# 1.8 Web сервера

* Запуск web сервера
* Запуск: **sudo /etc/init.d/nginx start**
* Чтение файла конфигурации
* Получение порта 80
* Открытие (создание) логов
* Понижение привилегий
* Запуск дочерних процессов / потоков (\*)
* Готов к обработке запроса

Файлы web сервера

* Конфиг: **/etc/nginx/nginx.conf**
* include **/etc/nginx/sites-enabled/\***
* Init-script: **/etc/init.d/nginx [start|stop|restart]**
* PID-file: **/var/run/nginx.pid**
* Error-log: **/var/log/nginx/error.log**
* Access-log: **/var/log/nginx/access.log**

Процессы web сервера

* Master (root, 1 процесс)
  + Чтение и валидация конфига
  + Открытие сокета(ов) и логов
  + Запуск и управление дочерними workers
  + Graceful restart, Binary updates
* Worker (www-data, 1 + процессов)
  + Обработка входящих запросов

Работа Worker процесса

1. Чтение HTTP запроса
2. Выбор virtual host
3. Выбор location
4. Проверка доступа
5. Чтение файла
6. Применение фильтров
7. Отправка HTTP ответа
8. Запись Access Log
9. Очистка

Модульная архитектура

* Web сервера – не монолитный
* Динамическая загрузка модулей – LoadModule
* Этапы обработки запроса и модули
* Дополнительны директивы, контексты
* Примеры: mod\_mime, mod\_mime\_magic, mod\_autoindex, mod\_rewrite, mod\_cgi, mod\_gzip

**Конфигурация web сервера**

Терминология

**Virtual host –** секция конфига web сервера, отвечающая за обслуживание определенного домена

**Location –** секция конфига, отвечающая за обслуживание определенной группы URL

Пример упрощенного конфига

**user www www;**

**error\_log /var/log/nginx.error\_log info;**

**http {**

**include conf/mime.types;**

**default\_type application/octet-stream;**

**log\_format simple ‘$remote\_addr $reqeust $status’;**

**server {**

**listen one.example.com;**

**server\_name one.example.com www.one.example.com;**

**access\_log /var/log/nginx.access\_log simple;**

**location / { root /www/one.example.comp; }**

**location ~\* ^.+\.(jpg|jpeg|gif)$ {**

**root /www/images;**

**acess\_log off;**

**expires 30d;**

**}**

**}**

**}**

Приоритеты location в nginx

1. **location = /img/1.jpg (точное совпадение)**
2. **location ^~ /pic/ (совпадение по префиксу)**
3. **location ~\* \.jpg$ (по регулярному выражению)**
4. **location /img/ (совпадение по префиксу)**

При одинаковом приоритете используется тот location, что находится выше в конфиге.

Секции и директивы

* **http –** конфигурация для HTTP сервера
* **server –** конфигурация домена (вирт. хоста)
* **server\_name –** имена доменов
* **location –** локейшен, группа URL
* **root, alias –** откуда нужно брать файлы
* **error\_log –** лог ошибок сервера
* **access\_log –** лог запросов

Отдача статических документов

**location ~\* ^.+\.(jpg|png|gif)$ {**

**root /wwww/images;**

**}**

**Преобразование:**

**/2017/06/ae2b6.png → /www/images/2017/ae2b5.png**

**location /sitemap/ {**

**alias /home/www/generated/;**

**}**

**Преобразование:**

**/sitemap/index.xml → /home/www/generated/index.xml**

**(alias заменяет часть совпавшего урла, работает только с совпадениями по префиксу)**

Атрибуты файлов и процессов

У процесса есть

* пользователь
* группа

У файла (или директории) есть

* пользователь (владелец)
* группа
* права доступа (read/write/execute)

Как узнать атрибуты?

