
$ sudo apt-add-repository ppa:gunicorn/ppa
```

Or this PPA can be added to your system manually by copying the lines below and adding them to your system's software sources:

```
deb http://ppa.launchpad.net/gunicorn/ppa/ubuntu lucid main
deb-src http://ppa.launchpad.net/gunicorn/ppa/ubuntu lucid main
```

Replace 'lucid' with your Ubuntu distribution series.

Signing key

```
1024R/5370FF2A
```

Fingerprint

```
FC7B41B54C9B8476D9EC22A2C6773E575370FF2A
```

2.2 Running Gunicorn

You can run Gunicorn by using commands or integrate with Django or Paster. For deploying Gunicorn in production see *Deploying Gunicorn*.

2.2.1 Commands

After installing Gunicorn you will have access to three command line scripts that can be used for serving the various supported web frameworks:

- `gunicorn`
- `gunicorn_django`
- `gunicorn_paster`

gunicorn

The first and most basic script is used to serve 'bare' WSGI applications that don't require a translation layer. Basic usage:

```
$ gunicorn [OPTIONS] APP_MODULE
```

Where `APP_MODULE` is of the pattern `$(MODULE_NAME):$(VARIABLE_NAME)`. The module name can be a full dotted path. The variable name refers to a WSGI callable that should be found in the specified module.

Example with test app:

```
$ cd examples
$ cat test.py
# -*- coding: utf-8 -
#
# This file is part of gunicorn released under the MIT license.
# See the NOTICE for more information.

def app(environ, start_response):
    """Simplest possible application object"""
    data = 'Hello, World!\n'
    status = '200 OK'
    response_headers = [
        ('Content-type','text/plain'),
        ('Content-Length', str(len(data)))
    ]
    start_response(status, response_headers)
    return iter([data])

$ gunicorn --workers=2 test:app
```

gunicorn_django

You might not have guessed it, but this script is used to serve Django applications. Basic usage:

```
$ gunicorn_django [OPTIONS] [SETTINGS_PATH]
```

By default `SETTINGS_PATH` will look for `settings.py` in the current directory.

Example with your Django project:

```
$ cd path/to/yourdjangoproject
$ gunicorn_django --workers=2
```

Note: If you run Django 1.4 or newer, it's highly recommended to simply run your application with the WSGI interface using the gunicorn command.

gunicorn_paster

Yeah, for Paster-compatible frameworks (Pylons, TurboGears 2, ...). We apologize for the lack of script name creativity. And some usage:

```
$ gunicorn_paster [OPTIONS] paste_config.ini
```

Simple example:

```
$ cd yourpasteproject
$ gunicorn_paste --workers=2 development.ini
```

2.2.2 Integration

Alternatively, we also provide integration for both Django and Paster applications in case your deployment strategy would be better served by such invocation styles.

Django ./manage.py

You can add a `run_gunicorn` command to your `./manage.py` simply by adding gunicorn to your `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ...
    "gunicorn",
)
```

Then you can run:

```
python manage.py run_gunicorn
```

paster serve

If you're wanting to keep on keeping on with the usual paster serve command, you can specify the Gunicorn server settings in your configuration file:

```
[server:main]
use = egg:gunicorn#main
host = 127.0.0.1
port = 5000
```

And then as per usual:

```
$ cd yourpasteproject
$ paster serve development.ini workers=2
```

2.3 Configuration Overview

Gunicorn pulls configuration information from three distinct places.

The first place that Gunicorn will read configuration from is the framework specific configuration file. Currently this only affects Paster applications.

The second source of configuration information is a configuration file that is optionally specified on the command line. Anything specified in the Gunicorn config file will override any framework specific settings.

Lastly, the command line arguments used to invoke Gunicorn are the final place considered for configuration settings. If an option is specified on the command line, this is the value that will be used.

Once again, in order of least to most authoritative:

1. Framework Settings
2. Configuration File
3. Command Line

Note: To check your configuration when using the command line or the configuration file you can run the following command:

```
$ gunicorn --check-config
```

It also allows you to know if your application can be launched.

2.3.1 Framework Settings

Currently, only Paster applications have access to framework specific settings. If you have ideas for providing settings to WSGI applications or pulling information from Django's settings.py feel free to open an issue to let us know.

Paster Applications

In your INI file, you can specify to use Gunicorn as the server like such:

```
[server:main]
use = egg:gunicorn#main
host = 192.168.0.1
port = 80
workers = 2
proc_name = brim
```

Any parameters that Gunicorn knows about will automatically be inserted into the base configuration. Remember that these will be overridden by the config file and/or the command line.

2.3.2 Configuration File

The configuration file should be a valid Python source file. It only needs to be readable from the file system. More specifically, it does not need to be importable. Any Python is valid. Just consider that this will be run every time you start Gunicorn (including when you signal Gunicorn to reload).

To set a parameter, just assign to it. There's no special syntax. The values you provide will be used for the configuration values.

For instance:

```
import multiprocessing

bind = "127.0.0.1:8000"
workers = multiprocessing.cpu_count() * 2 + 1
```

On a side note, Python's older than 2.6 can use sysconf to get the number of processors:

```
import os

def numCPUs():
    if not hasattr(os, "sysconf"):
        raise RuntimeError("No sysconf detected.")
    return os.sysconf("SC_NPROCESSORS_ONLN")
```

2.3.3 Command Line

If an option is specified on the command line, it overrides all other values that may have been specified in the app specific settings, or in the optional configuration file. Not all Gunicorn settings are available to be set from the command line. To see the full list of command line settings you can do the usual:

```
$ gunicorn -h
```

There is also a `--version` flag available to the command line scripts that isn't mentioned in the list of settings.

