

Как работает HTTP-сервер nginx

Дмитрий Васильев

Уже долгое время Apache является наиболее популярным HTTP-сервером с открытым исходным кодом. Но он может быть слишком громоздким для некоторых задач. И здесь на сцену выходят новые игроки – легковесные, асинхронные HTTP-серверы, одним из которых является nginx.

Вот уже более восьми лет я использую HTTP-сервер Apache, и в большинстве случаев он меня во всем устраивает. Но недавно, установив его на один из небольших серверов с Ubuntu Linux, я подумал, что Apache не очень целесообразно использует ресурсы сервера. И с этой мыслью решил попробовать какой-нибудь альтернативный вариант. Я уже давно слышал о высокопроизводительных асинхронных серверах Lighttpd (произносится lighty, <http://www.lighttpd.net>) и nginx (произносится engine x, <http://sysoev.ru/nginx>), но до этого момента не использовал их на практике. Надо также заметить, что по статистике Netcraft на октябрь 2008 года Lighttpd и nginx занимают, в рейтинге соответственно, 5-е и 6-е места: <http://survey.netcraft.com/Reports/200810>.

Теперь, когда у меня было два варианта, оставалось самое сложное – выбрать наиболее подходящий для моего случая. Это сложный выбор, т.к. оба имеют практически одинаковые возможности (хотя nginx также может использоваться как почтовый прокси-сервер, я не учитывал это в сравнении). Оба используются на высоко посещаемых сайтах, например, YouTube использует Lighttpd, а Rambler – nginx. Оба имеют достаточно хорошую документацию, в том числе nginx имеет до-

кументацию на русском. В итоге для себя я остановился на nginx по следующим причинам:

- Многие источники (например, <http://hostingfu.com/article/nginx-vs-lighttpd-for-a-small-vps>) отмечают, что Lighttpd имеет проблемы со стабильностью и утечки памяти. Даже если эти проблемы остались в прошлом, похоже, что автор не уделял им достаточно внимания.
- Nginx, в отличие от Lighttpd, имеет две официальные ветки разработки (стабильную и экспериментальную), что, на мой взгляд, говорит о большей заботе автора о конечных пользователях и качестве проекта в целом.
- Мне хотелось иметь возможность использовать сервер с языком Python (<http://www.python.org>) с помощью стандартного интерфейса WSGI (<http://ru.wikipedia.org/wiki/WSGI>). При этом для nginx существует дополнительный модуль `mod_wsgi`, а в Lighttpd это можно делать только через шлюз FastCGI > WSGI.

Принципы работы

Прежде чем обратиться к рассмотрению непосредственно сервера nginx, рассмотрим, чем отличается асинхронный (событийный) подход от параллельной обработки соединений в отдельных процессах, или потоках.

В простейшем случае при параллельной обработке соединений основной процесс занимается ожиданием входящих соединений и после этого отдает всю работу по обработке нового соединения дочернему процессу, или отдельному потоку. У данного подхода есть следующие достоинства:

- Это простая модель для программирования сервера и подключаемых модулей.
- Каждое соединение обрабатывается независимо, и таким образом возможная долгая обработка данных, выполняемая в одном процессе (потоке), не влияет на другие.

Недостатки:

- Такой подход плохо масштабируется. Например, 1000 одновременно открытых соединений может быть вполне нормальным числом, но 1000 одновременно работающих процессов, или потоков, могут быть проблемой.

При асинхронной обработке в случае одного процесса с одним потоком используется специальный системный вызов осуществляющих диспетчеризацию открытых сокетов и дескрипторов файлов. Этот системный вызов возвращает только сокеты и дескрипторы, готовые для открытия соединения, чтения или записи. Так как ос-

новное время обычно тратится на ожидание ввода-вывода и скорость работы процессора с памятью во много раз выше скорости ввода-вывода, можно успевать выполнять необходимую обработку данных, в то время как система ожидает их новую порцию. Достоинства данного подхода:

- Для многих применений только один процесс с одним потоком может обрабатывать больше соединений, чем в случае параллельной обработки.
- Так как один процесс с одним потоком может обрабатывать сразу несколько соединений для обработки большего количества соединений, используется меньше ресурсов.

