21 апреля 2012 в 17:02

Оптимизация flatpages проекта на django под минимальные системные требования. Статья-шутка

Веб-разработка*, Django*

Под катом много букв, но не беспокойтесь — вы всех их знаете.

Предыстория

Так интереснее

Когда-то давным-давно мы с моей ненаглядной пытались сделать свой первый маленький проект. Тогда я занимался только дизайном поэтому программиста пришлось нанять: я ему макеты, он нам верстку и само приложение. Помню хостинг нам обошелся где-то в 130-150 рублей — конечно это был LAMP.

Как это, обычно, бывает первый блин у нас не вышел. Прошло время и мы созрели для второго проекта. Теперь я мог все сделать сам. Накидав дизайн, я принялся за верстку, а затем и за само приложение. В этот раз ничего существенного: простой сайт с кучей статики — даже не интересно.

На хостинг я по привычке прикинул так: открываем страничку тарифов на VPS, выбираем первую строчку снизу — ну где-то рублей 800-1000. «Так много? Раньше мы платили рублей 200…» — супруга была озадачена. Действительно, почему больше-то? Ответ, конечно, очевиден: впс, как не крути, джанга там, туда-сюда, «это вам не джумля какаянить!» — а что джумла? Джумла, вон: 100 руб. живи месяц счастливо. Но это слишком простой ответ, так не интересно.

Задача

И так у нас есть задача: минимальная впс-ка и готовый проект на django — статика, парочка форм, админка и прочее. Синтетический минимум: 200 мб ОЗУ, 500 мгц проц, на винте — ну пусть будет два гига на все вместе с системой.

DB

Самое узкое место в задаче: ОЗУ. Возьмем sqlite. Wait. What?! Я серьезно, он не так уж плох. Его недостатки мы компенсируем кешем.

Кеш

Берем все самое мощное: nginx + memcached и научим работать с джангой.

Веб-сервер

Это будет uwsgi — быстрый и модный. Для для любителей сферически-вакуумных можно глянуть тесты.

Сессии

Сессии надо где-то хранить. Обычно я использую redis, но не здесь. Мемкешд тоже не подходит: учитывая архитектуру кеширования мы часто будем сбрасывать весь кеш, а вместе с ними и сессии *удалятся*. На помощь приходит новинка в джанге 1.4. Это cookie-based сессии. Принцип прост: все данные о сессии храниться в куках у пользователя. Сами данные не шифруются, но целостность обеспечивается за счет криптографической подписи. Подробнее можно почитать в доках.

Контент

Тут все просто берем MarkDown. «Почему маркдаун?». Он крут. Он офигенен. Даже эту статью я пишу на нем. Он поможет нам облегчить размер бд, он не содержит тегов. Текст в MD, порой, в половину легче треша из fckeditor alike редактора и раз в 100 приятнее глазу. К тому же, текст должен форматироваться стилями сайта, а не редактора: одни стили правят всем — прям как у саурона.

Софт

Возьмем Ubuntu server minimal, чтобы получить самый новый и самый быстрый софт: nginx, memcached, python 2.7. Вообще лучше что-нибудь полегче типа slitaz, puppy linux, slax — хотя бы потому-что минимум для убунты 128 мб ОЗУ, а настроенная «базовая система» + кое-какой софт у меня вышел больше гигабайта на винте. Серверный дистрибутив, *сэр*. Но мы решили получить удовольствие от статьи, в общем, будем знать меру.

Разработка

Django

Сначала нам нужно наладить наш конфиг джанги:

```
# Укажем новые бэкэнд к сессиям.
# Данные не шифруется, поэтому мы запретим их чтение через js.
SESSION_ENGINE = 'django.contrib.sessions.backends.signed_cookies'
SESSION_COOKIE_HTTPONLY = True

# https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-USE_I18N
# Выключим перевод в приложениях. Как написано в доках: This provides an easy way to turn it off, for performance. В данном слу
чае речь скорее об админке, но это не кешируемая часть, а значит нам это пригодится
USE_I18N = False

# https://docs.djangoproject.com/en/1.4/ref/settings/#use-l10n
# Используется для локализации данных в местный формат. Форматирование дат с pytils все равно не сравнится.
USE_L10N = False

# https://docs.djangoproject.com/en/1.4/ref/settings/#use-tz
# Для работы с часовыми поясами
USE_TZ = False

# Тут повыключаем, то что нам не нужно. По сути это экономия на спичках.
TEMPLATE_CONTEXT_PROCESSORS = (
    # default template context processors
    "django.contrib.auth.context_processors.auth",
    # "django.core.context_processors.debug",
    # "django.core.context_processors.i18n",
    # "django.core.context_processors.media",
    "django.core.context_processors.static",
    #"django.core.context_processors.tz",
    "django.contrib.messages.context_processors.messages",
    # required by django-admin-tools
    'django.core.context_processors.request',
    'orangetrans.utils.context_processors.common',
)

# Sentry мы использовать тут не можем, берем что есть в джанге.
# include_html вышлет на почту весь трейсбек, со всеми локальными переменными в случае эксепшена.
# Во флетпейджах как правило эксепшены очень редкие гости.
# https://docs.djangoproject.com/en/1.4/topics/logging/#django.utils.log.AdminEmailHandler
LOGGING = {
…
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler',
            'include_html': True,
        }
    },
…
}

# Следующее вышлет на почту уведомления о "сломанных" ссылках
SEND_BROKEN_LINK_EMAILS = True
```