2.3.4 Settings

This is an exhaustive list of settings for Gunicorn. Some settings are only able to be set from a configuration file. The setting name is what should be used in the configuration file. The command line arguments are listed as well for reference on setting at the command line.

Config File

config

- `-c FILE, --config FILE`
- `None`

The path to a Gunicorn config file.

Only has an effect when specified on the command line or as part of an application specific configuration.

Server Socket

bind

- `-b ADDRESS, --bind ADDRESS`
- `['127.0.0.1:8000']`

The socket to bind.

A string of the form: 'HOST', 'HOST:PORT', 'unix:PATH'. An IP is a valid HOST.

Multiple addresses can be bound. ex.:

```
$ gunicorn -b 127.0.0.1:8000 -b [::1]:8000 test:app
```

will bind the *test:app* application on localhost both on ipv6 and ipv4 interfaces.

backlog

- `--backlog INT`
- `2048`

The maximum number of pending connections.

This refers to the number of clients that can be waiting to be served. Exceeding this number results in the client getting an error when attempting to connect. It should only affect servers under significant load.

Must be a positive integer. Generally set in the 64-2048 range.

Worker Processes

workers

- `-w INT, --workers INT`
- `1`

The number of worker process for handling requests.

A positive integer generally in the 2-4 x $(NUM_CORES) range. You'll want to vary this a bit to find the best for your particular application's work load.

worker_class

- `-k STRING, --worker-class STRING`
- `sync`

The type of workers to use.

The default class (sync) should handle most 'normal' types of workloads. You'll want to read http://gunicorn.org/design.html for information on when you might want to choose one of the other worker classes.

A string referring to one of the following bundled classes:

- `sync`
- `eventlet` - Requires eventlet >= 0.9.7
- `gevent` - Requires gevent >= 0.12.2 (?)
- `tornado` - Requires tornado >= 0.2

Optionally, you can provide your own worker by giving gunicorn a python path to a subclass of gunicorn.workers.base.Worker. This alternative syntax will load the gevent class: `gunicorn.workers.ggevent.GeventWorker`. Alternatively the syntax can also load the gevent class with `egg:gunicorn#gevent`

worker_connections

- `--worker-connections INT`
- `1000`

The maximum number of simultaneous clients.

This setting only affects the Eventlet and Gevent worker types.

max_requests

- `--max-requests INT`
- `0`

The maximum number of requests a worker will process before restarting.

Any value greater than zero will limit the number of requests a work will process before automatically restarting. This is a simple method to help limit the damage of memory leaks.

If this is set to zero (the default) then the automatic worker restarts are disabled.

timeout

- `-t INT, --timeout INT`
- `30`

Workers silent for more than this many seconds are killed and restarted.

Generally set to thirty seconds. Only set this noticeably higher if you're sure of the repercussions for sync workers. For the non sync workers it just means that the worker process is still communicating and is not tied to the length of time required to handle a single request.

graceful_timeout

- `--graceful-timeout INT`
- `30`

Timeout for graceful workers restart.

Generally set to thirty seconds. How max time worker can handle request after got restart signal. If the time is up worker will be force killed.

keepalive

- `--keep-alive INT`
- `2`

The number of seconds to wait for requests on a Keep-Alive connection.

Generally set in the 1-5 seconds range.

Security

limit_request_line

- `--limit-request-line INT`
- `4094`

The maximum size of HTTP request line in bytes.

This parameter is used to limit the allowed size of a client's HTTP request-line. Since the request-line consists of the HTTP method, URI, and protocol version, this directive places a restriction on the length of a request-URI allowed for a request on the server. A server needs this value to be large enough to hold any of its resource names, including any information that might be passed in the query part of a GET request. Value is a number from 0 (unlimited) to 8190.

This parameter can be used to prevent any DDOS attack.

limit_request_fields

- `--limit-request-fields INT`
- `100`

Limit the number of HTTP headers fields in a request.

This parameter is used to limit the number of headers in a request to prevent DDOS attack. Used with the *limit_request_field_size* it allows more safety. By default this value is 100 and can't be larger than 32768.

limit_request_field_size

- `--limit-request-field_size INT`
- `8190`

Limit the allowed size of an HTTP request header field.

Value is a number from 0 (unlimited) to 8190. to set the limit on the allowed size of an HTTP request header field.

Debugging

debug

- `--debug`
- `False`

Turn on debugging in the server.

This limits the number of worker processes to 1 and changes some error handling that's sent to clients.

spew

- `--spew`
- `False`

Install a trace function that spews every line executed by the server.

This is the nuclear option.

check_config

- `--check-config`
- `False`

Check the configuration..

Server Mechanics

preload_app

- `--preload`
- `False`

Load application code before the worker processes are forked.

By preloading an application you can save some RAM resources as well as speed up server boot times. Although, if you defer application loading to each worker process, you can reload your application code easily by restarting workers.

daemon

- `-D, --daemon`
- `False`

Daemonize the Gunicorn process.

Detaches the server from the controlling terminal and enters the background.

pidfile

- `-p FILE, --pid FILE`
- `None`

A filename to use for the PID file.

If not set, no PID file will be written.

user

- `-u USER, --user USER`
- `1001`

Switch worker processes to run as this user.

A valid user id (as an integer) or the name of a user that can be retrieved with a call to pwd.getpwnam(value) or None to not change the worker process user.

group

- `-g GROUP, --group GROUP`
- `1001`

Switch worker process to run as this group.

A valid group id (as an integer) or the name of a user that can be retrieved with a call to pwd.getgrnam(value) or None to not change the worker processes group.

umask

- `-m INT, --umask INT`
- `0`

A bit mask for the file mode on files written by Gunicorn.