Недостатки:

- Программирование асинхронных приложений может быть сложнее, чем приложений с параллельной обработкой.
- В случае использования только одного процесса с одним потоком он может плохо масштабироваться, хотя и лучше, чем при параллельной обработке.
- В простейшем случае асинхронный подход не может использоваться, если необходима долгая обработка данных.

Как можно заметить, оба подхода имеют свои достоинства и недостатки. Чтобы объединить достоинства и уменьшить недостатки из обоих подходов, в настоящее время для асинхронных сетевых приложений наиболее популярен гибридный подход, при котором несколько асинхронных приложений работают параллельно в отдельных процессах, или потоках. Теперь рассмотрим, как это работает в nginx.

Особенности nginx

Nginx разрабатывается Игорем Сысоевым (<http://sysoev.ru>) с весны 2002 года. Первая публичная версия вышла осенью 2004 года. На данный момент сервер находится в активной разработке с точки зрения функциональности, но нет никаких нареканий с точки зрения стабильности. Последняя стабильная версия на данный момент – 0.6.32 (<http://sysoev.ru/nginx/nginx-0.6.32.tar.gz>).

Архитектурно nginx – это асинхронный сервер, который использует один главный процесс для приема соединений и несколько рабочих процессов для их обработки. Рабочие процессы выполняются от непривилегированного пользователя. Асинхронная диспетчеризация может осуществляться как старыми вызовами `select()` и `poll()`, так и с использованием современных подходов специфичных для различных операционных систем: `kqueue` (для FreeBSD, начиная с версии 4.1), `epoll` (для Linux, начиная с версии 2.6), `rt signals` (для Linux, начиная с версии 2.2.19), `/dev/poll` (для Solaris, начиная с версии 7) и `event ports` (для Solaris, начиная с версии 10). Также для оптимизации производительности используются исключаяющие лишнее копирование данных системные вызовы `sendfile()`, `sendfile64()` или `sendfilev()` и сведены к минимуму операции копирования данных внутри сервера.

Далее рассмотрим, как собрать и установить сервер, и подробнее остановимся на конфигурировании некоторых сценариев работы.

Сборка и установка

Хотя nginx уже доступен в пакетах для последних версий Ubuntu Linux, я решил собирать его из исходников, потому что хотел использовать более свежую стабильную версию и модуль `mod_wsgi` для поддержки WSGI-протокола Python. Если вам не нужно собирать nginx с `mod_wsgi`, вы можете пропустить все места, где он упоминается.

Итак, для сборки нам понадобится исходный код nginx, исходный код библиотеки совместимых с Perl регулярных выражений (PCRE) и исходный код модуля `mod_wsgi`. Здесь мы используем самые последние, на данный момент, версии `mod_wsgi` и библиотеки PCRE. При использовании более старых версий PCRE версия не должна быть ниже 4.4:

```
$ wget http://sysoev.ru/nginx/nginx-0.6.32.tar.gz
$ tar -xzf nginx-0.6.32.tar.gz
$ wget ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/pcre-7.8.tar.bz2
$ tar -xjf pcre-7.8.tar.bz2
$ mv pcre-7.8 pcre
$ wget -O mod_wsgi.tar.gz http://hg.mperillo.ath.cx/ \
  nginx/mod_wsgi/archive/tip.tar.gz
$ tar -xzf mod_wsgi.tar.gz
$ mv mod_wsgi-8994b058d2db mod_wsgi
```

Здесь я также переименовал директории для PCRE и `mod_wsgi`, чтобы в дальнейшем использовать для конфигурации один скрипт, не изменяя его. Автор `mod_wsgi` выбрал подход с применением патчей, чтобы успевать за изменениями в nginx, и поэтому мы должны приложить патч для нашей версии nginx. Запишем его как `mod_wsgi_nginx-0.6.32.patch`:

```
--- src/nginx_http_wsgi_handler.c 2008-03-26 22:35:15.000000000 +0300
+++ src/nginx_http_wsgi_handler.c 2008-10-06 16:55:07.000000000 +0400
@@ -71,7 +71,7 @@
    if (r->method == NGX_HTTP_GET ||
        r->method == NGX_HTTP_HEAD) {
        /* XXX not sure */
        rc = ngx_http_discard_body(r);
        rc = ngx_http_discard_request_body(r);

        if (rc != NGX_OK && rc != NGX_AGAIN) {
            return rc;
        }
    }
}
```