$ ps –o pid, euser, egroup, comm, args –C nginx

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PID | EUSER | EGROUP | COMMAND |  |
| 29731 | root | root | nginx: | master process |
| 29732 | www-data | www-data | nginx: | master process |
| 29733 | www-data | www-data | nginx: | master process |
| 29374 | www-data | www-data | nginx: | master process |
| 29737 | www-data | www-data | nginx: | master process |

$ ls –lag www/index.html

-rw-r--r-- 1 nuf users 156K Jul 1 15:22 www/index.html

Проверка доступа

Для того, чтобы открыть файл, необходимо иметь права на чтение r самого файла и на исполнение x директорий, в которых он находится. Наличие прав проверяется следующим образом:

Если совпадает пользователь: **-rw-r--r--**

Если совпадает группа: **–rw-r--r--**

Иначе: **–rw-r--r--**

**Модели обработки сетевых соединений**

Простейший TCP сервер

* Открываем, слушаем порт
* Входим в бесконечных цикл
* Устанавливаем сетевое соединение с клиентом
* Читаем входящие данные
* Открываем файл, читаем из него данные
* Отправляем обратно клиенту
* Закрытие файла и соединения

Блокирующий ввод - вывод

Приложение передают задачу ОС, далее ОС выполняя свою задачу, переводит процесс в режим ожидания. Далее при появлении данных в сокете процесс пробуждается и происходит возврат из системного вызова.

Решение проблемы ожидания

* множество потоков - multithreading
* множество процессов - prefork, pool of workers
* комбинированный подход

Плюсы и минусы prefork

+ простота в разработке

+ можно использовать любые библиотеки

- большое потребление памяти: 1 клиент = 1процесс

- проблема с долгоживущими соединениями

Плюсы и минусы multithreading по сравнению с prefork

+ экономия памяти: 1 клиент = 1 поток

- требует аккуратной работы с памятью

- ограничение в выборе библиотек

Неблокирующий ввод - вывод

Вместо усыпления процесса, ядро возвращает управление из функции со специальным кодом (EAGAIN / E\_WOULDBLOCK), далее приложение может заняться обработкой других клиентов. После, приложение может проверить повторно сделать вызов для первого сетевого соединения. К этому моменту времени в сокете могут быть данные, которые вернуться в приложение. Таким образом достигается параллельная работа с разными клиентами.

Event-driven разработка

* множество открытых файловых дескрипторов
* select, eoll, aio… - методы мультиплексирования
* последовательное исполнение → события

Плюсы и минусы

+ быстро, программа не блокируется

+ экономия памяти: 1 клиент = 1 объект

+ обработка большого количеств клиентов

+ обработка медленных или долгоживущих соединений

- тяжело программировать

- блокирующий вызов все портит (блочит всех клиентов)

Кто есть кто

* Apache - prefork, worker, threads, C
* Tomcat, Jetty - threads, Java
* Starman, Gunicorn - prefork, ЯВУ
* Nginx, lighttpd - ассинхронные, C
* Node.js, Tornaod - асинхронные, ЯВУ

**Практическая часть**

Задача

1. Создать структуру директорий

2. Настроить nginx

3. Включить свой nginx конфиг с помощью символической ссылки.

Решение

1. Создал структуру директорий

2. Создал файл /web/etc/nginx.conf

server {

listen 80;

error\_log /home/box/web/error.log;

access\_log /home/box/web/access.log;

location ^~ /uploads/ {

root /home/box/web;

}

location ~\* ^.\.\w+$ {

root /home/box/web/public;

}

location / {

return 404;

}

}

3. Создал init.sh файл

sudo ln -s /home/box/web/etc/nginx.conf /etc/nginx/sites-enabled/default

sudo /etc/init.d/nginx restart

И выполнил:

chomd +x init.sh

./init.sh

# 1.9 Архитектура frontend-backend

Общая архитектура

|  |  |  |  |
| --- | --- | --- | --- |
| Клиенты → | Frontend → | Backend → | Хранилища |
|  | ↓ |  |  |
|  | Хранилища |  |  |

Задачи Frontend (web) сервера

* отдача статических документов
* проксирование (reverse proxy)
* балансировка нагрузки
* кеширование
* сборка SSI (динамическая сборка web странички)
* авторизация, SLL, нарезка картинок, gzip