Note that this affects unix socket permissions.

A valid value for the os.umask(mode) call or a string compatible with int(value, 0) (0 means Python guesses the base, so values like “0”, “0xFF”, “0022” are valid for decimal, hex, and octal representations)

tmp_upload_dir

- `None`

Directory to store temporary request data as they are read.

This may disappear in the near future.

This path should be writable by the process permissions set for Gunicorn workers. If not specified, Gunicorn will choose a system generated temporary directory.

secure_scheme_headers

- `{'X-FORWARDED-PROTOCOL': 'ssl', 'X-FORWARDED-PROTO': 'https', 'X-FORWARDED-SSL': 'on'}`

A dictionary containing headers and values that the front-end proxy uses to indicate HTTPS requests. These tell gunicorn to set wsgi.url_scheme to “https”, so your application can tell that the request is secure.

The dictionary should map upper-case header names to exact string values. The value comparisons are case-sensitive, unlike the header names, so make sure they’re exactly what your front-end proxy sends when handling HTTPS requests.

It is important that your front-end proxy configuration ensures that the headers defined here can not be passed directly from the client.

x_forwarded_for_header

- `X-FORWARDED-FOR`

Set the X-Forwarded-For header that identify the originating IP address of the client connection to gunicorn via a proxy.

forwarded_allow_ips

- `127.0.0.1`

Front-end's IPs from which allowed to handle X-Forwarded-* headers. (comma separate).

Set to "*" to disable checking of Front-end IPs (useful for setups where you don't know in advance the IP address of Front-end, but you still trust the environment)

Logging

accesslog

- `--access-logfile FILE`
- `None`

The Access log file to write to.

"-" means log to stderr.

access_log_format

- `--access-logformat STRING`
- `"%(h)s %(l)s %(u)s %(t)s "%(r)s" %(s)s %(b)s "%(f)s" "%(a)s"`

The Access log format .

By default:

%(h)s %(l)s %(u)s %(t)s "%(r)s" %(s)s %(b)s "%(f)s" "%(a)s"

h: remote address l: '-' u: currently '-', may be user name in future releases t: date of the request r: status line (ex: GET / HTTP/1.1) s: status b: response length or '-' f: referer a: user agent T: request time in seconds D: request time in microseconds, p: process ID {Header}i: request header {Header}o: response header

errorlog

- `--error-logfile FILE, --log-file FILE`
- `-`

The Error log file to write to.

"-" means log to stderr.

loglevel

- `--log-level LEVEL`
- `info`

The granularity of Error log outputs.

Valid level names are:

- debug

- info
- warning
- error
- critical

logger_class

- `--logger-class STRING`
- `simple`

The logger you want to use to log events in gunicorn.

The default class (`gunicorn.glogging.Logger`) handle most of normal usages in logging. It provides error and access logging.

You can provide your own worker by giving gunicorn a python path to a subclass like gunicorn.glogging.Logger. Alternatively the syntax can also load the Logger class with *egg:gunicorn#simple*

logconfig

- `--log-config FILE`
- `None`

The log config file to use. Gunicorn uses the standard Python logging module's Configuration file format.

Process Naming

proc_name

- `-n STRING, --name STRING`
- `None`

A base to use with setproctitle for process naming.

This affects things like `ps` and `top`. If you're going to be running more than one instance of Gunicorn you'll probably want to set a name to tell them apart. This requires that you install the setproctitle module.

It defaults to 'gunicorn'.

default_proc_name

- `gunicorn`

Internal setting that is adjusted for each type of application.

Django

django_settings

- `--settings STRING`

- None

The Python path to a Django settings module.

e.g. 'myproject.settings.main'. If this isn't provided, the DJANGO_SETTINGS_MODULE environment variable will be used.

Server Mechanics

pythonpath

- --pythonpath STRING
- None

A directory to add to the Python path for Django.

e.g. '/home/djangoprojects/myproject'.

Server Hooks

on_starting

- ```
  def on_starting(server):
      pass
  ```

Called just before the master process is initialized.

The callable needs to accept a single instance variable for the Arbiter.

on_reload

- ```
  def on_reload(server):
      pass
  ```

Called to recycle workers during a reload via SIGHUP.

The callable needs to accept a single instance variable for the Arbiter.

when_ready

- ```
  def when_ready(server):
      pass
  ```

Called just after the server is started.

The callable needs to accept a single instance variable for the Arbiter.

pre_fork

- ```
  def pre_fork(server, worker):
      pass
  ```

Called just before a worker is forked.

The callable needs to accept two instance variables for the Arbiter and new Worker.

post_fork

- ```
  def post_fork(server, worker):
      pass
  ```

Called just after a worker has been forked.

The callable needs to accept two instance variables for the Arbiter and new Worker.

pre_exec

- ```
  def pre_exec(server):
      pass
  ```

Called just before a new master process is forked.

The callable needs to accept a single instance variable for the Arbiter.

pre_request

- ```
  def pre_request(worker, req):
      worker.log.debug("%s %s" % (req.method, req.path))
  ```

Called just before a worker processes the request.

The callable needs to accept two instance variables for the Worker and the Request.

post_request

- ```
  def post_request(worker, req, environ, resp):
      pass
  ```

Called after a worker processes the request.

The callable needs to accept two instance variables for the Worker and the Request.

worker_exit

- ```
  def worker_exit(server, worker):
      pass
  ```

Called just after a worker has been exited.

