Standalone WSGI Containers

There are popular servers written in Python that contain WSGI applications and serve HTTP. These servers stand alone when they run; you can proxy to them from your web server. Note the section on Proxy Setups if you run into issues.

Gunicorn

Gunicorn 'Green Unicorn' is a WSGI HTTP Server for UNIX. It's a pre-fork worker model ported from Ruby's Unicorn project. It supports both eventlet and greenlet. Running a Flask application on this server is quite simple:

```
$ gunicorn myproject:app
```

Gunicorn provides many command-line options – see `gunicorn -h`. For example, to run a Flask application with 4 worker processes (`-w 4`) binding to localhost port 4000 (`-b 127.0.0.1:4000`):

```
$ gunicorn -w 4 -b 127.0.0.1:4000 myproject:app
```

The `gunicorn` command expects the names of your application module or package and the application instance within the module. If you use the application factory pattern, you can pass a call to that:

```
$ gunicorn "myproject:create_app()"
```

uWSGI

uWSGI is a fast application server written in C. It is very configurable which makes it more complicated to setup than gunicorn.

Running uWSGI HTTP Router:

```
$ uwsgi --http 127.0.0.1:5000 --module myproject:app
```

For a more optimized setup, see configuring uWSGI and NGINX.

Gevent

Gevent is a coroutine-based Python networking library that uses greenlet to provide a high-level synchronous API on top of libev event loop:

```
from gevent.pywsgi import WSGIServer
from yourapplication import app

http_server = WSGIServer(('', 5000), app)
http_server.serve_forever()
```

Twisted Web

Twisted Web is the web server shipped with Twisted, a mature, non-blocking event-driven networking library. Twisted Web comes with a standard WSGI container which can be controlled from the command line using the `twistd` utility:

```
$ twistd web --wsgi myproject.app
```

This example will run a Flask application called `app` from a module named `myproject`.

Twisted Web supports many flags and options, and the `twistd` utility does as well; see `twistd -h` and `twistd web -h` for more information. For example, to run a Twisted Web server in the foreground, on port 8080, with an application from `myproject`:

```
$ twistd -n web --port tcp:8080 --wsgi myproject.app
```

Proxy Setups

If you deploy your application using one of these servers behind an HTTP proxy you will need to rewrite a few headers in order for the application to work. The two problematic values in the WSGI environment usually are `REMOTE_ADDR` and `HTTP_HOST`. You can configure your httpd to pass these headers, or you can fix them in middleware. Werkzeug ships a fixer that will solve some common setups, but you might want to write your own WSGI middleware for specific setups.

Here's a simple nginx configuration which proxies to an application served on localhost at port 8000, setting appropriate headers:

```
server {
    listen 80;

    server_name _;
```

```
    access_log  /var/log/nginx/access.log;
    error_log  /var/log/nginx/error.log;

    location / {
        proxy_pass         http://127.0.0.1:8000/;
        proxy_redirect     off;

        proxy_set_header   Host                 $host;
        proxy_set_header   X-Real-IP            $remote_addr;
        proxy_set_header   X-Forwarded-For      $proxy_add_x_forwarded_
        proxy_set_header   X-Forwarded-Proto    $scheme;
    }
}
```

If your httpd is not providing these headers, the most common setup invokes the host being set from `X-Forwarded-Host` and the remote address from `X-Forwarded-For`:

```
from werkzeug.middleware.proxy_fix import ProxyFix
app.wsgi_app = ProxyFix(app.wsgi_app, x_proto=1, x_host=1)
```

Trusting Headers:

Please keep in mind that it is a security issue to use such a middleware in a non-proxy setup because it will blindly trust the incoming headers which might be forged by malicious clients.

If you want to rewrite the headers from another header, you might want to use a fixer like this:

```
class CustomProxyFix(object):

    def __init__(self, app):
        self.app = app

    def __call__(self, environ, start_response):
        host = environ.get('HTTP_X_FHOST', '')
        if host:
            environ['HTTP_HOST'] = host
        return self.app(environ, start_response)

app.wsgi_app = CustomProxyFix(app.wsgi_app)
```