ИУ-10
Системное
Программное
Обеспечение

Apache vs Nginx: практический взгляд

Москва, 2022

Содержание

Введение

Apache и Nginx — 2 самых широко распространенных веб-сервера с открытым исходным кодом в мире. Вместе они обслуживают более 50% трафика во всем интернете. Оба решения способны работать с разнообразными рабочими нагрузками и взаимодействовать с другими приложениями для реализации полного веб-стека.

Несмотря на то, что у Apache и Nginx много схожих качеств, их нельзя рассматривать как полностью взаимозаменямые решения. Каждый из них имеет собственные преимущества и важно понимать какой веб-сервер выбрать в какой ситуации. В этой статье описано то, как каждый из этих веб-серверов ведет себя при различных условиях.

Общий обзор

Прежде чем погрузиться в различия между Apache и Nginx давайте бегло взглянем на предысторию каждого из этих проектов.

Apache

Apache HTTP Server был разработан Робертом Маккулом в 1995 году, а с 1999 года разрабатывается под управлением Apache Software Foundation — фонда развития программного обеспечения Apache. Так как HTTP сервер это первый и самый популярный проект фонда его обычно называют просто Apache.

Веб-север Apache был самым популярным веб-сервером в интернете с 1996 года. Благодаря его популярности у Apache сильная документация и интеграция со сторонним софтом.

Администраторы часто выбирают Apache из-за его гибкости, мощности и широкой распространенности. Он может быть расширен с помощью системы динамически загружаемых модулей и исполнять программы на большом количестве интерпретируемых языков программирования без использования внешнего программного обеспечения.

Nginx

В 2002 году Игорь Сысоев начал работу над Nginx для того чтобы решить проблему C10K — требование к ПО работать с 10 тысячами одновременных соединений. Первый публичный релиз был выпущен в 2004 году,

поставленная цель была достигнута благодаря асинхронной event-driven архитектуре.

Nginx начал набирать популярность с момента релиза благодаря своей легковесности (light-weight resource utilization) и возможности легко масштабироваться на минимальном железе. Nginx превосходен при отдаче статического контента и спроектирован так, чтобы передавать динамические запросы другому ПО предназначенному для их обработки.

Администраторы часто выбирают Nginx из-за его эффективного потребления ресурсов и отзывчивости под нагрузкой, а также из-за возможности использовать его и как веб-сервер, и как прокси.

Архитектура обработки соединений

Одно из самых существенных отличий между Apache и Nginx состоит в том как они обрабатывают соединения и отвечают на различные виды трафика.

Apache

Apache предоставляет несколько модулей мультипроцессинга (multi-processing modules, MPM), которые отвечают за то как запрос клиента будет обработан. Это позволет администраторам определять политику обработки соединений. Ниже представлен список MPM-модулей Apache:

- **mpm_prefork** — этот модуль создает по одному процессу с одним потоком на каждый запрос. Каждый процесс может обрабатывать только одно соединение в один момент времени. Пока число запросов меньше числа процессов этот MPM работает очень быстро. Однако производительность быстро падает когда число запросов начинает превосходить число процессов, поэтому в большинстве случаев это не самый лучший выбор. Каждый процесс потребляет значительный объем RAM, поэтому этот MPM сложно поддается масштабированию. Но он может быть использован вместе с компонентами, которые не созданы для работы в многопоточной среде. Например, PHP не является потокобезопасным, поэтому этот MPM рекомендуется использовать как безопасный метод работы с mod_php.

- **mpm_worker** — этот модуль создает процессы, каждый из которых может управлять несколькими потоками. Каждый поток может обрабтывать одно соединение. Потоки значительно более эффективны чем процессы, что означает что mpm_worker масштабируется значительно

лучше чем mpm_prefork. Так как потоков больше чем процессов это означает, что новое соединение может быть сразу обработано свободным потоком, а не ждать пока освободится процесс.

- **mpm_event** — этот модуль похож на mpm_worker, но оптимизирован под работу с keep-alive соединениями. Когда используется mpm_worker соединение будет удерживать поток вне зависимости от того активное это соединение или keep-alive. Mpm_event выделяет отдельные потоки для keep-alive соединений и отдельные потоки для активных соединений. Это позволяет модулю не погрязнуть в keep-alive соединениях, что необходимо для быстрой работы. Этот модуль был отмечен как стабильный в Apache версии 2.4.

Как вы можете видеть Apache предлагает гибкие возможности для выбора различных алгоритмов обработки соединений и запросов.

Nginx

Nginx появился на сцене позднее Apache, по этой причине, его разработчик был лучше осведомлен о проблемах конкурентности, с которыми сталкиваются сайты при масштабировании. Благодаря этим знаниям Nginx изначально был спроектирован на базе асинхронных неблокирующих event-driven алгоритмов.