Кеш

Это самая сильная часть проекта. Кеш будет нашим щитом перед медленным бекэндом. Многие, конечно, уже догадались о чем речь.

Как это будет работать? Все просто: юзер запрашивает страницу, нжинкс смотрит есть ли эта страница в кеше, если нет, джанга ее генерит, кладет в кеш и отдает нжинксу, в следующий раз фронтенд отдаст ее самостоятельно. Т.е. по факту достаточно пройтись по сайту разок и все последующие разы наш сайт будет работать со скоростью мемкешд + нжинкс.

Правда здорово? Ну почти, этот тип кеша хорошо подходит для нашего случая, для сложного проекта он обладает множеством минусов. Например, мы не можем обновлять заполняемую юзером форму, потому-что всякий раз заходя по этому урлу юзер будет получать все ту же форму из кеша: без заполненных данных и полученных ошибок. Хвала моде нам это не понадобится :)

Django

Форма у нас будет отправляться на аяксе, нам достаточно показать ее разок, а данные и ошибки у нас будут ходить уже на js. Фоллбек вариант без аякса (вы же всегда его делаете, так? :) мы кешировать не будем. Есть еще момент, форма у нас работает через пост. Начиная с какой-то версии джанги (1.2?) для POST запросов стала обязательна защита от csrf-атак. В нашем случае это запрос обратного звонка: одно поле номера и никаких личных данных. Чтобы выключить защиту для определенной вьюхи нужно просто завернуть его в декоратор csrf_exempt.

Для осуществления нашего плана напишем middleware. В интернетах есть статья как осуществить работу такой связки. Она написана для предыдущих версий джанги и без просмотра сырцов я не разобрался. Я ее немого подправлю и обновлю. Итак в middleware:

```
import re
from django.core.cache import cache
from django.conf import settings

class NginxMemCacheMiddleWare:
    def process_response(self, request, response):
        url = request.get_full_path()

        cache_it = not settings.DEBUG \
            and request.method == 'GET' \
            and response.status_code == 200

        if cache_it:
            stoplist = [
                x for x
                in settings.CACHE_IGNORE_REGEXPS
                if re.match(x, url)
            ]
            if not stoplist:
                cache.set(url, response.content)

        return response
```

Мы кешируем только GET, не кешируем во время разработки и 404. А почему? А потому-что боты в поисках админки или сам злоумышленник могут забить память всяким хламом.

Вернемся в конфиг:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': 'unix:/tmp/memcached.sock',
        'KEY_PREFIX': 'YOURSITE',
    }
}
```

Тут все просто: бэкэнд мемкешд, сокет и префикс. Кстати, pylibmc мне не удалось завести через сокет, чего-то там ему не нравится, да и все равно. Я взял python-memcached.

```
CACHE_IGNORE_REGEXPS = (
    re.compile(r'/admin.*'),
    re.compile(r'/some_url.*'),
)
```

Это список урлов запрещенных к кешированию, о котором говорилось чуть выше. Не забудьте добавить наш новый middleware в MIDDLEWARE_CLASSES в settings.

Кстати, для мониторинга состояния мемкешд можно воспользоваться программкой django-memcache-status. Она рисует симпатичную шкалу использованной отведенной памяти для мемкешд в админке и показывает еще кучу всякой инфы. Если честно, особого применения я ей не нашел :) просто разбавит и без того скучную для флетпейджев админку. К сожалению она не дружит с django-admin-tools.

Nginx

Конфиг для дев сервера, позже мы его обновим:

```
upstream dev {
    server 127.0.0.1:8000;
}

server {
    listen 80;
    server_name dev.local;
    charset utf-8;

    location ~^/(media|static) {
        root /home/user/project;
        access_log off;
        break;
    }

    location / {
        if ($request_method = POST) {
            proxy_pass http://dev;
            break;
        }
        default_type  "text/html; charset=utf-8";
        set $memcached_key "YOURSITE:1:$request_uri";
        memcached_pass unix:/tmp/memcached.sock;
        error_page 404 502 = @fallback;
    }

    location @fallback {
        proxy_pass http://dev;
    }
}
```

Видите эту строчку: 'set $memcached_key «YOURSITE:1:$request_uri»;'? Тут и происходит вся магия. Каждая страница хранится в кеше по ключу префикс + версия кеша + полный урл. Достаточно разок открыть страницу, джанга положит ее в кеш за ключом, а нжинкс в следующий раз по этому ключу ее достанет. Даже если выключить джангу сайт будет работоспособен. Пост-запросы мы сразу передаем бэкэнду, не дергая мемкешд. Еще момент: я использую гет-ключи для фаллбек режима если у юзера отключен js (у меня noscript :P): вкладки там всякие, так вот поэтому request_uri вместо uri — последнее урл без гет-ключей.

Начиная с джанги 1.3 нам не надо формировать весь ключ самостоятельно. Достаточно передать уникальную часть в методы set(), get(), delete(), а остальное джанга сделает самостоятельно. Уникальная часть в нашем случае это абсолютный урл страницы.

С привычным нам gzip'ом есть небольшая проблема: нжинкс не сжимает, то что берет из мемкешда. Я нашел в сети патч 4-х летней давности для версии 0.6 нжинкса, а на дворе уже 1.*. Если вы не обслуживаете ие6 вы можете включить компрессию в самой джанге и уже отдавать сжатый контент. В нашем случае старпер-браузер важен. Для всего остального gzip работать будет.

Memecached

Скажем memcached работать через сокет, /etc/memcached.conf:

```
#-s unix socket path to listen on (disables network support)
-s /tmp/memcached.sock
#-a access mask for unix socket, in octal (default 0700)
-a 0777
```

Статика

Я использую jquery и html5shim для IE. Хвала гуглу, он облегчит нам участь:

```
<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
<script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
```

Остальную статику сожмем с помощью django-compressor. Я использую less — django-compressor может еще и скомпилировать его в css. Если вы тоже используете less, не забудьте поставить node.js на сервер и сам компилятор. Для всей статики включим компрессию и склеим вместе: js с js, css с css. Красота.

Приложение также дает файлам уникальные имена, так что при изменении исходников будут скомпилированы новые с новыми именами, следовательно юзер их скачает по новой и не будет задаваться вопросом «почему все выглядит так странно?». Кстати,

джанга 1.4 тоже так может.

Локальный тест

Окей, мы почти закончили.
Возьмем любимую виртуальную машину, поставим систему и развернем наш проект. Заодно все подготовим к продакшену: обновим requirements для pip, которые, почему-то, всегда неактуальны и настроим uwsgi-сервер. С процессором есть один момент: виртуалбокс и вмваре плеер не умеют отдавать определенное количество герцов, ограничимся одним ядром.

По настройке uwsgi-сервера, я рекомендую почитать доки и еще статью на welinux. Объяснять это здесь не имеет смысла: взять пример конфига из доков, да запустить. Единственно я рекомендую указывать virtualenv, потому как большая часть проблем связана с pythonpath и окружением — ох, сколько времени я убил пока не догадался глянуть в доки. Судя по гуглу: я не один такой, народ извращается как может: добавляет пути в окружение из скриптов и т.д. А всего-то старое доброе rtfm.

И так, все готово. Откроем приватный режим в браузере, чтобы все по честноку, замерим за сколько мы получим страницу:

```
18 requests  |  284.93KB transferred  |  1.11s (onload: 1.12s, DOMContentLoaded: 979ms)
```

Включим кеш, сгенерим кеш, откроем новый приватный браузер:

```
18 requests  |  298.09KB transferred  |  291ms (onload: 293ms, DOMContentLoaded: 150ms)
```

Выполним POST запрос (потом объясню зачем). В нем выполняется пару раз get_or_create:

```
21 ms. Просто запомним.
```

Как видим есть прирост, как и немного больший вес без гзипа.
Круто? Круто.

Можно круче

Вот что мы сделаем:

- начиная с версии 1.3 джанга собирает всю статику всех приложений в папку со статикой (manage.py collectstatic) — лучше не придумаешь. Положим все в память;
- бд у нас это файл и обращаться к нему приложение будет как к файлу, положим и ее в память;
- настроим копирование бд раз в час на винт, чтобы спать спокойно.

Статика это 2 мегабайта и бд больше пары мегабайт не вырастит — берем 5 Мб. Создадим директорию в памяти:

```
# Укажем в конце /etc/fstab:
tmpfs /mnt/project/ tmpfs size=5M,mode=0777 0 0

# Создадим директорию
$ sudo mkdir /mnt/project/

# Смонтируем в первый раз ручками, при ребуте директория смонтируется автоматически:
$ sudo mount /mnt/project/

# mode=0777 позволит нам делать все, что нам захочется
$ mkdir /mnt/project/static
$ mkdir /mnt/project/db
```

Если вы правильно собрали свой проект, то вся статика у вас окажется в директориях приложений. Для этого статику нужно класть в static каждого приложения, подробнее и больше в доках. Так джанга сможет их найти и собрать в нужном месте.
Делаем:

```
# В локальных настройках джанги укажем путь до директории сбора статики.
# Наверняка забудете про компрессор, ему тоже надо обновиться:
STATIC_ROOT = COMPRESS_ROOT = '/mnt/project/static/'

# В директории проекта выполним (--noinput не будет задавать вопросы, а -c потрет все лишнее в директории назначения):
$ ./manage.py collectstatic --noinput -c

# Тоже самое для бд. Укажим в настроках полный путь:
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '/mnt/project/db/db',
    }
}

# Скопируем бд
```

```
$ cp -r project/db /mnt/project/db/db

# Ребутнем проект
$ killall -HUP uwsgi
```

Автоматизацию всего этого при старте и бекап по крону оставим в качестве домашнего задания.

Обновим конфиг нжинкса:

```
upstream project {
    server unix:/tmp/project.sock;
}

server {
    listen 80;
    server_name project.ru www.project.ru;
    if ($host = www.project.ru){
        rewrite ^(.*)$ http://project.ru$1 permanent;
    }

    location /static {
        root /mnt/project/;
        access_log off;
        expires max;
    }

    location /media {
        root /project/;
        access_log off;
        expires 5d;
    }

    location / {
        include /etc/nginx/uwsgi_params;
        if ($request_method = POST) {
            uwsgi_pass project;
        }
        default_type  "text/html; charset=utf-8";
        set $memcached_key "project:1:$request_uri";
        memcached_pass unix:/tmp/memcached.sock;
        error_page 404 502 = @fallback;
    }

    location @fallback {
        include /etc/nginx/uwsgi_params;
        uwsgi_pass project;
    }
}
```

Media у нас физически остался там же где и был. Статика переехала, остальное мы уже видели.
Стоит отметить строчку 'expires max' для статики: это ведь не круто. А что если завтра мы картиночку подправим, м? Если вы используете django-compressor, вы будите приятно удивлены увидев такое в css: /static/i/button-small.png?e59177bafc62.

Без кеша:

```
14 requests  |  284.93KB transferred  |  743ms (onload: 775ms, DOMContentLoaded: 708ms)
```

С кешем:

```
14 requests  |  298.09KB transferred  |  174ms (onload: 175ms, DOMContentLoaded: 140ms)
```

POST:

```
17ms
```

Есть прирост. Небольшой, но есть. А при большей посещаемости это станет заметно.

Бекапы

Обычно я настолько расточителен, что использую Dropbox, но здесь больше подойдет WebDav. Простую и достаточно подробную инструкцию вы найдете на MyDrive. Благо сейчас сервисов полно — выбирайте. Это позволит нам сэкономить место на винте и

вынесет бекапы в облако одновременно.

Итак, что мы сделали?

- мы сэкономили ресурсов заменив mysql на sqlite (и немного их потратили положив бд в память :) однако до поры добились большой скорости;
- использовали cookie-based сессии — они у пользователя, а значит нам не стоит заботится об их хранении, чистке и прочей ерунде;
- мы оптимизировали работу джанги, в основном админки, за счет отключения перевода, но это некешируемая часть, а значит это важно;
- нжинкс, джанга и мемкешд общаются через сокеты — незнаю насколько это точно прибавит, но должно быть быстрее чем через порты;
- мы использовали, наверное, самый быстрый кеш из существующих и при правильном использовании можем дать фору в скорости сайтам покруче;
- часть статики размещена во вне, а значит сайт будет грузиться еще быстрее: из разных источников или из локального кеша (штука-то популярная);
- свою статику мы сжали, положили в память и отдаем через нжинкс;
- бекапы? Checked.

Так в чем же шутка?

Мне было интересно решить задачу, просто так, из любопытства и непривычки работать в тесных условиях. В основном, решения безвредные и даже полезные, но некоторые несут исключительно спортивный характер: выбранный кеш непрактичен, а sqlite не позволит сильно расширить проект (бэк-офис для обработки звонков, например). Да и в здравом уме экономить на сервере я не буду, так что все это не более чем фан.

Вроде, все. Спасибо за внимание.

django, django-compress, uwsgi, nginx, memcached

комментарии (50) отслеживать новые: в почте в трекере

VBart, 21 апреля 2012 в 17:34 # +19

Так делать плохо:

```
if ($host = www.project.ru){
    rewrite ^(.*)$ http://project.ru$1 permanent;
}
```

правильно:

```
server {
    listen       80;
    server_name  www.project.ru;
    return       301 http://project.ru$request_uri;
}
```

О чем написано в первых же строчках:
nginx.org/en/docs/http/converting_rewrite_rules.html

magic4x, 21 апреля 2012 в 17:41 # ↑ +7

Кхм. Вы только что привели меня к мысли: я не читал доки от начала. Искал через гугл, когда в этом была необходимость и как следствие читал «кусками». Спасибо, за правку.

 **VBart**, 21 апреля 2012 в 18:34 # ↑ **+15**

К сожалению официальная документация в настоящий момент частично содержит устаревшую информацию, но работы над актуализацией ведутся.

Поэтому, помимо nginx.org, стоит также смотреть wiki.nginx.org, но самое интересное обычно содержится в списках рассылки: http://mailman.nginx.org/mailman/listinfo. Их регулярно читают разработчики, а также не мало очень опытных пользователей, и из них можно узнать как лучше настраивать те или иные вещи, одним из первых узнавать обо всех нововведениях, выпуске новых версий и уязвимостях, которые, увы, тоже случаются. Туда также можно направлять свои предложения и пожелания по улучшению, но предварительно стоит хорошенько погуглить.

Всем, кто хочет научиться хорошо и безопасно настраивать nginx, рекомендуется к прочтению:

- How nginx processes a request
- Converting rewrite rules
- if is evil
- Pitfalls
- Server names

И обязательно документацию по таким важным директивам, как location, server_name, root, alias, try_files, error_page, map.

p.s. ещё маленькая придирка:

> Веб-сервер
>
> Это будет uwsgi — быстрый и модный.

uWSGI (uwsgi маленькими буквами — это название протокола) все-таки правильнее называть не веб-сервером, а сервером приложений (в роли веб-сервера у вас nginx), о чем сами авторы по ссылке и пишут.

 **magic4x**, 21 апреля 2012 в 18:46 # ↑ **+8**

Ваш комментарий интереснее моей статьи :)
За мной пинта или две. В общем будете у нас Петербурге..!

 **akzhan**, 21 апреля 2012 в 17:45 # **+8**

Не думаю, что при отдаче статики из временной файловой системы будет большой выигрыш.

Современные ОС очень грамотно кэшируют отдачу файлов.

То же касается и замены MySQL на SQLite. Ведь если оттюнить MySQL, вполне можно добиться отдачи всех ответов из кэша запросов.

 **magic4x**, 21 апреля 2012 в 18:06 # ↑ **0**

Выигрыш действительно небольшой. По поводу mysql: я решил что сам по себе сервис тоже требует ресурсов, и еще буквально перед публикацией, подумав я решил удалить следующее:

```
# В настройках кеша
'TIMEOUT': 0,

# В модели флетпейджа
def save(self, *args, **kwargs):
        if self.id:
            cache.delete(self.get_absolute_url())
        super(FlatPage, self).save(*args, **kwargs)
```

С таким кешированием тяжело работать, но если знать как — проблем никаких.

 **akzhan**, 21 апреля 2012 в 18:18 # ↑ **0**

Посмотрите лучше Octopress. Немного о подобных движках есть тут — Goblog: Самодельный статический движок для блога на Go.

 **Deepwalker**, 22 апреля 2012 в 00:58 # ↑ **+2**

Или если не выходить за рамки питона, то flask-freeze

 **B_dot**, 21 апреля 2012 в 21:36 # ↑ **+3**

tmpfs имеет смысл использовать когда нету контроля над кешем. Например в OpenVZ где дисковый кеш расшарен под все виртуалки и соседняя виртуалка с активным i/o может вымывать из кеша файлы вашей виртуалки.

В нормальной виртуализации (xen, kvm и подобные) это действительно не имеет особого смысла.

Что касается MySQL — будет больше жрать памяти. При таком размере базы сам mysql будет занимать больше места в ОЗУ.

 **Azy**, 21 апреля 2012 в 18:06 # **–2**

Пару мелких проектов я захостил у locum.ru. Django за 150р в месяц — это ок.

 **viperet**, 21 апреля 2012 в 18:14 # **0**

В ваш конфиг я бы добавил запрет на запросы кроме GET/POST, или $request_method в ключ кеша. А так — кто-то пошлет HEAD запрос на страницу и у вас в кеш ляжет пустая старница.

 **magic4x**, 22 апреля 2012 в 00:15 # ↑ **0**

Это?