The callable needs to accept two instance variables for the Arbiter and the just-exited Worker.

nworkers_changed
  ```

- ```
  def nworkers_changed(server, new_value, old_value):
      pass
  ```

Called just after num_workers has been changed.

The callable needs to accept an instance variable of the Arbiter and two integers of number of workers after and before change.

If the number of workers is set for the first time, old_value would be None.

Server Mechanics

proxy_protocol

- `--proxy-protocol`
- `False`

Enable detect PROXY protocol (PROXY mode).

Allow using Http and Proxy together. It's may be useful for work with stunnel as https frondend and gunicorn as http server.

PROXY protocol: http://haproxy.1wt.eu/download/1.5/doc/proxy-protocol.txt

Example for stunnel config:

```
[https]
```

protocol = proxy accept = 443 connect = 80 cert = /etc/ssl/certs/stunnel.pem key = /etc/ssl/certs/stunnel.key

proxy_allow_ips

- `--proxy-allow-from`
- `127.0.0.1`

Front-end's IPs from which allowed accept proxy requests (comma separate).

Ssl

keyfile

- `--keyfile FILE`
- `None`

SSL key file

certfile

- `--certfile FILE`
- `None`

SSL certificate file
```

Logging

syslog_addr

- `--log-syslog-to SYSLOG_ADDR`
- `udp://localhost:514`

Address to send syslog messages

syslog

- `--log-syslog`
- `False`

Log to syslog.

syslog_prefix

- `--log-syslog-prefix SYSLOG_PREFIX`
- `None`

makes gunicorn use the parameter as program-name in the syslog entries.

All entries will be prefixed by gunicorn.<prefix>. By default the program name is the name of the process.

syslog_facility

- `--log-syslog-facility SYSLOG_FACILITY`
- `user`

Syslog facility name

2.4 Deploying Gunicorn

We strongly recommend to use Gunicorn behind a proxy server.

2.4.1 Nginx Configuration

Although there are many HTTP proxies available, we strongly advise that you use Nginx. If you choose another proxy server you need to make sure that it buffers slow clients when you use default Gunicorn workers. Without this buffering Gunicorn will be easily susceptible to denial-of-service attacks. You can use slowloris to check if your proxy is behaving properly.

An example configuration file for fast clients with Nginx:

```
worker_processes 1;

user nobody nogroup;
pid /tmp/nginx.pid;
error_log /tmp/nginx.error.log;
```

```
events {
    worker_connections 1024;
    accept_mutex off;
}

http {
    include mime.types;
    default_type application/octet-stream;
    access_log /tmp/nginx.access.log combined;
    sendfile on;

    upstream app_server {
        server unix:/tmp/gunicorn.sock fail_timeout=0;
        # For a TCP configuration:
        # server 192.168.0.7:8000 fail_timeout=0;
    }

    server {
        listen 80 default;
        client_max_body_size 4G;
        server_name _;

        keepalive_timeout 5;

        # path for static files
        root /path/to/app/current/public;

        location / {
            # checks for static file, if not found proxy to app
            try_files $uri @proxy_to_app;
        }

        location @proxy_to_app {
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header Host $http_host;
            proxy_redirect off;

            proxy_pass   http://app_server;
        }

        error_page 500 502 503 504 /500.html;
        location = /500.html {
            root /path/to/app/current/public;
        }
    }
}
```

If you want to be able to handle streaming request/responses or other fancy features like Comet, Long polling, or Web sockets, you need to turn off the proxy buffering. **When you do this** you must run with one of the async worker classes.

To turn off buffering, you only need to add `proxy_buffering off;` to your `location` block:

```
...
location / {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;
    proxy_buffering off;
```

```
    if (!-f $request_filename) {
        proxy_pass http://app_server;
        break;
    }
}
...
```

2.4.2 Using Virtualenv

To serve an app from a Virtualenv it is generally easiest to just install Gunicorn directly into the Virtualenv. This will create a set of Gunicorn scripts for that Virtualenv which can be used to run applications normally.

If you have Virtualenv installed, you should be able to do something like this:

```
$ mkdir ~/venvs/
$ virtualenv ~/venvs/webapp
$ source ~/venvs/webapp/bin/activate
$ ~/venvs/webapp/bin/easy_install -U gunicorn
$ deactivate
```

Then you just need to use one of the three Gunicorn scripts that was installed into `~/venvs/webapp/bin`.

Note: You can force the installation of Gunicorn in your Virtualenv by passing `-I` or `--ignore-installed` option to pip:

```
$ source ~/venvs/webapp/bin/activate
$ pip install -I gunicorn
```

2.4.3 Monitoring

Note: Make sure that when using either of these service monitors you do not enable the Gunicorn's daemon mode. These monitors expect that the process they launch will be the process they need to monior. Daemonizing will fork-exec which creates an unmonitored process and generally just confuses the monitor services.

Gaffer

Using Gafferd and gafferctl

Gaffer can be used to monitor gunicorn. A simple configuration is:

```
[process:gunicorn]
cmd = gunicorn -w 3 test:app
cwd = /path/to/project
```

Then you can easily manage Gunicorn using gafferctl.

Using a Procfile

Create a `Procfile` in your project:

```
gunicorn = gunicorn -w 3 test:app
```

You can any other applications that should be launched at the same time.

Then you can start your gunicorn application using gafferp.:

```
gafferp start
```

If gafferd is launched you can also load your Procfile in it directly:

```
gafferp load
```

All your applications will be then supervised by gafferd.

