NGINX – многофункциональное ПО: оно может выступать в роли веб-сервера и балансировщика нагрузки, кешировать контент и быть API-шлюзом. Ещё одно неоспоримое преимущество NGINX – надежность. Неудивительно, что с каждым днем все больше сайтов во всем мире отдают предпочтение этому веб-серверу.

Из данной книги вы узнаете, как получить максимальную отдачу от NGINX с открытым исходным кодом и NGINX Plus. Вы получите простые рекомендации по вопросам разного уровня сложности – начиная с установки ПО и настройки основных функций до устранения неполадок. В книге приводится описание новых функций NGINX с открытым исходным кодом, таких как поддержка gRPC, сервер активной доставки HTTP/2 и алгоритм балансировки нагрузки Random with Two Choices для кластерных сред, а также новых функций NGINX Plus.

Книга предназначена для администраторов и разработчиков сайтов.

Экономия
Ваши затраты сокращаются более чем на 80% по сравнению с использованием доставки аппаратных приложений. При этом вы получаете брандмауэры для веб-приложений с желаемыми производительностью и функционалом.

Пониженная сложность
Комплексный балансировщик нагрузки, кеширование содержимого, веб-сервер и брандмауэр веб-приложений препятствуют разрастанию инфраструктуры.

Эксклюзивное решение
Только NGINX Plus предлагает аутентификацию на основе JWT, высокую доступность, API NGINX Plus и другие расширенные функции.

NGINX WAF
Пробная версия NGINX WAF на базе ModSecurity доступна при загрузке пробной версии NGINX Plus.

NGINX Скачайте на nginx.com/freetrial

ISBN 978-5-97060-790-9

O'REILLY®

NGINX Книга рецептов

Дерек де Йонге

ДМК ИЗДАТЕЛЬСТВО

Дерек де Йонге

NGINX. Книга рецептов

Продвинутые рецепты высокопроизводительной балансировки нагрузки

2019 UPDATE

NGINX Cookbook

Advanced Recipes for High Performance Load Balancing

Derek DeJonghe

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

NGINX. Книга рецептов

Продвинутые рецепты высокопроизводительной балансировки нагрузки

Дерек де Йонге

Перевод с английского Беликова Д. А.

Москва, 2020 ДМК ПРЕСС издательство

УДК 004.42
ББК 32.972
Й11

Й11 Дерек де Йонге

NGINX. Книга рецептов. / пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2020. – 176 с.: ил.

ISBN 978-5-97060-790-9

Из этой книги вы узнаете, как получить максимальную отдачу от NGINX с открытым исходным кодом и NGINX Plus. Вы получите простые рекомендации по вопросам разного уровня сложности – начиная с установки ПО и настройки основных функций до устранения неполадок. Автор описывает новые функции NGINX с открытым исходным кодом, такие как поддержка gRPC, сервер активной доставки HTTP/2 и алгоритм балансировки нагрузки Random with Two Choices для кластерных сред, а также новые функции NGINX Plus.

Издание предназначено для администраторов и разработчиков сайтов.

УДК 004.42
ББК 32.972

ISBN 978-1-49196-893-2 (англ.)
ISBN 978-5-97060-790-9 (рус.)

Оглавление

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте **www.dmkpress.com**, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и O'Reilly очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты **dmkpress@gmail.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую предоставлять вам качественные материалы.

Предисловие

Добро пожаловать в обновленное издание «NGINX. Книга рецептов». Прошло почти два года, с тех пор как издательство O'Reilly опубликовало первую версию книги. С того времени многое изменилось, кроме одного: каждый день все больше и больше сайтов по всему миру отдают предпочтение NGINX. Сегодня их 300 млн, почти вдвое больше, когда вышло первое издание.

Есть много причин, по которым использование NGINX набирает популярность спустя 14 лет после его появления. Это швейцарский армейский нож: NGINX может быть веб-сервером, балансировщиком нагрузки, кешировать контент и быть API-шлюзом. Но, возможно, еще более важным фактом является то, что он надежен.

Эта книга покажет вам, как получить максимальную отдачу от NGINX с открытым исходным кодом и NGINX Plus. Вы найдете свыше 150 страниц, содержащих простые в использовании рецепты, охватывающие все от того, как правильно установить NGINX, настроить все основные функции до отладки и устранения неполадок.

Эта обновленная версия также включает в себя описание новых функций с открытым исходным кодом, таких как поддержка gRPC, сервер активной доставки HTTP/2 и алгоритм балансировки нагрузки Random with Two Choices для кластерных сред, а также новые функции NGINX Plus, такие как поддержка совместного использования состояния, новый API NGINX Plus и хранилище типа ключ/значение. На этих страницах рассказывается почти все, что вам нужно знать о NGINX.

Мы надеемся, что вам понравится эта книга и она будет способствовать вашему успеху в создании и развертывании приложений, которые все мы используем.

Файсаль Меном,
менеджер по маркетингу продукции, NGINX Inc.

Глава 1

Основы

1.0. Введение

Чтобы начать работу с NGINX с открытым исходным кодом или NGINX Plus, для начала необходимо установить их в системе и познакомиться с основами. В этой главе вы узнаете, как установить NGINX, где находятся основные файлы конфигурации и команды администрирования. Вы также узнаете, как проверить свою установку и выполнить запросы к серверу, установленному по умолчанию.

1.1. Установка на компьютер с Debian/Ubuntu

Задача

Вам необходимо установить NGINX с открытым исходным кодом на машине с Debian или Ubuntu.

Решение

Создайте файл с именем */etc/apt/sources.list.d/nginx.list* следующего содержания:

```
deb http://nginx.org/packages/mainline/OS/ CODENAME nginx
deb-src http://nginx.org/packages/mainline/OS/ CODENAME nginx
```

Измените файл, заменив буквосочетание `OS` в конце URL-адреса на `ubuntu` или `debian` в зависимости от дистрибутива. Замените слово `CODENAME` на кодовое имя вашего дистрибутива: `jessie` или `stretch`, если это Debian, либо `trusty`, `xenial`, `artful` или `bionic`, если это Ubuntu. Затем выполните приведенные ниже команды:

```
wget http://nginx.org/keys/nginx_signing.key
apt-key add nginx_signing.key
```

```
apt-get update
apt-get install -y nginx
/etc/init.d/nginx start
```

Обсуждение

Файл, который вы только что создали, инструктирует систему управления пакетами `apt` использовать Официальный репозиторий пакетов NGINX. С помощью следующих далее команд вы скачиваете ключ подписи пакета NGINX GPG и импортируете его в apt. Предоставление apt ключа подписи активирует систему, чтобы выполнить проверку пакетов из репозитория. Команда `apt-get-update` велит системе apt обновить свои листинги пакетов из известных репозиториев. После обновления списка пакетов вы можете установить NGINX Open Source из официального репозитория NGINX. После его установки последняя команда запускает NGINX.

1.2. Установка на компьютер с RedHat/CentOS

Задача

Вам необходимо установить NGINX с открытым исходным кодом на компьютер с RedHat или CentOS.

Решение

Создайте файл с именем */etc/yum.repos.d/nginx.repo* следующего содержания:

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/mainline/OS/OSRELEASE/$basearch/
gpgcheck=0
enabled=1
```

Измените файл, заменив буквосочетание `OS` в конце URL-адреса на `rhel` или `centos` в зависимости от дистрибутива. Замените слово `OSRELEASE` на цифру 6 или 7 для версии 6.x или 7.x соответственно. Затем выполните приведенные ниже команды:

```
yum -y install nginx
systemctl enable nginx
```

```
systemctl start nginx
firewall-cmd --permanent --zone=public --add-port=80/tcp
firewall-cmd -reload
```

Обсуждение

Файл, который вы только что создали, инструктирует систему управления пакетами yum использовать Официальный репозиторий пакетов NGINX. С помощью следующих далее команд вы устанавливаете NGINX Open Source из Официального репозитория, сообщаете systemd активировать NGINX во время загрузки и велите ей запустить его сейчас. Команды брандмауэра открывают порт 80 для протокола TCP, который является значением порта по умолчанию для HTTP. С помощью последней команды вы выполняете перезагрузку брандмауэра для фиксации изменений.

1.3. Установка NGINX Plus

Задача

Вам необходимо установить NGINX Plus.

Решение

Посетите страницу по адресу http://cs.nginx.com/repo_setup. В раскрывающемся меню выберите устанавливаемую вами ОС и следуйте инструкциям. Эти инструкции аналогичны инструкциям по установке решений с открытым исходным кодом; однако вам необходимо установить сертификат для аутентификации в репозитории NGINX Plus.

Обсуждение

NGINX постоянно обновляет данное руководство по установке репозитория с инструкциями по установке NGINX Plus. В зависимости от вашей операционной системы и ее версии эти инструкции несколько отличаются, но есть кое-что общее. Вы должны войти на портал NGINX, чтобы скачать сертификат и ключ к вашей системе, которые используются для прохождения аутентификации в репозитории NGINX Plus.

1.4. Проверка установки

Задача

Вам нужно оценить правильность установки NGINX и проверить версию.

Решение

Вы можете убедиться, что NGINX установлен правильно и проверить его версию, используя приведенную ниже команду:

```
$ nginx -v
nginx version: nginx/1.15.3
```

Как показывает этот пример, в ответе отображается версия.

Можно подтвердить, что NGINX работает с помощью приведенной ниже команды:

```
$ ps -ef | grep nginx
root      1738    1 0 19:54 ? 00:00:00 nginx: master process
nginx     1739 1738 0 19:54 ? 00:00:00 nginx: worker process
```

Команда `ps` выводит список запущенных процессов. Отправив его в `grep`, можно искать конкретные слова в выводе. В этом примере мы используем `grep` для поиска `nginx`. Результат показывает два запущенных процесса, `master` и `worker`. Если NGINX запущен, вы всегда будете видеть главный процесс. Чтобы увидеть инструкции по запуску NGINX, см. следующий раздел. Чтобы понять, как запустить NGINX в качестве демона, примените методологии init.d или systemd.

Чтобы убедиться, что NGINX возвращает запросы правильно, используйте свой браузер и отправьте запрос на ваш компьютер или воспользуйтесь командой `curl`:

```
$ curl localhost
```

Вы увидите страницу приветствия NGINX.

Обсуждение

Команда `nginx` позволяет взаимодействовать с двоичным файлом NGINX, чтобы проверить версию, вывести список установленных модулей, протестировать конфигурации и отправить сигналы в главный процесс. Чтобы обслуживать запросы, NGINX должен быть запущен. Команда

`ps` – верный способ определить, работает ли NGINX в качестве демона или в приоритетном режиме. В исходной конфигурации, предоставляемой по умолчанию в NGINX, запускается HTTP-сервер статического сайта на порту 80. Этот сайт можно протестировать, отправив HTTP-запрос на компьютер на локальном хосте, а также на IP-адрес хоста и имя хоста.

1.5. Ключевые файлы, команды и каталоги

Задача

Вам нужно разобраться в важных каталогах и командах NGINX.

Решение

Файлы и каталоги NGINX

/etc/nginx/

Каталог */etc/nginx/* является корневым каталогом конфигурации по умолчанию сервера NGINX. В этом каталоге вы найдете файлы конфигурации, которые инструктируют NGINX, как вести себя.

/etc/nginx/nginx.conf

Файл */etc/nginx/nginx.conf* является точкой входа конфигурации по умолчанию, используемой службой NGINX. Этот файл конфигурации устанавливает глобальные параметры для таких вещей, как рабочий процесс, настройка, ведение журнала, загрузка динамических модулей и ссылки на другие файлы конфигурации NGINX. В исходной конфигурации файл */etc/nginx/nginx.conf* включает в себя блок `http` верхнего уровня, который содержит все файлы конфигурации в каталоге, описанном далее.

/etc/nginx/conf.d/

Каталог */etc/nginx/conf.d/* содержит файл конфигурации HTTP-сервера по умолчанию. Файлы в этом каталоге, оканчивающиеся на *.conf*, включены в блок `http` верхнего уровня из файла */etc/nginx/nginx.conf*. Рекомендуется использовать операторы `include` и упорядочивать свою конфигурацию таким образом, чтобы ваши файлы конфигурации были краткими. В некоторых репозиториях пакетов эта папка называется *site-enabled*, а файлы конфигурации связываются из папки с именем *site-available*; данная конвенция является устаревшей.

/var/log/nginx/

Каталог */var/log/nginx/* является местоположением журнала по умолчанию для NGINX. В этом каталоге вы найдете файлы *access.log* и *error.log*. Файл *access.log* содержит запись каждого запроса, который обслуживает NGINX. В файле *error.log* содержатся события с ошибками и отладочная информация, если модуль отладки включен.

Команды NGINX

`nginx -h`

Показывает справочное меню NGINX.

`nginx -v`

Показывает версию NGINX.

`nginx -V`

Показывает версию NGINX, информацию о сборке и аргументы конфигурации, где даны модули, встроенные в двоичный файл NGINX.

`nginx -t`

Проверяет конфигурацию NGINX.

`nginx -T`

Проверяет конфигурацию NGINX и выводит ее проверенной на экран. Эта команда полезна при поиске поддержки.

`nginx -s signal`

Флаг `-s` отправляет сигнал главному процессу NGINX. Вы можете отправлять такие сигналы, как `stop`, `quit`, `reload` и `reopen`. `stop` немедленно прекращает процесс NGINX, `quit` останавливает процесс NGINX после завершения обработки запросов в полете, `reload` перезагружает конфигурацию, `reopen` дает NGINX команду повторно открывать файлы журнала.

Обсуждение

Освоив эти ключевые файлы, каталоги и команды, вы можете приступить к работе с NGINX. Обладая данными знаниями, вы можете изменять файлы конфигурации по умолчанию и тестировать свои изменения с помощью команды `nginx -t`. Если ваш тест пройдет успешно, вы также будете знать, как проинструктировать NGINX для перезагрузки конфигурации с помощью команды `nginx -s reload`.

1.6. Обслуживание статического контента

Задача

Вам необходимо обслуживать статический контент с помощью NGINX.

Решение

Перепишите конфигурацию HTTP-сервера по умолчанию, расположенную в */etc/nginx/conf.d/default.conf*, используя приведенный ниже пример конфигурации NGINX:

```
server {
    listen 80 default_server;
    server_name www.example.com;

    location / {
        root /usr/share/nginx/html;
        # alias /usr/share/nginx/html;
        index index.html index.htm;
    }
}
```

Обсуждение

Данная конфигурация обслуживает статические файлы по протоколу HTTP через порт 80 из каталога */usr/share/nginx/html/*. Первая строка в этой конфигурации определяет новый блок `server`. Это определяет новый контекст, который NGINX будет слушать. Вторая строка дает NGINX команду слушать порт 80, а параметр `default_server` указывает NGINX использовать этот сервер в качестве контекста по умолчанию для порта 80. Директива `server_name` определяет имя хоста или имена запросов, которые должны быть направлены на этот сервер. Если конфигурация не определила этот контекст как `default_server`, NGINX будет направлять запросы на этот сервер, только если заголовок HTTP-узла соответствует значению, указанному в директиве `server_name`.

Блок `location` определяет конфигурацию на основе пути в URL-адресе. Путь или часть URL-адреса после домена называется URI. NGINX лучше всего будет соответствовать запрошенному URI для блока `location`. В этом примере используется символ / для сопоставления всех запросов. Директива `root` показывает NGINX, где искать статические файлы при

обслуживании контента для данного контекста. URI запроса добавляется к значению директивы `root` при поиске запрошенного файла. Если бы мы указали префикс URI для директивы `location`, он был бы включен в добавленный путь, если только мы не использовали каталог `alias` вместо `root`. И наконец, директива `index` предоставляет NGINX файл по умолчанию или список файлов для проверки в случае, если в URI не указан дальнейший путь.

1.7. Аккуратная перезагрузка

Задача

Вам необходимо перезагрузить конфигурацию, не отбрасывая пакеты.

Решение

Используйте метод `reload` для аккуратной перезагрузки конфигурации без остановки сервера:

```
$ nginx -s reload
```

В этом примере выполняется перезагрузка системы NGINX с использованием двоичного файла NGINX для отправки сигнала главному процессу.

Обсуждение

Перезагрузка конфигурации NGINX без остановки сервера предоставляет возможность изменять конфигурации на лету, не отбрасывая никаких пакетов. В динамичной среде с высокой продолжительностью работы в какой-то момент вам потребуется изменить конфигурацию балансировки нагрузки. NGINX позволяет делать это, сохраняя балансировщик нагрузки в сети. Эта функция предоставляет бесчисленные возможности, такие как повторный запуск управления конфигурацией в реальной среде или создание модуля с поддержкой приложений и кластеров для динамической настройки и перезагрузки NGINX, чтобы удовлетворять потребностям среды.

Глава 2

Высокопроизводительная балансировка нагрузки

2.0. Введение

Сегодня пользователям интернета требуется производительность и безотказная работа. Для этого запускается несколько копий одной и той же системы, и нагрузка распределяется между ними. По мере увеличения нагрузки в рабочее состояние может вводиться еще одна копия такой системы. Этот метод называется *горизонтальным масштабированием*. Программная инфраструктура приобретает все большую популярность благодаря своей гибкости, открывая огромный мир возможностей. Независимо от того, является ли вариант использования небольшим (как набор из двух для обеспечения высокой доступности) или крупнее (из тысяч по всему миру), необходимо создать решение для балансировки нагрузки, которое будет таким же динамичным, как и инфраструктура. NGINX удовлетворяет эту потребность несколькими способами, между серверами HTTP, TCP и UDP, о чем мы расскажем в этой главе.

При балансировке нагрузки важно, чтобы влияние на клиента было только положительным. Многие современные веб-архитектуры используют уровни приложений без фиксации состояния, храня состояние в системе с общей памятью или базах данных. Однако такое происходит не у всех. Состояние сеанса чрезвычайно ценно и обширно в интерактивных приложениях. Оно может храниться локально на сервере приложений по ряду причин; например, в приложениях, для которых обрабатываемые данные настолько велики, что нагрузка на сеть слишком высока по производительности. Когда состояние хранится

локально на сервере приложений, пользователю крайне важно, чтобы последующие запросы продолжали доставляться на тот же сервер. Еще один аспект этой ситуации заключается в том, что серверы не должны быть освобождены до завершения сеанса. Для работы с масштабными приложениями, которые фиксируют состояние, требуется интеллектуальный балансировщик нагрузки. NGINX Plus предлагает несколько способов решения этой проблемы путем отслеживания куки-файлов или маршрутизации. В этой главе рассматривается постоянство сеансов, поскольку оно касается балансировки нагрузки с помощью NGINX и NGINX Plus.

Также важно обеспечить работоспособность приложения, которое обслуживает NGINX. По ряду причин приложения могут не работать. Это может быть вызвано подключением к сети, сбоем в работе сервера или приложения среди прочего. Прокси-серверы и балансировщики нагрузки должны быть достаточно умными, чтобы обнаруживать сбои в работе вышестоящих серверов и прекращать передачу трафика к ним; в противном случае клиент будет пребывать в состоянии ожидания, получая лишь сообщения о тайм-ауте. Способ смягчить ухудшение качества обслуживания в случае сбоя сервера состоит в том, чтобы прокси-сервер проверял работоспособность вышестоящих серверов. NGINX предлагает два типа проверки работоспособности: пассивный, доступный в версии с открытым исходным кодом, и активный, доступный только в NGINX Plus. Активные проверки работоспособности через регулярные промежутки времени установят соединение или выполнят запрос к вышестоящему серверу и смогут убедиться, что ответ верный. Пассивные проверки работоспособности контролируют соединение или ответы вышестоящего сервера, когда клиенты выполняют запрос или устанавливают соединение. Возможно, вы захотите использовать пассивные проверки, чтобы уменьшить нагрузку на свои вышестоящие серверы, и активные – чтобы определить отказ вышестоящего сервера, прежде чем клиент столкнется со сбоем. В конце этой главы рассматривается мониторинг работоспособности вышестоящих серверов приложений, для которых вы выполняете балансировку нагрузки.

2.1. Балансировка нагрузки для HTTP

Задача

Вам необходимо распределить нагрузку между двумя или более серверами HTTP.

Решение

Используйте HTTP-модуль NGINX для выполнения балансировки по HTTP-серверам с использованием блока `upstream`:

```
upstream backend {
    server 10.10.12.45:80 weight=1;
    server app.example.com:80 weight=2;
}
server {
    location / {
        proxy_pass http://backend;
    }
}
```

Эта конфигурация выполняет балансировку нагрузки по двум HTTP-серверам на порту 80. Параметр `weight` указывает NGINX передавать вдвое больше подключений на второй сервер, а значение этого параметра по умолчанию устанавливается на 1.

Обсуждение

Модуль `upstream` управляет балансировкой нагрузки для HTTP. Этот модуль определяет пул адресатов – любую комбинацию сокетов Unix, IP-адресов и записей DNS или их комбинацию. Модуль `upstream` также определяет, каким образом любой отдельный запрос назначается любому из вышестоящих серверов.

Каждый получатель в восходящем потоке определяется в восходящем пуле директивой `server`. В этой директиве указывается сокет Unix, IP-адрес или полное доменное имя, а также ряд необязательных параметров. Необязательные параметры дают больший контроль над маршрутизацией запросов. Эти параметры включают в себя вес сервера в алгоритме балансировки; находится ли сервер в режиме ожидания, доступен он или недоступен; и как определить, что сервер недоступен. NGINX Plus предоставляет ряд других удобных параметров, таких как ограничения подключения к серверу, расширенный контроль над DNS-преобразованием и возможность медленного увеличения количества подключений к серверу после его запуска.

2.2. Балансировка нагрузки для TCP

Задача

Вам необходимо распределить нагрузку между двумя или более серверами TCP.

Решение

Используйте модуль `stream` для балансировки нагрузки на TCP-серверы с использованием блока `upstream`:

```
stream {
    upstream mysql_read {
        server read1.example.com:3306 weight=5;
        server read2.example.com:3306;
        server 10.10.12.34:3306 backup;
    }

    server {
        listen 3306;
        proxy_pass mysql_read;
    }
}
```

Блок `server` в этом примере дает NGINX команду слушать TCP-порт 3306, выполняет балансировку нагрузки между двумя репликами на чтение базы данных MySQL и перечисляет еще одну в качестве резервной копии, в которой будет передаваться трафик, если указанные первичные будут остановлены. Эту конфигурацию не нужно добавлять в папку `conf.d`, поскольку она включена в блок `http`; вместо этого следует создать другую папку с именем `stream.conf.d`, открыть блок `stream` в файле `nginx.conf` и включить в состав новую папку для потоковых конфигураций.

Обсуждение

Балансировка нагрузки TCP определяется модулем `stream`. Модуль `stream`, как и модуль HTTP, позволяет определять пулы вышестоящих серверов и настраивать слушающий сервер. При настройке сервера для прослушивания данного порта вы должны определить порт для прослушивания или – по желанию – адрес и порт. Оттуда нужно настроить

адресат, будь то прямой обратный прокси-сервер на другой адрес или восходящий пул ресурсов.

Блок `upstream` для балансировки нагрузки TCP очень похож на тот, что используется для протокола HTTP, в том смысле, что он определяет восходящие ресурсы как серверы, настроенные с использованием сокета Unix, IP-адреса или полностью определенного доменного имени (FQDN), а также вес сервера, максимальное количество подключений, DNS-решатели и периоды вывода в рабочий режим; а также является ли сервер активным, выключен или находится в режиме резервного копирования.

NGINX Plus предлагает еще больше возможностей для балансировки нагрузки для протокола TCP.

Эти расширенные функции, предлагаемые в NGINX Plus, можно встретить на протяжении всей книги. Проверка работоспособности всей балансировки нагрузки будет рассмотрена позже в этой главе.

2.3. Балансировка нагрузки UDP

Задача

Вам необходимо распределить балансировку нагрузку между двумя или более серверами UDP.

Решение

Используйте модуль `stream` для балансировки нагрузки по UDP-серверам, взяв блок `upstream`, определенный как udp:

```
stream {
    upstream ntp {
        server ntp1.example.com:123 weight=2;
        server ntp2.example.com:123;
    }

    server {
        listen 123 udp;
        proxy_pass ntp;
    }
}
```

Данный раздел распределяет нагрузку между двумя вышестоящими NTP-серверами, использующими протокол UDP. Задать балансировку

нагрузки для UDP так же просто, как использовать параметр `udp` в директиве `listen`.

Если службе, для которой выполняется балансировка нагрузки, требуется отправить несколько пакетов туда и обратно между клиентом и сервером, можно указать параметр `reuseport`. Среди этих служб: OpenVPN, VoIP, решения для виртуальных рабочих столов и протокол датаграмм безопасности транспортного уровня (DTLS). Ниже приводится пример использования NGINX для обработки соединений Open VPN и их посредничества (прокси) для запущенной локально службы Open VPN:

```
stream {
    server {
        listen 1195 udp reuseport;
        proxy_pass 127.0.0.1:1194;
    }
}
```

Обсуждение

Вы можете спросить: «Зачем нужен балансировщик нагрузки, когда у меня может быть несколько хостов в записи DNS A или SRV?» Ответ заключается в том, что есть не только альтернативные алгоритмы балансировки, с помощью которых выполняют балансировку нагрузки, а можно делать это поверх самих DNS-серверов. Службы UDP составляют большую часть служб, от которых мы зависим в таких сетевых системах, как DNS, NTP и VoIP. Балансировка нагрузки UDP для кого-то может быть менее привычной, но столь же полезной в мире масштабирования.

Можно найти балансировку нагрузки для протокола UDP в модуле `stream`, как и для TCP, и настроить его в основном таким же образом. Главное отличие состоит в том, что директива `listen` указывает на то, что открытый сокет предназначен для работы с датаграммами. При работе с датаграммами существуют другие директивы, которые могут применяться там, где их нет в TCP, например директива `proxy_response`, указывающая NGINX, сколько ожидаемых ответов можно отправить с вышестоящего сервера. По умолчанию эта цифра не ограничена, до тех пор пока не будет достигнут лимит времени `proxy_time out`.

Параметр `reuseport` дает NGINX указание создать индивидуальный сокет прослушивания для каждого рабочего процесса. Это позволяет ядру распределять входящие соединения между рабочими процессами для обработки нескольких пакетов, которые отправляются между кли-

ентом и сервером. `reuseport` работает только на ядрах Linux 3.9 и выше, DragonFly BSD и FreeBSD 12 и выше.

2.4. Методы балансировки нагрузки

Задача

Циклический (round-robin) метод балансировки нагрузки не подходит для вашего варианта использования, потому что у вас разные рабочие нагрузки или пулы серверов.

Решение

Используйте один из методов балансировки нагрузки NGINX, такой как наименьшее количество подключений, наименьшее время, общее хеширование, хеширование IP или случайный метод:

```
upstream backend {
    least_conn;
    server backend.example.com;
    server backend1.example.com;
}
```

В этом примере для алгоритма балансировки нагрузки для внутреннего восходящего пула задано наименьшее количество активных подключений. Все алгоритмы распределения нагрузки, за исключением общего хеша, случайного и наименьшего времени, – это отдельные директивы, такие как в предыдущем примере. Параметры этих директив объясняются далее.

Обсуждение

Не все запросы или пакеты несут равный вес. Учитывая это, циклический или даже взвешенный циклический метод, использованный в предыдущих примерах, не будет отвечать потребностям всех приложений или потока трафика. NGINX предоставляет ряд алгоритмов распределения нагрузки, которые можно использовать для соответствия конкретным случаям использования. Помимо возможности выбора этих алгоритмов или методов балансировки нагрузки, вы также можете настроить их. Для восходящих пулов HTTP, TCP и UDP доступны следующие методы балансировки нагрузки.

Циклический алгоритм

Это метод балансировки нагрузки по умолчанию, который распределяет запросы в порядке списка серверов в восходящем пуле. Вы также можете принять во внимание вес для взвешенного карусельного метода, который можно использовать, если емкости вышестоящих серверов разнятся. Чем выше целочисленное значение для такого веса, тем более предпочтительным будет сервер в данном карусельном методе. Алгоритм, стоящий за весом, – это просто статистическая вероятность некоего среднего взвешенного.

Наименьшее количество подключений

Данный метод распределяет нагрузку, выступая посредником для текущего запроса к вышестоящему серверу с наименьшим количеством открытых подключений. Этот метод, равно как и карусельный, учитывает веса при принятии решения, в какой сервер отправлять подключение. Имя директивы – `least_conn`.

Наименьшее время

Доступный только в NGINX PLUS, метод наименьшего времени является родственным методу наименьшего количества подключений в том, что он выступает посредником для того сервера, у которого наименьшее число подключений, но при этом предпочтение отдается серверу с наименьшим значением среднего времени обработки запросов. Данный метод один из самых искушенных алгоритмов балансировки нагрузки и соответствует потребностям веб-приложений с высокой производительностью. Этот алгоритм дополнительно оценивает наименьшее время соединения, поскольку некое меньшее число подключений не обязательно означает самый быстрый отклик. Для данной директивы необходимо определять параметр `header` или `last_byte`. Когда определен `header`, используется время на получение отклика от заголовка. Когда определен `last_byte`, применяется значение времени для получения полного отклика. Имя директивы – `least_time`.

Общее хеширование

Администратор определяет хеш с заданным текстом, переменными запроса или времени выполнения или тем и другим. NGINX распределяет нагрузку между серверами, создавая хеш для текущего запроса и помещая его в вышестоящие серверы. Этот метод очень полезен, когда вам нужен дополнительный контроль над

тем, куда отправляются запросы, или для определения того, какой вышестоящий сервер, скорее всего, будет кешировать данные. Обратите внимание, что при добавлении или удалении сервера из пула хешированные запросы будут перераспределены. Этот алгоритм имеет необязательный параметр, `consistent`, чтобы минимизировать эффект перераспределения. Имя директивы – `hash`.

Случайный метод

Данный метод используется, чтобы дать NGINX команду выбрать произвольный сервер из группы, учитывая вес сервера. Необязательный параметр `two [method]` указывает NGINX случайным образом выбрать два сервера, а затем использовать предоставленный метод балансировки нагрузки для распределения нагрузки между ними. По умолчанию используется `least_conn`, если `two` передается без метода. Имя директивы для случайной балансировки нагрузки – `random`.

Хеширование IP-адреса

Этот метод работает только для HTTP. Он использует значение IP-адреса клиента в качестве самого хеширования. Несколько отличаясь от использования удаленной переменной в общем хешировании, этот алгоритм использует первые три октета адреса IPv4 или весь адрес IPv6 целиком. Метод гарантирует, что клиенты выставляются посредником на один и тот же вышестоящий сервер, до тех пор пока этот сервер доступен, что очень полезно, когда состояние сеанса имеет значение и не обрабатывается общей памятью приложения. Этот метод также учитывает параметр `weight` при распределении значения хеша. Имя директивы – `ip_hash`.

2.5. Директива sticky cookie

Задача

Вам необходимо привязать нижестоящего клиента к вышестоящему серверу, используя NGINX Plus.

Решение

Используйте директиву `sticky cookie`, чтобы дать указание NGINX Plus создать и отслеживать куки:

```
upstream backend {
    server backend1.example.com;
    server backend2.example.com;
    sticky cookie
           affinity
           expires=1h
           domain=.example.com
           httponly
           secure
           path=/;
}
```

В этой конфигурации мы создаем и отслеживаем куки, который связывает нижестоящего клиента с вышестоящим сервером. В этом примере файл куки носит название `affinity`. Он устанавливается для example.com со сроком истечения один час, его нельзя использовать на стороне клиента, можно отправлять только по протоколу HTTPS, и он действителен для всех путей.

Обсуждение

При использовании параметра `cookie` в директиве `sticky` создается куки при первом запросе, который содержит информацию о вышестоящем сервере. NGINX Plus отслеживает этот куки-файл, позволяя ему и дальше направлять последующие запросы на тот же сервер. Первый позиционный параметр для параметра `cookie` – имя куки-файла, который мы создадим и будем отслеживать. Другие параметры предлагают дополнительный контроль, информируя браузер о соответствующем использовании, например о времени истечения, домене, пути и возможности использования куки на стороне клиента либо можно ли передавать его по незащищенным протоколам.

2.6. Директива sticky learn

Задача

Вам необходимо привязать нижестоящего клиента к вышестоящему серверу, используя существующий куки-файл с помощью NGINX Plus.

Решение

Используйте директиву `sticky learn` для обнаружения и отслеживания куки, которые создаются восходящим приложением:

```
upstream backend {
    server backend1.example.com:8080;
    server backend2.example.com:8081;

    sticky learn
           create=$upstream_cookie_cookiename
           lookup=$cookie_cookiename
           zone=client_sessions:2m;
}
```

В этом примере NGINX дано указание искать и отслеживать сеансы путем поиска куки с именем `COOKIENAME` в заголовках ответа и осуществлять поиск существующих сеансов путем поиска того же куки в заголовках запросов. Это сходство сеанса хранится в зоне общей памяти размером 2 Мб, которая может отслеживать около 16 000 сеансов. Имя куки всегда будет зависеть от приложения. Обычно используемые имена, такие как `jsessionid` или `phpsessionid`, – как правило, значения по умолчанию, установленные в приложении или конфигурации сервера приложений.

Обсуждение

Когда приложения создают собственные куки состояния сеанса, NGINX Plus может обнаружить их в ответах на запросы и отслеживать их. Этот тип отслеживания куки выполняется, если директиве `sticky` предоставлен параметр `learn`. Общая память для отслеживания куки указывается при использовании параметра `zone` с именем и размером. NGINX Plus направлен на поиск куки в ответе, полученном от вышестоящего сервера через спецификацию параметра `create`, и ищет ранее зарегистрированную привязку к серверу, используя параметр `lookup`. Значением этих параметров являются переменные, предоставляемые модулем `HTTP`.

2.7. Директива sticky route

Задача

Вам необходим детальный контроль над вашими постоянными сеансами к вышестоящему серверу с помощью NGINX Plus.

Решение

Используйте директиву `sticky` с параметром `route`, чтобы применить переменные относительно запроса на маршрут:

```
map $cookie_jsessionid $route_cookie {
    ~.+\.(?P<route>\w+)$ $route;
}

map $request_uri $route_uri {
    ~jsessionid=.+\.(?P<route>\w+)$ $route;
}

upstream backend {
    server backend1.example.com route=a;
    server backend2.example.com route=b;

    sticky route $route_cookie $route_uri;
}
```

В этом примере вы пытаетесь извлечь идентификатор сеанса Java, сначала из куки устанавливая соответствие самого идентификатора сеанса Java некой переменной при помощи самого первого блока `map`, а затем путем поиска в URI запроса параметра с именем `jsessionid`, устанавливая соответствие полученного значения переменной при помощи второго блока `map`. Директива `sticky` с параметром `route` передает любое количество переменных. Первое ненулевое или непустое значение берется для маршрута. Если используется файл `jsessionid`, запрос направляется на `backend1`; если используется параметр URI, запрос направляется на `backend2`. Хотя этот пример основан на распространенном идентификаторе сеанса Java, то же самое применительно и к другой технологии сеанса, такой как `phpsessionid`, или любому гарантированному уникальному идентификатору, который ваше приложение генерирует для идентификатора сеанса.

Обсуждение

Иногда вам может потребоваться направить трафик на конкретный сервер, располагая при этом более детальным контролем. Параметр `route` для директивы `sticky` создан для достижения этой цели. Sticky route обеспечивает более эффективный контроль, фактическое отслеживание и связываемость в отличие от алгоритма балансировки нагрузки с общим хешированием. Сначала клиент направляется в вышестоящий сервер на основе указанного маршрута, а затем последовательные запросы будут содержать информацию о маршрутизации в куки или URI. Sticky route принимает ряд позиционных параметров, которые оцени-

ваются. Первая непустая переменная используется для маршрутизации на сервер. Блоки `map` могут использоваться для выборочного анализа переменных и сохранения их в качестве других переменных, которые будут использоваться в маршрутизации. По сути, директива `sticky route` создает сеанс внутри зоны общей памяти NGINX Plus для отслеживания любого идентификатора сеанса клиента, указанного вами для вышестоящего сервера, последовательно доставляя запросы с этим идентификатором сеанса тому же вышестоящему серверу, что и его исходный запрос.

2.8. Осушение соединения

Задача

Вам необходимо аккуратно удалить серверы для обслуживания или по другим причинам, продолжая обслуживать сеансы с помощью NGINX Plus.

Решение

Используйте параметр `drain` через API NGINX Plus, подробно описанный в главе 5, чтобы NGINX прекратил отправлять новые подключения, которые еще не отслеживаются:

```
$ curl -X POST -d '{"drain":true}' \
'    http://nginx.local/api/3/http/upstreams/backend/servers/0'

{
    "id":0,
    "server":"172.17.0.3:80",
    "weight":1,
    "max_conns":0,
    "max_fails":1,
    "fail_timeout":
    "10s","slow_start":
    "0s",
    "route":"",
    "backup":false,
    "down":false,
    "drain":true
}
```

Обсуждение

Когда состояние сеанса хранится локально на сервере, соединения и постоянные сеансы должны быть очищены перед его удалением из пула. Осушение соединений – это процесс, который позволяет естественным образом завершаться сеансам сервера по истечении времени, прежде чем удалять сам сервер из соответствующего пула восходящего потока. Можно настроить осушение для конкретного сервера, добавив параметр `drain` в директиву `server`. Когда задан параметр `drain`, NGINX Plus прекращает отправку новых сеансов на этот сервер, но позволяет текущим сеансам продолжать обслуживаться в течение продолжительности их сеанса. Вы также можете переключить конфигурацию, добавив параметр `drain` в директиву вышестоящего сервера.

2.9. Пассивные проверки работоспособности

Задача

Вам необходимо выполнить пассивную проверку работоспособности вышестоящих серверов.

Решение

Используйте проверку работоспособности NGINX с балансировкой нагрузки, чтобы убедиться, что применяются только работоспособные вышестоящие серверы:

```
upstream backend {
    server backend1.example.com:1234 max_fails=3 fail_timeout=3s;
    server backend2.example.com:1234 max_fails=3 fail_timeout=3s;
}
```

В этой конфигурации мы пассивно отслеживаем работоспособность, устанавливая для директивы `max_fails` значение, равное трем, а для `fail_timeout` – три секунды. Эти параметры директив работают одинаково и в потоковых, и HTTP-серверах.

Обсуждение

Пассивная проверка работоспособности доступна в версии NGINX с открытым исходным кодом. Пассивный мониторинг отслеживает сбойные или отключенные соединения, когда они проходят через NGINX по запросу клиента. Пассивные проверки работоспособности активирова-

ны по умолчанию; упомянутые здесь параметры позволяют настроить их поведение. Мониторинг работоспособности важен для всех видов балансировки нагрузки, не только с точки зрения взаимодействия с пользователем, но и для обеспечения непрерывности бизнеса. NGINX выполняет пассивный мониторинг вышестоящих серверов HTTP, TCP и UDP, чтобы убедиться, что они работоспособны.

2.10. Активные проверки работоспособности

Задача

Вам необходимо выполнить активную проверку работоспособности ваших вышестоящих серверов с помощью NGINX Plus.

Решение

Для протокола HTTP используйте директиву `health_check` в блоке location:

```
http {
    server {
        ...
        location / {
            proxy_pass http://backend;
            health_check interval=2s
                fails=2
                passes=5
                uri=/
                match=welcome;
        }
    }
    # status is 200, content type is "text/html",
    # and body contains "Welcome to nginx!"
    match welcome {
        status 200;
        header Content-Type = text/html;
        body ~ "Welcome to nginx!";
    }
}
```

В этой конфигурации проверки работоспособности для HTTP-серверов мы проверяем работоспособность вышестоящих серверов, отправ-

ляя HTTP-запрос в URI `'/'` каждые две секунды. Вышестоящие серверы должны пройти пять последовательных проверок работоспособности, чтобы считаться исправными. Если они не проходят две последовательные проверки, то считаются неисправными. Ответ, полученный от вышестоящего сервера, должен соответствовать определенному блоку `match`, который определяет код состояния как 200, значение заголовка Content-Type как `'text/html'` и строку `"Welcome to nginx!"` в теле ответа. У блока `match` есть три директивы: статус, заголовок и тело. У всех этих директив также есть флаги сравнения.

Проверка работоспособности для служб TCP/UDP очень похожа:

```
stream {
    ...
    server {
        listen 1234;
        proxy_pass stream_backend;
        health_check interval=10s
            passes=2
            fails=3;
        health_check_timeout 5s;
    }
    ...
}
```

В данном примере сервер TCP настроен на прослушивание порта `1234` и выполнение посредничества к некоему набору серверов восходящего потока, для которого он и выполняет активные проверки жизнеспособности. Соответствующая директива `health_check` получает все те же параметры, что и HTTP, за исключением **uri**, а значение версии потока имеет параметр для переключения протокола проверки на **udp**. В данном примере значение интервала установлено на 10 с, требуются два прохода, для того чтобы он рассматривался как работоспособный, и три отказа, чтобы принимать его неработоспособным. Такая активная проверка потока также способна удостоверять получаемый отклик от своего сервера верхнего уровня. Тем не менее соответствующий блок **match** для серверов потока имеет только два параметра: `send` и `expect`. Значением директивы `send` являются подлежащие отправке сырые данные, в то время как `expect` представляет собой точный отклик или некое регулярное выражение для проверки соответствия.

Обсуждение

При активных проверках работоспособности в NGINX Plus постоянно отправляются запросы к исходным серверам, чтобы проверить их работоспособность. Эти проверки могут измерять больше, чем просто код ответа. В NGINX Plus, активный мониторинг работоспособности HTTP осуществляется на основе ряда критериев приемлемости ответа от вышестоящего сервера. Можно настроить активную проверку работоспособности для определения частоты проверки вышестоящих серверов, того, сколько раз сервер должен пройти эту проверку, чтобы считаться исправным, сколько раз он мог потерпеть неудачу, перед тем как считаться неисправным, и каким должен быть ожидаемый результат. Параметр `match` указывает на блок match, который определяет критерии приемлемости для ответа. Он также определяет данные для отправки на вышестоящий сервер при использовании в контексте потока для TCP/UDP. Эти функции позволяют NGINX постоянно обеспечивать работоспособность вышестоящих серверов.

2.11. Медленный запуск

Задача

Ваше приложение нуждается в прогреве перед полной эксплуатационной нагрузкой.

Решение

Используйте параметр `slow_start` в директиве `server`, чтобы постепенно увеличивать количество подключений за указанное время, когда сервер повторно вводится в восходящий пул балансировки нагрузки:

```
upstream {
    zone backend 64k;
    server server1.example.com slow_start=20s;
    server server2.example.com slow_start=15s;
}
```

Данная настройка директивы `server` будет медленно прогревать трафик к вышестоящим серверам после их представления в пуле. `server1` будет плавно наращивать количество подключений в течение 20 с, а `server2` – в течение 15 с.

Обсуждение

Медленный запуск – это концепция постепенного увеличения количества запросов, передаваемых на сервер в течение определенного периода времени. Медленный запуск позволяет прогревать приложение, заполняя кеши, инициируя соединения с базами данных без перегрузки соединениями, как только он запускается. Это свойство вступает в силу, когда сервер, который не прошел проверку работоспособности, снова начинает проходить ее и возвращается в пул балансировки нагрузки.

2.12. Проверки работоспособности TCP

Задача

Вам необходимо проверить работоспособность вышестоящего TCP-сервера и удалить неисправные серверы из пула.

Решение

Используйте директиву `health_check` в блоке `server` для активной проверки работоспособности:

```
stream {
    server {
        listen       3306;
        proxy_pass   read_backend;
        health_check interval=10 passes=2 fails=3;
    }
}
```

В этом примере активно отслеживаются вышестоящие серверы. Вышестоящий сервер будет считаться неисправным, если не сможет ответить на три или более TCP-соединения, инициированных NGINX. NGINX выполняет проверку каждые 10 с. Сервер будет считаться исправным только после двух проверок работоспособности.

Обсуждение

Работоспособность TCP можно проверить с помощью NGINX Plus как пассивно, так и активно. Пассивный мониторинг работоспособности осуществляется путем регистрации обмена данными между клиентом и вышестоящим сервером. Если вышестоящий сервер приводит к тайм-ауту или отклоняет подключения, пассивная проверка работоспо-

собности сочтет этот сервер неисправным. Активные проверки работоспособности инициируют собственные настраиваемые проверки для определения работоспособности; они не только проверяют соединение с вышестоящим сервером, но и могут ожидать определенного ответа.

Глава 3

Управление трафиком

3.0. Введение

NGINX и NGINX Plus также классифицируются как контроллеры веб-трафика. Можно использовать NGINX для интеллектуальной маршрутизации трафика и управления потоком на базе множества атрибутов. В этой главе рассматривается способность NGINX разделять клиентские запросы на основе процентов, использовать географическое расположение клиентов и контролировать поток трафика в виде ограничения скорости, соединения и пропускной способности. Читая эту главу, помните, что можно смешивать и сочетать эти функции, чтобы получить неисчерпаемые возможности.

3.1. A/B-тестирование

Задача

Вам нужно разделить клиентов между двумя или более версиями файла или приложения для проверки приемлемости.

Решение

Используйте модуль `split_clients`, чтобы направить процент своих клиентов в другой восходящий пул:

```
split_clients "${remote_addr}AAA" $variant {
    20.0%    "backendv2";
    *        "backendv1";
}
```

Директива `split_clients` хеширует строку, которую вы предоставили в качестве первого параметра, и делит этот хеш на предоставленные проценты, чтобы отобразить значение переменной, которая приводится в

качестве второго параметра. Третий параметр – это объект, содержащий пары ключ/значение, где ключ – это процентный вес, а значение – это величина, которая должна быть назначена. Ключ может быть обозначен либо в процентах, либо в виде звездочки. Звездочка обозначает остаток целого, после того как приняты все проценты. Значение переменной `$variant` будет составлять `backendv2` для 20 % клиентских IP-адресов и `backendv1` для оставшихся 80 %.

В этом примере `backendv1` и `backendv2` обозначают пулы вышестоящих серверов и могут использоваться с директивой `proxy_pass`:

```
location / {
    proxy_pass http://$variant
}
```

При использовании переменной `$variant` наш трафик будет расщеплен между двумя различными пулами серверов.

Обсуждение

Данный тип A/B-тестирования полезен при тестировании различных типов функций маркетинга и внешнего интерфейса для показателей конверсии на сайтах электронной коммерции. Обычно приложения используют тип развертывания под названием *canary release*. В этом типе развертывания трафик медленно переключается на новую версию. Разделение клиентов между различными версиями своего приложения может быть полезно при развертывании новых версий кода, чтобы ограничить радиус взрыва в случае ошибки. Независимо от причины разделения клиентов между двумя различными наборами приложений NGINX упрощает этот процесс благодаря использованию модуля `split_clients`.

См. также:

Документация по split_clients (http://nginx.org/en/docs/http/ngx_http_split_clients_module.html)

3.2. Использование модуля GeoIP и базы данных

Задача

Вам необходимо установить базу данных GeoIP и активировать ее встраиваемые переменные в NGINX для регистрации и указания вашему приложению местоположения ваших клиентов.

Решение

Официальный репозиторий пакетов NGINX с открытым исходным кодом, который мы настраивали в главе 1 при установке NGINX, предоставляет пакет `nginx-module-geoip`. При использовании репозитория пакетов NGINX Plus этот пакет называется `nginx-plus-modulegeoip`. Эти пакеты устанавливают динамическую версию модуля GeoIP.

RHEL/CentOS NGINX с открытым исходным кодом:

```
# yum install nginx-module-geoip
```

Debian/Ubuntu NGINX с открытым исходным кодом:

```
# apt-get install nginx-module-geoip
```

RHEL/CentOS NGINX Plus:

```
# yum install nginx-plus-module-geoip
```

Debian/Ubuntu NGINX Plus:

```
# apt-get install nginx-plus-module-geoip
```

Скачайте базы данных GeoIP стран и городов и распакуйте их:

```
# mkdir /etc/nginx/geoip
# cd /etc/nginx/geoip
# wget "http://geolite.maxmind.com/\
download/geoip/database/GeoLiteCountry/GeoIP.dat.gz"
# gunzip GeoIP.dat.gz
# wget "http://geolite.maxmind.com/\
download/geoip/database/GeoLiteCity.dat.gz"
# gunzip GeoLiteCity.dat.gz
```

С помощью этого набора команд мы создаем каталог *geoip* в каталоге */etc/nginx*, перемещаемся туда, скачиваем и распаковываем пакеты.

Теперь, когда база данных GeoIP стран и городов находится на локальном диске, вы можете дать указание модулю GeoIP использовать ее для предоставления встраиваемых переменных на основе IP-адреса клиента:

```
load_module "/usr/lib64/nginx/modules/ngx_http_geoip_module.so";

http {
```

```
    geoip_country /etc/nginx/geoip/GeoIP.dat;
    geoip_city /etc/nginx/geoip/GeoLiteCity.dat;
...
}
```

Директива `load_module` динамически загружает модуль из его пути в файловой системе. Она действительна только в основном контексте. Директива `geoip_country` берет путь к файлу *GeoIP.dat*, содержащему базу данных, отображая IP-адреса в коды стран, и действительна только в контексте протокола HTTP.

Обсуждение

Директивы `geoip_country` и `geoip_city` предоставляют ряд встраиваемых переменных, доступных в этом модуле. Директива `geoip_country` активирует переменные, которые позволяют различать страну происхождения вашего клиента. Эти переменные включают в себя `$geoip_country_code`, `$geoip_country_code3` и `$geoip_country_name`. Переменная кода страны возвращает код страны из двух букв, а переменная с цифрой 3 на конце возвращает код страны, состоящий из трех букв.

Переменная имени страны возвращает полное название страны.

Директива `geoip_city` активирует довольно много переменных. Она активирует все те же переменные, что и директива `geoip_country`, только с другими именами, такими как `$geoip_city_country_code`, `$geoip_city_country_code3` и `$geoip_city_country_name`. Другие переменные включают в себя `$geoip_city`, `$geoip_city_continent_code`, `$geoip_latitude`, `$geoip_longitude` и `$geoip_postal_code` – все они описывают возвращаемое значение. `$geoip_region` и `$geoip_region_name` описывают регион, территорию, штат, провинцию, федеральную землю и т. п. Регион – это код из двух букв, где название региона – полное имя. `$geoip_area_code` действительна только в США. Она возвращает трехзначный телефонный код города.

Используя эти переменные, вы можете регистрировать информацию о вашем клиенте.

При желании можно передать эту информацию своему приложению в качестве заголовка или переменной или использовать NGINX для маршрутизации своего трафика особым образом.

См. также:

Обновления для GeoIP (https://github.com/maxmind/geoipupdate).

3.3. Ограничение доступа в зависимости от страны

Задача

Вам необходимо ограничить доступ из определенных стран согласно требованиям договора или приложения.

Решение

Отобразите коды стран, которые вы хотите заблокировать или разрешить в переменную:

```
load_module
  "/usr/lib64/nginx/modules/ngx_http_geoip_module.so";

http {
    map $geoip_country_code $country_access {
        "US"    0;
        "RU"    0;
        default 1;
    }
    ...
}
```

В результате этого значение новой переменной `$country_access` будет равно 1 или 0. Если клиентский IP-адрес происходит из США или России, значение переменной будет установлено на 0. Для любой другой страны значение переменной будет установлено на 1. Теперь в нашем блоке `server` мы будем использовать оператор `if`, чтобы запретить доступ любому, кто не из США или России:

```
server {
    if ($country_access = '1') {
      return 403;
    }
    ...
}
```

Если значение переменной `$country_access` установлено на 1, оператор `if` будет оценивать это как `True`. В этом случае сервер вернет ошибку 403 unauthorized. В противном случае сервер будет работать как обычно. Та-

ким образом, блок с `if` применяется, только чтобы отказать в доступе лицам не из США или России.

Обсуждение

Это короткий, но простой пример того, как разрешить доступ только лицам из двух стран. Его можно развить в зависимости от ваших потребностей. Можно использовать этот же метод, чтобы разрешить или заблокировать доступ на основе любой из встраиваемых переменных, доступных из модуля GeoIP.

3.4. Поиск исходного клиента

Задача

Вам необходимо найти исходный IP-адрес клиента, поскольку перед сервером NGINX находятся прокси-сервера.

Решение

Используйте директиву `geoip_proxy` для определения диапазона IP-адресов прокси-сервера и директиву `geoip_proxy_recursive` для поиска исходного IP-адреса:

```
load_module "/usr/lib64/nginx/modules/ngx_http_geoip_module.so";

http {
    geoip_country /etc/nginx/geoip/GeoIP.dat;
    geoip_city /etc/nginx/geoip/GeoLiteCity.dat;
    geoip_proxy 10.0.16.0/26;
    geoip_proxy_recursive on;
...
}
```

Директива `geoip_proxy` определяет диапазон бесклассовой адресации, в котором находятся наши прокси-серверы, и указывает NGINX использовать заголовок `X-Forwarded-For`, чтобы найти клиентский IP-адрес. Директива `geoip_proxy_recursive` дает NGINX указание рекурсивно просматривать заголовок `X-Forwarded-For`, чтобы найти последний известный IP-адреса клиента.

Обсуждение

Вы можете обнаружить, что при использовании прокси-сервера перед NGINX NGINX будет выбирать IP-адрес прокси, а не клиента. Для этого

можно использовать директиву `geoip_proxy`, чтобы указать NGINX взять заголовок `X-Forwarded-For`, когда соединения открываются из заданного диапазона. Директива `geoip_proxy` принимает адрес или диапазон бесклассовой адресации. Когда есть несколько прокси, передающих трафик перед NGINX, можно использовать директиву `geoip_proxy_recursive` для рекурсивного поиска по адресам `X-Forwarded-For`, чтобы найти исходящего клиента. Возможно, вы захотите использовать нечто подобное при использовании балансировщиков нагрузки, таких как AWS ELB, балансировщик нагрузки Google или Azure перед NGINX.

3.5. Ограничение подключений

Задача

Вам необходимо ограничить количество подключений на основе предопределенного ключа, такого как IP-адрес клиента.

Решение

Создайте зону общей памяти для удержания метрик подключений и используйте директиву `limit_conn` для ограничения открытых соединений:

```
http {
    limit_conn_zone $binary_remote_addr zone=limitbyaddr:10m;
    limit_conn_status 429;
    ...
    server {
        ...
            limit_conn limitbyaddr 40;
        ...
    }
}
```

В этой конфигурации мы создаем зону общей памяти с именем `limitbyaddr`. В качестве предопределенного ключа используется IP-адрес клиента в двоичном виде. Размер зоны общей памяти установлен на 10 Мб. Директива `limit_conn` принимает два параметра: имя `limit_conn_zone` и количество разрешенных соединений. Директива `limit_conn_status` устанавливает ответ, когда соединения ограничены статусом 429, что указывает на слишком большое количество запросов. Директивы `limit_conn` и `limit_conn_status` действительны в контексте HTTP, сервера и местоположения.

Обсуждение

Ограничение количества соединений на основе ключа может быть использовано для защиты от злоупотреблений и справедливого распределения ресурсов между всеми вашими клиентами. Важно быть осторожным с предопределенным ключом. Использование IP-адреса, как мы это делали в предыдущем примере, может быть опасным, если в одной сетевой среде, происходящей из одного и того же IP-адреса, находится большое количество пользователей, например когда речь идет о *преобразовании сетевых адресов*. Вся группа клиентов будет ограничена. Директива `limit_conn_zone` действует только в контексте HTTP. Вы можете использовать любое количество переменных, доступных для NGINX в контексте HTTP, чтобы создать строку, применяемую для ограничения. Использование переменной, которая может идентифицировать пользователя на уровне приложения, например куки-файла сессии, может быть более чистым решением в зависимости от варианта использования. Директива `limit_conn_status` по умолчанию выдает ошибку 503, service unavailable. Возможно, вы предпочтете использовать 429, поскольку служба доступна и ответы с кодом 5xx указывают на ошибку сервера, тогда как ответы с кодом 4xx указывают на ошибку клиента.

3.6. Ограничение скорости

Задача

Вам необходимо ограничить скорость запросов с помощью предварительно определенного ключа, такого как IP-адрес клиента.

Решение

Используйте модуль ограничения скорости, чтобы ограничить скорость запросов:

```
http {
    limit_req_zone $binary_remote_addr
        zone=limitbyaddr:10m rate=1r/s;
    limit_req_status 429;
    ...
    server {
        ...
            limit_req zone=limitbyaddr burst=10 nodelay;
```

```
        ...
    }
}
```

В этом примере конфигурации мы создаем зону общей памяти с именем `limitbyaddr`. Предопределенный ключ – это IP-адрес клиента в двоичной форме. Размер зоны общей памяти установлен на 10 Мб. Зона устанавливает скорость с помощью аргумента ключевого слова. Директива `limit_req` принимает два необязательных аргумента ключевого слова: `zone` и `burst`. `zone` необходим, чтобы указать директиве, какую зону ограничения запросов к общей памяти использовать. Когда скорость запросов для данной зоны превышена, запросы задерживаются, до тех пор пока не будет достигнут их максимальный размер пакета, обозначенный аргументом ключевого слова `burst`. Аргумент ключевого слова `burst` по умолчанию равен нулю. `limit_req` также принимает третий необязательный параметр, `nodelay`. Этот параметр позволяет клиенту использовать свой пакет без задержки до ограничений. `limit_req_status` устанавливает состояние, возвращаемое клиенту как определенный код состояния HTTP; по умолчанию это 503. `limit_req_status` и `limit_req` действительны в контексте HTTP, сервера и местоположения. Директива `limit_req_zone` действительна только в контексте HTTP. Ограничение скорости поддерживает кластеры в NGINX Plus, новая функция в версии R16.

Обсуждение

Модуль ограничения скорости – очень мощное средство для защиты от чрезмерно быстрых запросов, в то же время он предоставляет качественное обслуживание всем. Есть много причин для ограничения скорости запроса, и одна из них – это безопасность. Можно отбить атаку методом полного перебора, установив очень строгое ограничение на своей странице входа. Можно установить разумный лимит для всех запросов, тем самым нарушив планы злоумышленников, которые могут попытаться нарушить обслуживание вашего приложения или растратить ресурсы. По конфигурации модуль ограничения скорости очень похож на предыдущий модуль ограничения соединений, описанный в рецепте 3.5, и большая часть этих концепций применима и здесь. Можно указать скорость, с которой ограничиваются запросы, в запросах в секунду или в минуту. Когда ограничение скорости достигнуто, инцидент регистрируется. Существует также директива, которой нет в примере, `limit_req_log_level`, – она по умолчанию выдает ошибку, но это

может быть и информационное сообщение, уведомление или предупреждение. Новинка: в версии R16 NGINX Plus ограничение скорости теперь поддерживает кластеры (см. рецепт 12.5, где приводится пример синхронизации зоны).

3.7. Ограничение пропускной способности

Задача

Вам необходимо ограничить пропускную способность скачивания для каждого клиента для своих активов.

Решение

Используйте директивы `limit_rate` и `limit_rate_after`, чтобы ограничить скорость ответа клиенту:

```
location /download/ {
    limit_rate_after 10m;
    limit_rate 1m;
}
```

Конфигурация этого блока местоположения указывает, что для URI с префиксом *download* скорость, с которой ответ отправится клиенту, будет ограничена после 10 Мб до скорости 1 Мб/с. Ограничение пропускной способности осуществляется для каждого соединения, поэтому вы можете захотеть установить ограничение соединения, а также ограничение пропускной способности, где это применимо.

Обсуждение

Ограничение пропускной способности для определенных соединений позволяет NGINX распределять пропускную способность загрузки между всеми клиентами таким образом, который вы указываете. Все это делают две директивы: `limit_rate_after` и `limit_rate`. Директиву `limit_rate_after` можно установить практически в любом контексте: HTTP, сервер, местоположение и оператор `if`, если он находится внутри местоположения. Директива `limit_rate` применима в тех же контекстах, что и `limit_rate_after`; однако ее также можно настроить путем установки переменной с именем `$limit_rate`. Директива `limit_rate_after` указывает, что скорость соединения не должна ограничиваться, до тех пор пока не будет передан определенный объем данных. Директива `limit_rate` опре-

деляет ограничение скорости для данного контекста в байтах в секунду по умолчанию. Однако вы можете указать `m` для обозначения мегабайт или `g` для обозначения гигабайт. В обеих директивах по умолчанию установлено значение 0 – оно означает, что скорость скачивания вообще не ограничивается. Этот модуль позволяет программно изменять ограничение скорости клиентов.

Глава 4

Массивно масштабируемое кеширование контента

4.0. Введение

Кеширование ускоряет обслуживание контента, сохраняя ответы на запросы, которые будут снова обслуживаться в будущем. Кеширование контента снижает нагрузку на вышестоящие серверы, кешируя полный ответ, вместо того чтобы выполнять вычисления и снова делать один и тот же запрос. Кеширование повышает производительность и снижает нагрузку, а это означает, что вы можете выполнять обслуживание быстрее с меньшим количеством ресурсов. Масштабирование и распределение серверов кеширования в стратегических местах может оказать существенное влияние на пользовательский опыт. Это оптимально – размещать контент ближе к потребителю для обеспечения лучшей производительности. Вы также можете кешировать свой контент рядом со своими пользователями. Это шаблон сетей доставки содержимого или CDN. С помощью NGINX вы кешируете свой контент везде, где можно разместить сервер NGINX, что позволяет вам эффективно создавать собственную сеть доставки содержимого. С помощью кеширования в NGINX вы также можете пассивно кешировать и обслуживать кешированные ответы в случае отказа восходящего потока.

4.1. Кеширование зон

Задача

Вам необходимо кешировать контент и определять, где хранится кеш.

Решение

Используйте директиву `proxy_cache_path`, чтобы определить зоны кеша общей памяти и расположение контента:

```
proxy_cache_path /var/nginx/cache
                 keys_zone=CACHE:60m
                 levels=1:2
                 inactive=3h
                 max_size=20g;
proxy_cache CACHE;
```

В этом примере определения кеша мы создаем каталог для кешированных ответов в файловой системе в */var/nginx/cache* и пространство общей памяти с именем `CACHE` с 60 Мб памяти. Мы устанавливаем уровни структуры каталогов, определяем выпуск кешированных ответов после того, как они не были запрошены в течение 3 ч, и определяем максимальный размер кеша в 20 Гб. Директива `proxy_cache` сообщает конкретному контексту использовать зону кеширования. Директива `proxy_cache_path` действительна в контексте HTTP, а директива `proxy_cache` – в контексте HTTP, сервера и местоположения.

Обсуждение

Чтобы настроить кеширование в NGINX, необходимо объявить путь и зону для использования. Зона кеширования в NGINX создается с помощью директивы `proxy_cache_path`. Она обозначает местоположение, где будет храниться кешированная информация, и пространство общей памяти, где будут храниться активные ключи и метаданные ответа. Необязательные параметры для этой директивы обеспечивают больший контроль над тем, как поддерживается кеш и как к нему выполняется доступ. Параметр `levels` определяет, как создается структура файла. Значение – это разделенная двоеточиями величина, которая объявляет длину имен подкаталогов, с максимумом в три уровня. NGINX осуществляет кеширование на основе ключа кеша, который представляет собой хешированное значение. Затем NGINX сохраняет результат в предоставленной файловой структуре, используя ключ кеша в качестве пути к файлу и разбивая каталоги на основе значения `levels`. Параметр `inactive` позволяет контролировать время, в течение которого элемент кеша будет размещаться после его последнего использования. Размер кеша настраивается с использованием параметра `max_size`. Другие параметры

относятся к процессу загрузки кеша, который загружает ключи кеша в зону общей памяти из файлов, кешированных на диске.

4.2. Хеш-ключи кеширования

Задача

Вам необходимо контролировать, как ваш контент кешируется и просматривается.

Решение

Используйте директиву `proxy_cache_key` наряду с переменными, чтобы определить, на чем основывается кеширование – на попадании или на промахе:

```
proxy_cache_key "$host$request_uri $cookie_user";
```

Этот хеш-ключ будет указывать NGINX кешировать страницы на основе запрашиваемого хоста и URI, а также куки, который определяет пользователя. При этом вы можете кешировать динамические страницы, не обслуживая контент, созданный для другого пользователя.

Обсуждение

По умолчанию директива `proxy_cache_key`, которая подходит для большинства случаев использования, – это `"$scheme$proxy_host$request_uri"`. Используемые переменные включают в себя схему, протокол HTTP или HTTPS, `proxy_host`, куда отправляется запрос, и URI запроса. Все это вместе отражает URL-адрес, на который NGINX передает запрос. Вы можете обнаружить, что существует множество других факторов, определяющих уникальный запрос для каждого приложения, например аргументы запроса, заголовки, идентификаторы сеанса и т. д., для которых вам понадобится создать собственный хеш-ключ[1]. Выбор подходящего хеш-ключа очень важен и должен быть продуман с пониманием приложения. Выбрать ключ кеша для статического контента обычно довольно просто – достаточно использовать имя хоста и URI. Выбор ключа кеша для довольно динамичного содержимого, такого как страницы, для

[1] Любая комбинация текста или переменных, доступных для NGINX, может быть использована для формирования ключа кеша. Список переменных доступен на странице: http://nginx.org/en/docs/varindex.html.

приложения панели мониторинга требует дополнительных знаний о том, как пользователи взаимодействуют с приложением, и о степени различия между пользовательскими интерфейсами. Из соображений безопасности вы можете отказаться представлять кешированные данные от одного пользователя другому без полного понимания контекста. Директива `proxy_cache_key` настраивает значение строки, выступающей в виде хеша для ключа кеширования. Ее можно установить в контексте блоков HTTP, сервера и местоположения, обеспечивая гибкий контроль над тем, как кешируются запросы.

4.3. Обход кеширования

Задача

Вам нужна возможность обойти кеширование.

Решение

Используйте директиву `proxy_cache_bypass` с непустым или ненулевым значением. Один из способов сделать это – установить переменную внутри блоков местоположения, которые вы не хотите кешировать, равную 1:

```
proxy_cache_bypass $http_cache_bypass;
```

Эта конфигурация дает NGINX указание обойти кеширование, если для заголовка HTTP-запроса `cache_bypass` установлено любое значение, отличное от 0.

Обсуждение

Существует ряд сценариев, которые требуют, чтобы запрос не кешировался. Для этого NGINX предоставляет директиву `proxy_cache_bypass`, поэтому, когда значение является непустым или ненулевым, запрос будет отправлен на вышестоящий сервер, а не извлечен из кеша. Различные потребности и сценарии обхода кеша будут зависеть от варианта использования ваших приложений. Методы обхода кеша могут быть такими же простыми, как использование заголовка запроса или ответа, или такими сложными, как совместная работа нескольких блоков map. По многим причинам у вас может возникнуть желание обойти кеширование. Одна из важных причин состоит в устранении неполадок и отладке. Проблемы с воспроизведением могут быть трудными, если

вы постоянно извлекаете кешированные страницы или если ваш ключ кеша связан с идентификатором пользователя. Возможность обойти кеш очень важна. Опции включают в себя (но не ограничиваются) обход кеша, когда установлен определенный куки-файл, заголовок или аргумент запроса. Вы также можете полностью отключить кеш для заданного контекста, такого как блок местоположения, используя директиву `proxy_cache off;`.

4.4. Производительность кеширования

Задача

Вам необходимо повысить производительность путем кеширования на стороне клиента.

Решение

Используйте заголовки управления кешированием на стороне клиента:

```
location ~* \.(css|js)$ {
    expires 1y;
    add_header Cache-Control "public";
}
```

Этот блок `location` указывает, что клиент может кешировать содержимое файлов CSS и JavaScript. Директива `expires` указывает клиенту, что его кешированный ресурс будет недействительным через год. Директива `add_header` добавляет к ответу HTTP-заголовок ответа `Cache-Control` со значением `public`, что позволяет любому серверу кеширования по пути кешировать ресурс. Если мы указываем значение `private`, кешировать значение разрешено только клиенту.

Обсуждение

Производительность кеширования зависит от многих факторов, в первую очередь, от скорости диска. В конфигурации NGINX есть много вещей, которые можно сделать для повышения производительности кеширования. Один из вариантов – установить заголовки ответа таким образом, чтобы клиент фактически кешировал ответ и вообще не выполнял запрос к NGINX, просто обслуживая его из собственного кеша.

4.5. Продувка

Задача

Вам необходимо отменить действие кеширования для какого-то объекта.

Решение

Используйте функцию продувки NGINX Plus, директиву `proxy_cache_purge` и непустую переменную или переменную с нулевым значением:

```
map $request_method $purge_method {
    PURGE 1;
    default 0;
}
server {
    ...
    location / {
        ...
        proxy_cache_purge $purge_method;
    }
}
```

В этом примере кеш конкретного объекта будет очищен, если он будет запрошен методом `PURGE`. Ниже приведен пример очистки кеша файла `main.js` с помощью команды `curl`:

```
$ curl -XPURGE localhost/main.js
```

Обсуждение

Обычный способ обработки статических файлов – поместить хеш файла в имя файла. Это гарантирует, что при развертывании нового кода и содержимого ваша сеть доставки содержимого распознает его как новый файл, поскольку URI изменился. Однако это не совсем подходит для динамического контента, для которого вы установили ключи кеша, не соответствующие этой модели. Для каждого сценария кеширования у вас должен быть способ очистки кеша. NGINX Plus предоставил простой метод очистки кешированных ответов. Директива `proxy_cache_purge` при передаче ненулевого или непустого значения будет очищать кешированные элементы, соответствующие запросу. Простой способ настроить очистку – отобразить метод запроса для `PURGE`. Однако вы можете

использовать это в сочетании с модулем `geo_ip` или простой аутентификацией, чтобы никто не мог очистить важные элементы кеша. NGINX также позволяет использовать знак `*`, благодаря чему можно очищать элементы кеша, которые соответствуют общему префиксу URI. Чтобы использовать подстановочные знаки, вам понадобится настроить директиву `proxy_cache_path` с аргументом `purger=on`.

4.6. Директива slice

Задача

Вам необходимо повысить эффективность кеширования, сегментируя файл на фрагменты.

Решение

Используйте директиву `slice` и ее встраиваемые переменные, чтобы разделить результат кеширования на фрагменты:

```
proxy_cache_path /tmp/mycache keys_zone=mycache:10m;
server {
    ...
    proxy_cache mycache;
    slice 1m;
    proxy_cache_key $host$uri$is_args$args$slice_range;
    proxy_set_header Range $slice_range;
    proxy_http_version 1.1;
    proxy_cache_valid 200 206 1h;

    location / {
        proxy_pass http://origin:80;
    }
}
```

Обсуждение

Данная конфигурация определяет зону кеширования и активирует ее для сервера. Затем используется директива `slice`, чтобы указать NGINX разделить ответ на файловые сегменты размером 1 Мб. Файлы кеша хранятся в соответствии с директивой `proxy_cache_key`. Обратите внимание на использование встраиваемой переменной с именем `slice_range`. Эта же переменная используется в качестве заголовка при отправке запро-

са к источнику, и версия HTTP этого запроса обновляется до HTTP/1.1, поскольку версия 1.0 не поддерживает запросы диапазона байтов. Срок действия кеша устанавливается для кодов ответа 200 или 206 на один час, а затем определяются местоположение и происхождение.

Модуль Cache Slice был разработан для доставки видео HTML5, в котором используются запросы в диапазоне байтов для передачи псевдопотока в браузер. По умолчанию NGINX может обслуживать запросы диапазона байтов из своего кеша. Если запрос для байтового диапазона выполнен для некешированного содержимого, NGINX запрашивает весь файл из источника. Когда вы используете модуль Cache Slice, NGINX запрашивает только необходимые сегменты из источника. Запросы диапазона, которые превышают размер фрагмента, включая весь файл, инициируют подзапросы для каждого из требуемых сегментов, а затем эти сегменты кешируются. Когда все сегменты кешируются, ответ собирается и отправляется клиенту, позволяя NGINX более эффективно кешировать и обслуживать контент, запрашиваемый в диапазонах. Модуль Cache Slice следует использовать только для больших файлов, которые не меняются. NGINX проверяет ETag всякий раз, когда он получает сегмент из источника. Если ETag в источнике изменяется, NGINX прерывает транзакцию, потому что кеш больше не действителен. Если содержимое меняется, а файл меньше или ваш источник может обрабатывать скачки нагрузки во время процесса заполнения кеша, лучше использовать модуль Cache Lock, описанный в блоге, ссылка на который указана ниже.

См. также:
https://www.nginx.com/blog/smart-efficient-byte-range-caching-nginx/

Глава 5

Программируемость и автоматизация

5.0. Введение

Программируемость означает способность взаимодействовать с чем-либо посредством программирования. API для NGINX Plus как раз предоставляет это: способность взаимодействовать с настройками и поведением NGINX Plus через HTTP-интерфейс. Этот API дает возможность перенастроить NGINX Plus, добавляя или удаляя вышестоящие серверы через HTTP-запросы. Функция хранилища типа ключ/значение в NGINX Plus обеспечивает еще один уровень динамической конфигурации – вы применяете HTTP-вызовы для ввода информации, которую NGINX Plus может использовать для динамической маршрутизации или управления трафиком. В этой главе будет рассказано об API NGINX Plus и модуле хранилища типа ключ/значение, предоставляемых этим API.

Инструменты управления конфигурацией автоматизируют установку и настройку серверов, что является бесценным в эпоху облачных вычислений. Разработчикам крупных веб-приложений больше не нужно настраивать серверы вручную; вместо этого они могут использовать один из множества доступных инструментов управления конфигурацией. С помощью этих инструментов инженеры могут писать конфигурации и кодировать один раз, чтобы создать множество серверов с одинаковой конфигурацией в воспроизводимом, тестируемом и модульном виде. В этой главе рассматриваются некоторые самые популярные доступные инструменты управления конфигурацией и то, как их использовать для установки NGINX и шаблона базовой конфигурации. Эти примеры очень просты, но они демонстрируют, как запустить сервер NGINX, используя разные платформы.

5.1. API NGINX Plus

Задача

У вас есть динамичная среда, и вам нужно перенастроить NGINX Plus на лету.

Решение

Сконфигурируйте API NGINX Plus, чтобы разрешить добавление и удаление серверов через API-вызовы:

```
upstream backend {
    zone http_backend 64k;
}
server {
    # ...
    location /api {
        api [write=on];
        # Директивы, ограничивающие доступ к API
        # См. главу 7
    }

    location = /dashboard.html {
        root /usr/share/nginx/html;
    }
}
```

В этой конфигурация NGINX Plus создается вышестоящий сервер с зоной общей памяти, активируется API в блоке `location /api` и предоставляется местоположение для панели инструментов NGINX Plus.

Можно использовать API для добавления серверов, когда они подключаются к сети:

```
$ curl -X POST -d '{"server":"172.17.0.3"}' \
    'http://nginx.local/api/3/http/upstreams/backend/servers/'
{
    "id":0,
    "server":"172.17.0.3:80",
    "weight":1,
    "max_conns":0,
    "max_fails":1,
```

```
    "fail_timeout":"10s",
    "slow_start":"0s",
    "route":"",
    "backup":false,
    "down":false
}
```

Используя команду `curl` в этом примере, мы делаем запрос к NGINX Plus, чтобы добавить новый сервер в конфигурацию backend-upstream. HTTP-метод – это POST, а объект JSON передается в качестве тела. API NGINX Plus – это RESTful; следовательно, в URI запроса есть параметры. Формат URI выглядит так:

```
/api/{version}/http/upstreams/{httpUpstreamName}/servers/
```

Можно использовать API NGINX Plus для перечисления серверов в восходящем пуле:

```
$ curl 'http://nginx.local/api/3/http/upstreams/backend/servers/'
[
  {
    "id":0,
    "server":"172.17.0.3:80",
    "weight":1,
    "max_conns":0,
    "max_fails":1,
    "fail_timeout":"10s",
    "slow_start":"0s",
    "route":"",
    "backup":false,
    "down":false
  }
]
```

Используя команду `curl` в этом примере, мы отправляем запрос в NGINX Plus, чтобы вывести список всех серверов в восходящем пуле с именем backend. В настоящее время у нас есть только один сервер, который мы добавили в предыдущем вызове. Запрос вернет объект вышестоящего сервера, который содержит все настраиваемые параметры для сервера.

Используйте API NGINX Plus для осушения подключений от вышестоящего сервера и подготовки его к постепенному удалению из восходя-

щего пула. Подробную информацию об осушении подключения можно найти в рецепте 2.8:

```
$ curl -X PATCH -d '{"drain":true}' \
    'http://nginx.local/api/3/http/upstreams/backend/servers/0'
{
    "id":0,
    "server":"172.17.0.3:80",
    "weight":1,
    "max_conns":0,
    "max_fails":1,
    "fail_timeout":
    "10s","slow_start":
    "0s",
    "route":"",
    "backup":false,
    "down":false,
    "drain":true
}
```

Здесь мы указываем, что методом запроса является `PATCH`, передаем тело JSON, предписывая ему выполнить осушение подключения для сервера, и указываем идентификатор сервера, добавляя его в URI. Мы нашли идентификатор сервера, перечислив серверы в восходящем пуле с помощью предыдущей команды `curl`.

NGINX Plus начнет осушение подключений. Этот процесс может занять столько же времени, сколько и сеансы приложения. Чтобы проверить, сколько активных соединений обслуживает сервер, который вы начали применять, используйте приведенный ниже вызов и найдите атрибут `active` подлежащего осушению сервера:

```
$ curl 'http://nginx.local/api/3/http/upstreams/backend'
{
    "zone" : "http_backend",
    "keepalive" : 0,
    "peers" : [
        {
          "backup" : false,
          "id" : 0,
          "unavail" : 0,
          "name" : "172.17.0.3",
```