```
cache_it = not settings.DEBUG \
    and request.method == 'GET' \
    and response.status_code == 200
```

 funca, 22 апреля 2012 в 14:53 # ↑ **+1**

тогда по логике в nginx должно быть аналогичное условие. что-то типа:

```
    location / {
         if ($request_method = GET) {
            default_type  "text/html; charset=utf-8";
            set $memcached_key "YOURSITE:1:$request_uri";
            memcached_pass unix:/tmp/memcached.sock;
            error_page 404 502 = @fallback;
            break;
        }
    try_files $uri @fallback;
    }
```

 **marazmiki**, 21 апреля 2012 в 18:26 # **+3**

Зачем катавасия с мемкешем? Если уж единовременно генерировать странички, можно сразу класть их в виде html-файлов в корень nginx'a (кстати, это уже было в Симпсонах). Заодно и память сэкономите, на ВДСах за её расходом приходится следить.

А вообще статья хорошая, конечно =)

 **magic4x**, 22 апреля 2012 в 00:14 # ↑ **0**

Во-первых мемкеш быстрее чем диск, во-вторых в кеше можно еще хранить всякие полезные штуки типа слоу-квери: менюшки, например, то что часто используется: в контекстных процессорах.

 **Deepwalker**, 22 апреля 2012 в 01:01 # ↑ **+5**

Мемкеш быстрее, чем чтение с диска. Но обычно, как уже говорилось выше, файл в кеше ОС, и получается что работа с мемкешом медленнее — вам еще и через сокеты прогнать инфу надо.

 **magic4x**, 22 апреля 2012 в 09:41 # ↑ **0**

Еще выше говорилось почему это может быть полезно.

 **Deepwalker**, 23 апреля 2012 в 00:23 # ↑ **0**

В случае одной особой виртуализации. Но в общем да, про это тоже помнить полезно. Общий вывод — под OpenVZ придется пользоваться memcache, и это будет медленнее, чем если бы у нас все было окей с обычным кешом ОС.

 **marazmiki**, 22 апреля 2012 в 08:55 # ↑ **0**

Уж сколько раз тут уже упоминалось про кеш ОС :-)

Для менюшек и прочего кеширование использовать, разумеется, можно (хоть и непонятно зачем: страницы-то целиком в запоминаются в мемкеше и при их вызове к джанге даже обращения нет). Но ради такой мелочи можно и locmem использовать.

 **niko83**, 21 апреля 2012 в 20:32 # **+8**

Увидел в загаловках «стаья — шутка» и подумал что статья безполезная х*ня, открыл тупо поржать с прикола. А в итоге: с удавольствием прочитал, увидил много полезных для себя ссылок, посмотрел предложенные варианты оптимизации, кое-что наматал на ус. Остался даволен. Спасибо.

 **gigimon**, 21 апреля 2012 в 22:16 # **+1**

За способы оптимиизации конечно большой плюс, но в целом за подход… Я рад, что вы подписали шутка :)

200 мб озу и 500 мгц, хватает на достаточно большое количество юзеров и приличный сайт (ну, если саму логику кодить прилично), поэтому проблемы такой оптимизации не совсем понял.

magic4x, 21 апреля 2012 в 23:11 # ↑ +1

200мб это много и мало — смотря как к этому подойти. Все зависит от того насколько ваше приложение отличается от «hello world». Я в работе использую кучу батареек и при этом являюсь ярым сторонником DRY: например, less вместо css, а ему нужен nodejs. Однако оптимизация — это, почти, всегда: тут убавил, там прибавил и при определенной степени «комфорта» разработки возникают определнные «трудности». Потеря в скорости, при этом, не допустима.
Если убрать мемкешд и статику из памяти сайт будет работать, но не так быстро как я могу себе это позволить.

marazmiki, 22 апреля 2012 в 00:01 # ↑ 0

Вы правы в том, что less и DRY — это хорошо. В остальном — нет :) less нужно компилировать ровно один раз, на своей девелоперской машине, при деплое. Заодно кучу css-файлов можно смержить в один, удалить лишние пробелы и гзипнуть на всякий случай.

С coffeescript проделываются аналогичные вещи: компиляция — мерж — yuicompressor… На выходе получается один минифицированный (или даже обфусцированный) js, который уже и отправляется на сервер.

Такая вот подготовленная статика молча лежит и ~~каши не просит~~ не компилируется. При желании её вообще можно на народ.ру закинуть.

Зачем Вам less на сервере? =)

magic4x, 22 апреля 2012 в 00:12 # ↑ 0

А зачем мне треш в репе? На каждый коммит удалять старые ксс/джс и добавлять новые? Временные файлы должны быть временными и я не хочу помнить о таких вещах.

VolCh, 22 апреля 2012 в 00:30 # ↑ 0