Runit

A popular method for deploying Gunicorn is to have it monitored by runit. Here is an example service definition:

```
#!/bin/sh

GUNICORN=/usr/local/bin/gunicorn
ROOT=/path/to/project
PID=/var/run/gunicorn.pid

APP=main:application

if [ -f $PID ]; then rm $PID; fi

cd $ROOT
exec $GUNICORN -c $ROOT/gunicorn.conf.py --pid=$PID $APP
```

Save this as `/etc/sv/[app_name]/run`, and make it executable (`chmod u+x /etc/sv/[app_name]/run`). Then run `ln -s /etc/sv/[app_name] /etc/service/[app_name]`. If runit is installed, gunicorn should start running automatically as soon as you create the symlink.

If it doesn't start automatically, run the script directly to troubleshoot.

Supervisor

Another useful tool to monitor and control Gunicorn is Supervisor. A simple configuration is:

```
[program:gunicorn]
command=/path/to/gunicorn main:application -c /path/to/gunicorn.conf.py
directory=/path/to/project
user=nobody
autostart=true
autorestart=true
redirect_stderr=True
```

2.4.4 Logging

Logging can be configured by using various flags detailed in the configuration documentation or by creating a logging configuration file. Send the `USR1` signal to rotate logs if you are using the logrotate utility:

```
kill -USR1 $(cat /var/run/gunicorn.pid)
```

2.5 Design

A brief description of an architecture of Gunicorn.

2.5.1 Server Model

Gunicorn is based on the pre-fork worker model. This means that there is a central master process that manages a set of worker processes. The master never knows anything about individual clients. All requests and responses are handled completely by worker processes.

Master

The master process is a simple loop that listens for various process signals and reacts accordingly. It manages the list of running workers by listening for signals like TTIN, TTOU, and CHLD. TTIN and TTOU tell the master to increase or decrease the number of running workers. CHLD indicates that a child process has terminated, in this case the master process automatically restarts the failed worker.

Sync Workers

The most basic and the default worker type is a synchronous worker class that handles a single request at a time. This model is the simplest to reason about as any errors will affect at most a single request. Though as we describe below only processing a single request at a time requires some assumptions about how applications are programmed.

Async Workers

The asynchronous workers available are based on Greenlets (via Eventlet and Gevent). Greenlets are an implementation of cooperative multi-threading for Python. In general, an application should be able to make use of these worker classes with no changes.

Tornado Workers

There's also a Tornado worker class. It can be used to write applications using the Tornado framework. Although the Tornado workers are capable of serving a WSGI application, this is not a recommended configuration.

2.5.2 Choosing a Worker Type

The default synchronous workers assume that your application is resource bound in terms of CPU and network bandwidth. Generally this means that your application shouldn't do anything that takes an undefined amount of time. For instance, a request to the internet meets this criteria. At some point the external network will fail in such a way that clients will pile up on your servers.

This resource bound assumption is why we require a buffering proxy in front of a default configuration Gunicorn. If you exposed synchronous workers to the internet, a DOS attack would be trivial by creating a load that trickles data to the servers. For the curious, Slowloris is an example of this type of load.

Some examples of behavior requiring asynchronous workers:

- Applications making long blocking calls (Ie, external web services)
- Serving requests directly to the internet

- Streaming requests and responses
- Long polling
- Web sockets
- Comet

2.5.3 How Many Workers?

DO NOT scale the number of workers to the number of clients you expect to have. Gunicorn should only need 4-12 worker processes to handle hundreds or thousands of requests per second.

Gunicorn relies on the operating system to provide all of the load balancing when handling requests. Generally we recommend `(2 x $num_cores) + 1` as the number of workers to start off with. While not overly scientific, the formula is based on the assumption that for a given core, one worker will be reading or writing from the socket while the other worker is processing a request.

Obviously, your particular hardware and application are going to affect the optimal number of workers. Our recommendation is to start with the above guess and tune using TTIN and TTOU signals while the application is under load.

Always remember, there is such a thing as too many workers. After a point your worker processes will start thrashing system resources decreasing the throughput of the entire system.

2.6 Community

Use these channels to communicate about the project.

2.6.1 Mailing list

The **user** mailing list is general discussion and support list for Gunicorn users.

- To **subscribe**, send an email to user+subscribe@gunicorn.org
- To **unsubscribe**, send an email to user+unsubscribe@gunicorn.org
- Finally, to post a message to the list use the address to user@gunicorn.org

The archive for this list can also be browsed online .

2.6.2 Irc

The Gunicorn channel is on the Freenode IRC network. You can chat with other on #gunicorn channel.

2.6.3 Issue Tracking

Bug reports, enhancement requests and tasks generally go in the Github issue tracker.

2.7 FAQ

2.7.1 WSGI Bits

How do I set SCRIPT_NAME?

By default `SCRIPT_NAME` is an empy string. The value could be set by setting `SCRIPT_NAME` in the environment or as an HTTP header.

2.7.2 Server Stuff

How do I reload my application in Gunicorn?

You can gracefully reload by sending HUP signal to gunicorn:

```
$ kill -HUP masterpid
```

How might I test a proxy configuration?

The Slowloris script is a great way to test that your proxy is correctly buffering responses for the synchronous workers.

How can I name processes?

If you install the Python package setproctitle Gunicorn will set the process names to something a bit more meaningful. This will affect the output you see in tools like `ps` and `top`. This helps for distinguishing the master process as well as between masters when running more than one app on a single machine. See the proc_name setting for more information.

Gunicorn fails to start with upstart

Make sure you run gunicorn with `--daemon` option.

Why is there no HTTP Keep-Alive?

The default Sync workers are designed to run behind Nginx which only uses HTTP/1.0 with its upstream servers. If you want to deploy Gunicorn to handle unbuffered requests (ie, serving requests directly from the internet) you should use one of the async workers.

2.7.3 Worker Processes

How do I know which type of worker to use?

Read the *Design* page for help on the various worker types.

What types of workers are there?

