

КАК СТАТЬ АВТОРОМ



Выгорание в IT Обучение ИБ: что с ним, на ваш взгляд, н...

ANGIE VBart

11 июн 2015 в 18:52

NGINX изнутри: рожден для производительности и масштабирования

🕒 8 мин 🏃 144K

Высокая производительность*

Перевод

Автор оригинала: Owen Garrett



NGINX вполне заслуженно является одним из лучших по производительности серверов, и всё это благодаря его внутреннему устройству. В то время, как многие веб-серверы и серверы приложений используют простую многопоточную модель, NGINX выделяется из общей массы своей нетривиальной событийной архитектурой, которая позволяет ему с легкостью масштабироваться до сотен тысяч параллельных соединений.

Инфографика *Inside NGINX* сверху вниз проведет вас по азам устройства процессов к иллюстрации того, как NGINX обрабатывает множество соединений в одном процессе. Данная статья рассмотрит всё это чуть более детально.

Подготавливаем сцену: модель NGINX процессов



Чтобы лучше представлять устройство, сперва необходимо понять как NGINX запускается. У NGINX есть один мастер-процесс (который от имени суперпользователя выполняет такие операции, как чтение конфигурации и открытие портов), а также некоторое количество рабочих и вспомогательных процессов.

```
# service nginx restart
* Restarting nginx
# ps -ef --forest | grep nginx
root      32475      1  0 13:36 ?          00:00:00 nginx: master process /usr/sbin/nginx
                                                -c /etc/nginx/nginx.conf
nginx     32476  32475  0 13:36 ?          00:00:00 \_ nginx: worker process
nginx     32477  32475  0 13:36 ?          00:00:00 \_ nginx: worker process
nginx     32479  32475  0 13:36 ?          00:00:00 \_ nginx: worker process
nginx     32480  32475  0 13:36 ?          00:00:00 \_ nginx: worker process
nginx     32481  32475  0 13:36 ?          00:00:00 \_ nginx: cache manager process
nginx     32482  32475  0 13:36 ?          00:00:00 \_ nginx: cache loader process
```

На 4-х ядерном сервере мастер-процесс NGINX создает 4 рабочих процесса и пару вспомогательных кэш-процессов, которые управляют содержимым кэша на жестком диске.

Почему архитектура всё же важна?

Одна из фундаментальных основ любого Unix-приложения – это процесс или поток (с точки зрения ядра Linux процессы и потоки практически одно и то же – вся разница в разделении

адресного пространства). Процесс или поток – это самодостаточный набор инструкций, который операционная система может запланировать для выполнения на ядре процессора. Большинство сложных приложений параллельно запускают множество процессов или потоков по двум причинам:

- Чтобы одновременно задействовать больше вычислительных ядер;
- Процессы и потоки позволяют проще выполнять параллельные операции (например обрабатывать множество соединений одновременно).

Процессы и потоки сами по себе расходуют дополнительные ресурсы. Каждый такой процесс или поток потребляет некоторое количество памяти, а кроме того они постоянно подменяют друг друга на процессоре (т. н. переключение контекста). Современные серверы могут справляться с сотнями активных процессов и потоков, но производительность сильно страдает, как только заканчивается память или огромное количество операций ввода-вывода приводит к слишком частой смене контекста.

Наиболее типичный подход к построению сетевого приложения – это выделять для каждого соединения отдельный процесс или поток. Такая архитектура проста для понимания и легка в реализации, но при этом плохо масштабируется когда приложению приходится работать с тысячами соединений одновременно.

Как же работает NGINX?

NGINX использует модель с фиксированным числом процессов, которая наиболее эффективно задействует доступные ресурсы системы:

- Единственный мастер-процесс выполняет операции, которые требуют повышенных прав, такие, как чтение конфигурации и открытие портов, а затем порождает небольшое число дочерних процессов (следующие три типа).
- Загрузчик кэша запускается на старте чтобы загрузить данные кэша, расположенные на диске, в оперативную память, а затем завершается. Его работа спланирована так, чтобы не потреблять много ресурсов.
- Кэш-менеджер просыпается периодически и удаляет объекты кэша с жесткого диска, чтобы поддерживать его объем в рамках заданного ограничения.
- Рабочие процессы выполняют всю работу. Они обрабатывают сетевые соединения, читают данные с диска и пишут на диск, общаясь с бэкенд-серверами.