Можно не удалять, а синкать. Хотя, конечно, многое зависит от инструментов деплоя. А воспринимать css/js как временные, если им не нужна динамическая (на каждый запрос) генерация, имхо, несколько не верно. Это как считать исполняемые бинарники (pe, elf, ...) временными файлами и раздавать пользователям исходники — запускаешь калькулятор? «упс, временный экзешник удалился уже, подождите пока идёт компиляция и сборка...»

magic4x, 22 апреля 2012 в 02:32 # ↑ 0

Статика генерится раз при старте приложения, и только если исходные файлы изменены.
Компрессор самостоятельно компилирует, жмет и подставляет новые имена файлов в шаблоны. Я вообще об этом не думаю: мне только залить изменения на сервер и ребутнуть приложение.

DLag, 22 апреля 2012 в 03:08 # ↑ 0

Зачем ребутать приложение при изменении в CSS?

magic4x, 22 апреля 2012 в 22:06 # ↑ 0

Компрессор генерит цсс из лесс и записывает новые имена в шаблоны при старте приложения. Затем нужно зачистить кеш, чтобы страницы получили новые стили. Да и в ребуте приложения я ничего плохого не вижу, даже наоборот время от времени будет полезно: утечки там всякие.

akzhan, 22 апреля 2012 в 01:42 # ↑ +4

За компиляцию и доставку отвечает не система контроля версий, а система разворачивания приложений.

Обычно ассеты компилируются, пакуются и заливаются на целевой сайт без VCS. Либо компилируются на целевом хосте перед симлинком на новую версию приложения.

magic4x, 22 апреля 2012 в 02:34 # ↑ 0

Тут ответил.

funca, 22 апреля 2012 в 14:33 # ↑ 0

> Если вы тоже используете less, не забудьте поставить node.js на сервер и сам компилятор

есть куча доводов против того, чтобы тащить средства разработки на продакшен.
а в целом, спасибо за статью, очень познавательно.

marazmiki, 22 апреля 2012 в 08:48 # ↑ +1

Действительно, зачем треш в репе? У меня вообще папка public/ (в ней media и static/ хранятся) находится в .gitignore и создаётся при развёртывании динамически.

Так что тут тоже полностью согласен: трешу в репозитории не место :-)

Deepwalker, 22 апреля 2012 в 01:06 # 0

Я тоже как-то раз так загнался — хотелось платить за сайт по-минимуму. В итоге сделал генерацию статики, и один активный скрипт на cgi для отправки заказа (магазин). Кидаю готовый контент на GAE, проект мелкий, пока лимиты не выбирает. Корзина в куках. Последний раз переделал на flask-freeze.

NeleGALL, 22 апреля 2012 в 06:29 # +1

Лично для меня содержание поста (пока) не применимо, но перед вашей манерой письма снимаю шляпу. Читается на одном дыхании.

MechanisM, 22 апреля 2012 в 10:12 # +4

Я бы на вашем месте разместил всю статику на Github Pages. Там шустрые nginx и все сжато и отлично подходит в качестве личного мини-CDN. Просто с каждой версией компилить css из less, js из coffeescript итд а потом коммитить в репозиторий и не забыть положить в репозиторий файлик CNAME где прописать например static.site.com и все — ваша статика не занимает на сервере место и не напрягает ваш nginx лишний раз. Все раздается из облака в Rackspace Cloud(именно там располагаются сервера Github Pages). А еще можно вообще спрятать весь сайт за крепкими плечами cloudflare.com

MechanisM, 22 апреля 2012 в 10:19 # ↑ +2

А вообще мне кажется в вашей ситуации varnish бы сделал больше чем делает memcached.

magic4x, 22 апреля 2012 в 22:10 # ↑ 0

Да, я в курсе под гитхаб :)

> Просто с каждой версией компилить css из less, js из coffeescript итд а потом коммитить в репозиторий и не забыть положить в репозиторий файлик CNAME где прописать например static.site.com и все

«Просто… и все» — вы мне мое начальство напоминаете :) То, что вы описали нисколечки не DRY, но как один из способов оптимизации — гуд. Жирный плюс вам.

magic4x, 22 апреля 2012 в 22:16 # ↑ 0

Тьфу, *про гитхаб
Да еще и пробелище такой влепил после цитаты. Уважаемый, Хабр, почему вы не трете лишние br, м?

MechanisM, 23 апреля 2012 в 09:15 # ↑ +1

Ну в том-же Twitter Bootstrap лежит файлик Makefile можно посмотреть как он сделан и что он делает и сделать нечто подобное для своего проекта и добавить там еще пару строчек чтобы все коммитилось в репозиторий. Чем не DRY. А файлик CNAME 1 раз только положить надо ну и создать CNAME или A DNS-запись.
Кстати раньше часто пользовал django-compressor а теперь перешел на django-pipeline или jingo-minify.