```
            "requests" : 0,
            "received" : 0,
            "state" : "draining",
            "server" : "172.17.0.3:80",
            "active" : 0,
            "weight" : 1,
            "fails" : 0,
            "sent" : 0,
            "responses" : {
                "4xx" : 0,
                "total" : 0,
                "3xx" : 0,
                "5xx" : 0,
                "2xx" : 0,
                "1xx" : 0
            },
            "health_checks" : {
                "checks" : 0,
                "unhealthy" : 0,
                "fails" : 0
            },
            "downtime" : 0
        }
    ],
    "zombies" : 0
}
```

После осушения всех подключений используйте API NGINX Plus, чтобы полностью удалить сервер из восходящего пула:

```
$ curl -X DELETE \
    'http://nginx.local/api/3/http/upstreams/backend/servers/0'
[]
```

Команда `curl` выполняет запрос с методом `DELETE` на тот же URI, который использовался для обновления состояния серверов. Метод `DELETE` дает команду NGINX удалить сервер. Этот API-вызов возвращает все серверы и их идентификаторы, которые по-прежнему остаются в пуле. Поскольку мы начали с пустого пула, добавили только один сервер через API, осушили его и затем удалили, теперь у нас снова есть пустой пул.

Обсуждение

Эксклюзивный API NGINX Plus позволяет динамическим серверам приложений добавлять и удалять себя в конфигурацию NGINX на лету. Когда серверы подключаются к сети, они могут зарегистрироваться в пуле, и NGINX начнет отправлять его для нагрузки. Когда сервер необходимо удалить, он может запросить у NGINX Plus осушение своих подключений, а затем удалить себя из восходящего пула до закрытия. Это позволяет инфраструктуре за счет автоматизации уменьшать и увеличивать масштаб без вмешательства человека.

См. также:
Документация по NGINX Plus API Swagger (https://demo.nginx.com/swagger-ui/).

5.2. Хранилище типа ключ/значение

Задача

Вам нужен NGINX Plus для принятия динамических решений по управлению трафиком на основе входных данных из приложений.

Решение

Настройте хранилище типа ключ/значение, поддерживающее кластеры и API, а затем добавьте ключи и значения:

```
keyval_zone zone=blacklist:1M;
keyval $remote_addr $num_failures zone=blacklist;

server {
    # ...
    location / {
        if ($num_failures) {
            return 403 'Forbidden';
        }
        return 200 'OK';
    }
}
server {
    # ...
    # Директивы, ограничивающие доступ к API
```

```
    # См. главу 6
    location /api {
        api write=on;
    }
}
```

В этой конфигурации NGINX Plus используется каталог `keyval_zone` для создания зоны общей памяти хранилища типа ключ/значение с именем `blacklist` и устанавливается лимит памяти в 1 Мб. Затем директива `keyval` отображает значение ключа, соответствующего первому параметру `$remote_addr`, в новую переменную с именем `$num_failures` из зоны. Эта новая переменная затем используется для определения того, должен ли NGINX Plus обслуживать запрос или вернуть ошибку 403 Forbidden code.

После запуска сервера NGINX Plus с этой конфигурацией вы можете использовать команду `curl` и ожидать ответа 200 OK.

```
$ curl 'http://127.0.0.1/'
OK
```

Теперь добавьте IP-адрес локального компьютера в хранилище типа ключ/значение со значением 1:

```
$ curl -X POST -d '{"127.0.0.1":"1"}' \
  'http://127.0.0.1/api/3/http/keyvals/blacklist'
```

Команда `curl` отправляет запрос с использованием метода POST по протоколу HTTP с объектом JSON, содержащим объект ключ/значение для отправки в зону общей памяти черного списка. URI API-интерфейса хранилища типа ключ/значение форматируется как показано ниже:

```
/api/{version}/http/keyvals/{httpKeyvalZoneName}
```

IP-адрес локального компьютера теперь добавлен в зону ключ/значение с именем `blacklist` со значением 1. В следующем запросе NGINX Plus ищет `$remote_addr` в зоне ключ/значение, находит запись и отображает значение в переменную `$num_failures`. Эта переменная затем оценивается в операторе `if`. Когда у переменной есть значение, `if` оценивается как True, а NGINX Plus возвращает ошибку 403 Forbidden:

```
$ curl 'http://127.0.0.1/'
Forbidden
```

Вы можете обновить или удалить ключ, сделав запрос с методом `PATCH`:

```
$ curl -X PATCH -d '{"127.0.0.1":null}' \
  'http://127.0.0.1/api/3/http/keyvals/blacklist'
```

NGINX Plus удаляет ключ, если значение равно нулю, и запросы снова возвращают ответ 200 OK.

Обсуждение

Хранилище типа ключ/значение, эксклюзивная функция NGINX Plus, позволяет приложениям вводить информацию в NGINX Plus. В приведенном примере переменная `$remote_addr` используется для создания динамического черного списка. Вы можете заполнить это хранилище любым ключом, который NGINX Plus может иметь в качестве переменной (например, куки-файлом сессии), и предоставить NGINX Plus внешнее значение. В NGINX Plus R16 хранилище типа ключ/значение стало поддерживать кластеры, а это означает, что вы должны предоставить обновление ключ/значение только одному серверу NGINX Plus, и все прочие получат эту информацию.

См. также:
https://www.nginx.com/blog/dynamic-bandwidth-limits-nginx-plus-key-value-store/

5.3. Установка с использованием приложения Puppet

Задача

Вам нужно установить и настроить NGINX с использованием Puppet, чтобы управлять конфигурацией NGINX в виде кода и соответствовать остальным конфигурациям Puppet.

Решение

Создайте модуль, который устанавливает NGINX, управляет файлами, которые вам нужны, и гарантирует, что NGINX работает:

```
class nginx {
    package {"nginx": ensure => 'installed',}
    service {"nginx":
        ensure => 'true',
        hasrestart => 'true',
```

```
        restart => '/etc/init.d/nginx reload',
    }
    file { "nginx.conf":
        path => '/etc/nginx/nginx.conf',
        require => Package['nginx'],
        notify => Service['nginx'],
        content => template('nginx/templates/nginx.conf.erb'),
        user=>'root',
        group=>'root',
        mode='0644';
    }
}
```

Этот модуль использует утилиту управления пакетами, чтобы гарантировать, что пакет NGINX установлен. Он также гарантирует, что NGINX запущен и включен во время загрузки. Конфигурация сообщает Puppet, что у службы есть команда перезапуска с директивой `hasrestart`, и мы можем переопределить команду `restart`, перезагрузив NGINX. Файловый ресурс будет управлять и шаблонировать файл *nginx.conf* с использованием языка шаблонов Embedded Ruby (ERB). Шаблонирование файла произойдет после установки пакета NGINX благодаря директиве `require`. Однако файловый ресурс уведомит службу NGINX о перезагрузке благодаря директиве `notify`. Шаблонный файл конфигурации не включен. Тем не менее файл конфигурации NGINX по умолчанию установить просто, либо это может быть очень сложно, если использовать циклы языков шаблонов ERB или EPP и подстановку переменных.

Обсуждение

Puppet – это инструмент управления конфигурацией на базе языке программирования Ruby. Модули встроены в предметно-ориентированный язык и вызываются через файл манифеста, который определяет конфигурацию для данного сервера. Puppet может быть запущен в конфигурации «ведущий–ведомый» или без ведущего. С Puppet манифест запускается на ведущем устройстве, а затем отправляется на ведомое устройство. Это важно, потому что это гарантирует, что ведомое устройство поставляется только с той конфигурацией, которая предназначена для него, и без дополнительных настроек, предназначенных для других серверов. Существует множество чрезвычайно развитых общедоступных модулей для Puppet. Эти модули помогут вам начать работу с

вашей конфигурацией. Общедоступный модуль NGINX из voxpupuli на GitHub создаст шаблоны конфигурации NGINX за вас.

См. также:
https://puppet.com/docs
https://puppet.com/docs/puppet/latest/types/package.html
https://puppet.com/docs/puppet/latest/types/service.html
https://puppet.com/docs/puppet/latest/types/file.html
https://puppet.com/docs/puppet/latest/lang_template.html
https://github.com/voxpupuli/puppet-nginx

5.4. Установка с использованием системы Chef

Задача

Вам нужно установить и настроить NGINX, используя Chef, чтобы управлять конфигурациями NGINX в виде кода и соответствовать остальным конфигурациям Chef.

Решение

Создайте кулинарную книгу с рецептом установки NGINX, настройте файлы конфигурации через шаблоны и обеспечьте перезагрузку NGINX, после того как конфигурация будет введена в действие. Ниже приводится пример рецепта:

```
package 'nginx' do
  action :install
end

service 'nginx' do
  supports :status => true, :restart => true, :reload => true
  action [ :start, :enable ]
end

template 'nginx.conf' do
  path "/etc/nginx.conf"
  source "nginx.conf.erb"
  owner 'root'
```

```
  group 'root'
  mode '0644'
  notifies :reload, 'service[nginx]', :delayed
end
```

Блок `package` устанавливает NGINX. Блок `service` гарантирует, что NGINX запускается и включается при загрузке, а затем объявляет остальной части Chef, что служба `nginx` будет поддерживать применительно к действиям. Блок `template` формирует файл ERB и помещает его в */etc/nginx.conf* с владельцем и группой root, а также устанавливает значение `mode` на 644 и уведомляет службу `nginx` о перезагрузке, но ждет конца запуска Chef, объявляемое оператором `:delayed`. Шаблонный файл конфигурации не включен. Тем не менее файл конфигурации NGINX по умолчанию установить просто, либо это может быть очень сложно, если использовать циклы языков шаблонов ERB или EPP и подстановку переменных.

Обсуждение

Chef – это система управления конфигурациями на базе языка Ruby. Она может работать в режиме «ведущий–ведомый» или в режиме соло, ныне известном как Chef Zero. У Chef есть огромное сообщество с большим количеством общедоступных книг с рецептами, которое именуется Supermarket. Эти сборники рецептов можно устанавливать и поддерживать с помощью утилиты командной строки Berkshelf. Chef чрезвычайно мощная система, и то, что мы продемонстрировали, является лишь небольшим примером. Общедоступная книга с рецептами NGINX в Supermarket чрезвычайно гибкая и предоставляет варианты простой установки NGINX из диспетчера пакетов или из исходного кода, а также возможность компилировать и устанавливать множество различных модулей, создавать шаблоны основных конфигураций.

См. также:
https://docs.chef.io
https://docs.chef.io/resource_package.html
https://docs.chef.io/resource_service.html
https://docs.chef.io/resource_template.html
https://supermarket.chef.io/cookbooks/nginx

5.5. Установка с использованием системы Ansible

Задача

Вам необходимо установить и настроить NGINX, используя Ansible, чтобы управлять конфигурациями NGINX в виде кода и соответствовать остальным конфигурациям Ansible.

Решение

Создайте плейбук Ansible для установки NGINX и управления файлом *nginx.conf*. Ниже приведен пример файла задачи для плейбука, чтобы установить NGINX. Убедитесь, что он работает и создает шаблон файла конфигурации:

```
- name: NGINX | Installing NGINX
  package: name=nginx state=present

- name: NGINX | Starting NGINX
  service:
    name: nginx
    state: started
    enabled: yes

- name: Copy nginx configuration in place.
  template:
    src: nginx.conf.j2
    dest: "/etc/nginx/nginx.conf"
    owner: root
    group: root
    mode: 0644
  notify:
    - reload nginx
```

Блок `package` устанавливает NGINX. Блок `service` гарантирует, что NGINX запущен и включен при загрузке. Блок `template` устанавливает файл шаблона *Jinja2* и помещает результат в */etc/nginx.conf* с root в качестве владельца и группы, а также устанавливает значение `mode` в 644 и уведомляет службу `nginx` о перезагрузке. Шаблонный файл конфигурации не включен. Тем не менее файл конфигурации NGINX по

умолчанию установить просто, либо это может быть очень сложно, если использовать циклы языка шаблонов Jinja2 и подстановку переменных.

Обсуждение

Ansible – это широко распространенное и мощное средство управления конфигурациями, написанное на языке Python. Конфигурация задач идет в формате YAML, а язык шаблонов Jinja2 используется для файлов шаблонов. Ansible предлагает графический интерфейс для управления и мониторинга работы Ansible Tower по подписке. Тем не менее обычно он используется с локальных компьютеров или для создания серверов непосредственно к клиенту или в модели без ведомого устройства. Ansible отправляет SSH-пакеты на свои серверы и запускает конфигурации. Как и у других систем управления конфигурациями, у Ansible существует большое сообщество под названием Ansible Galaxy, где можно найти очень сложные роли для использования в своих плейбуках.

См. также:
https://docs.ansible.com
https://docs.ansible.com/ansible/latest/modules/package_module.html
https://docs.ansible.com/ansible/latest/modules/service_module.html
https://docs.ansible.com/ansible/latest/modules/template_module.html
https://galaxy.ansible.com

5.6. Установка с использованием SaltStack

Задача

Вам необходимо установить и настроить NGINX с использованием системы SaltStack, чтобы управлять конфигурациями NGINX в виде кода и соответствовать остальным конфигурациям SaltStack.

Решение

Установите NGINX через модуль управления пакетами и управляйте файлами конфигураций, которые вам нужны. Ниже приводится пример файла состояния (*sls*), который установит пакет `nginx` и гарантирует, что служба работает, включается при загрузке и перезагружается, если в файл конфигурации внесены изменения:

```
nginx:
  pkg:
    - installed
  service:
    - name: nginx
    - running
    - enable: True
    - reload: True
    - watch:
      - file: /etc/nginx/nginx.conf

/etc/nginx/nginx.conf:
  file:
    - managed
    - source: salt://path/to/nginx.conf
    - user: root
    - group: root
    - template: jinja
    - mode: 644
    - require:
      - pkg: nginx
```

Это базовый пример установки NGINX с помощью утилиты управления пакетами и управления файлом *nginx.conf*. Пакет NGINX установлен, а служба запущена и активируется при загрузке. С помощью SaltStack можно объявлять файл, управляемый Salt, как показано в примере, и шаблонизированный многими различными языками шаблонов. Шаблонный файл конфигурации не включен. Тем не менее файл конфигурации NGINX по умолчанию установить просто, либо это может быть очень сложно, если использовать циклы языка шаблонов Jinja2 и подстановку переменных. Эта конфигурация также указывает, что NGINX должно быть установлено до управления файлом из-за оператора `require`. После того как файл будет на месте, NGINX перезагружается благодаря установленной в его службе директиве `watch` и перезагружается, а не перезапускается, так как значение директивы `reload` установлено в `True`.

Обсуждение

SaltStack – это мощный инструмент управления конфигурациями, который определяет состояния сервера в YAML. Модули для SaltStack

могут быть написаны на языке Python. Salt предоставляет язык шаблонов Jinja2 для состояний, а также для файлов. Однако для файлов существует множество других опций, таких как Mako, сам Python и др. Salt работает в конфигурации «ведущий–ведомый», а также в конфигурации «без ведущего». Ведомые устройства носят название миньоны. Однако транспорт взаимодействия «ведущий–ведомый» в SaltStack отличается от других систем. Работая с Salt, вы можете выбрать протоколы ZeroMQ, TCP или RAET для передачи агенту; или можно не использовать агента, и вместо этого ведущее устройство может применять проколол SSH. Поскольку транспортный уровень по умолчанию асинхронный, SaltStack построен таким образом, чтобы иметь возможность доставить свое сообщение большому числу миньонов с низкой нагрузкой на главный сервер.

См. также:

https://docs.saltstack.com/en/latest/

https://docs.saltstack.com/en/latest/ref/states/all/salt.states.pkg.html#salt.states.pkg.installed

https://docs.saltstack.com/en/latest/ref/states/all/salt.states.file.html#salt.states.file.managed

https://docs.saltstack.com/en/latest/topics/jinja/index.html

5.7. Автоматизация конфигураций с помощью Consul

Задача

Вам необходимо автоматизировать конфигурацию NGINX, чтобы реагировать на изменения в вашей среде с помощью Consul.

Решение

Используйте демон `consul-template` и файл шаблона, чтобы создать шаблон файла конфигурации NGINX на ваш выбор:

```
upstream backend { {{range service "app.backend"}}
    server {{.Address}}:{{end}}
}
```

Этот пример представляет собой файл шаблона Consul, который формирует шаблон конфигурации блока `upstream backend`. Этот шаблон будет

осуществлять итерацию узлов в Consul, идентифицированном как `app.backend`. Для каждого узла в Consul шаблон создаст директиву сервера с IP-адресом этого узла.

Демон `consul-template` запускается из командной строки и может использоваться для перезагрузки NGINX при каждом изменении установленного шаблона файла настройки:

```
# consul-template -consul consul.example.internal -template \
 template:/etc/nginx/conf.d/upstream.conf:"nginx -s reload"
```

Данная команда указывает демону `consul-template` подключиться к кластеру Consul в `consul.example.internal` и использовать файл с именем *template* в текущем рабочем каталоге для выработки файла шаблона и вывода сгенерированного содержимого в */etc/nginx/conf.d./upstream.conf* с последующей перезагрузкой NGINX при каждом изменении данного файла шаблона. Флаг `-template` принимает строку файла template, расположение вывода и команду, запускаемую после того, как происходит процесс создания шаблона; эти три переменные разделены двоеточием. Если в выполняемой команде есть пробелы, убедитесь, что она заключена в двойные кавычки. Флаг `-consul` сообщает демону, с каким кластером Consul устанавливать соединение.

Обсуждение

Consul – это мощный инструмент для обнаружения служб и хранилище настроек. Он хранит информацию об узлах, а также пары ключ/значение в структуре, напоминающей каталог, и обеспечивает взаимодействие с restful API. Consul также предоставляет DNS-интерфейс для каждого клиента, что позволяет выполнять поиск доменных имен узлов, подключенных к кластеру. Демон `consul-template` – отдельный проект, который использует кластеры Consul; этот инструмент вырабатывает шаблоны файлов в ответ на изменения в узлах Consul, службах или парах ключ/значение. Это делает Consul очень мощным выбором для автоматизации NGINX. Используя `consul-template`, вы также можете дать демону указание выполнить команду после изменения шаблона. Благодаря этому можно перезагрузить конфигурацию NGINX и позволить оживить вашу конфигурацию наряду с вашей средой. С помощью Consul можно настроить проверку работоспособности для каждого клиента для проверки работоспособности предполагаемой службы. Благодаря такому выявлению отказов можно соответствующим образом

снабжать шаблонами свою конфигурацию NGINX, чтобы отправлять трафик только в жизнеспособные хосты.

См. также:

https://www.consul.io

https://www.hashicorp.com/blog/introducing-consul-template.html

https://github.com/hashicorp/consul-template

Глава 6

Аутентификация

6.0. Введение

NGINX может аутентифицировать клиентов. Запросы аутентификации клиента с помощью NGINX освобождают от лишней работы и предоставляют возможность останавливать запросы без аутентификации, чтобы они не доходили до ваших серверов приложений. Модули, доступные для NGINX с открытым исходным кодом, включают в себя базовую аутентификацию и подзапросы аутентификации. Эксклюзивный модуль NGINX Plus для проверки веб-токенов JSON (JWT) обеспечивает интеграцию со сторонними поставщиками проверки подлинности, которые используют стандарт аутентификации OpenID Connect.

6.1. Базовая HTTP-аутентификация

Задача

Вам необходимо защитить свое приложение или контент с помощью базовой HTTP-аутентификации.

Решение

Создайте файл в приведенном ниже формате, где пароль зашифрован или хеширован в одном из разрешенных форматов:

```
# comment
name1:password1
name2:password2:comment
name3:password3
```

Имя пользователя – это первое поле, пароль – второе поле, а разделитель – это двоеточие. Существует необязательное третье поле, кото-

рое можно использовать для комментирования каждого пользователя. NGINX может понимать несколько различных форматов паролей, один из которых заключается в том, зашифрован ли пароль с помощью функции языка C `crypt()`. Эта функция предоставляется командной строке командой `openssl passwd`. После установки `openssl` вы можете создавать зашифрованные строки пароля с помощью этой команды:

```
$ openssl passwd MyPassword1234
```

Выводом будет строка, которую NGINX может использовать в вашем файле пароля.

Используйте директивы `auth_basic` и `auth_basic_user_file` внутри своей конфигурации NGINX для активации базовой аутентификации:

```
location / {
    auth_basic           "Private site";
    auth_basic_user_file conf.d/passwd;
}
```

Вы можете использовать директивы `auth_basic` в контексте HTTP, сервера или местоположения. Директива `auth_basic` принимает строковый параметр, который отображается во всплывающем окне базовой аутентификации, при появлении пользователя без аутентификации. Файл `auth_basic_user_file` указывает путь к файлу пользователя.

Обсуждение

Вы можете создать пароли базовой аутентификации несколькими способами и в различных форматах с различной степенью безопасности. Команда `htpasswd` от Apache также может генерировать пароли. Команды `openssl` и `htpasswd` могут генерировать пароли с помощью алгоритма `apr1`, который также способно понимать и NGINX. Пароль тоже может находиться в формате SHA-1, используемом протоколом LDAP и сервером Dovecot. NGINX поддерживает дополнительные форматы и алгоритмы хеширования; тем не менее многие из них считаются небезопасными, потому что их можно легко взломать, применяя атаки методом грубого перебора.

Можно использовать базовую аутентификацию для защиты контекста всего хоста NGINX, конкретных виртуальных серверов или даже просто определенных блоков location. Базовая аутентификация не заменит аутентификацию пользователя для веб-приложения, но может помочь защитить конфиденциальную информацию.

Под капотом базовая аутентификация выполняется сервером, возвращающим ошибку 401 unauthorized с заголовком ответа `WWW-Authenticate`. Этот заголовок будет иметь значение `Basic realm = "ваша строка"`. Этот ответ заставляет браузер запрашивать имя пользователя и пароль. Имя пользователя и пароль объединяются и разделяются двоеточием, затем кодируются в base64 и отправляются в заголовке запроса с именем `Authorization`. В заголовке запроса `Authorization` будут указаны `Basic` и закодированная строка `user: password`. Сервер декодирует заголовок и сверяется с `auth_basic_user_file`. Поскольку строка с паролем и именем пользователя закодирована всего лишь в кодировке base64, рекомендуется использовать протокол HTTPS с базовой аутентификацией.

6.2. Подзапросы аутентификации

Задача

У вас есть сторонняя система аутентификации, для которой вам нужны аутентифицированные запросы.

Решение

Используйте `http_auth_request_module`, чтобы отправить запрос службе аутентификации для проверки личности перед обслуживанием запроса:

```
location /private/ {
    auth_request     /auth;
    auth_request_set $auth_status $upstream_status;
}

location = /auth {
    internal;
    proxy_pass              http://auth-server;
    proxy_pass_request_body off;
    proxy_set_header        Content-Length "";
    proxy_set_header        X-Original-URI $request_uri;
}
```

Директива `auth_request` принимает параметр URI, который должен быть местным внутренним местоположением. Директива `auth_request_set` позволяет установить переменные из подзапроса аутентификации.

Обсуждение

Модуль `http_auth_request_module` активирует аутентификацию для каждого запроса, обрабатываемого сервером NGINX. Модуль выполняет подзапрос перед обработкой оригинала, чтобы определить, имеет ли запрос доступ к ресурсу, который он запрашивает. Весь исходный запрос передается в это местоположение подзапроса. Местоположение аутентификации действует как типичный прокси-сервер для подзапроса и отправляет исходный запрос, включая его тело и заголовки. Код состояния HTTP подзапроса определяет, будет ли предоставлен доступ. Если подзапрос возвращается с кодом состояния HTTP 200, аутентификация прошла успешно и запрос выполнен. Если подзапрос возвращает ошибку 401 или 403, то же самое будет возвращено для исходного запроса.

Если ваша служба аутентификации не запрашивает тело запроса, можно удалить его с помощью директивы `proxy_pass_request_body`, как было продемонстрировано ранее. Используя такой метод, вы уменьшите размер запроса и время. Поскольку тело ответа отбрасывается, заголовок `Content-Length` должен быть установлен в пустую строку. Если вашей службе аутентификации нужно знать URI, к которому обращается запрос, вам понадобится поместить это значение в настраиваемый заголовок, который ваша служба аутентификации проверяет и верифицирует. Если есть нечто, что вы хотите сохранить из подзапроса в службу аутентификации, например заголовки ответа или другая информация, можно использовать директиву `auth_request_set` для создания новых переменных из данных ответа.

6.3. Валидация токенов в формате JWT

Задача

Вам нужно проверить JWT-токен, прежде чем запрос будет обработан с помощью NGINX Plus.

Решение

Используйте модуль аутентификации на основе JWT от NGINX Plus, чтобы проверить подпись токена и встроить утверждения и заголовки JWT в качестве переменных NGINX:

```
location /api/ {
    auth_jwt          "api";
```

```
    auth_jwt_key_file conf/keys.json;
}
```

Данная конфигурация позволяет проверять токены JWT для этого местоположения. Директиве `auth_jwt` передается строка, которая используется как область аутентификации. Директива `auth_jwt` принимает необязательный параметр токена переменной, которая содержит JWT. По умолчанию заголовок `Authentication` используется в соответствии со стандартом JWT. Директиву `auth_jwt` также можно использовать для отмены эффектов требуемой аутентификации JWT из унаследованных конфигураций. Чтобы отключить аутентификацию, передайте ключевое слово в директиву `auth_jwt` без всего. Чтобы отменить унаследованные требования аутентификации, передайте ключевое слово `off` в директиву `auth_jwt`, и ничего больше. `auth_jwt_key_file` принимает один параметр, который является путем к файлу ключа в стандартном формате JSON Web Key.

Обсуждение

NGINX Plus может проверять JWT-токены в отличие от типа веб-шифрования JSON, где весь токен шифруется. NGINX Plus может проверять сигнатуры, подписанные с помощью алгоритмов HS256, RS256 и ES256. Используя NGINX Plus для проверки токена, вы можете сэкономить время и ресурсы, необходимые для создания подзапроса к службе аутентификации.

NGINX Plus расшифровывает заголовок и полезную нагрузку JWT, а также подставляет значения стандартных заголовков и заявок во встроенные переменные, которые вы применяете.

См. также:
https://tools.ietf.org/html/rfc7515
https://tools.ietf.org/html/rfc7518
https://tools.ietf.org/html/rfc7519
http://nginx.org/en/docs/http/ngx_http_auth_jwt_module.html#variables
https://www.nginx.com/blog/authenticating-api-clients-jwt-nginx-plus/

6.4. Создание веб-ключей в формате JSON

Задача

Вам нужен ключ в формате JSON для использования в NGINX Plus.

Решение

NGINX Plus использует формат JSON Web Key (JWK), как указано в стандарте RFC. Этот стандарт допускает массив ключей объектов в файле JWK.

Ниже приводится пример того, как может выглядеть файл ключа:

```
{"keys":
  [
    {
      "kty":"oct",
      "kid":"0001",
      "k":"OctetSequenceKeyValue"
    },
    {
      "kty":"EC",
      "kid":"0002"
      "crv":"P-256",
      "x": "XCoordinateValue",
      "y": "YCoordinateValue",
      "d": "PrivateExponent",
      "use": "sig"
    },
    {
      "kty":"RSA",
      "kid":"0003"
      "n": "Modulus",
      "e": "Exponent",
      "d": "PrivateExponent"
    }
  ]
}
```

Показанный файл JWK демонстрирует три начальных типа ключей, отмеченных в стандарте RFC. Формат этих ключей также является частью стандарта RFC. Атрибут `kty` – это тип ключа. В этом файле показаны три типа ключей: Octet Sequence (oct), EllipticCurve (EC) и тип RSA. Атрибут `kid` – это идентификатор ключа. Другие атрибуты этих ключей указаны в стандарте для этого типа ключа. Обратитесь к документации RFC для получения дополнительной информации.

Обсуждение

Для создания JWK существует множество библиотек на разных языках. Рекомендуется создать службу ключей, которая будет центральным органом JWK для создания и ротации ваших ключей с регулярным интервалом. Для повышения безопасности рекомендуется делать ваши JWK такими же безопасными, как и сертификаты SSL/TLS. Защитите свой ключевой файл соответствующими правами доступа пользователя и группы. Хранение их в памяти на вашем хосте – лучший вариант. Это можно сделать, создав файловую систему в памяти, такую как ramfs. Ротация ключей с регулярным интервалом также важна; вы можете создать службу ключей, которая создает открытые и закрытые ключи и предлагает их приложению и NGINX через API.

См. также:
https://tools.ietf.org/html/rfc7517

6.5. Аутентификация пользователей с помощью существующего протокола единого входа OpenID Connect

Задача

Вам нужно перенести проверку подлинности с помощью OpenID Connect в NGINX Plus.

Решение

Используйте модуль JWT, поставляемый с NGINX Plus, для защиты местоположения или сервера и дайте директиве `auth_jwt` указание использовать `$cookie_auth_token` в качестве токена для проверки:

```
location /private/ {
    auth_jwt "Google Oauth" token=$cookie_auth_token;
    auth_jwt_key_file /etc/nginx/google_certs.jwk;
}
```

В этой конфигурации мы направляем NGINX Plus для защиты пути */private/* URI с проверкой JWT. В OpenID Connect используется куки-файл `auth_token`, а не токен на предъявителя по умолчанию. Таким образом, вы должны указать NGINX искать токен в этом куки-файле, а не в располо-

жении по умолчанию NGINX Plus. Для местоположения `auth_jwt_key_file` задан произвольный путь, который мы рассмотрим в рецепте 6.6.

Обсуждение

Данная конфигурация демонстрирует, как можно проверить веб-токен JSON OpenID Connect JSON Google OAuth 2.0 с помощью NGINX Plus. Модуль аутентификации JWT NGINX Plus для протокола HTTP способен проверить любой JWT-токен, который соответствует RFC для спецификации JSON Web Signature, мгновенно активируя любые полномочия системы единого входа, использующие JWT-токены на уровне NGINX Plus. Протокол OpenID 1.0 – это уровень поверх протокола аутентификации OAuth 2.0, который добавляет идентичность, позволяя использовать JWT для подтверждения личности пользователя, отправившего запрос. С помощью подписи токена NGINX Plus может проверить, что токен не был изменен, с того момента как он был подписан. Таким образом, Google использует метод асинхронной подписи и позволяет распространять общедоступные JWK, храня при этом свой закрытый JWK в тайне.

NGINX Plus также может управлять потоком с кодом подтверждения для OpenID Connect 1.0, что позволяет NGINX Plus выступать в роли ретранслятора для OpenID Connect. Эта возможность обеспечивает интеграцию с большинством основных поставщиков удостоверений, включая CA Single Sign-On (ранее SiteMinder), ForgeRock OpenAM, Keycloak, Okta, OneLogin и Ping Identity. Дополнительную информацию и справочную реализацию NGINX Plus в качестве ретранслятора для аутентификации с OpenID Connect можно найти в репозитории OpenID Connect GitHub NGINX Inc (https://github.com/nginxinc/nginx-openid-connect).

См. также:

https://www.nginx.com/blog/authenticating-users-existing-applications-openid-connect-nginx-plus/

https://openid.net/connect/

6.6. Получение ключа в формате JSON от Google

Задача

Вам необходимо получить ключ в формате JSON от Google, чтобы использовать его при проверке токенов OpenID Connect с помощью NGINX Plus.

Решение

Используйте Cron для запроса нового набора ключей каждый час, чтобы гарантировать, что ключи всегда актуальны:

```
0 * * * * root wget https://www.googleapis.com/oauth2/v3/ \
  certs-0 /etc/nginx/google_certs.jwk
```

Этот фрагмент кода представляет собой строку из файла crontab. Unix-подобные системы имеют множество опций, где могут находиться файлы crontab. У каждого пользователя будет пользовательский crontab, а также ряд файлов и каталогов в каталоге */etc/*.

Обсуждение

Cron – это распространенный способ запуска запланированной задачи в Unix-подобной системе. Следует регулярно выполнять ротацию ключей в формате JSON, чтобы обеспечить их безопасность и, в свою очередь, безопасность своей системы. Чтобы у вас всегда был самый актуальный ключ от Google, вам нужно регулярно проверять наличие новых JWK. Cron – один из способов сделать это.

См. также:
https://linux.die.net/man/8/cron

Глава 7

Контроль безопасности

7.0. Введение

Безопасность осуществляется по уровням, и в вашей модели безопасности должно быть несколько уровней, чтобы она была действительно прочной. В этой главе мы рассмотрим различные способы защиты ваших веб-приложений с помощью NGINX и NGINX Plus. Вы можете использовать многие из этих методов обеспечения безопасности в сочетании друг с другом, чтобы помочь повысить защиту. Ниже приводится ряд разделов, посвященных вопросам безопасности, где рассматриваются функции NGINX и NGINX Plus, которые могут помочь при защите вашего приложения. Вы можете заметить, что в этой главе не затрагивается одна из крупнейших функций безопасности NGINX, модуль ModSecurity 3.0, который превращает NGINX в брандмауэр веб-приложений. Чтобы узнать больше о возможностях WAF, скачайте «ModSecurity 3.0 и NGINX: Краткое руководство по началу работы» (https://www.nginx.com/resources/library/modsecurity-3-nginx-quick-start-guide/).

7.1. Доступ на основе IP-адреса

Задача

Вам необходимо контролировать доступ на основе IP-адреса клиента.

Решение

Используйте модуль доступа по протоколу HTTP для управления доступом к защищенным ресурсам:

```
location /admin/ {
    deny 10.0.0.1;
```

```
    allow 10.0.0.0/20;
    allow 2001:0db8::/32;
    deny all;
}
```

Данный блок разрешает доступ с любого адреса IPv4 в 10.0.0.0/20, кроме 10.0.0.1, доступ с IPv6-адресов в подсети `2001:0db8::/32` и возвращает ошибку 403 для запросов, исходящих с любого другого адреса. Директивы `allow` и `deny` действительны в контексте HTTP, сервера и местоположения. Правила проверяются последовательно, пока не будет найдено соответствие для удаленного адреса.

Обсуждение

Защита ценных ресурсов и услуг в интернете должна осуществляться в слоях. NGINX предоставляет возможность быть одним из этих слоев. Директива `deny` блокирует доступ к данному контексту, в то время как директива `allow` может использоваться для разрешения подмножеств заблокированного доступа. Вы можете использовать IP-адреса, IPv4 или IPv6, диапазоны блоков бесклассовой адресации, ключевое слово `all` и сокет Unix. Обычно при защите ресурса можно разрешить блок внутренних IP-адресов и запретить отовсюду.

7.2. Разрешение совместного использования ресурсов между разными источниками

Задача

Вы обслуживаете ресурсы из другого домена и вам нужно разрешить совместное использование ресурсов между разными источниками (CORS), чтобы браузеры могли использовать эти ресурсы.

Решение

Измените заголовки на основе метода `request`, чтобы активировать CORS:

```
map $request_method $cors_method {
  OPTIONS 11;
  GET 1;
  POST 1;
  default 0;
```

```
}
server {
    ...
    location / {
      if ($cors_method ~ '1') {
          add_header 'Access-Control-Allow-Methods'
             'GET,POST,OPTIONS';
          add_header 'Access-Control-Allow-Origin'
             '*.example.com';
          add_header 'Access-Control-Allow-Headers'
                     'DNT,
                     Keep-Alive,
                     User-Agent,
                     X-Requested-With,
                     If-Modified-Since,
                     Cache-Control,
                     Content-Type';
      }
      if ($cors_method = '11') {
          add_header 'Access-Control-Max-Age' 1728000;
          add_header 'Content-Type' 'text/plain; charset=UTF-8';
          add_header 'Content-Length' 0;
          return 204;
      }
    }
}
```

В этом примере происходит много всего, что было сжато при использовании карты для группировки методов `GET` и `POST`. Метод запроса `OPTIONS` возвращает *предварительный* запрос клиенту о правилах CORS этого сервера. Методы `OPTIONS`, `GET` и `POST` разрешены правилами CORS. Установка заголовка `Access-Control-Allow-Origin` позволяет также использовать контент, обслуживаемый этим сервером, на страницах происхождения, которые соответствуют этому заголовку. Предварительный запрос можно кешировать для клиента на 1 728 000 секунд, или 20 дней.

Обсуждение

Такие ресурсы, как JavaScript, используют технологию CORS, когда запрашиваемый ими ресурс находится в домене, отличном от их собственного. Когда запрос рассматривается как предполагающий совмест-

ное применение источников, браузер должен соблюдать правила CORS. Браузер не будет использовать ресурс, если у него нет заголовков, которые конкретно разрешают его использование. Чтобы наши ресурсы могли использоваться другими поддоменами, мы должны установить заголовки CORS, что можно сделать с помощью директивы `add_header`. Если запрос – это метод `GET`, `HEAD` или `POST` со стандартным типом контента, и у запроса нет специальных заголовков, браузер выполнит запрос и проверит только источник. Другие методы запроса заставят браузер выполнить предварительный запрос, чтобы проверить условия сервера, которому он будет подчиняться для этого ресурса. Если вы не установите эти заголовки надлежащим образом, браузер выдаст ошибку при попытке использовать этот ресурс.

7.3. Шифрование на стороне клиента

Задача

Вам необходимо зашифровать трафик между вашим сервером NGINX и клиентом.

Решение

Используйте один из SSL-модулей, например `ngx_http_ssl_module` или `ngx_stream_ssl_module`, для шифрования трафика:

```
http { # All directives used below are also valid in stream
    server {
        listen 8433 ssl;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers HIGH:!aNULL:!MD5;
        ssl_certificate /etc/nginx/ssl/example.pem;
        ssl_certificate_key /etc/nginx/ssl/example.key;
        ssl_certificate /etc/nginx/ssl/example.ecdsa.crt;
        ssl_certificate_key /etc/nginx/ssl/example.ecdsa.key;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 10m;
    }
}
```

Данная конфигурация настраивает сервер на прослушивание порта 8443, зашифрованного с помощью SSL. Сервер принимает протоколы SSL версий TLSv1.2 и TLSv1.3. Для применения данным сервером рас-

крываются два набора местоположений сертификата и пар ключей. Серверу предписывается применять наивысшую предлагаемую его клиентом неприступность при ограничении небольшим числом, не являющимся безопасным. Поскольку мы предоставили некую пару ключей ECC (Elliptic Curve Cryptopgraphy), наивысшим приоритетом обладает именно шифрование ECC. Имеющиеся кеш SSL сеанса и тайм-аут делают для исполнителей возможным кешировать и сохранять параметры сеанса на заданный промежуток времени. Существует большое число прочих параметров кеширования сеанса, которые способствуют производительности и безопасности для любых видов вариантов применения. Вы можете использовать параметры кеширования сеанса совместно друг с другом. Однако если определять параметр без значения по умолчанию, это отключит значение по умолчанию, которое встроено в кеширование сеанса.

Обсуждение

Безопасные транспортные уровни являются наиболее распространенным способом шифрования информации при передаче. На момент написания этих строк предпочтительнее использовать протокол TLS вместо SSL, потому что версии SSL с 1 по 3 теперь считаются небезопасными. Хотя имя протокола может отличаться, TLS по-прежнему устанавливает уровень защищенных сокетов. NGINX позволяет вашей службе защищать информацию между вами и вашими клиентами, что, в свою очередь, защищает клиента и ваш бизнес. При использовании подписанного сертификата необходимо объединить сертификат с цепочкой сертификатов центра сертификации. Когда вы объединяете свой сертификат и цепочку, ваш сертификат должен находиться над цепочкой в файле. Если ваш центр сертификации предоставил большое количество файлов в цепочке, он также может указать порядок их размещения. Кеш SSL-сессии повышает производительность благодаря отсутствию согласования шифров и версий SSL/TLS.

При тестировании было установлено, что сертификаты ECC быстрее, чем аналогичные сертификаты RSA. Размер ключа меньше, что приводит к возможности обслуживать больше SSL/TLS-соединений и с более быстрым квитированием. NGINX позволяет настраивать несколько сертификатов и ключей, а затем обслуживать оптимальный сертификат для клиентского браузера. Это дает вам возможность воспользоваться преимуществами новой технологии, но при этом обслуживать старых клиентов.

См. также:
https://wiki.mozilla.org/Security/Server_Side_TLS#Recommended_configurations
https://ssl-config.mozilla.org
https://www.ssllabs.com/ssltest/

7.4. Восходящее шифрование

Задача

Вам требуется шифровать трафик между NGINX и конкретной службой восходящего потока, а также устанавливать особые правила согласования для регулировок надежности или когда данный восходящий поток пребывает вне вашей безопасной сетевой среды.

Решение

Для задания правил SSL вам следует применить директивы SSL для своего модуля посредника HTTP:

```
location / {
    proxy_pass https://upstream.example.com;
    proxy_ssl_verify on;
    proxy_ssl_verify_depth 2;
    proxy_ssl_protocols TLSv1.2;
}
```

Эти директивы посредника устанавливают особые правила SSL чтобы подчиняться NGINX. Эти настраиваемые директивы гарантируют, что NGINX удостоверяет: данный сертификат и цепочка в данной службе восходящего потока подтверждены до двух сертификатов в глубину. Соответствующая директива `proxy_ssl_protocols` определяет, что этот NGINX будет применять только TLS с версией 1.2. По умолчанию NGINX не проверяет сертификаты восходящего потока и применяет все версии TLS.

Обсуждение

Имеющиеся директивы для нашего модуля посредника HTTP очень многочисленны, а если вам требуется шифровать трафик восходящего потока, следует, по крайней мере, включить установление подлинности. Вы можете выступать посредником поверх HTTPS, изменяя протокол в том значении, которое передается в соответствующую директиву

proxy_pass. Тем не менее это не подтверждает имеющийся сертификат восходящего потока. Прочие директивы, такие как proxy_ssl_certificate и proxy_ssl_certificate_key, позволяют вам ограничивать шифрование восходящего потока расширенной безопасностью. Вы также можете предписать proxy_ssl_crl или некий список аннулированных сертификатов, перечисляющий сертификаты, которые более не рассматриваются как допустимые. Такие директивы посредника SSL помогают упрочнять каналы взаимодействия вашей системы внутри вашей собственной сетевой среды или поверх общедоступного интернета.

7.5. Безопасность местоположения

Задача

Вам требуется при помощи ключа безопасности защитить блок location.

Решение

Используйте модуль безопасных ссылок и директиву secure_link_secret, чтобы ограничить доступ к ресурсам для пользователей, у которых есть защищенная ссылка:

```
location /resources {
    secure_link_secret mySecret;
    if ($secure_link = "") { return 403; }

    rewrite ^ /secured/$secure_link;
}

location /secured/ {
    internal;
    root /var/www;
}
```

Данная конфигурация создает некий внутренний и повернутый в общедоступную сторону блок location. Развернутый в сторону общей доступности блок location/resources будет возвращать ошибку 403 Forbidden всякий раз, когда URI его запроса содержит некую строку хеширования md5, которая не может быть проверена тем кодом безопасности, что предоставляется в установленной директиве secure_link_secret. Значение переменной $secure_link является некой пустой строкой, пока значение хеша URI не удостоверено.

Обсуждение

Предоставление безопасности при помощи кода безопасности является великолепным способом гарантии того, что ваши файлы защищены. Такой код безопасности применяется в сочетании со значением URI. Данная строка далее хешируется посредством `md5`, и в получаемом URI применяются шестнадцатеричные цифры данного хеша `md5`. Этот хеш помещается в надлежащую ссылку и вычисляется NGINX. NGINX обладает информацией относительно значений, запрашиваемых пути и файла, поскольку они пребывают в URI после значения хеша. NGINX также знает ваш код безопасности (secret), поскольку он предоставляется через соответствующую директиву `secure_link_secret`. NGINX имеет возможность быстро удостоверять значение хеша `md5` и сохранять получаемый URI в значении переменной `$secure_link`. Если же значение хеша не подтверждается, значение переменной устанавливается в некую пустую строку. Важно отметить, что те аргументы, которые передаются в `secure_link_secret`, обязаны быть статической строкой; они не могут быть переменной.

7.6. Генерация безопасного соединения при помощи ключа безопасности

Задача

Вам требуется сгенерировать в своем приложении некую безопасную ссылку при помощи какого-то кода безопасности.

Решение

Имеющийся в NGINX модуль безопасной ссылки принимает значения шестнадцатеричных цифр некой хешированной `md5` строки, причем такая строка представляет собой сцепление значения URI пути и самого кода безопасности (secret). Отталкиваясь от последнего рецепта 7.5, мы создадим защищенную ссылку, которая будет работать с предыдущим примером конфигурации, с учетом того что имеется некий файл, представленный в */var/www/secured/index.html*. Для выработки требуемых значений шестнадцатеричных цифр `md5` мы можем применить команду Unix `openssl`:

```
$ echo -n 'index.htmlmySecret' | openssl md5 -hex
(stdin)= a53bee08a4bf0bbea978ddf736363a12
```

Здесь мы показываем значение защищаемого нами URI, *index.html*, сцепляемое с нашим кодом безопасности, `mySecret`. Эта строка передается в соответствующую команду `openssl` для вывода шестнадцатеричных цифр `md5`.

Ниже приводится пример построения в Python тех же самых шестнадцатеричных цифр с применением библиотеки `hashlib`, которая входит в состав Стандартной библиотеки Python:

```
import hashlib
hashlib.md5.(b'index.htmlmySecret').hexdigest()
'a53bee08a4bf0bbea978ddf736363a12'
```

Теперь, когда у нас имеются данные шестнадцатеричные цифры, мы можем применять их в некоем URL. Нашим примером будет www.example.com, выполняющий некий запрос для своего файла */var/www/secured/index.html* через location */resources*. Наш полный URL-адрес будет выглядеть так:

```
www.example.com/resources/a53bee08a4bf0bbea978ddf736363a12/\
index.html
```

Обсуждение

Выработка цифровых значений может выполняться множеством способов, причем на множестве языков программирования. Что нужно помнить: значение пути URI следует перед самим кодом безопасности, в этой строке нет никаких символов перехода на новую строку, а также применяются шестнадцатеричные цифры из соответствующего хеширования `md5`.

7.7. Безопасность местоположения при помощи ограниченной даты

Задача

Вам требуется защитить некое местоположение при помощи ссылки, которая имеет срок истечения в будущем и является специфичной для некоего клиента.

Решение

Для настройки срока истечения действия воспользуйтесь прочими включенными в обсуждаемый модуль безопасных ссылок директивами и применяйте переменные в своей защищенной ссылке:

```
location /resources {
    root /var/www;
    secure_link $arg_md5,$arg_expires;
    secure_link_md5 "$secure_link_expires$uri$remote_addr mySecret";
    if ($secure_link = "") { return 403; }
    if ($secure_link = "0") { return 410; }
}
```

Приводимая директива `secure_link` получает два разделяемых запятой параметра. Значением первого параметра является переменная, которая содержит значение хеша `md5`. Данный пример использует некий аргумент HTTP `md5`. Вторым параметром является переменная, которая содержит значение времени истечения срока данной ссылки, исчисляемого в формате времени эпохи Unix. Следующая директива `secure_link_md5` получает некий отдельный параметр, объявляющий значение формата той строки, которая применяется для построения значения хеша `md5`. Как и для прочих конфигураций, когда это значение хеша не проходит процедуру удостоверения, значение переменной `$secure_link` устанавливается в некую пустую строку. Тем не менее при данном применении в случае соответствия значения хеша, но при истечении срока действия значение переменной `$secure_link` будет установлено на 0.

Обсуждение

Данное применение защищенной ссылки является более гибким и выглядит более понятным, чем тот `secure_link_secret`, который показан в рецепте 7.5. При помощи этих директив мы способны применять любое число доступных для NGINX переменных в своей хешируемой строке. Использование специфичных для пользователя переменных в такой строке хеша усилит вашу безопасность, поскольку у пользователей не будет возможности выставлять на продажу ссылки к защищенным ресурсам. Рекомендуется применять переменные, подобные `$remote_addr` или `$http_x_forwarded_for`, либо некий заголовок куки сеанса, вырабатываемый вашим приложением. Значения аргументов для `secure_link` может поступать из любых предпочитаемых вами переменных, и они могут именоваться так, чтобы лучше соответствовать вашим потребностям. Значения условий относительно значения переменной `$secure_link` устанавливаются для возврата известных кодов HTTP для Forbidden и Gone. Значение 410 HTTP, Gone прекрасно подходит для просроченных ссылок, поскольку данное условие рассматривается как неизменное.

7.8. Генерация ссылки с ограниченным сроком

Задача

Вам требуется выработать ссылку со сроком истечения.

Решение

Сгенерируйте некий временной штамп (timestamp) для значения срока истечения в формате эпохи Unix. В системе Unix можно выполнить проверку, применив значение даты, как это продемонстрировано в приведенном ниже примере:

```
$ date -d "2020-12-31 00:00" +%s --utc
1609372800
```

Далее вам требуется сцепить строку своего хеша для установления значения строки, настраиваемой при помощи директивы `secure_link_md5`. В данном случае нашей используемой строкой будет `1293771600/resources/index.html127.0.0.1 mySecret`. Значение хеша `md5` слегка отличается от просто шестнадцатеричных цифр. Это хеш `md5` в двоичном формате, кодированный base64, причем символы плюса (+) транслируются в дефисы (-), слеши (/) – в подчеркивания (_), а символы равенства (=) удаляются. Вот пример в системе Unix:

```
$ echo -n '1609372800/resources/index.html127.0.0.1 mySecret' \
  | openssl md5 -binary \
  | openssl base64 \
  | tr +/ -_ \
  | tr -d =
TG6ck3OpAttQ1d7jW3JOcw
```

Теперь, когда у нас есть свой хеш, мы можем воспользоваться им в качестве некоего аргумента совместно с датой истечения срока:

```
'/resources/index.html?md5=TG6ck3OpAttQ1d7jW3JOcw&expires=1609372800'
```

Далее приводится более практичный пример на Python, применяющий относительное время для срока истечения, устанавливающий значение срока истечения ссылки на один час с момента его выработки. На момент написания этих строк этот пример работал с Python 2.7 b 3.x с использованием стандартной библиотеки Python:

```
from datetime import datetime, timedelta
from base64 import b64encode
import hashlib

# Set environment vars
resource = b'/resources/index.html'
remote_addr = b'127.0.0.1'
host = b'www.example.com'
mysecret = b'mySecret'

# Generate expire timestamp
now = datetime.utcnow()
expire_dt = now + timedelta(hours=1)
expire_epoch = str.encode(expire_dt.strftime('%s'))

# md5 hash the string
uncoded = expire_epoch + resource + remote_addr + mysecret
md5hashed = hashlib.md5(uncoded).digest()

# Base64 encode and transform the string
b64 = b64encode(md5hashed)
unpadded_b64url = b64.replace(b'+', b'-')\
    .replace(b'/', b'_')\
    .replace(b'=', b'')

# Format and generate the link
linkformat = "{}{}?md5={}?expires={}"
securelink = linkformat.format(
    host.decode(),
    resource.decode(),
    unpadded_b64url.decode(),
    expire_epoch.decode()
)
print(securelink)
```

Обсуждение

При наличии данного шаблона мы имеем возможность вырабатывать некую защищенную ссылку в особом формате, которая может применяться в URI. Данный код безопасности (secret) предоставляет

защиту при помощи использования переменных, которые никогда не отправляются определенному клиенту. У вас есть возможность применять столько переменных, сколько вам потребуется, чтобы сделать свое местоположение безопасным. Хеширование `md5` и кодирование base64 являются распространенными, обладают малым весом и доступны почти во всех языках программирования.

7.9. Перенаправление HTTPS

Задача

Вам требуется перенаправлять незашифрованные запросы в HTTPS.

Решение

Воспользуйтесь повторной записью для отправки всего HTTP-трафика в HTTPS:

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name _;
    return 301 https://$host$request_uri;
}
```

В данной конфигурации мы слушаем порт 80 в качестве своего сервера по умолчанию как для IPv4, так и для IPv6, а также для любых названий хостов. Оператор `return` возвращает некий код 301, постоянно выполняя перенаправление в имеющийся в том же самом хосте сервер HTTP и подставляя URI запроса.

Обсуждение

Важно всегда выполнять перенаправление в HTTPS, когда он доступен. Вы можете обнаружить, что вам требуется перенаправлять не все запросы, а только те, которые содержат чувствительную информацию, подлежащую передаче между клиентом и сервером. В данном случае вы можете пожелать поместить в оператор `return` только конкретные расположения, например */login*.

7.10. Перенаправление на HTTPS, когда SSL/TLS прекращается до NGINX

Задача

Вам требуется выполнять перенаправление в HTTPS, однако у вас истек срок SSL/TLS на уровне перед NGINX.

Решение

Чтобы выяснить, требуется ли вам выполнить перенаправление, воспользуйтесь стандартным заголовком `X-Forwarded-Proto`:

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name _;
    if ($http_x_forwarded_proto = 'http') {
        return 301 https://$host$request_uri;
    }
}
```

Данная конфигурация во многом схожа с перенаправлением HTTPS. Тем не менее в данной конфигурации мы выполняем перенаправление, только когда значение заголовка `X-Forwarded-Proto` эквивалентно HTTP.

Обсуждение

Достаточно распространен случай, когда у вас может завершиться срок действия SSL/TLS на некоем уровне перед NGINX. Одной из причин может быть то, что вы делаете нечто подобное сбережению затрат на вычисления. Тем не менее вам требуется быть уверенным, что все запросы являются HTTPS, но при этом сам уровень с прекращенным SSL/TLS не имеет возможности перенаправления. Он способен, однако, устанавливать заголовки посредника. Эта конфигурация работает с такими уровнями, как Amazon Web Services Elastic Load Balancer, который выполняет разгрузку SSL/TLS без дополнительных затрат. Это удобный трюк для обеспечения защиты вашего HTTP-трафика.

7.11. Строгая безопасность доставки HTTP

Задача

Вам требуется проинформировать браузеры, что им никогда не следует отправлять запросы поверх HTTP.

Решение

Воспользуйтесь расширением HSTS (HTTP Strict Transport Security), устанавливая заголовок `Strict-Transport-Security`:

```
add_header Strict-Transport-Security max-age=31536000;
```

Данная конфигурация устанавливает значение заголовка `Strict-Transport-Security` на максимальный возраст в один год. Он будет инструктировать соответствующий браузер всегда осуществлять некое внутреннее перенаправление при попытке выполнения к данному домену запросов HTTP, чтобы все запросы выполнялись поверх HTTPS.

Обсуждение

Для некоторых приложений один перехваченный атакой с человеком в середине запрос HTTP может стать концом всей компании. Если сообщение некой содержащей чувствительную информацию формы отправляется по HTTP, перенаправление HTTPS из NGINX не спасет вас; взлом уже произошел. Данный вариант улучшения безопасности информирует браузер запросов никогда не делать никаких запросов HTTP, а следовательно, они никогда не отправляются в незащищенном виде.

См. также:
https://tools.ietf.org/html/rfc6797
https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet

7.12. Удовлетворение любого числа методов безопасности

Задача

Вам требуется предоставить множество способов передачи безопасности закрытой площадке.

Решение

Для указания того, что вы желаете удовлетворять неким или даже всем применяемым методам безопасности, для выдачи NGINX инструкции об этом примените директиву `satisfy`:

```
location / {
    satisfy any;

    allow 192.168.1.0/24;
    deny all;

    auth_basic           "closed site";
    auth_basic_user_file conf/htpasswd;
}
```

Данная конфигурация сообщает NGINX, что запрашивающий `location`/ пользователь должен удовлетворять одному из установленных методов безопасности: либо эти запросы должны происходить с блока CIDR *192.168.1.0/24*, либо быть способными поддерживать некие имя пользователя и пароль, которые могут быть найдены в обозначенном файле *conf/htpasswd*. Данная директива `satisfy` имеет всего два варианта: `any` или `all`.

Обсуждение

Описанная директива `satisfy` является великолепным способом предложения множества способов аутентификации для вашего веб-приложения. Определяя `any` для директивы `satisfy`, данный пользователь обязан соответствовать одной из предоставленных защит. Определяя в директиве `satisfy` значение `all`, запрашивающий пользователь должен соответствовать всем выставленным требованиям безопасности. Данная директива может применяться совместно с `http_access_module`, описанном в рецепте 7.1, `http_auth_basic_module`, детализированном в рецепте 6.1, `http_auth_request_module`, описанном в рецепте 6.2 и представленном в рецепте 6.3 `http_auth_jwt_module`. Обсуждаемая директива `satisfy` поможет вам достичь этого для местоположений и серверов, которые требуют глубоких правил защиты.

7.13. Динамичное ослабление DDoS

Задача

Вам необходимо динамичное ослабление DDoS (Distributed Denial of Service).

Решение

Для создания поддерживающего кластер ограничения скорости запросов и автоматического черного списка воспользуйтесь NGINX Plus:

```
limit_req_zone   $remote_addr zone=per_ip:1M rate=100r/s sync;
                 # Cluster-aware rate limit
limit_req_status 429;

keyval_zone zone=sinbin:1M timeout=600 sync;
              # Cluster-aware "sin bin" with
              # 10-minute TTL
keyval $remote_addr $in_sinbin zone=sinbin;
              # Populate $in_sinbin with
              # matched client IP addresses

server {
    listen 80;
    location / {
        if ($in_sinbin) {
            set $limit_rate 50; # Restrict bandwidth of bad clients
        }

        limit_req zone=per_ip;
              # Apply the rate limit here
        error_page 429 = @send_to_sinbin;
              # Excessive clients are moved to
              # this location
        proxy_pass http://my_backend;
    }

    location @send_to_sinbin {
        rewrite ^ /api/3/http/keyvals/sinbin break;
```