Документация NGINX рекомендует в большинстве случаев настраивать число рабочих

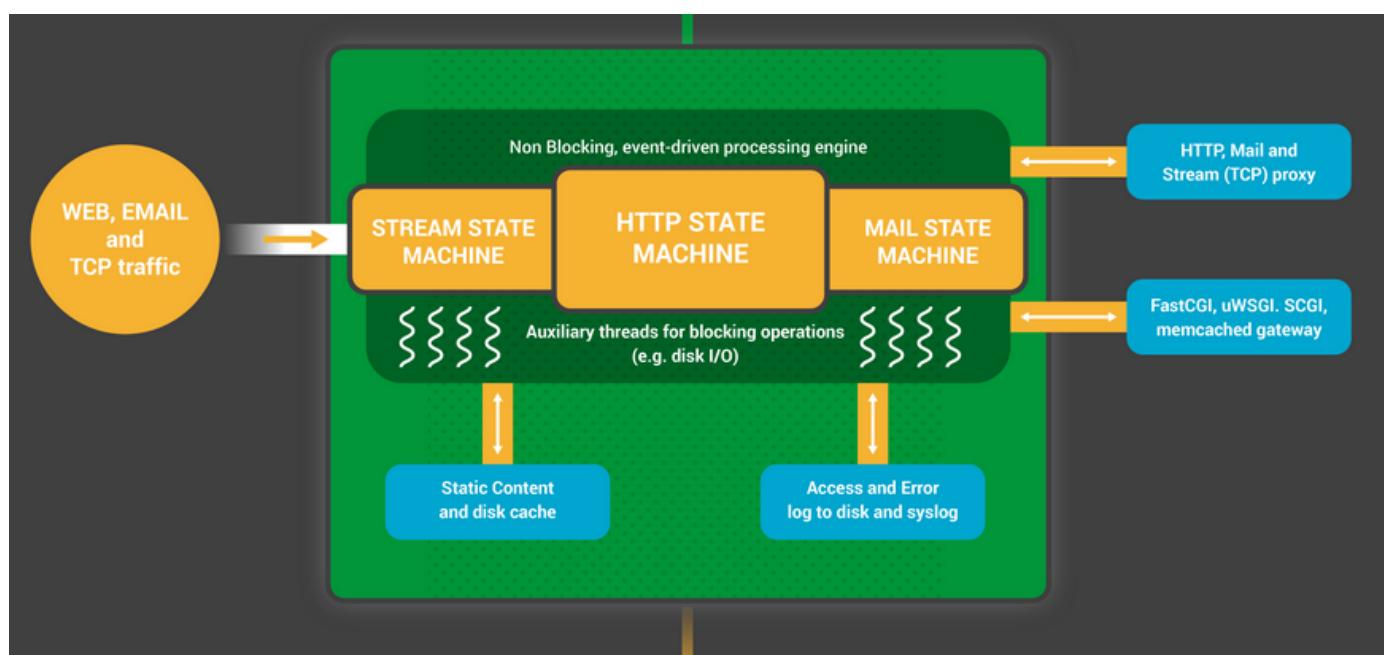
процессов равное количеству ядер процессора, что позволяет использовать системные ресурсы максимально эффективно. Вы можете задать такой режим с помощью директивы `worker_processes auto` в конфигурационном файле:

```
worker_processes auto;
```

Когда NGINX находится под нагрузкой, то в основном заняты рабочие процессы. Каждый из них обрабатывает множество соединений в неблокирующем режиме, минимизируя количество переключений контекста.

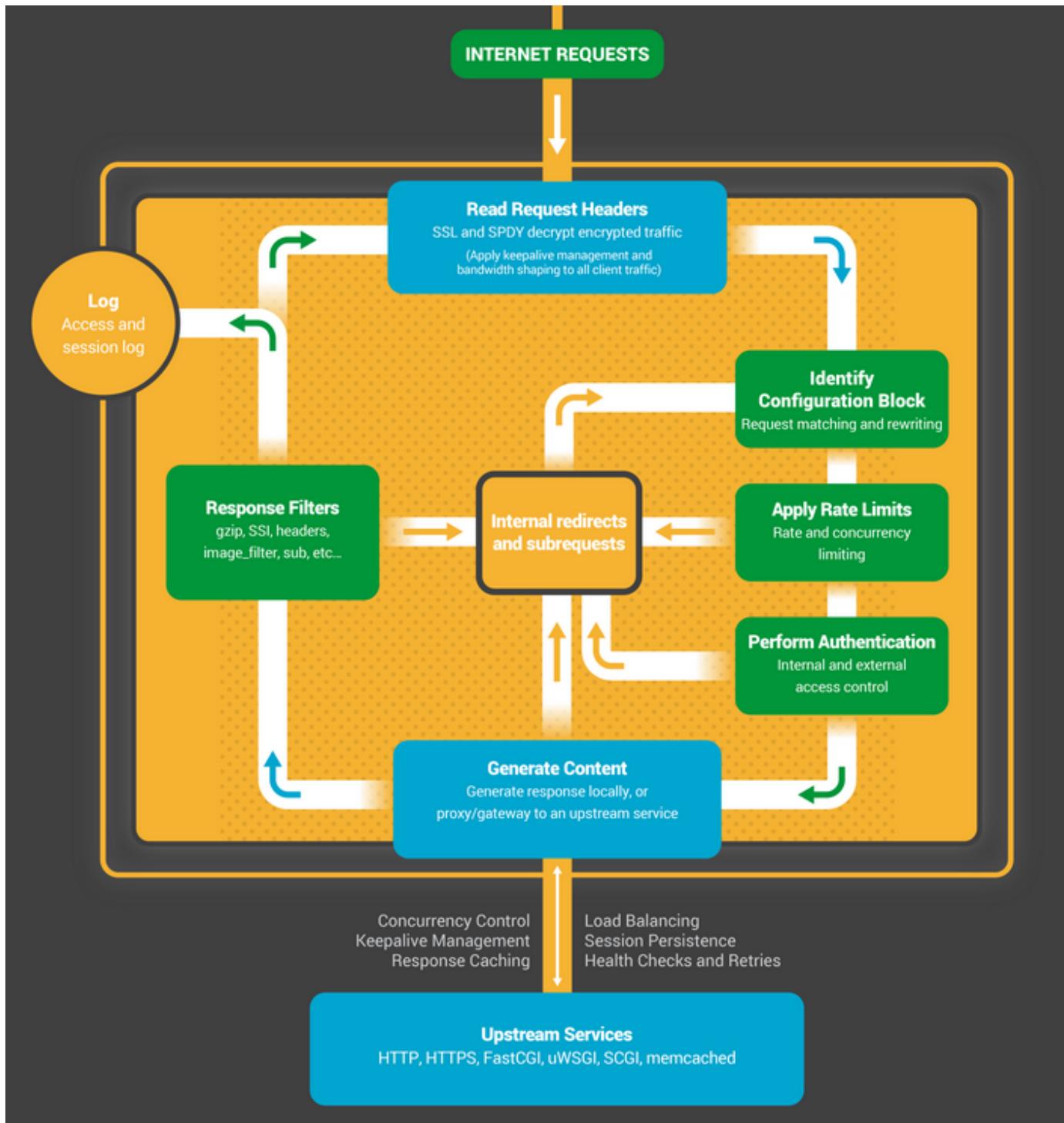
Каждый рабочий процесс однопоточен и работает независимо, принимая новые соединения и обрабатывая их. Процессы взаимодействуют друг с другом используя разделяемую память для данных кэша, сессий и других общих ресурсов.