И затем приложим:

```
$ patch -d mod_wsgi -p0 < mod_wsgi_nginx_0.6.32.patch
```

Для сборки Nginx с SSL также необходимо установить библиотеку OpenSSL и для использования модуля `ngx_http_gzip_module` библиотеку `zlib`. Для сборки `mod_wsgi` понадобится установить пакет `python-dev`:

```
$ sudo apt-get install libssl-dev
$ sudo apt-get install zlib1g-dev
$ sudo apt-get install python-dev
$ touch nginx.sh
$ chmod a+x nginx.sh
$ vi nginx.sh
```

Для редактирования скрипта конфигурации я использую текстовый редактор Vi, но вы можете использовать любой другой удобный для вас редактор. Для сборки я написал следующий скрипт:

```
#!/bin/sh
./configure \
  --prefix=/var/lib/nginx \
  --sbin-path=/usr/sbin/nginx \
  --conf-path=/etc/nginx/nginx.conf \
  --pid-path=/var/run/nginx.pid \
  --error-log-path=/var/log/nginx/error.log \
  --http-log-path=/var/log/nginx/access.log \
  --lock-path=/var/lock/nginx \
  --http-client-body-temp-path=/var/lib/nginx/client_body \
  --http-proxy-temp-path=/var/lib/nginx/proxy \
  --http-fastcgi-temp-path=/var/lib/nginx/fastcgi \
  --user=www-data \
  --group=www-data \
  --without-select_module \
  --without-poll_module \
  --without-http_ssi_module \
  --without-http_geo_module \
  --without-http_referer_module \
  --without-http_memcached_module \
  --without-http_limit_zone_module \
  --without-http_empty_gif_module \
  --without-http_browser_module \
  --without-http_upstream_ip_hash_module \
  --with-http_ssl_module \
  --with-http_gzip_static_module \
  --with-pcre=../pcre \
  --add-module=../mod_wsgi
```

По умолчанию nginx устанавливается в директорию `/usr/local/nginx` и использует поддиректории для файлов конфигурации и различных временных файлов, но я привык придерживаться стандарта иерархии файловой системы (<http://www.pathname.com/fhs/>), и поэтому в начале скрипта идут опции переназначения различных директорий. Затем идут опции установки пользователя и группы, под которыми будут работать рабочие процессы. В случае если данные пользователь и группа не созданы в вашей системе, их надо будет создать вручную. После этого находятся опции, включающие и отключающие различные модули, в том числе отключение поддержки системных вызовов `select()` и `poll()`, т.к. в моем случае, на Linux 2.6, будет использоваться вызов `epoll()`. Если вы не знаете какой системный вызов может использоваться в вашей системе, оставьте это решение скрипту конфигурации, который выберет наиболее оптимальный вариант. Соответственно если вам нужны другие пути к файлам и директориям, другой пользователь и группа, или другие модули, вы можете внести изменения в эти секции. После того как все готово со скриптом конфигурации, можно начать конфигурацию:

```
$ cd nginx-0.6.32
$ ../nginx.sh
```

В конце своей работы скрипт конфигурации должен вывести следующую информацию, где мы можем убедиться, что все было сделано правильно:

```
Configuration summary
+ using PCRE library: ../pcre
+ using system OpenSSL library
+ md5 library is not used
+ sha1 library is not used
+ using system zlib library

nginx path prefix: "/var/lib/nginx"
nginx binary file: "/usr/sbin/nginx"
nginx configuration prefix: "/etc/nginx"
nginx configuration file: "/etc/nginx/nginx.conf"
nginx pid file: "/var/run/nginx.pid"
nginx error log file: "/var/log/nginx/error.log"
nginx http access log file: "/var/log/nginx/access.log"
nginx http client request body temporary files: "/var/lib/nginx/client_body"
nginx http proxy temporary files: "/var/lib/nginx/proxy"
nginx http fastcgi temporary files: "/var/lib/nginx/fastcgi"
```

Теперь можно начать компиляцию:

```
$ make
```

Если компиляция прошла успешно, устанавливаем nginx:

```
$ sudo make install
```

При установке для конфигурации выше будет создана директория `/var/lib/nginx`, бинарный файл сервера `nginx` будет установлен в директорию `/usr/sbin` (в случае присутствия старого файла он переименовывается в `nginx.old`), будет создана директория `/etc/nginx`, в которую будет перенесена конфигурация по умолчанию, и будут созданы директории `/var/log/nginx` и `/var/lib/nginx` с нужными правами доступа.