Mox, 22 апреля 2012 в 13:07 # –2

Я бы просто на локум задешево положил проект :)

Azy, 22 апреля 2012 в 15:20 # ↑ –1

Меня выше заминусовали за локум (

xSkyFoXx, 22 апреля 2012 в 13:27 # 0

Если я правильно Вас понял, основная цель состояла в том, что минимизировать расходы на проект.
В последнее время для маленьких или тестовых проектов я начал использовать google app engine. Получается бесплатно (квоты).
Кроме того, нет необходимости разбираться с хостингом, а разворачивание проекта в боевой режим сводится к нажатию кнопки «Deploy».
Если вы хотите использовать MySQL — Cloud SQL к вашим услугам. При этом для запросов из app engine там нет ограничений.
Ваша статья понравилась! Спасибо.

achekalin, 22 апреля 2012 в 20:48 # 0

Мне очень понравилась подводка к основному тексту: сначала рассказчик не знает ничего, и держит проект на php на shared-хостинге, затем (с

точки зрения текста — внезапно) раскачивает свой skill, и легким движением руки настраивает Django на VPS. :)

Это я не в критику — просто побольше бы таких историй успеха. Уж больно большой прыжок в знаниях и в подходе нам показали. Интересно, а в жизни этот прыжок сколько времени занял?

magic4x, 22 апреля 2012 в 21:57 # ↑ +1

В январе 2010 я сделал свою первую верстку: и сразу интернет-портал (дивы, ие6 и более). У нас были проблемы с программистом, мне часто приходилось лезть в шаблоны mako (это я как-то еще понимал).

В августе 2010 я начал изучать джангу: читал книжку в свободное время («Django. Разработка веб-приложений на Python»). Где-то в октябре до меня дошло, что я начал не с того конца и взялся за Лутца («изучаем питон»). Дальше я работал с готовым проектом (мне его оставили в наследство) по немногу увеличивая скилы.

Весной 2011 я сделал первый проект самостоятельно от дизайна до кода и развертки: интернет-магазин. Потом еще один, но уже сложнее и больше. С августа 2011 я встрял в один большой проект вторым программистом, где первый — чистой воды гений: подобный тимлид все равно что допинг. До этого августа уже становилось скучно: одно и тоже целыми днями. В ноябре проект развалился и я потихоньку стал киснуть. В декабре 2011 это скука просто била по мозгу и я начал делать свой маленький проект, который забросил в феврале в долгий ящик, теперь наверное придется все переписывать :(В феврале мы с супругой решили попробовать свои силы в своем деле, 5 апреля мы гордо сказали «понеслася!» и удалили авторизация с нжинкс :) На основе полученного опыта при его разработке я и написал статью. За кулисами осталось еще много интересного, но в основном по верстке. Возможно я их как-нибудь соберу в отдельную статью: ведь немало нового можно узнать из комментариев, как, например, тут.

2010 и по сей день: 2,5 года выходит. Прыжок может и большой, но пробелов в знаниях, хватает. Вот и вся «история успеха» :)

achekalin, 22 апреля 2012 в 22:14 # ↑ 0

Промашка вышла, не туда ответил :)

VolCh, 23 апреля 2012 в 00:36 # ↑ +1

Большая часть касается не джанги и не python. Если задаться вопросом оптимизации php проекта то всё почти также будет, разве что вместо uwsgi php-fpm ставить или phpdaemon в режиме http-сервер (не уверен, что есть смысл) или true fastcgi

magic4x, 23 апреля 2012 в 11:14 # ↑ 0

Я сначала так и написал, затем удалил вместе с остальным «объемом», потому-как статья стала ну совсем большой. И справедливо решил: раз в блоге «веб-разработка», кто захочет — посмотрит.

Да, большая часть — идеи без зависимостей.

achekalin, 22 апреля 2012 в 22:13 # 0

Серьезно, и с искренним уважением: настоящая success story. Побольше бы таких!

А статью, скажу за всех — ЖДЕМ-с!

seriyPS, 23 апреля 2012 в 00:39 # 0

А кеширование Nginx вам не подходит? wiki.nginx.org/HttpUwsgiModule#uwsgi_cache ИМХО гораздо проще организовать.

hardtop, 23 апреля 2012 в 09:55 # 0

Хорошо написано. Но если надо кешировать сайт при малых мощностях или большой нагрузке — есть отличный StaticGenerator Pro, написанный Alrond-ом. При первом обращении к странице генерится html-файл прямо в папку — откуда потом nginx берет напрямую, минуя джангу.

Внезапно: спутник Электро-Л снял затмение по нашей просьбе

Нужны ли программисту бесплатные плюшки?

Разбор задач финала чемпионата мира про программированию ACM ICPC 2013