Внутри рабочего процесса



Каждый рабочий процесс NGINX инициализируется с заданной конфигурацией и набором слушающих сокетов, унаследованных от мастер-процесса.

Рабочие процессы начинают с ожидания событий на слушающих сокетах (см. также `accept_mutex` и разделяемые сокеты). События извещают о новых соединениях. Эти соединения попадают в конечный автомат – наиболее часто используемый предназначен для обработки HTTP, но NGINX также содержит конечные автоматы для обработки потоков TCP трафика (модуль `stream`) и целого ряда протоколов электронной почты (SMTP, IMAP и POP3).



Конечный автомат в NGINX по своей сути является набором инструкций для обработки запроса. Большинство веб-серверов выполняют такую же функцию, но разница кроется в реализации.

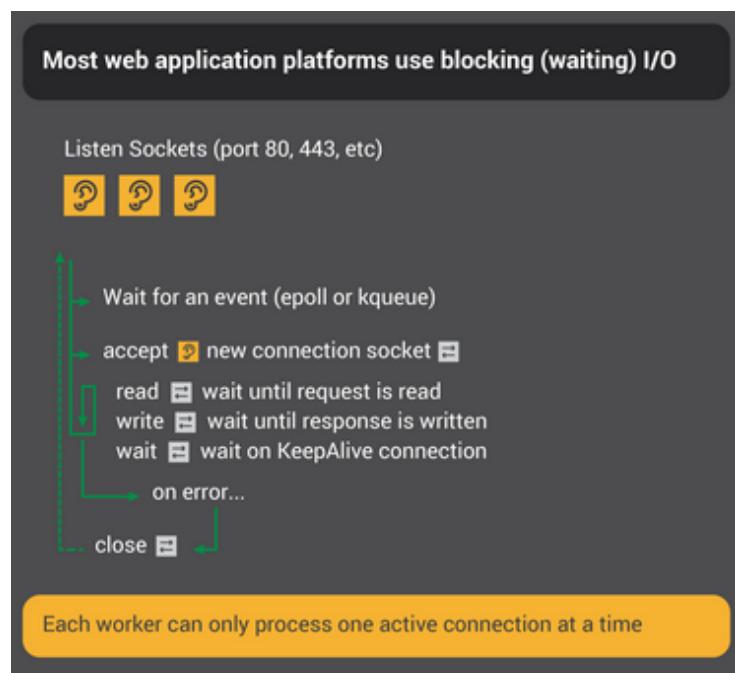
Устройство конечного автомата

Конечный автомат можно представить себе в виде правил для игры в шахматы. Каждая HTTP транзакция – это шахматная партия. С одной стороны шахматной доски веб-сервер – гроссмейстер, который принимает решения очень быстро. На другой стороне – удаленный клиент, браузер, который запрашивает сайт или приложение по относительно медленной сети.

Как бы то ни было, правила игры могут быть очень сложными. Например, веб-серверу может потребоваться взаимодействовать с другими ресурсами (проксировать запросы на бэкенд) или обращаться к серверу аутентификации. Сторонние модули способны ещё сильнее усложнить обработку.

Блокирующийся конечный автомат

Вспомните наше определение процесса или потока, как самодостаточного набора инструкций, выполнение которых операционная система может назначать на конкретное ядро процессора. Большинство веб-серверов и веб-приложений используют модель, в которой для «игры в шахматы» приходится по одному процессу или потоку на соединение. Каждый процесс или поток содержит инструкции, чтобы сыграть одну партию до конца. Все это время процесс, выполняясь на сервере, проводит большую часть времени заблокированным в ожидании следующего хода от клиента.



- Процесс веб-сервера ожидает новых соединений (новых партий инициированных клиентами) на слушающих сокетах.
- Получив новое соединение, он играет партию, блокируясь после каждого хода в ожидании ответа от клиента.
- Когда партия сыграна, процесс веб-сервера может находиться в ожидании желания клиента начать следующую партию (это соответствует долгоживущим keepalive-соединениям). Если соединение закрыто (клиент ушел или наступил таймаут), процесс возвращается к встрече новых клиентов на слушающих сокетах.

Важный момент, который стоит отметить, заключается в том, что каждое активное HTTP-соединение (каждая партия) требует отдельного процесса или потока (гроссмейстера).

Такая архитектура проста и легко расширяема с помощью сторонних модулей (новых «правил»). Однако, в ней существует огромный дисбаланс: достаточно легкое HTTP-соединение, представленное в виде файлового дескриптора и небольшого объема памяти, соотносится с отдельным процессом или потоком, достаточно тяжелым объектом в операционной системе. Это удобно для программирования, но весьма расточительно.

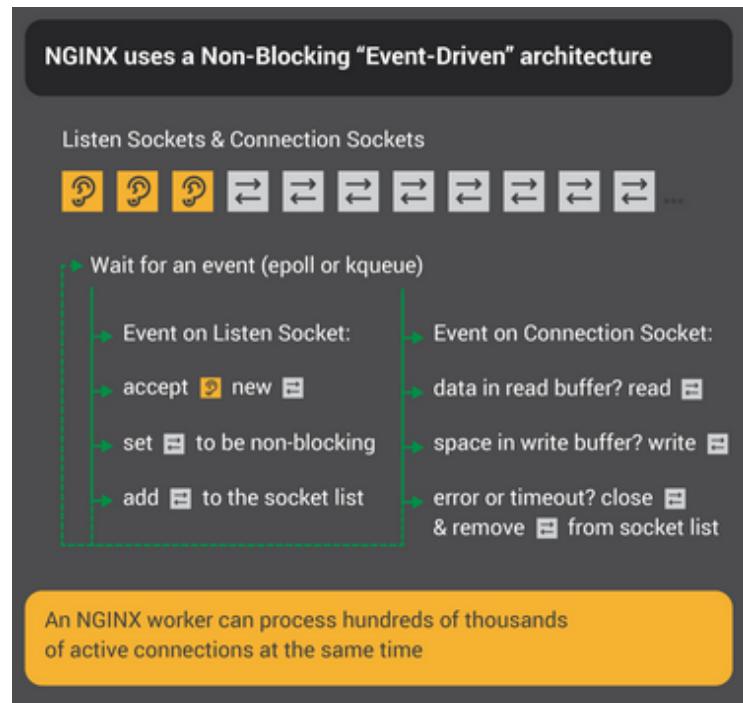
NGINX, как настоящий Гроссмейстер

Вероятно вы слышали о сеансах одновременной игры, когда один гроссмейстер играет на множестве шахматных полей сразу с десятками противников?