Check out the configuration docs for worker_class

How can I figure out the best number of worker processes?

Here is our recommendation for tuning the number of workers.

How can I change the number of workers dynamically?

TTIN and TTOU signals can be sent to the master to increase or decrease the number of workers.

To increase the worker count by one:

```
$ kill -TTIN $masterpid
```

To decrease the worker count by one:

```
$ kill -TTOU $masterpid
```

2.7.4 Kernel Parameters

When dealing with large numbers of concurrent connections there are a handful of kernel parameters that you might need to adjust. Generally these should only affect sites with a very large concurrent load. These parameters are not specific to Gunicorn, they would apply to any sort of network server you may be running.

These commands are for Linux. Your particular OS may have slightly different parameters.

How can I increase the maximum number of file descriptors?

One of the first settings that usually needs to be bumped is the maximum number of open file descriptors for a given process. For the confused out there, remember that Unices treat sockets as files.

```
$ sudo ulimit -n 2048
```

How can I increase the maximum socket backlog?

Listening sockets have an associated queue of incoming connections that are waiting to be accepted. If you happen to have a stampede of clients that fill up this queue new connections will eventually start getting dropped.

```
$ sudo sysctl -w net.core.somaxconn="2048"
```

2.8 Changelog

2.8.1 0.17.2 / 2013-01-07

- optimize readline
- make imports errors more visiblle when loading an app or a logging class
- fix tornado worker: don't pass ssl options if there are none
- fix PEP3333: accept only bytetrings in the response body
- fix support on CYGWIN platforms

2.8.2 0.17.1 / 2013-01-05

- add syslog facility name setting
- fix `--version` command line argument
- fix wsgi url_scheme for https

2.8.3 0.17.0 / 2012-12-25

- allows gunicorn to bind to multiple address
- add SSL support
- add syslog support
- add nworkers_changed hook
- add response arg for post_request hook
- parse command line with argparse (replace deprecated optparse)
- fix PWD detection in arbiter
- miscellenaeous PEP8 fixes

2.8.4 0.16.1 / 2012-11-19

- Fix packaging

2.8.5 0.16.0 / 2012-11-19

- **Added support for Python 3.2 & 3.3**
- Expose –pythonpath command to all gunicorn commands
- Honor $PORT environment variable, useful for deployement on heroku
- Removed support for Python 2.5
- Make sure we reopen the logs on the console
- Fix django settings module detection from path
- Reverted timeout for client socket. Fix issue on blocking issues.
- Fixed gevent worker

2.8.6 0.15.0 / 2012-10-18

- new documentation site on http://docs.gunicorn.org
- new website on http://gunicorn.org
- add haproxy PROXY protocol support
- add ForwardedAllowIPS option: allows to filter Front-end's IPs allowed to handle X-Forwarded-* headers.
- add callable hooks for paster config
- add x-forwarded-proto as secure scheme default (Heroku is using this)

- allows gunicorn to load a pre-compiled application
- support file reopening & reexec for all loggers
- initialize the logging config file with defaults.
- set timeout for client socket (slow client DoS).
- NoMoreData, ChunkMissingTerminator, InvalidChunkSize are now IOError exceptions
- fix graceful shutdown in gevent
- fix limit request line check

2.8.7 0.14.6 / 2012-07-26

- fix gevent & subproces
- fix request line length check
- fix keepalive = 0
- fix tornado worker

2.8.8 0.14.5 / 2012-06-24

- fix logging during daemonisation

2.8.9 0.14.4 / 2012-06-24

- new –graceful-timeout option
- fix multiple issues with request limit
- more fixes in django settings resolutions
- fix gevent.core import
- fix keepalive=0 in eventlet worker
- fix handle_error display with the unix worker
- fix tornado.wsgi.WSGIApplication calling error
- **breaking change**: take the control on graceful reload back. graceful can't be overrided anymore using the on_reload function.

2.8.10 0.14.3 / 2012-05-15

- improvement: performance of http.body.Body.readline()
- improvement: log HTTP errors in access log like Apache
- improvment: display traceback when the worker fails to boot
- improvement: makes gunicorn work with gevent 1.0
- examples: websocket example now supports hybi13
- fix: reopen log files after initialization

- fix: websockets support
- fix: django1.4 support
- fix: only load the paster application 1 time

2.8.11 0.14.2 / 2012-03-16

- add validate_class validator: allows to use a class or a method to initialize the app during in-code configuration
- add support for max_requests in tornado worker
- add support for disabling x_forwarded_for_header in tornado worker
- gevent_wsgi is now an alias of gevent_pywsgi
- Fix gevent_pywsgi worker

2.8.12 0.14.1 / 2012-03-02

- fixing source archive, reducing its size

2.8.13 0.14.0 / 2012-02-27

- check if Request line is too large: You can now pass the parameter `--limit-request-line` or set the `limit_request_line` in your configuration file to set the max size of the request line in bytes.
- limit the number of headers fields and their size. Add `--limit-request-field` and `limit-request-field-size` settings
- add `p` variable to the log access format to log pidfile
- add `{HeaderName}o` variable to the logo access format to log the response header HeaderName
- request header is now logged with the variable `{HeaderName}i` in the access log file
- improve error logging
- support logging.configFile
- support django 1.4 in both gunicorn_django & run_gunicorn command
- improve reload in django run_gunicorn command (should just work now)
- allows people to set the `X-Forwarded-For` header key and disable it by setting an empty string.
- fix support of Tornado
- many other fixes.