Nginx создает процессы-воркеры каждый из которых может обслуживать тысячи соединений. Воркеры достигают такого результата благодаря механизму основанному на быстром цикле, в котором проверяются и обрабатываются события. Отделение основной работы от обработки соединений позволяет каждому воркеру заниматься своей работой и отвлекаться на обработку соединений только тогда когда произошло новое событие.

Каждое соединение, обрабатываемое воркером, помещается в event loop вместе с другими соединениями. В этом цикле события обрабатываются асинхронно, позволяя обрабатывать задачи в неблокирующей манере. Когда соединение закрывается оно удаляется из цикла.

Этот подход к обработке соединений позволяет Nginx'у невероятно масштабироваться при ограниченных ресурсах. Поскольку сервер однопоточный и он не создает процессы под каждое соединение, использование памяти и CPU относительно равномерно, даже при высоких нагрузках.

Статический и динамический контент

Если рассматривать жизненные примеры, то основные различия между Apache и Nginx в том как они обрабатывают запросы к статическому и динамическому контенту.

Apache

Apache может раздавать статический контент используя стандартные file-based методы. Производительность таких операций зависит от выбранного MPM.

Apache также может раздавать динамический контент встраивая интерпретатор нужного языка в каждого воркера. Это позволяет обрабатывать запросы к динамическому содержимому средствами самого веб-сервера и не полагаться на внешние компоненты. Интерпретаторы языков могут быть подключены к Apache с помощью динамически загружаемых модулей.

Возможность обрабатывать динамический контент средствами самого Apache упрощает конфигурирование. Нет необходимости настраивать взаимодействие с дополнительным софтом, динамический модуль может быть легко отключен в случае изменившихся требований.

Nginx

Nginx не имеет возможности самостоятельно обрабатывать запросы к динамическому контенту. Для обработки запросов к PHP или другому динамическому контенту Nginx должен передать запрос внешнему процессору для исполнения, подождать пока ответ будет сгенерирован и получить его. Затем результат может быть отправлен клиенту.

Для администраторов это означает, что нужно настроить взаимодействие Nginx с таким процессором используя один из протоколов, который известен Nginx'у (http, FastCGI, SCGI, uWSGI, memcache). Это может немного усложнить процесс настройки, в особенности когда вы будете пытаться предугадать какое число соединений разрешить, так как будет использоваться дополнительное соединение с процессором на каждый пользовательский запрос.

Однако, этот метод имеет и свои преимущества. Так как интерпретатор не встроен в каждого воркера, то оверхед, связанный с этим, будет иметь место только при запросах к динамическому контенту. Статический контент будет возвращен клиенту простым способом и запросы к интерпретатору будут выполняться только тогда когда они нужны. Apache тоже может работать в такой манере, но тогда это лишит его всех преимуществ

описанных в предыдущем разделе.

Распределенная конфигурация против централизованной

Для администраторов одним из очевидных отличий этих двух веб-серверов является наличие у Apache возможности задавать конфигурацию на уровне директории.

Apache

Apache имеет опцию, которая позволяет включить конфигурирование на уровне директорий. Если эта опция включена, то Apache будет искать конфигурационные директивы в директориях с контентом в специальных скрытых файлах, которые называются .htaccess.

Так как такие конфигурационные файлы находятся в директриях с контентом, Apache вынужден при обработке каждого запроса проверять не содержит ли каждый компонент запрашиваемого пути файл .htaccess и исполнять директивы в найденных файлах. Это позволяет децентрализовать конфигурирование веб-сервера, что позволяет реализовать на уровне директорий модификацию URL'ов (URL rewrite), ограничения доступа, авторизацию и аутентификацию и даже политики кеширования.

Несмотря на то, что все описанное выше может быть настроено и в основном конфигурационном файле Apache, файлы .htaccess имеют ряд преимуществ. Во-первых, эти файлы интерпретируются как только они найдены по запрашиваемому пути, что позволяет менять конфигурацию на лету не перезагружая веб-сервер. Во вторых, это позволяет дать возможность непривилегированным пользователям контролировать определынные аспекты собственных веб-приложений с помощью .htaccess.

Это дает простой способ таким приложениям как системы управления контентом (CMS) конфигурировать собственное окружение не имея доступа к конфигурационному файлу веб-сервера. Это также может быть использовано шаред хостингами, чтобы сохранить контроль над основным конфигурационным файлом и дать клиентам контроль над конфигурацией определенных директорий.

Nginx

Nginx не интерпретирует файлы .htaccess и не предоставляет механизм конфигурирования на уровне директорий за пределами основного конфи-

гурационного файла. Этот подход может показаться менее гибким чем в случае с Apache, но он имеет свои преимущства.