Кирил Георгиев на турнире в Болгарии сыграл параллельно 360 партий. Его итоговый результат составил: 284 победы, 70 вничью и 6 поражений.

Таким же образом рабочий процесс NGINX «играет в шахматы». Каждый рабочий процесс (помните – обычно всего один на вычислительное ядро) является гроссмейстером, способным играть сотни (а на самом деле сотни тысяч) партий одновременно.



1. Рабочий процесс ожидает событий на слушающих сокетах и сокетах соединений.
2. На сокетах происходят события и процесс их обрабатывает:

- Событие на слушающем сокете означает, что пришел новый клиент для начала игры. Рабочий процесс создает новый сокет соединения.
- Событие на сокете соединений сигнализирует, что клиент сделал ход. Рабочий процесс ему мгновенно отвечает.

Рабочий процесс, обрабатывая сетевой трафик, никогда не блокируется, ожидая очередного хода от оппонента (клиента). После того как процесс сделал свой ход, он немедленно переходит к другим доскам, на которых игроки ожидают хода, или встречает новых у двери.

Почему так получается быстрее, чем блокирующаяся многопоточная архитектура?

Каждое новое соединение создает файловый дескриптор и потребляет небольшой объем памяти в рабочем процессе. Это очень малые накладные расходы на соединение. Процессы NGINX могут оставаться привязанными к конкретным ядрам процессора. Переключения контекста происходят достаточно редко и в основном когда не осталось больше работы.

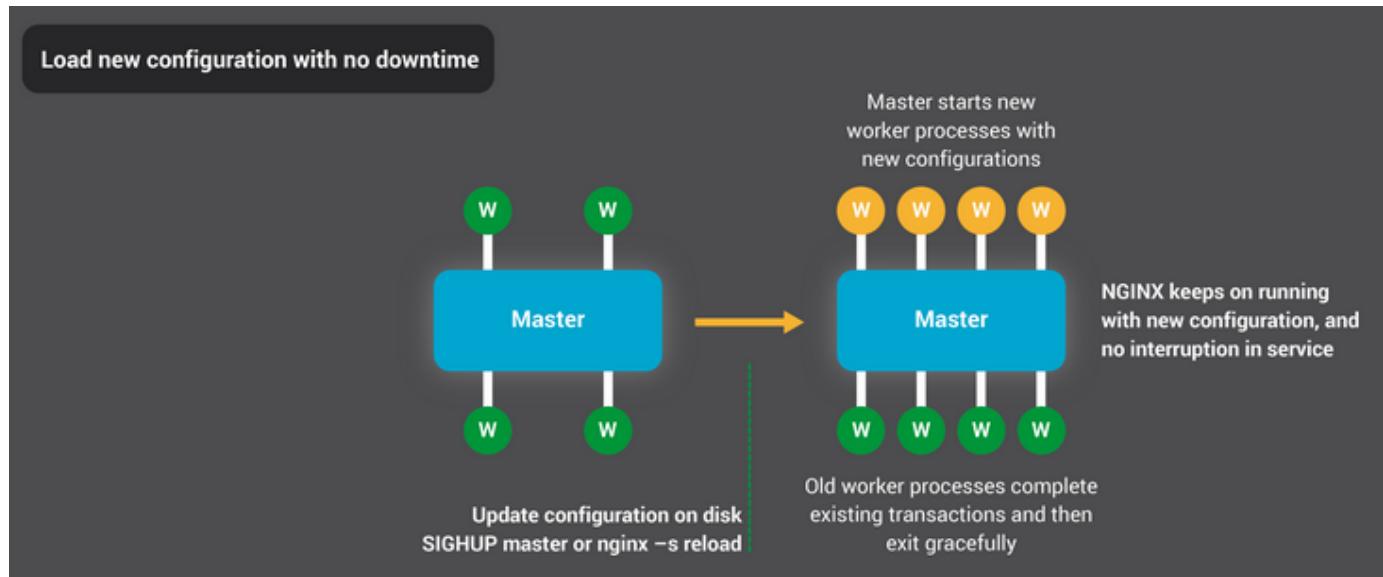
В блокирующемся подходе, с отдельным процессом на каждое соединение, требуется сравнительно большой объем дополнительных ресурсов, и переключения контекста с одного процесса на другой происходят гораздо чаще.

Дополнительную информацию по теме можно также узнать из статьи об архитектуре NGINX от Андрея Алексеева, вице-президента по развитию и сооснователя компании NGINX, Inc.

С адекватной настройкой системы, NGINX прекрасно масштабируется до многих сотен тысяч параллельных HTTP соединений на каждый рабочий процесс и уверенно поглощает всплески трафика (толпы новых игроков).

Обновление конфигурации и исполняемого кода

Архитектура NGINX с малым количеством рабочих процессов позволяет достаточно эффективно обновлять конфигурацию и даже его собственный исполняемый код на лету.