2.9 History

2.9.1 Changelog - 2013

0.17.2 / 2013-01-07

- optimize readline

- make imports errors more visiblle when loading an app or a logging class
- fix tornado worker: don't pass ssl options if there are none
- fix PEP3333: accept only bytetrings in the response body
- fix support on CYGWIN platforms

0.17.1 / 2013-01-05

- add syslog facility name setting
- fix `--version` command line argument
- fix wsgi url_scheme for https

2.9.2 Changelog - 2012

0.17.0 / 2012-12-25

- allows gunicorn to bind to multiple address
- add SSL support
- add syslog support
- add nworkers_changed hook
- add response arg for post_request hook
- parse command line with argparse (replace deprecated optparse)
- fix PWD detection in arbiter
- miscellenaeous PEP8 fixes

0.16.1 / 2012-11-19

- Fix packaging

0.16.0 / 2012-11-19

- **Added support for Python 3.2 & 3.3**
- Expose –pythonpath command to all gunicorn commands
- Honor $PORT environment variable, useful for deployement on heroku
- Removed support for Python 2.5
- Make sure we reopen the logs on the console
- Fix django settings module detection from path
- Reverted timeout for client socket. Fix issue on blocking issues.
- Fixed gevent worker

0.15.0 / 2012-10-18

- new documentation site on http://docs.gunicorn.org
- new website on http://gunicorn.org
- add haproxy PROXY protocol support
- add ForwardedAllowIPS option: allows to filter Front-end's IPs allowed to handle X-Forwarded-* headers.
- add callable hooks for paster config
- add x-forwarded-proto as secure scheme default (Heroku is using this)
- allows gunicorn to load a pre-compiled application
- support file reopening & reexec for all loggers
- initialize the logging config file with defaults.
- set timeout for client socket (slow client DoS).
- NoMoreData, ChunkMissingTerminator, InvalidChunkSize are now IOError exceptions
- fix graceful shutdown in gevent
- fix limit request line check

0.14.6 / 2012-07-26

- fix gevent & subproces
- fix request line length check
- fix keepalive = 0
- fix tornado worker

0.14.5 / 2012-06-24

- fix logging during daemonisation

0.14.4 / 2012-06-24

- new –graceful-timeout option
- fix multiple issues with request limit
- more fixes in django settings resolutions
- fix gevent.core import
- fix keepalive=0 in eventlet worker
- fix handle_error display with the unix worker
- fix tornado.wsgi.WSGIApplication calling error
- **breaking change**: take the control on graceful reload back. graceful can't be overrided anymore using the on_reload function.

0.14.3 / 2012-05-15

- improvement: performance of http.body.Body.readline()
- improvement: log HTTP errors in access log like Apache
- improvment: display traceback when the worker fails to boot
- improvement: makes gunicorn work with gevent 1.0
- examples: websocket example now supports hybi13
- fix: reopen log files after initialization
- fix: websockets support
- fix: django1.4 support
- fix: only load the paster application 1 time

0.14.2 / 2012-03-16

- add validate_class validator: allows to use a class or a method to initialize the app during in-code configuration
- add support for max_requests in tornado worker
- add support for disabling x_forwarded_for_header in tornado worker
- gevent_wsgi is now an alias of gevent_pywsgi
- Fix gevent_pywsgi worker

0.14.1 / 2012-03-02

- fixing source archive, reducing its size

0.14.0 / 2012-02-27

- check if Request line is too large: You can now pass the parameter `--limit-request-line` or set the `limit_request_line` in your configuration file to set the max size of the request line in bytes.
- limit the number of headers fields and their size. Add `--limit-request-field` and `limit-request-field-size` settings
- add `p` variable to the log access format to log pidfile
- add `{HeaderName}o` variable to the logo access format to log the response header HeaderName
- request header is now logged with the variable `{HeaderName}i` in the access log file
- improve error logging
- support logging.configFile
- support django 1.4 in both gunicorn_django & run_gunicorn command
- improve reload in django run_gunicorn command (should just work now)
- allows people to set the `X-Forwarded-For` header key and disable it by setting an empty string.
- fix support of Tornado
- many other fixes.

2.9.3 Changelog - 2011

0.13.4 / 2011-09-23

- fix util.closerange function used to prevent leaking fds on python 2.5 (typo)

0.13.3 / 2011-09-19

- refactor gevent worker
- prevent leaking fds on reexec
- fix inverted request_time computation

0.13.2 / 2011-09-17

- Add support for Tornado 2.0 in tornado worker
- Improve access logs: allows customisation of the log format & add request time
- Logger module is now pluggable
- Improve graceful shutdown in Python versions >= 2.6
- Fix post_request root arity for compatibility
- Fix sendfile support
- Fix Django reloading

0.13.1 / 2011-08-22

- Fix unix socket. log argument was missing.

0.13.0 / 2011-08-22

- Improve logging: allows file-reopening and add access log file compatible with the apache combined log format
- Add the possibility to set custom SSL headers. X-Forwarded-Protocol and X-Forwarded-SSL are still the default
- New *on_reload* hook to customize how gunicorn spawn new workers on SIGHUP
- Handle projects with relative path in django_gunicorn command
- Preserve path parameters in PATH_INFO
- post_request hook now accepts the environ as argument.
- When stopping the arbiter, close the listener asap.
- Fix Django command *run_gunicorn* in settings reloading
- Fix Tornado worker exiting
- Fix the use of sendfile in wsgi.file_wrapper

0.12.2 / 2011-05-18

- Add wsgi.file_wrapper optimised for FreeBSD, Linux & MacOSX (use sendfile if available)
- Fix django run_gunicorn command. Make sure we reload the application code.
- Fix django localisation
- Compatible with gevent 0.14dev