Основное преимущество перед использованием .htaccess — это улучшенная производительность. Типичная установка Apache позволяет использовать файлы .htaccess в любой директории, поэтому веб-сервер при каждом запросе вынужден проверять наличие этого файла во всех родительских директориях запрошенного файла. Если найден один или более таких файлов, то все они должны быть прочитаны и интерпретированы.

Так как Nginx не позволяет переопределять конфиги на уровне директорий, он может обрабатывать запросы быстрее, ведь ему достаточно сделать один directory lookup и прочитать один конфигурационный файл на каждый запрос (предполагается, что файл найден там где он должен быть по соглашению).

Второе преимущество связано с безопасностью. Распределенная конфигурация на уровне директорий в Apache возлагает ответственность за безопасность на обычных пользователей, которые вряд ли способны решить эту задачу качественно. То что администратор контролирует весь сервер предотвращает ошибки безопасности, которые могут возникнуть если дать пользователям доступ к конфигурации.

Имейте ввиду, что вы можете отключить поддержку .htaccess в Apache, если сказанное выше произвело на вас впечатление.

Интерпретация базирующаяся на файлах и URI

То как веб-сервер интерпретирует запрос и сопоставляет его с ресурсом в системе это еще одна отличительная особенность в этих двух серверах.

Apache

Apache имеет возможность интерпретировать запрос как физический ресурс в файловой системе или как URI, который требует дополнительной обработки. Первый тип запросов использует конфигурационные блоки <Directory> или <File>, второй — блоки <Location>.

Так как Apache изначально был спроектирован как веб-сервер, он по умолчанию интерпретирует запросы как ресурсы в файловой системе. Он берет document root веб-сервера и дополняет его частью запроса, которая следует за именем хоста и номером порта, чтобы найти запрашиваемый файл. В общем случае, иерархия файловой системы представленная в вебе доступна как дерево документов.

Apache предоставляет ряд альтернатив на случай когда запрос не соответствует файлу в файловой системе. Использование блоков <Location> это метод работы с URI без отображения на файловую систему. Также возможно использовать регулярные выражения, которые позволяют задать более гибкие настройки для всей файловой системы.

Так как Apache может оперировать и с файловой системой, и с webspace, то он в основном опирается на методы работы с файловой системой. Это видно в некоторых решениях в дизайне архитектуры веб-сервера, например, в использовании файлов .htaccess для конфигурирования на уровне директорий. Документация к Apache не рекомендует использовать URI-блоки для ограничения доступа для запросов к файловой системе.

Nginx

Nginx создан, чтобы работать и в качестве веб-сервера, и в качестве прокси-сервера. По этой причине он работает в первую очередь с URI, транслируя их при необходимости в запросы к файловой системе.

Эта особенность прослеживается в том как для Nginx конструируются и интерпретируются конфигурационные файлы. В Nginx нет способа создать конфигурацию для заданной директории, вместо этого он парсит URI.

Например, основными конфигурационными блоками в Nginx являются <server> и <location>. В блоке <server> определяется хост, который будет обслуживаться, блок <location> отвечает за обработку части URI, которая идет после имени хоста и номера порта. Таким образом, запрос интерпретируется как URI, а не как путь в файловой системе.

В случае запросов к статическим файлам все запросы должны быть отображены (mapped) на путь в файловой системе. Сначала Nginx выбирает блоки server и location, которые будут использованы для обработки запроса и затем объединяет document root с URI, в соответствии с конфигурацией.

Эти подходы (интерпретация запроса как пути в файловой системе и как URI) могут показаться похожими, но тот факт что Nginx рассматривает запросы как URI, а не как пути в файловой системе позволяет ему легче справляться одновременно и с ролью веб-сервера, и с ролью прокси. Nginx конфигурируется так, чтобы отвечать на различные шаблоны запросов. Nginx не обращается к файловой системе до тех пор пока он не готов обслужить запрос, что объясняет почему он не реализует ничего похожего на файлы .htaccess.

Модули

И Apache, и Nginx могут быть расширены при помощи системы модулей, но способы реализации модульной системы принципиально отличаются.

Apache

Система модулей Apache позволяет динамически загружать и выгружать модули, чтобы удовлетворить ваши потребности, в то время как ваш сервер запущен. Ядро Apache всегда доступно, в то время как модули можно включать и выключать, чтобы добавить или удалить функциональность из основного сервера.

Apache использует эту функциональность для решения широкого круга задач. Благодаря зрелости платформы существует огромное множество модулей, которые могут изменять ключевые особенности сервера, например модуль mod_php позволяет включать PHP-интерпретатор в кажого воркера.

Использование модулей не ограничивается лишь обработкой динамических запросов. Среди других возможностей модулей: изменение URL'ов (URL rewrite), аутентификация клиентов, защита сервера, логгирование, кеширование, сжатие, проксирование, ограничение частоты запросов, шифрование. Динамические модули могут значительно расширить функциональность ядра.