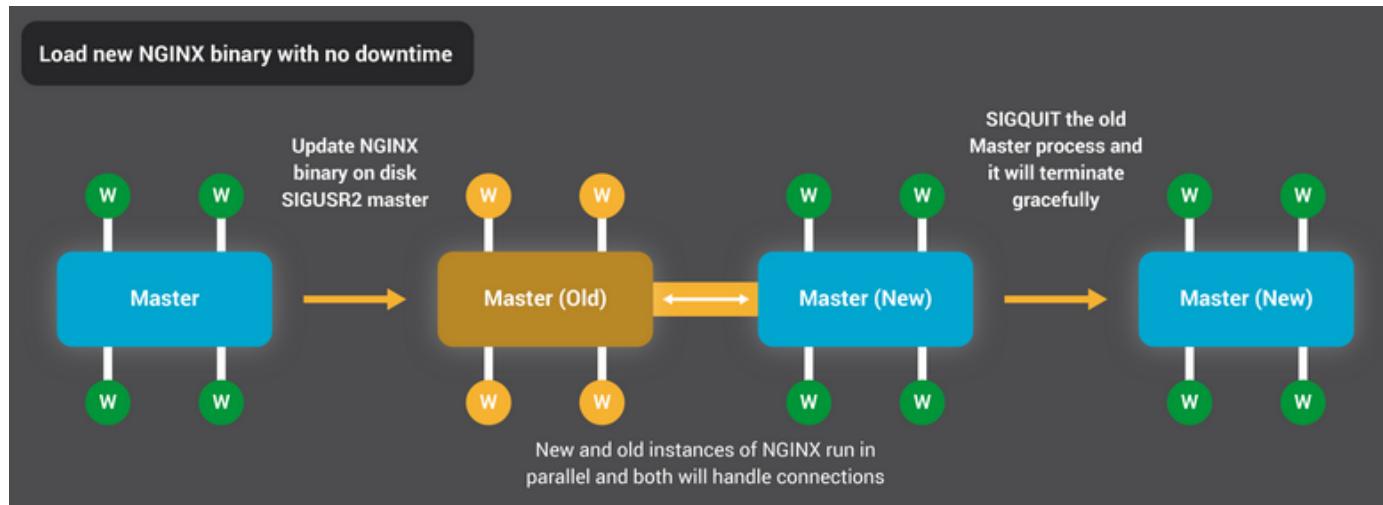
Обновление конфигурации NGINX – очень простая, легковесная и надежная процедура. Она заключается в простой отправке мастер-процессу сигнала SIGHUP.

Когда рабочий процесс получает SIGHUP, он производит несколько операций:

1. Перезагружает конфигурацию и порождает новый набор рабочих процессов. Эти новые рабочие процессы сразу начинают принимать соединения и обрабатывать трафик (используя новые настройки).
2. Сигнализирует старые рабочие процессы о плавном завершении. Они перестают принимать новые соединения. Как только завершается обработка текущих HTTP-запросов, соединения закрываются (никаких затянувшихся keep-alive соединений). Как только все соединения закрыты, рабочий процесс завершается.

Данная процедура может вызвать небольшой всплеск нагрузки на процессор и память, но в общем это практически незаметно на фоне затрат на обработку активных соединений. Вы можете перезагружать конфигурацию несколько раз в секунду (и есть немало пользователей NGINX, кто так делает). В редких случаях могут возникнуть проблемы, когда слишком много поколений рабочих процессов NGINX ожидают закрытия соединений, но они быстро разрешаются.

Обновление исполняемого кода NGINX – это Святой Грааль высокой доступности сервисов. Вы можете обновлять сервер на лету, без потери соединений, простоя ресурсов и каких-либо перерывов в обслуживании клиентов.



Процесс обновления исполняемого кода использует схожий с перезагрузкой конфигурации подход. Новый мастер-процесс NGINX запускается параллельно со старым и получает от него дескрипторы слушающих сокетов. Оба процесса загружены и их рабочие процессы обрабатывают трафик. Затем вы можете отдать команду старому мастер-процессу на плавное завершение.

Вся процедура подробно описана в документации.

Подведем итоги

Мы дали поверхностный обзор того, как функционирует NGINX. Под этими простыми описаниями скрывается более чем десятилетний опыт разработки и оптимизации, который позволяет NGINX демонстрировать выдающуюся производительность на широком спектре оборудования и реальных задачах, оставаясь надежным и безопасным, как того требуют современные веб-приложения.

Если вы хотите узнать больше по данной теме, то рекомендуем для ознакомления:

- [Installing and Tuning NGINX for Performance](#) (вебинар; слайды на Speaker Deck)
- [Tuning NGINX for Performance](#)
- [The Architecture of Open Source Applications – NGINX](#)
- [Множественные сокеты в NGINX 1.9.1 \(использование опции SO_REUSEPORT\)](#)

Теги: nginx, архитектура, высокая производительность, масштабируемость, внутреннее устройство

Хабы: Высокая производительность

ANGIE 281 0
Карма Рейтинг

Валентин Бартенев @VBart

Руководитель разработки ООО «Веб-Сервер»

Подписаться



Сайт Github

Реклама

РЕКЛАМА • RUVD.S.HABR.IO

пришло время
убраться
на орбиту на орбите

GRU VDS

Комментарии 32



• **пскма** 11 июня 2015 в 15:45



Для меня это была интересная статья.

Хотя бы теперь понятна принципиальная разница между apache и nginx.

Но у меня другой вопрос. А что если сделать аппаратный http сервер? Можно например засшить логику http сервера в ПЛИС. Я пытался так сделать и даже моя простейшая модель http сервера когда-то заработала:

marsohod.org/index.php/projects/marsohod2/290-websrv

Но как мне узнать вообще такие идеи кому интересны или нет?

Я знаю, что мир сейчас наоборотдвигается в сторону виртуализации и всяких SDN (Software Defined Network), но может и чисто аппаратные решения имеют свои плюсы?

По крайней мере сделать в ПЛИС/FPGA web сервер, который будет отдавать статику по HTTP – я не думаю что это сильно сложно.