0.12.1 / 2011-03-23

- Add "on_starting" hook. This hook can be used to set anything before the arbiter really start
- Support bdist_rpm in setup
- Improve content-length handling (pep 3333)
- Improve Django support
- Fix daemonizing (#142)
- Fix ipv6 handling

2.9.4 Changelog - 2010

0.12.0 / 2010-12-22

- Add support for logging configuration using a ini file. It uses the standard Python logging's module Configuration file format and allows anyone to use his custom file handler
- Add IPV6 support
- Add multidomain application example
- Improve gunicorn_django command when importing settings module using DJANGO_SETTINGS_MODULE environment variable
- Send appropriate error status on http parsing
- Fix pidfile, set permissions so other user can read it and use it.
- Fix temporary file leaking
- Fix setpgrp issue, can now be launched via ubuntu upstart
- Set the number of workers to zero on WINCH

0.11.2 / 2010-10-30

- Add SERVER_SOFTWARE to the os.environ
- Add support for django settings environement variable
- Add support for logging configuration in Paster ini-files
- Improve arbiter notification in asynchronous workers
- Display the right error when a worker can't be used
- Fix Django support

- Fix HUP with Paster applications
- Fix readline in wsgi.input

0.11.1 / 2010-09-02

- Implement max-requests feature to prevent memory leaks.
- Added 'worker_exit' server hook.
- Reseed the random number generator after fork().
- Improve Eventlet worker.
- Fix Django command *run_gunicorn*.
- Fix the default proc name internal setting.
- Workaround to prevent Gevent worker to segfault on MacOSX.

0.11.0 / 2010-08-12

- Improve dramatically performances of Gevent and Eventlet workers
- Optimize HTTP parsing
- Drop Server and Date headers in start_response when provided.
- Fix latency issue in async workers

0.10.1 / 2010-08-06

- Improve gevent's workers. Add "egg:gunicorn#gevent_wsgi" worker using gevent.wsgi and "egg:gunicorn#gevent_pywsgi" worker using gevent.pywsgi . **"egg:gunicorn#gevent"** using our own HTTP parser is still here and is **recommended** for normal uses. Use the "gevent.wsgi" parser if you need really fast connections and don't need streaming, keepalive or ssl.
- Add pre/post request hooks
- Exit more quietly
- Fix gevent dns issue

0.10.0 / 2010-07-08

- New HTTP parser.
- New HUP behaviour. Re-reads the configuration and then reloads all worker processes without changing the master process id. Helpful for code reloading and monitoring applications like supervisord and runit.
- Added a preload configuration parameter. By default, application code is now loaded after a worker forks. This couple with the new HUP handling can be used for dev servers to do hot code reloading. Using the preload flag can help a bit in small memory VM's.
- Allow people to pass command line arguments to WSGI applications. See: examples/alt_spec.py
- Added an example gevent reloader configuration: examples/example_gevent_reloader.py.
- New gevent worker "egg:gunicorn#gevent2", working with gevent.wsgi.
- Internal refactoring and various bug fixes.

- New documentation website.

0.9.1 / 2010-05-26

- Support https via X-Forwarded-Protocol or X-Forwarded-Ssl headers
- Fix configuration
- Remove -d options which was used instead of -D for daemon.
- Fix umask in unix socket

0.9.0 / 2010-05-24

- Added *when_ready* hook. Called just after the server is started
- Added *preload* setting. Load application code before the worker processes are forked.
- Refactored Config
- Fix pidfile
- Fix QUIT/HUP in async workers
- Fix reexec
- Documentation improvements

0.8.1 / 2010-04-29

- Fix builtins import in config
- Fix installation with pip
- Fix Tornado WSGI support
- Delay application loading until after processing all configuration

0.8.0 / 2010-04-22

- Refactored Worker management for better async support. Now use the -k option to set the type of request processing to use
- Added support for Tornado

0.7.2 / 2010-04-15

- Added –spew option to help debugging (installs a system trace hook)
- Some fixes in async arbiters
- Fix a bug in start_response on error

0.7.1 / 2010-04-01

- Fix bug when responses have no body.

0.7.0 / 2010-03-26

- Added support for Eventlet and Gevent based workers.
- Added Websockets support
- Fix Chunked Encoding
- Fix SIGWINCH on OpenBSD
- Fix PEP 333 compliance for the write callable.

0.6.5 / 2010-03-11

- Fix pidfile handling
- Fix Exception Error

0.6.4 / 2010-03-08

- Use cStringIO for performance when possible.
- Fix worker freeze when a remote connection closes unexpectedly.

0.6.3 / 2010-03-07

- Make HTTP parsing faster.
- Various bug fixes

0.6.2 / 2010-03-01

- Added support for chunked response.
- Added proc_name option to the config file.
- Improved the HTTP parser. It now uses buffers instead of strings to store temporary data.
- Improved performance when sending responses.
- Workers are now murdered by age (the oldest is killed first).

0.6.1 / 2010-02-24

- Added gunicorn config file support for Django admin command
- Fix gunicorn config file. -c was broken.
- Removed TTIN/TTOU from workers which blocked other signals.

0.6.0 / 2010-02-22

- Added setproctitle support
- Change privilege switch behavior. We now work like NGINX, master keeps the permissions, new uid/gid permissions are only set for workers.

0.5.1 / 2010-02-22

- Fix umask
- Added Debian packaging

0.5.0 / 2010-02-20

- Added configuration file handler.
- Added support for pre/post fork hooks
- Added support for before_exec hook
- Added support for unix sockets
- Added launch of workers processes under different user/group
- Added umask option
- Added SCRIPT_NAME support
- Better support of some exotic settings for Django projects
- Better support of Paste-compatible applications
- Some refactoring to make the code easier to hack
- Allow multiple keys in request and response headers