Запуск и управление

В простейшем случае мы можем запустить `nginx` следующим образом:

```
$ sudo /usr/sbin/nginx
```

После первого запуска можно зайти по адресу `http://localhost/` и увидеть страницу приветствия. Для остановки сервера можно выполнить:

```
$ sudo killall nginx
```

Конечно, такой способ управления лучше всего использовать только для первого тестирования, а для обычного использования и в последующем автоматического запуска лучше написать скрипт управления и положить его в директорий `/etc/init.d` под именем `nginx`, не забывая дать права на исполнение:

```
#!/bin/sh
#
# /etc/init.d/nginx: start and stop nginx http server
#

PID_FILE=/var/run/nginx.pid
CAT=/bin/cat
NGINX=/usr/sbin/nginx
START_STOP_DAEMON=/sbin/start-stop-daemon

. /lib/lsb/init-functions

do_start()
{
    $START_STOP_DAEMON --start --quiet \
      --pidfile $PID_FILE --exec $NGINX \
      RETVAL=$?

    if [ $RETVAL = 0 ]; then
        log_success_msg "Server started at pid" \
          $(($CAT $PID_FILE))
    elif [ $RETVAL = 1 ]; then
        log_failure_msg "Server already running at pid" \
          $(($CAT $PID_FILE))
    else
        log_failure_msg "Error starting server"
    fi

    return $RETVAL
}
```

```

do_stop()
{
    $START_STOP_DAEMON --stop --quiet \
    --retry=TERM/30/KILL/5 --pidfile $PID_FILE \
    --exec $NGINX
    RETVAL=$?

    if [ $RETVAL = 0 ]; then
        log_success_msg "Server stopped"
    elif [ $RETVAL = 1 ]; then
        log_failure_msg "Server already stopped"
    else
        log_failure_msg "Error stopping server"
    fi

    return $RETVAL
}

is_running()
{
    if [ -f $PID_FILE ]; then
        kill -n 0 $(cat $PID_FILE)
        if [ $? = 0 ]; then
            return 1
        else
            return 0
        fi
    fi
    return 0
}

case "$1" in
    start)
        log_begin_msg
        do_start
        log_end_msg $?
        ;;

    stop)
        log_begin_msg
        do_stop
        log_end_msg $?
        ;;

    status)
        log_begin_msg
        is_running
        if [ $? = 0 ]; then
            log_success_msg "Server not running"
            log_end_msg 1
        else
            log_success_msg "Server running at pid" \
                $(cat $PID_FILE)
            log_end_msg 0
        fi
        ;;

    restart)
        log_begin_msg
        do_stop
        if [ $? = 0 ] || [ $? = 1 ]; then
            do_start
        fi
        log_end_msg $?
        ;;

    reload)
        log_begin_msg
        is_running
        if [ $? = 0 ]; then
            log_failure_msg "Server not running"
            log_end_msg 1
        else
            $NGINX -t
            if [ $? = 0 ]; then
                kill -HUP $(cat $PID_FILE)
                if [ $? = 0 ]; then
                    log_success_msg "Server configuration \
                        reloaded"
                    log_end_msg 0
                else
                    log_failure_msg "Error while \
                        reloading configuration"
                    log_end_msg 1
                fi
            fi
        fi
    esac
exit $?

```

```

        fi
    else
        log_failure_msg "Configuration errors found"
        log_end_msg 1
    fi
fi
;;

*)
    log_success_msg "Usage: /etc/init.d/nginx \
        {start|stop|status|restart|reload}"
    exit 3
    ;;
esac

exit $?