+6 Ответить



• **maksqwe** 11 июня 2015 в 16:01



Думаю, тот же высокочастотный трейдинг? Где под конкретные протоколы делаются свои FPGA железки.

К примеру, хотя не совсем то: www.es.ele.tue.nl/mamps/web_server/files/report.pdf

+1 Ответить

...

 zup
11 июн 2015 в 16:45

так ведь есть подобные решения на рынке – f5 networks ltm, citrix netscaler. в том числе и для обеспечения low latency, что критично для трейдеров, но там сильно ограничены возможности анализа/модификации пейлоада в угоду большей производительности.

+1 Ответить

...

• НЛО прилетело и опубликовало эту надпись здесь

 lexoge
11 июн 2015 в 17:05

А какие плюсы у аппаратного решения, кроме скорости?

Имхо, у аппаратного решения много минусов:

- * подбор кадров (сколько в мире специалистов по pgsql, и сколько специалистов по этому решению?)
- * нет стандартной реализации (нужно будет сделать и протестировать 1U железку)
- * массовая замена железа (вместо стандартного сервера нужно будет ставить эту железку)
- * соответствие по функционалу (нужно, чтобы аппаратное решение поддерживало весь требуемый функционал)
- * объем настройки (нужно переделать все конфиги)
- * сложность при правке конфига (перепрошивать железку?)
- * сложность обновления (достаточно будет перепрошить железку, или нужно новую железку?)

Поэтому (опять же, имхо) «скорость» станет существенным фактором, если прирост будет не на 15-30%, а в 5-10 раз или больше.

Т.е. оно должно финансово переплюнуть все затраты на переход.

А ведь железо дешевеет, т.е. стандартный подход все дешевеет и дешевеет, а следовательно «профит» от железки должен быть все больше и больше, чтобы оправдать себя.

Лично мне не кажется, что будет большой плюс в скорости.

Сами по себе *pgsql добавляют очень маленькие накладные расходы, т.е. очень эффективно используют аппаратные ресурсы.

Вот если задействовать ресурсы GPU…

Но и при этом nginx – это же не просто отдача статики.
Это ещё куча логики + куча доп. модулей.
Чего стоит обработка запросов с использованием тар и регулярок.
Ещё всякие fastcgi, uwsgi и прочие *gi, без которых nginx теряет половину привлекательности.
А у многих ещё прикручены memcache/redis/lua/perl.
Если все это реализовывать в ПЛИС, то сколько будет стоить разработка?

0 Ответить



- НЛО прилетело и опубликовало эту надпись здесь

lexoge 11 июн 2015 в 17:21

Я в этом не специалист, но предположу, что там другой процессор.
А значит, его не следует сравнивать с x86/x64, как мы сравниваем i3 и i5.
Например, вспомним bitcoin.
Сравните Mhash/s у CPU и FPGA.
en.bitcoin.it/wiki/Non-specialized_hardware_comparison#CPUs.2FAPUs
en.bitcoin.it/wiki/Mining_hardware_comparison

1 Ответить



- НЛО прилетело и опубликовало эту надпись здесь

zup 11 июн 2015 в 17:30

Вот в целом вы правильно пишите, но тоже надо понимать, что вендоры прилагают массу усилий, чтобы создать некую экосистему вокруг своих продуктов.

я по работе часто сталкиваюсь и с «аппаратными» контроллерами доставки приложений. это целый сегмент рынка.
они жутко программируемы, в них используются криptoакселераторы и всяческие аппаратные прибамбасы для ускорения и оптимизации.

и каждый вендор зомбирует своими преимуществами – организуют тренинги и обучение, проводят вебинары и презентации для заказчиков.
у таких компаний есть свои клиенты и куча заказов, потому что не везде есть годные специалисты, способные обеспечить доступность приложений на кластере nginx, а после ухода из компании не каждый новый специалист будет способен всю эту инфраструктуру поддерживать.

а с таким вот «аппаратным» контроллером вы просто посыпаете человека на курсы, он читает подробную документацию, которой сопровождается оборудование и без особых проблем инфраструктура продолжает жить.

чаще всего решает цена

0 Ответить



lexoge 11 июня 2015 в 17:35

Согласен.
Но для России, насколько я знаю, цена на такие аппаратные решения очень кусается.
А с падением курса рубля она стала кусаться только сильнее.
Т.е. сейчас стоимость специалистов упала по сравнению со стоимостью оборудования
(в том числе и такого).
Могу предположить, что в других странах ситуация противоположна – оборудование
дешевеет, специалисты дорожают.
И там выбор в сторону аппаратных решений реальнее.

0 Ответить



ANGIE VBart 11 июня 2015 в 17:39

Она кусается не только для России. Многие клиенты с удовольствием отмечают, что NGINX Plus с поддержкой им обошелся буквально на порядок дешевле.

+1 Ответить



lexoge 11 июня 2015 в 17:32

К своему ответу ещё хочу добавить, что сейчас сервер на nginx может отдавать, например 80 Гбит/с с себя.
А не с себя (т.е. просто проксировать) и того больше.
Поэтому нет вопроса «как бы это ускорить?», скорее стоит вопрос «чем все это нагрузить?».

0 Ответить



robert_augapetyan 11 июня 2015 в 17:51

Какой самый эффективный способ прицепить свои обработчики на C/C++ к nginx?
Если через написание модуля – то планируется ли какая-либо вменяемая документация API, кроме известной статьи 10-летней давности?
Если через fcgi – насколько данное решение будет уступать модулю по производительности?

0 Ответить



НЛО прилетело и опубликовало эту надпись здесь

 V Bart

11 июня 2015 в 18:06



*CGI/HTTP должно хватить для большинства задач. Едва ли вы упретесь в протокол, зато получится гибко, стабильно и это будет на порядок легче поддерживать.

Планы были, но пока отсутствует понятный, формализованный, стабильный и безопасный API для сторонних модулей, любая такая документация будет неполноценной и быстро устаревать. На данный момент самая вменяемая документация – исходный код.