Nginx

Nginx тоже имеет систему модулей, но она сильно отличается от подхода используемого в Apache. В Nginx модули загружаются не динамически, а должны быть выбраны и скомпилированы с ядром сервера.

Для многих пользователей по этой причине Nginx кажется менее гибким. Это особенно относится к пользователям, кто имеет мало опыта ручной сборки приложений и предпочитают использовать системы управления пакетами. Обычно разработчики дистрибутивов стремятся создать пакеты для всех часто используемых модулей, но если вам нужен какой-то нестандартный модуль вам придется собрать его из исходников самостоятельно.

Тем не менее, модули в Nginx очень полезны и востребованы, вы можете определить чего вы хотите от сервера и включить только те модули, что вам нужны. Некоторые пользователи считают такой подход более безопасным так как произвольные модули не могут быть подключены к серверу.

Модули Nginx реализуют те же возможности, что и модули Apache: проксирование, сжатие данных, ограничение частоты запросов, логгирование, модификация URL'ов, гео-локация, аутентификация, шифрование,

потоковое вещание, почтовые функции.

Поддержка, совместимость, экосистема и документация

В процессе использования приложения важными являются экосистема созданная вокруг него и возможность получения поддержки.

Apache

Так как Apache пользуется популярностью такое длительное время с поддержкой у него нет проблем. Легко можно найти большое количество документации как от разработчиков Apache, так и от сторонних авторов. Эта документация покрывает все возможные сценарии использования Apache, включая взаимодействие с другими приложениями.

Существует много инструментов и веб-проектов идущих в комплекте со средствами запуска самих себя из под Apache. Это относится как к самим проектам, так и к системам управления пакетами.

В общем случае Apache будет иметь больше поддержки в сторонних проектах просто потому он доступен на рынке длительное время. Администраторы также обычно имеют больше опыта работы с Apache, так как большинство людей начинают работу с шаред-хостингов где Apache более популярен из-за поддержки файлов .htaccess.

Nginx

Nginx обычно используется там, где предъявляются повышенные требования к производительности и в некоторых областях он все еще является догоняющим.

В прошлом было сложно найти вменяемую поддержку по этому веб-серверу на английском языке, так как на ранних этапах разработка и документация велись на русском языке. Вместе с ростом интереса к проекту документация была переведена на английский и теперь можно найти достаточное количество документации и от разработчиков веб-сервера, и от сторонних авторов.

Разработчики стороннего ПО также начинают поддерживать работу с Nginx и некоторые из них уже предлагают на выбор пользователя конфиги для работы или с Apache, или с Nginx. И даже без поддержки приложением работы с Nginx не составляет большого труда написать свой конфиг

для интеграции приложения с Nginx.

Совместное использование Apache и Nginx

После того как вы ознакомились с плюсами и минусами Apache и Nginx у вас должно появиться представление того, какой из серверов больше подходит под ваши задачи. Однако, можно достигнуть лучших результатов используя оба сервера вместе.

Распространенной схемой использования является размещение Nginx перед Apache в качестве реверс-прокси. В такой конфигурации Nginx называют фронтендом, а Apache — бэкендом. При таком подходе Nginx будет обслуживать все входящие запросы клиентов и мы получим выигрыш из-за его возможности обрабатывать множество конкурентных запросов.

Nginx будет самостоятельно обслуживать статический контент, а для динамического контента, например для запросов к PHP-страницам, будет передавать запрос к Apache, который будет рендерить страницу, возвращать ее Nginx'у, а тот в свою очередь будет передавать ее пользователю.

Такая конфигурация очень популярна, Nginx используется в ней для сортировки запросов. Он обрабатывает сам те запросы которые может и передает Apache только запросы, которые не может обслужить сам, снижая таким образом нагрузку на Apache.

Эта конфигурация позволяет горизонтально масштабировать приложение: вы можете установить несколько бэкенд серверов за одним фронтендом и Nginx будет распределять нагрузку между ними, увеличивая тем самым отказоустойчивость приложения.

Заключение

Как вы можете видеть и Apache, и Nginx — это мощные, гибкие и функциональные инструменты. Для того чтобы выбрать сервер под ваши задачи необходимо определиться с требованиями к нему и провести тесты на всех возможных паттернах использования вашего приложения.

Между этими проектами есть значительные различия, которые могут оказать влияние на производительность, возможности и время реализации необходимое для внедрения и запуска каждого из решений. Выбор является серией компромиссов и не стоит пренебрегать тестами. В конечном итоге, не существует одного универсального веб-сервера под все возможные задачи, поэтому важно найти решение максимально соответствующее вашем задачам и целям.