```

Дадим скрипту нужные права:

```
$ sudo chmod a+x /etc/init.d/nginx
```

И теперь можно управлять nginx через наш скрипт, используя команды start, stop, status, restart и reload. Чтобы nginx запускался при старте системы можно воспользоваться следующей командой:

```
$ sudo update-rc.d nginx defaults
```

Эта команда создаст ссылки на скрипт /etc/init.d/nginx в каталогах, соответствующих различным уровням запуска: /etc/rc0.d – /etc/rc6.d.

Настраиваем статический сайт

Когда мы разобрались с управлением сервером, рассмотрим детали настройки статического сайта.

Вот пример основного файла конфигурации для статических сайтов:

```

user www-data www-data;

worker_processes 4;

error_log /var/log/nginx/error.log info;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    charset utf-8;
    server_tokens off;
    sendfile on;

    gzip on;
    gzip_http_version 1.0;
    gzip_comp_level 6;
    gzip_proxied any;
    gzip_disable "MSIE [1-6]\.(?!.*SV1)";
    gzip_buffers 16 8k;
    gzip_types text/plain text/html text/css
        application/x-javascript text/xml application/xml
        application/javascript text/js
        application/xml+rss text \
            /javascript application/atom+xml;

    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;

    include /etc/nginx/example.ru.conf
}

```


Рассмотрим опции по порядку:

- Опция `user` описывает, от имени какого пользователя и группы будут выполняться рабочие процессы.
- Опция `worker_processes` определяет количество рабочих процессов. Количество рабочих процессов – параметр, который влияет на производительность и в основном зависит от ожидаемой нагрузки на сервер и требуемой функциональности сайтов. В простейшем случае может быть только один рабочий процесс, но если используется компрессия данных, или SSL, то их количество может быть равно количеству процессоров или в два раза большему числу. В данном случае я выбрал 4 рабочих процесса, т.к. пример со статическим сайтом использует и компрессию, и SSL и запускается на двухпроцессорной машине. В случае если используются не совсем свойственные nginx синхронные модули, как например `mod_wsgi`, нужно выбрать количество рабочих процессов в зависимости от ожидаемого количества одновременных запросов и скорости их обработки.
- Опции `error_log` и `pid` описывают соответственно путь к файлу журнала ошибок вместе с уровнем логгинга и путь к файлу, в котором будет храниться идентификатор главного процесса сервера.
- Секция `events` описывает опции, связанные с обработкой событий. Опция `worker_connections` описывает ограничение на количество одновременно обрабатываемых рабочими процессами соединений. Таким образом, в случае статического сайта максимальное количество одновременных соединений с сервером можно рассчитать по формуле: `worker_processes * worker_connections`. Также в этой секции можно выбрать метод обработки событий, например `use epoll`, и другие настройки, связанные с обработкой событий.
- Секция `http` описывает конфигурацию HTTP-сервера и всех сайтов. Опции `include` и `default_type` описывают каталог MIME-типов и тип по умолчанию соответственно. Опция `charset` определяет кодировку, которая будет добавлена в заголовок ответа `Content-Type`. Если ее не указывать, браузеры будут использовать собственные кодировки по умолчанию как кодировку контента. Опция `server_tokens` управляет выводом номера версии сервера в ответах. Опция `sendfile` включает использование системного вызова, оптимизирующего посылку файлов. Набор опций, начинающихся на `gzip`, описывает параметры сжатия ответов. Набор опций, начинающихся на `SSL`, описывает настройки SSL-сессий. Опция `include` подключает другой файл конфигурации, в данном случае непосредственно конфигурацию сайта.

Конфигурация сайта может выглядеть следующим образом:

Протокол WSGI

До определенного момента для связи веб-приложения, написанного на Python и HTTP-сервера, использовалось множество подходов. Например, это могли быть протоколы CGI и FastCGI или модуль `mod_python` для Apache. Также многие приложения, например Zope, использовали жестко интегрированные HTTP-серверы. Стандартный протокол WSGI (<http://www.python.org/dev/peps/pep-0333>) был впервые предложен в декабре 2003 года и на данный момент поддерживается подавляющим большинством серверов и веб-приложений, написанных на Python. Протокол подробно опи-

сывает порядок передачи параметров приложению, передачи ответа серверу и обработку ошибок. Кроме того, учитываются расширения, специфичные для различных серверов. Простейшее приложение с использованием WSGI может выглядеть так (см. листинг).