 0 Ответить evnuih
11 июня 2015 в 18:25

Наверное, если делать API для модулей, то совсем без копирования в памяти, а иначе-то оно и по fcgi неплохо.

 0 Ответить robert_аугаретян
12 июня 2015 в 00:30

Какой из зоопарка решений (<https://blog.inventic.eu/2013/11/fastcgi-c-library-for-all-platforms-windows-mac-and-linux/>) порекомендуете? Нужно, чтобы была мультиплатформенная обработка асинхр. событий (не хардкоднутый select) и желательно не Boost.ASIO. Есть такие?

 0 Ответить V Bart
12 июня 2015 в 00:59

Если вопрос ко мне, то не могу ничего порекомендовать, поскольку нет такого опыта, не пользовался. Вообще протокол в самом базовом виде достаточно простой и можно свое написать за день.

И мне кажется обработкой событий не fastcgi библиотека должна заниматься.

 0 Ответить robert_аугаретян
12 июня 2015 в 01:02

Для чего-то он там есть (встречал как раз Boost.ASIO и select)…

 0 Ответить Mercury13
11 июня 2015 в 22:25

Хорошо, есть вопрос.

Допустим, я хочу сделать игровой сервер (несколько игроков в общем мире). Игра

нединамичная и потому протокол TCP. Как лучше реализовать обмен, чтобы потоков было поменьше?

 0 Ответить  

o  **VBart**
ANGIE 11 июня 2015 в 22:30

Так же, как это делает nginx, асинхронно обрабатывая сокеты в неблокирующемся режиме.

 0 Ответить  

o  **Megsugy13**
11 июня 2015 в 22:38

В моей архитектуре было два потока на игрока: один занимается отправкой, второй – приёмом. Почему два? А потому, что сообщение с игрока A, после того, как обрабатывается, будет разослано всем. Но всё равно многовато. Может, от второго потока удастся избавиться асинхронным вводом-выводом? – всё равно разбираться и разбираться.

Потому я и написал это в статью про NGINX, что у него есть чему научиться. А вот с чего начать?

 0 Ответить  

o НЛО прилетело и опубликовало эту надпись здесь

o  **cgeat0g**
12 июня 2015 в 03:33

IOCP в связке с пулом потоков работает. Для игр вот как раз статейка про него.

 0 Ответить  

o  **waleks**
12 июня 2015 в 01:31

а почему бы вам не попробовать связку nginx->node.js->websocket? возможно, вам подойдет это решение…

 0 Ответить  

o  **lexoge**
12 июня 2015 в 18:04

А вот с чего начать?

Начать можно вот с этого: www.gnu.org/software/libc/manual/html_node/Server-Example.html

Очень простая штука на самом деле.

Простая, портабельная и эффективная, как топор.

Я с ней ещё 15 лет назад баловался, когда писал сетевые программки.

Для двух игроков вам этого хватит за глаза.

Когда освоите её и захочется стильтного, быстрого и современного, легко сможете переключиться на это:

banu.com/blog/2/how-to-use-epoll-a-complete-example-in-c

Это в случае, если у вас сервер linux only.

+1

Ответить

...

- НЛО прилетело и опубликовало эту надпись здесь

UAZMQJ
13 июня 2015 в 01:24

Буду тем парнем, который скажет про Erlang.

0

Ответить

...

- НЛО прилетело и опубликовало эту надпись здесь

solus_gex
14 июня 2015 в 06:16

статья полезная, но слишком много метафор и эмоций – для большинства, кроме совсем уж дилетантов, «оно не надо»

-3

Ответить

...

VFedyk
14 июня 2015 в 11:55

Странно, что вы считаете полезным свой комментарий.

+4

Ответить

...

vazic
15 июня 2015 в 08:07

Как раз недавно использовал NGINX для нагрузочного тестирования F5 LTM, где объект этой статьи выступал в роли backend. Без тонкой настройки сетевых параметров ядра, каждый легко обрабатывал 50000 запросов в секунду на статике. (уперся в способности сетевой карты)

Нащупать потолок у F5 не получилось только на чистом http трафике (на 500к запросов он даже не потел). С TLS и всячими IDP функциями все оказалось прозаичнее

+1

Ответить

...

Вы можете оставлять комментарии только к свежим публикациям

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

 alizar
21 час назад

Гений 21 века. Чем сейчас занимается Фабрис Беллар

 Простой  9 мин  21K

Ретроспектива

 +106

 94

 19

 Lunathecat
17 часов назад

Собираем электрогитару из конструктора

 Простой  8 мин  5.4K

Туториал

 +50

 29

 14

 Dukagav
4 часа назад

Переезд на Астра Линукс

 Простой  8 мин  5K

 +12

 13

 13

 ViktorSergeev
19 часов назад

LaserPecker LP4: портативный лазерный гравер последнего поколения. Что он умеет?

 3 мин  6.2K



+8



14



10



DaneSoul

56 минут назад

Иероглифы – хаос или система? Разбираемся из чего они состоят, как работают и в чём их сила



Простой



15 мин



572

FAQ



+7



15

0



Suvitruf

19 часов назад

Недельный геймдев: #154 – 31 декабря, 2023



Простой



4 мин



2.3K

Дайджест



+6



3

0



ko_ya

18 часов назад

Машина из трилогии «Назад в будущее»: Джон Делореан и его изобретения



Простой



3 мин



5.6K

Обзор



+3



16

11



VerZsuT

19 часов назад

Что такое react-afc



Средний



3 мин



4.2K

Обзор



+3



19

8



MountainGoat

9 часов назад

Начинаем продолжать: обработка исходников с помощью ИИ в оффлайне

Средний 17 мин 2.1K

Обзор

+2

29

2

Город засыпает, просыпается география: угадываем IT-города России

Турбо

Показать лучшие за всё время

МИНУТОЧКУ ВНИМАНИЯ



Помоги хабраслизию найти самые праздничные видеоигры



Разыскивается Дед Мороз: срочно нужна ваша помощь



Подарки победителям охоты за секретами

ВОПРОСЫ И ОТВЕТЫ

Как настроить редирект с url на другую папку?

nginx · Простой · 0 ответов

Как правильно развернуть несколько web-приложений с помощью docker compose на одном сервере?

nginx · Простой · 1 ответ

Как использовать proxy_pass для https?

nginx · Простой · 1 ответ

Как сделать в NGINX proxy_pass на адрес из значения аргумента в запросе?

nginx · Простой · 1 ответ

Как сделать два сервера на одном компьютере доступными в интернет?