Reverse proxy

* frontend (slow) читает запрос от клиента
* frontend (fast) передает запрос свободному backend
* backend генерирует страницу
* backend (fast) возвращает ответ frontend серверу
* frontend (slow) возвращает ответ клиенту

**Настройка проксироваия в nginx**

proxy\_set\_header Host $host; (передача в другой домен)

proxy\_set\_header X-Real-IP $remote\_addr; (ip клиента)

location / {

proxy\_pass http://backend; (передача в upstream)

}

location /parent/ {

proxy\_pass http://www.partner.com; (передача бекэнд)

}

location ~ \.\w\w\w?\w?$ {

root /www/static;

}

Настройка upstream в nginx

Upstream - группа серверов, работающие под общим именем

upstream backend {

server back1.example.com:8080 weight=1 max\_fails=3;

server back2.example.com:8080 weibht=2;

server unix:/tmp/backend.sock;

server backup1.example.com:8080 backup;

server backup2.example.com:8080 backup;

}

**Backend (application) server**

Роль application сервера заключается в исполнении бизнес-логики приложения и генерации динамических документов.

На каждый HTTP запрос application сервер запускает некоторый обработчик в приложении. Это может быть функция, класс или программа, в зависимости от технологии.

Протоколы запуска приложения

* Servlets и др. специализированное API
* mod\_perl, mod\_python, mod\_php
* CGI
* FastCGI
* SCGI
* PSGI, WSGI, Rack

CGI - Common Gateway Inteface

* Метод, QueryString, заголовки запроса - через переменные окружения
* Тело запроса передается через STDIN
* Заголовки и тело ответа возвращаются через STDOUT
* HTTP код ответа передаются через псевдо заголовок Status
* Поток ошибок STDERR направляется в лог ошибок сервера

При запросе постоянно приходится запускать скрипт, что замедляет время ответа сервера.

Переменные окружения CGI

REQUEST\_METHOD - метод запроса

PATH\_INFO - путь из URL

QUERY\_STRING - фрагмент URL после ?

REMOTE\_ADDR - ip адрес пользователя

CONTENT\_LEGHT - длина тела запроса

HTTP\_COOKIE - Заголовок Cookie

HTTP\_ANY\_HEADER\_NAME - любой другой http заголовок

FastCGI и SCGI

Основная проблема CGI - низкая производительность. Протоколы FastCGI и SCGI призваны решить эту проблему путем демонизации приложения. Иногда это возможно сделать даже без изменения кода CGI приложения.

WSGI - актуальный протокол

WSGI, PSGI, Rack - протоколы вызова функции обработчика из application сервера. Сам application server при этом может выполняться в отдельном процессе или совпадать с web сервером. Как правило, при использовании этих протоколов в качестве application сервера выступает отдельный легковесный процесс.

Web Server Gateway Interface

* Обработчик - функция или класс (callable)
* Метод, QueryString, заголовки запроса - через аргумент environ
* Тело запроса передается через file-handle wsgi.input
* HTTP код ответа и заголовки ответа передаются через вызов функции start\_response
* Тело ответа возвращается в виде списка (iterable) из обработчика
* Поток ошибо должен быть направлен в file-handle wsgi.stderr

Переменные environ dict

* CGI-like переменные: REQUEST\_URL, …
* wsgi.version - версия WSGI протокола
* wsgi.url\_scheme - схема текущего URL
* wsgi.input - file-handle для чтения тела запроса
* wsgi.errors - file-handle для вывода ошибок
* wsgi.multithreaded - …
* wsgi.mulitprocess - …

Развертывание WSGI

|  |  |  |
| --- | --- | --- |
| Web Server | ← HTTP → | Application Server |
|  |  | WSGI |
|  |  | Framework |
|  |  | Application |

Что ложится на приложение?

Анализ PATH\_INFO и выбор конкретного обработчика

Разбор конкретных заголовков, например Cookie

Разбор QUERY\_STRING (?a=1&b=2)

Разбор тела запроса:

* x-www-form-urlencoded
* multipart/form-data

Вывод правильных заголовков ответа