Таким образом, протокол позволяет, например, начать отлаживать веб-приложение, используя стандартный пакет `wsgiref` (появился в Python 2.5), и затем перейти на `mod_wsgi` (с nginx, или Apache) или HTTP-сервер из асинхронного каркаса Twisted.

```
def application(environ, start_response):
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain')]
    start_response(status, response_headers)
    return ['Hello world!\n']
```

```
server {
    listen      80;
    server_name example.ru *.example.ru;

    error_log   /var/log/nginx/example.ru/error.log info;
    access_log  /var/log/nginx/example.ru/access.log;

    location / {
        root    /home/www/example.ru/data/;
        index   index.html;

        if ($host != "example.ru") {
            rewrite ^(.*) http://example.ru$1 permanent;
        }
    }
}

server {
    listen      443;
    server_name example.ru;

    error_log   /var/log/nginx/example.ru/error.log info;
    access_log  /var/log/nginx/example.ru/access.log;

    ssl on;
    ssl_certificate /etc/nginx/cert/example.ru.pem;
    ssl_certificate_key /etc/nginx/cert/example.ru.pem;

    auth_basic "Example admin place";
    auth_basic_user_file /home/www/example.ru/.htpasswd;

    location / {
        root    /home/www/example.ru/data/admin/;
        index   index.html;
    }
}
```

В данном случае файл `/etc/nginx/example.ru.conf` описывает две версии одного сайта – обычную HTTP-версию и HTTPS-версию, защищенную паролем. Секции `server` описывают конфигурацию виртуальных серверов.

Рассмотрим опции подробнее:

- Опция `listen` определяет адрес и порт, на котором сервер будет принимать запросы. Здесь также можно задать дополнительные опции, управляющие приемом соединений. В том числе с помощью этой опции можно объединить конфигурацию HTTP- и HTTPS-серверов в одной секции, хотя в примере этого не сделано.
- Опция `server_name` задает имя и псевдонимы сайта.
- Опции `error_log` и `access_log` определяют путь к файлу журнала ошибок с уровнем логгирования и журналу запросов соответственно.

- Секция `location` объединяет конфигурационные параметры в зависимости от URI запроса.
- Опции `root` и `index` описывают корневую директорию сайта и имя индексного файла по умолчанию.
- Опции `if` и `rewrite` используются для изменения URI запроса. В данном примере все запросы для других доменов перенаправляются на `example.ru`.
- Опции, начинающиеся с `ssl_`, описывают путь к SSL-сертификату и приватному ключу соответственно.
- Опции, начинающиеся с `auth_basic`, описывают текст комментария при вводе пароля и путь к файлу со списком паролей соответственно, в случае использования простой HTTP-авторизации. Файл со списком паролей может быть создан утилитой `htpasswd` из пакета сервера Apache. На данный момент `nginx` не поддерживает других методов авторизации кроме простой HTTP-авторизации.

Настраиваем HTTP-прокси

Часто HTTP-сервер является лишь посредником (прокси), стоящим перед другим HTTP-сервером. Например, такая конфигурация нужна, если один из сайтов сделан с применением технологии, которая уже имеет свой HTTP-сервер, или посредник используется как балансировщик нагрузки для нескольких стоящих за ним серверов. В `nginx` эта функциональность поддерживается модулем `ngx_http_proxy_module`.

Также с помощью модуля `ngx_http_memcached_module` `nginx` можно использовать совместно с `Memcached` (<http://www.danga.com/memcached>) для кэширования контента. Рассмотрим пример простого сайта, который перенаправляет запросы на сервер, сделанный с использованием технологии Zope 3 (<http://www.zope.org>). Сначала добавим следующие опции в секцию `http` файла `/etc/nginx/nginx.conf`:

```
proxy_set_header    Host            $host;
proxy_set_header    X-Real-IP       $remote_addr;
proxy_set_header    X-Forwarded-For 1
    $proxy_add_x_forwarded_for;

include /etc/nginx/zope3.example.ru.conf
```