Nginx · Средний · 1 ответ
Больше вопросов на Хабр Q&A

Реклама

РЕКЛАМА · RUVDS.HABR.IO

⋮



пришло
время
убраться
~~на орбиту~~
на орбите



ЧИТАЮТ СЕЙЧАС

Переезд на Астра Линукс

5K 13

Гений 21 века. Чем сейчас занимается Фабрис Беллар

21K 19

Солнце поздравило землян с Новым годом необычайно мощной вспышкой 1 января 2024 года

21K 18

Иероглифы – хаос или система? Разбираемся из чего они состоят, как работают и в чём их сила

572

0

Мой вам подарок к Новому году или как наконец запустил то, что надо было давно запустить…

24K

70

Город засыпает, просыпается география: угадываем IT-города России

Турбо

ИСТОРИИ



Ищем Деда Мороза

Он куда-то запропался, и Хабролизен пытается найти его на огромной Фабрике подарков.

Примкните к поискам и унесите припрятанные там подарки. Еще искать? Выбирайте локации.



Слово новичкам

Осенью Хабр провёл воркшопы для компаний-новичков: мы рассказывали, как общаться с айтишниками в хабрологе, чтобы всем это приносило больше пользы.

И вот — подборка первых статей от обучавшихся компаний.



Как Дедушке Морозу выбрать стек

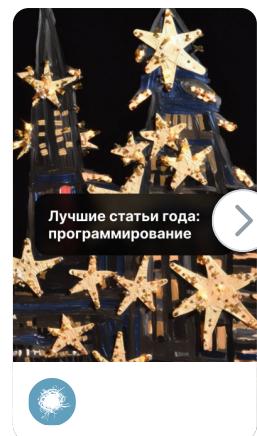
Мороз — то скромный боссъ инфраструктуры, который не выдержит: а ведь нужно выбирать стеки для серверов и для помощников создать виртуальные рабочие места!



В Новый Год с Яндекс 360

ТАК, Я ВСЁ ПРОДУМАЛ ДЕЛУЧКА. ВОТ ПОСМОТРИ, КАКОЙ У МЕНЯ СТАРЫЙ НАРДИК, С НИМ НАМ ВСЁ ПО ЛЧУЧ!

А вы ЕЛКУ НАРДИКАЕМ, ДАВАЙ К НАМ!



Программерский топ

Лучшие статьи года: программирование



[Ищем Деда Мороза](#)

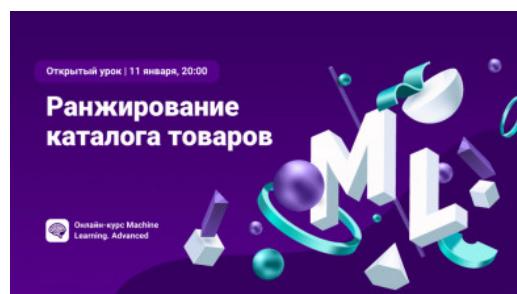
[Слово новичкам](#)

[Как Дедушке Морозу выбрать стек](#)

[В Новый Год с Яндекс 360](#)

[Программерский топ](#)

БЛИЖАЙШИЕ СОБЫТИЯ

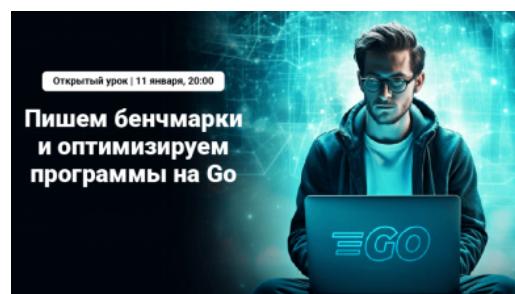


Открытый урок «Ранжирование каталога товаров»

11 января 20:00

Онлайн

[Подробнее в календаре](#)



Открытый вебинар «Пишем бенчмарки и оптимизируем программы на Go»

11 января 20:00

Онлайн

[Подробнее в календаре](#)

TECH TALKS VLADIMIR



12 января / 2024

МИТАП #2:

Встречи и разговоры про насущные проблемы в IT. Онлайн и офлайн.

Vladimir Tech t

- 📅 12 января
- ⌚ 18:00 - 21:00
- 📍 Онлайн

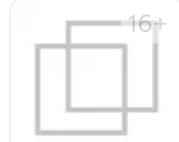
[Подробнее в календаре](#)

РЕКЛАМА • 16+ Я

 practicum.yandex.ru

 16+ HTML & CSS

18 500 ₽
Курс Профессиональная вёрстка на HTML и CSS

 16+

18 500 ₽
Курс Мид фронтенд-разработчик

Ваш аккаунт

[Профиль](#)[Трекер](#)[Диалоги](#)[Настройки](#)[ППА](#)

Разделы

[Статьи](#)[Новости](#)[Хабы](#)[Компании](#)[Авторы](#)[Песочница](#)

Информация

[Устройство сайта](#)[Для авторов](#)[Для компаний](#)[Документы](#)[Соглашение](#)[Конфиденциальность](#)

Услуги

[Корпоративный блог](#)[Медийная реклама](#)[Нативные проекты](#)[Образовательные программы](#)[Стартапам](#)[Настройка языка](#)[Техническая поддержка](#)