Эти опции устанавливают дополнительные заголовки, которые будут переданы вспомогательному серверу, и подключают файл конфигурации сайта. Конфигурация сайта может быть такой:

```
server {
    listen      80;
    server_name zope3.example.ru;

    error_log 1
        /var/log/nginx/zope3.example.ru/error.log info;
    access_log 1
        /var/log/nginx/zope3.example.ru/access.log;

    error_page 502 503 504 /maintenance.html;
    location /maintenance.html {
        alias /home/www/zope3.example.ru/data/index.html;
    }

    location / {
        proxy_pass http://127.0.0.1:8080 1
            /++skin++example/sites/example 1
            /++vh++http:zope3.example.ru:80/++/;
    }
}
```

Рассмотрим не описанные выше опции:

- Опция `error_page` задает URI, который будет показываться для заданных кодов ошибок. В данном случае это ошибки, связанные с недоступностью вспомогательного сервера, или невозможностью обработать запрос.
- Опция `alias` задает замену для указанного пути. Таким образом для страницы сайта `/maintenance.html` будет возвращено содержимое файла `/home/www/zope3.example.ru/data/index.html`, который может содержать страницу с описанием того, почему сервер в данный момент не доступен.
- Опция `proxy_pass` задает адрес и путь на вспомогательном сервере, на который будет отображаться путь на сайте. Таким образом, в случае доступа к `http://example.zope3.ru/page.html` пользователю будет возвращен результат обработки запроса вспомогательным сервером по адресу `http://127.0.0.1:8080/++skin++example/sites/example/++vh++http:zope3.example.ru:80/++/page.html`.

Настраиваем PHP

На данный момент наиболее оптимальным способом связать `nginx` и PHP является протокол `FastCGI`. При этом PHP должен работать в отдельном процессе, что не является особой проблемой, так как PHP уже имеет собственный `FastCGI`-демон. Для управления демоном PHP можно использовать следующий скрипт, который в свою очередь можно сохранить как `/etc/init.d/php-fastcgi` и дать ему права на исполнение:

```
#!/bin/sh

EXEC_AS_USER=www-data

FCGI_HOST=127.0.0.1
FCGI_PORT=9000

PHP_FCGI_CHILDREN=5
PHP_FCGI_MAX_REQUESTS=1000

DAEMON=/usr/bin/php-cgi
PIDFILE=/var/run/php-fastcgi.pid
PHP_CONFIG_FILE=/etc/php5/cgi/php.ini
START_STOP_DAEMON=/sbin/start-stop-daemon

export PHP_FCGI_CHILDREN PHP_FCGI_MAX_REQUESTS
DAEMON_ARGS="-q -b $FCGI_HOST:$FCGI_PORT 1
    -c $PHP_CONFIG_FILE"

. /lib/lsb/init-functions

do_start()
{
    $START_STOP_DAEMON --start --quiet 1
    --pidfile $PIDFILE --exec $DAEMON --test || return 1
    $START_STOP_DAEMON --start --quiet 1
    --pidfile $PIDFILE --exec $DAEMON 1
    --background --make-pidfile --chuid $EXEC_AS_USER 1
    --startas $DAEMON -- $DAEMON_ARGS || return 2
}

do_stop()
{
    $START_STOP_DAEMON --stop --quiet 1
    --retry=TERM/30/KILL/5 1
    --pidfile $PIDFILE # --name $DAEMON
    RETVAL="$?"
    [ "$RETVAL" = 2 ] && return 2
    $START_STOP_DAEMON --stop --quiet --oknodo 1
    --retry=0/30/KILL/5 --exec $DAEMON
    [ "$?" = 2 ] && return 2
    rm -f $PIDFILE
    return "$RETVAL"
}
```

```

case "$1" in
  start)
    log_begin_msg
    do_start
    log_end_msg $?
    ;;
  stop)
    log_begin_msg
    do_stop
    log_end_msg $?
    ;;
  restart)
    log_begin_msg
    do_stop
    if [ $? = 0 ] || [ $? = 1 ]; then
      do_start
    fi
    log_end_msg $?
    ;;
  *)
    echo "Usage: php-fastcgi {start|stop|restart}"
    exit 3
    ;;
esac

```

После старта демона он начнет принимать запросы по адресу 127.0.0.1:9000, и для сайта можно использовать следующий файл конфигурации:

```

error_log  /var/log/nginx/php.example.ru/error.log info;
access_log /var/log/nginx/php.example.ru/access.log;

location / {
    root /home/www/php.example.ru/data/;
    index index.html;
}

location ~ \.php$ {
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;
    include /etc/nginx/fastcgi_params;
    fastcgi_param SCRIPT_FILENAME /home/www/php.example.ru/data$fastcgi_script_name;
}

```

Кроме уже известных опций, здесь есть секция, которая определяет действия для всех путей, оканчивающихся на .php:

- Опция `fastcgi_pass` определяет адрес FastCGI-сервера. В нашем случае это адрес, где запущен демон PHP.
- Опция `fastcgi_index` задает имя индексного файла для FastCGI.
- Опцией `include` мы подключаем файл с параметрами, которые будут передаваться в FastCGI-запросах. К сожалению, мы не можем один раз подключить эти параметры на более верхнем уровне, т.к. следующая опция `fastcgi_param`, которая в свою очередь задает еще один дополнительный параметр, отменяет все определения параметров более высокого уровня.

Настраиваем Python с WSGI

WSGI является протоколом, который связывает HTTP-сервер и приложение на Python. На данный момент уже есть большое количество серверов и приложений, работающих по этому протоколу, что позволяет использовать их как взаимозаменяемые компоненты. В примере я подключаю к nginx известный трекер ошибок Trac (<http://trac.edgewall.org>). Прежде всего нам нужно создать файл с описанием параметров, передаваемых по WSGI. Файл можно записать как `/etc/nginx/wsgi_params` по аналогии с `fastcgi_params`:

```

wsgi_var REQUEST_METHOD $request_method;
wsgi_var QUERY_STRING $query_string;

wsgi_var CONTENT_TYPE $content_type;
wsgi_var CONTENT_LENGTH $content_length;

wsgi_var SERVER_NAME $server_name;
wsgi_var SERVER_PORT $server_port;

wsgi_var SERVER_PROTOCOL $server_protocol;

wsgi_var REQUEST_URI $request_uri;
wsgi_var DOCUMENT_URI $document_uri;
wsgi_var DOCUMENT_ROOT $document_root;

wsgi_var SERVER_SOFTWARE $nginx_version;

wsgi_var REMOTE_ADDR $remote_addr;
wsgi_var REMOTE_PORT $remote_port;
wsgi_var SERVER_ADDR $server_addr;

wsgi_var REMOTE_USER $remote_user;

```

После этого необходимо добавить в `/etc/nginx/nginx.conf` в секцию `http` опции:

```

wsgi_temp_path /var/lib/nginx/wsgi;
include /etc/nginx/wsgi_params;

```

Теперь сделаем WSGI-скрипт для подключения Trac и положим его как `/home/www/wsgi.example.ru/wsgi.py`, не забывая сделать исполняемым:

```

#!/usr/bin/env python2.4

import os

os.environ['TRAC_ENV'] = "/home/www/wsgi.example.ru/trac"
os.environ['PYTHON_EGG_CACHE'] = "/var/tmp"

import trac.web.main

application = trac.web.main.dispatch_request

```

И последний момент — конфигурация сайта:

```

server {
    listen 80;
    server_name wsgi.example.ru;

    error_log /var/log/nginx/wsgi.example.ru/error.log info;
    access_log /var/log/nginx/wsgi.example.ru/access.log;

    location / {
        wsgi_pass /home/www/wsgi.example.ru/wsgi.py;
    }
}

```

Здесь опция `wsgi_pass` определяет имя скрипта, который будет выполняться в рамках рабочего процесса. Соответственно, так как скрипты выполняются синхронно в рамках рабочих процессов, для оптимизации производительности сайта с WSGI, возможно, придется увеличить значение опции `worker_processes` в `nginx.conf`.

Заключение

Я постарался охватить широкий спектр вопросов, связанных с nginx. Конечно, многие детали пока остались за кадром, но о них можно прочитать на следующих сайтах: <http://sysoev.ru/nginx> и <http://wiki.codemongers.com/Main>. Надеюсь, статья помогла вам расширить кругозор и выбрать правильный HTTP-сервер, подходящий для ваших задач. 