```
            # Set the URI of the
            # "sin bin" key-val
        proxy_method POST;
        proxy_set_body '{"$remote_addr":"1"}';
        proxy_pass http://127.0.0.1:80;
    }

    location /api/ {
        api write=on;
        # directives to control access to the API
    }
}
```

Обсуждение

Это решение использует некий синхронизируемый предел частот обращений и синхронизируемое хранилище ключ/значение для динамических откликов на DDoS атаки и смягчения их действия. Параметр `sync`, предоставленный директивам `limit_req_zone` и `keyval_zone`, синхронизирует зону общей памяти с другими компьютерами в активном-активном кластере NGINX Plus. Данный пример выявляет клиентов, которые отправляют более 100 запросов в секунду вне зависимости от того, какой из узлов NGINX Plus получает такой запрос. После того как некий клиент превышает установленный предел частот, его IP адрес добавляется в некое хранилище ключ/значение Sin Bin (Скамейка штрафников) через выполнение запроса API NGINX Plus. Эта Скамейка штрафников синхронизируется по всему кластеру. Последующие запросы от попадающих на Скамейку штрафников клиентов являются предметом очень низкого ограничения полосы пропускания, причем вне зависимости от того, какой из узлов NGINX Plus получает их. Ограничение полосы пропускания более предпочтительно, чем полное отклонение запросов, ибо оно не становится очевидным сигналом для такого клиента, что смягчение DDoS атаки вступило в действие. После 10 мин данный клиент автоматически удаляется со Скамейки штрафников.

Глава 8

HTTP/2

8.0. Введение

HTTP/2 является основной ревизией протокола HTTP. Большая часть выполненной в этой версии работы сосредоточена на его транспортном уровне, например разрешение мультиплексирования полного запроса и отклика поверх отдельного подключения TCP. За счет применения сжатия полей заголовка HTTP были достигнуты высокие показатели эффективности, к тому же была добавлена поддержка приоритетов для запросов. Другим крупным добавлением в этот протокол стала возможность для конкретного сервера активно доставлять сообщения своему клиенту. В данной главе описываются подробности базовой конфигурации для включения в NGINX HTTP/2, а также настройки поддержки gRPC и активной доставки сервера HTTP/2.

8.1. Базовая настройка

Задача

Вы желаете воспользоваться преимуществами HTTP/2.

Решение

Включите HTTP/2 в своем сервере NGINX:

```
server {
    listen 443 ssl http2 default_server;

    ssl_certificate server.crt;
    ssl_certificate_key server.key;
    ...
}
```

Обсуждение

Для включения HTTP/2 вам просто требуется добавить соответствующий параметр `http2` к своей директиве `listen`. Однако подвох в том, что, хотя сам протокол и не требует обертки своего подключения в SSL/TLS, некоторые реализации клиентов HTTP/2 поддерживают исключительно HTTP/2 поверх шифрованных подключений. Еще одно предостережение состоит в том, что в самой спецификации HTTP/2 несколько комплектов шифров TLS 1.2 занесено в черный список и, следовательно, будут отказывать при квитировании (handshake). Применяемые в NGINX по умолчанию шифры не находятся в этом черном списке. Чтобы проверить, что ваша установка правильная, можно установить подключаемый модуль для браузеров Chrome и Firefox, который указывает, когда какой-либо сайт применяет HTTP/2 или из командной строки при помощи утилиты `nghttp`.

См. также:

https://tools.ietf.org/html/rfc7540#appendix-A

https://chrome.google.com/webstore/detail/http2-and-spdy-indicator/mpbpobfflnpcgagjijhmgnchggcjblin

https://addons.mozilla.org/en-US/firefox/addon/http2-indicator/

8.2. gRPC

Задача

Вам требуется прекратить, проинспектировать, маршрутизировать или осуществить балансировку нагрузки вызовов метода gRPC.

Решение

Для посредничества (прокси) подключений gRPC воспользуйтесь NGINX.

```
server {
    listen 80 http2;

    location / {
        grpc_pass grpc://backend.local:50051;
    }
}
```

В этой конфигурации NGINX слушает порт 80 для HTTP/2-трафика без шифрования, а также выступает посредником для машины с названием **backend.local** на порту 50051. Установленная директива grpc_pass указывает NGINX на необходимость рассматривать такое подключение как некий вызов gRPC. Указание grpc:// в самом начале местоположения нашего бэкенд-сервера не требуется; тем не менее это непосредственно указывает на то, что данное бэкенд-подключение не шифруется.

Чтобы воспользоваться шифрованием TLS между клиентом и NGINX и прерывать такое шифрование перед передачей вызова серверу приложений, включите SSL и HTTP/2, как вы это делали в первом разделе:

```
server {
    listen 443 ssl http2 default_server;

    ssl_certificate server.crt;
    ssl_certificate_key server.key;
    location / {
        grpc_pass grpc://backend.local:50051;
    }
}
```

Данная конфигурация прекращает TLS в NGINX и передает соответствующее взаимодействие gPRC в имеющееся приложение поверх незашифрованного HTTP/2.

Для настройки NGINX на шифрование всего взаимодействия gRPC со своим сервером приложения предоставьте повсеместное шифрование трафика, просто изменив имеющуюся директиву grpc_pass, определив grpcs:// перед уже имеющейся информацией о сервере (обратите внимание на добавление символа s, обозначающего безопасное взаимодействие):

```
    grpc_pass grpcs://backend.local:50051;
```

Вы также можете применять NGINX для маршрутизации вызовов к различным серверным службам на основе значения URI gRPC, которые включают пакеты, службы и методы. Для этого воспользуйтесь директивой location.

```
location /mypackage.service1 {
    grpc_pass grpc://backend.local:50051;
}
location /mypackage.service2 {
```

```
        grpc_pass grpc://backend.local:50052;
    }
    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
    }
```

Данный пример конфигурации применяет директиву `location` для направления входящего трафика HTTP/2 между двумя обособленными службами gRPC, а также `location` для обслуживания статического содержимого. Вызовы метода для установленной службы `mypackage.service1` направляются в сервер `backend.local` по порту `50051`, а вызовы для `mypackage.service2` отправляются в порт `50052`. `location /`, перехватывают все прочие запросы HTTP и обслуживают статическое содержимое. Это демонстрирует, как NGINX способно обслуживать gRPC и не-gRPC под одной и той же конечной точкой HTTP/2 и выполнять маршрутизацию надлежащим образом.

Вызовы балансировки нагрузки также похожи на трафик HTTP без gRPC:

```
upstream grpcservers {
    server backend1.local:50051;
    server backend2.local:50051;
}
server {
    listen 443 ssl http2 default_server;

    ssl_certificate server.crt;
    ssl_certificate_key server.key;
    location / {
        grpc_pass grpc://grpcservers;
    }
}
```

Обсуждение

NGINX обладает возможностью получать, выступать посредником, выполнять балансировку нагрузки, осуществлять маршрутизацию и прекращать шифрование для вызовов gRPC. Имеющийся модуль gRPC позволяет NGINX устанавливать, изменять или сбрасывать заголовки вызова gRPC, выставлять тайм-ауты для запросов и устанавливать

спецификации восходящего SSL/TLS. Так как gRPC выполняет взаимодействие поверх протокола HTTP/2, вы можете настроить NGINX на прием gRPC и не-gRPC веб-трафика в одной и той же конечной точке.

8.3. Сервер активной доставки HTTP/2

Задача

Вам требуется активно доставлять своему клиенту содержимое упреждающим образом.

Решение

Воспользуйтесь функциональностью NGINX активной доставки HTTP/2.

```
server {
    listen 443 ssl http2 default_server;

    ssl_certificate server.crt;
    ssl_certificate_key server.key;
    root /usr/share/nginx/html;

    location = /demo.html {
        http2_push /style.css;
        http2_push /image1.jpg;
    }
}
```

Обсуждение

Для применения активной доставки сервера HTTP/2 ваш сервер обязан быть настроенным для HTTP/2, как это было нами сделано в рецепте 7.1. Здесь вы можете указать NGINX на необходимость упреждающей активной доставки определенных файлов при помощи директивы `http2_push`. Эта директива получает один параметр, значение полного пути URI того файла, который активно доставляется вашему клиенту.

NGINX также способно автоматически активно доставлять клиентам ресурсы, если выступает посредником (прокси) в приложениях, содержащих некий заголовок отклика HTTP с названием `Link`. Такой заголовок имеет возможность указывать NGINX на упреждающую загрузку предписанных ресурсов. Чтобы включить данную функциональность, добавьте в настройки NGINX `http2_push_preload on;`.

Глава 9

Управление сложными потоками медиа

9.0. Введение

В этой главе рассматриваются медиапотоки через NGINX в форматах MPEG-4 или Flash Video. NGINX широко применяется для массивного распространения и потокового содержимого. NGINX поддерживает форматы стандартов отрасли и потоковые технологии, которые будут обсуждены в данной главе. NGINX Plus делает возможным фрагментирование содержимого на лету при помощи модуля HTTP Live Stream, а также возможность доставки HTTP Dynamic Streaming, который уже является фрагментированным типом медиа. NGINX естественным образом допускает пределы для полос пропускания, а расширенные свойства NGINX Plus предлагают пределы скорости передачи битов, что делает возможной доставку наиболее действенным способом при резервировании имеющихся ресурсов сервера для охвата большего числа пользователей.

9.1. Обслуживание MP4 и FLV

Задача

Вам требуется отправлять поток цифровых данных, изначально представленных в формате MPEG-4 (MP4) или Flash Video (FLV).

Решение

Обозначьте блок местоположения HTTP как *.mp4* или *.flv*. NGINX будет управлять потоком медиа, применяя прогрессивные выгрузки или псевдопотоки HTTP с поддержкой поисков:

```
http {
    server {
        ...

        location /videos/ {
            mp4;
        }
        location ~ \.flv$ {
            flv;
        }
    }
}
```

Блок местоположения данного примера сообщает, что файлы в каталоге *videos* представлены в формате с типом MP4 и могут направляться потоком с поддержкой прогрессивной выгрузки. Второй блок местоположения указывает NGINX, что все файлы, завершающиеся на *.flv*, представлены в формате FLV и могут направляться потоком с поддержкой псевдопотоков HTTP.

Обсуждение

Потоковые видео- и аудиофайлы в NGINX являются неким образцом отдельной директивы. Прогрессивная выгрузка делает возможным для клиента инициировать воспроизведение таких медиа до того, как завершится выгрузка всего файла. NGINX поддерживает операции поиска по неким пока не выгруженным порциям этих медиа в обоих форматах.

9.2. Организация потоков с помощью HLS

Задача

Вам требуется поддержка HTTP Live Streaming (HLS) для кодированного H.264/AAC содержимого, упакованного в файлы MP4.

Решение

Воспользуйтесь модулем HLS NGINX Plus с сегментацией в реальном масштабе времени, пакетизацией и мультиплексированием, а также с контролем буферизации фрагментов и др. в качестве аргументов перенаправления HLS:

```
location /hls/ {
    hls; # Use the HLS handler to manage requests

    # Serve content from the following location
    alias /var/www/video;
    # HLS parameters
    hls_fragment            4s;
    hls_buffers         10 10m;
    hls_mp4_buffer_size     1m;
    hls_mp4_max_buffer_size 5m;
}
```

Данный блок местоположения демонстрирует направления NGINX для потоков медиа HLS из каталога */var/www/video*, причем с фрагментированием медиа на сегменты по четыре секунды. Общее число буферов HLS устанавливается в 10 с размером по 10 Мб. Начальный размер буфера MP4 устанавливается в 1 Мб с максимальным значением в 5 Мб.

Обсуждение

Доступный в NGINX Plus модуль HLS предоставляет свои возможности для мультиплексирования транспортировки файлов с медиа MP4 на лету. Существует множество директив, которые предоставляют вам возможность контроля над тем, как фрагментируются и буферизуются ваши медиа. Определенный блок местоположения должен быть настроен для обслуживания медиа как некоего потока HLS с соответствующим обработчиком HLS. Значение фрагментации HLS устанавливается в числе секунд, что указывает NGINX на необходимость фрагментирования по длине времени. Общее число буферируемых данных устанавливается с помощью директивы `hls_buffers`, определяющей количество буферов и их размер. Данному клиенту разрешается начинать воспроизведение его медиа после осуществления накопления буферизации определенного объема, определяемого `hls_mp4_buffer_size`. Однако может потребоваться больший размер буфера, если метаданные для этого видео могут превышать первоначальный размер буфера. Такой объем блокируется сверху значением `hls_mp4_max_buffer_size`. Эти переменные буферизации позволяют NGINX оптимизировать практику, получаемую конечным пользователем; выбор правильных значений для этих директив требует знания целевой аудитории и ваших медиа. Например, если имеющиеся пачки ваших медиа составляют видеофайлы, а ваша целевая аудитория имеет широкую полосу пропускания, можете выполнить оптимизацию

большим размером максимума буфера и фрагментированием с большей длиной. Это позволит вам изначально выгружать без ошибок метаданные относительно содержимого, а вашим пользователям получать фрагменты с большей длиной.

9.3. Организация потоков с помощью HDS

Задача

Вам требуется поддержка HDS (HTTP Dynamic Streaming) Adobe, который уже был фрагментирован и обособлен от своих метаданных.

Решение

Для предложения своим пользователям Adobe Adaptive Streaming воспользуйтесь модулем поддержки файлов FLV (`F4F`) из NGINX Plus:

```
location /video/ {
    alias /var/www/transformed_video;
    f4f;
    f4f_buffer_size 512k;
}
```

Данный пример указывает NGINX Plus на необходимость обслуживать изначально фрагментированное видео из некоего местоположения на диске своему клиенту при помощи модуля `F4F` NGINX Plus. Значение размера буфера для его индексного файла (*.f4x*) устанавливается в 512 Кб.

Обсуждение

Модуль `F4F` позволяет NGINX обслуживать для конечных пользователей изначально фрагментированные медиа. Сама настройка этого проста и состоит в применении обработчика `F4F` внутри некоего блока местоположения HTTP. Значение директивы `f4f_buffer_size` настраивает величину размера буфера для файла индексации данного типа медиа.

9.4. Пределы полосы пропускания

Задача

Вам требуется лимитировать полосу пропускания нисходящего медиапотока клиентов без воздействия на практику их просмотра.

Решение

Примените поддержку побитовой скорости NGINX Plus для файлов медиа MP4:

```
location /video/ {
    mp4;
    mp4_limit_rate_after 15s;
    mp4_limit_rate       1.2;
}
```

Данная настройка позволяет клиентам вашего нисходящего потока выгружать 15 с до применения ограничения битовой скорости. После 15 с данному клиенту разрешается выгружать медиа с соотношением 120 % к установленной скорости обмена данных, что делает возможным этому клиенту всегда выполнять выгрузку быстрее воспроизведения.

Обсуждение

Ограничение скорости обмена NGINX Plus позволяет вашему потоковому серверу динамично ограничивать полосу пропускания для всего подлежащего обслуживанию медиафайла, позволяя клиентам выгрузку даже быстрее, чем им это требуется, чтобы обеспечить некую бесшовную практику пользователям. Обсуждаемый обработчик MP4, описанный в рецепте 9.1, озаглавливает этот блок местоположения для форматов MP4 медиа. Ограничивающие скорость директивы, такие как `mp4_limit_rate_after`, задают NGINX ограничивать скорость трафика только через указанное время в секундах. Другой директивой, вовлеченной в ограничение скорости MP4, является `mp4_limit_rate`, которая определяет значение скорости обмена данными для медиа. Значение 1, предоставляемое для директивы `mp4_limit_rate`, означает, что NGINX ограничивает полосу пропускания скоростью воспроизведения этого типа медиа (1/1). Предоставление для данной директивы значения `mp4_limit_rate`, большего единицы, позволит пользователям выполнять выгрузку быстрее, чем они просматривают, с тем чтобы они были способны осуществлять буферизацию медиа и незаметно просматривать его в процессе его выгрузки.

Глава 10

Развертывание в облачных решениях

10.0. Введение

Появление поставщиков облачных решений изменило ландшафт хостинга веб-приложений. Такой процесс, как подготовка новой машины раньше занимал часы или месяцы; теперь ее можно создать максимально быстро, как щелчок мышью или API-вызов. Эти облачные провайдеры сдают в аренду свои виртуальные машины – эта модель носит название *инфраструктура как услуга* (IaaS) – или управляемые программные решения, такие как базы данных, по модели pay-per-use, что означает, что вы платите только за то, что используете. Это позволяет инженерам создавать целые среды для тестирования в любой момент и отключать их, когда они больше не нужны. Облачные провайдеры также позволяют приложениям масштабироваться по горизонтали в зависимости от потребности в производительности в любой момент. В этой главе рассматриваются базовые развертывания с использованием NGINX и NGINX Plus на нескольких основных платформах облачных провайдеров.

10.1. Автоматическая настройка в AWS

Задача

Вам нужно автоматизировать настройку серверов NGINX в Amazon Web Services, чтобы машины могли автоматически настраивать сами себя.

Решение

Используйте `UserData`, а также предварительно настроенный образ машины Amazon. Создайте образ машины Amazon (AMI) с NGINX и любыми установленными пакетами программного обеспечения. Используйте `UserData` для настройки любых специфичных для среды конфигураций во время выполнения.

Обсуждение

Есть три модели мышления при выполнении настройки на AWS.

Настройка при загрузке

Начните с обычного образа Linux, затем запустите управление конфигурацией или сценарии оболочки во время загрузки для настройки сервера. Этот шаблон медленно запускается и может быть подвержен ошибкам.

Полностью настроенные образы машины Amazon

Полностью настройте сервер, затем запишите образ машины Amazon для использования. Этот шаблон запускается очень быстро и аккуратно. Тем не менее он менее гибок по отношению окружающей его среде, и поддержание множества образов может быть сложным.

Частично настроенные образы машины Amazon

Это смесь двух миров. Частично настроенный образ – когда требования к программному обеспечению устанавливаются и записываются в образ машины Amazon, а настройка среды выполняется во время загрузки. Этот шаблон является гибким по сравнению с предыдущим шаблоном и быстрым по сравнению с настройкой при загрузке.

Если вы решите частично или полностью настроить свои образы, этот процесс можно автоматизировать. Чтобы собрать конвейер сборки образов машины Amazon, предлагается использовать несколько инструментов:

Управление конфигурациями

Инструменты управления конфигурацией определяют состояние сервера в коде, например какую версию NGINX следует запускать

и от какого пользователя он должен работать, какой DNS-решатель использовать и куда выполнять проксирование. Этот код управления конфигурациями может располагаться в системе управления версиями как программный проект. Популярные инструменты управления конфигурациями – Ansible, Chef, Puppet и SaltStack, которые были описаны в главе 5.

Packer от компании HashiCorp

Packer используется для автоматизации управления конфигурациями практически на любой виртуальной или облачной платформе, а также для записи образа машины в случае успешного запуска. В основном Packer создает виртуальную машину на выбранной вами платформе, отправляет SSH-пакеты в виртуальную машину, запускает любую заданную вами настройку и записывает образ. Можно использовать Packer для запуска инструмента управления конфигурациями и надежной записи образа машины в соответствии с вашими требованиями.

Для подготовки конфигураций среды во время загрузки можно использовать `UserData` для запуска команд в первый раз, когда экземпляр загружается. Если вы берете частично настроенный образ, можно воспользоваться этим для настройки элементов среды во время загрузки. Примерами конфигураций на основе среды могут быть имена серверов, которые нужно прослушивать, резолвер, который будет использоваться, доменное имя для проксирования или пул вышестоящих серверов.

`UserData` – это строка в кодировке base64, которая скачивается при первой загрузке и запуске. `UserData` может быть такой же простой, как и файл окружения, доступ к которому осуществляется другими процессами начальной загрузки в вашем образе машины Amazon, или же это может быть скрипт, написанный на любом языке, который существует в образе.

Обычно `UserData` – это bash-скрипт, который определяет переменные или скачивает переменные для передачи их в управление конфигурациями. Управление конфигурациями обеспечивает правильную настройку системы, вырабатывает шаблоны файлов конфигурации на базе переменных среды и выполняет перезагрузку служб. После запуска `UserData` ваша машина с NGINX должна быть полностью настроена очень надежным образом.

10.2. Маршрутизация в узлы NGINX без ELB

Задача

Вам необходимо маршрутизировать трафик в несколько активных узлов NGINX или создать отработку отказа «активный-пассивный» для достижения высокой доступности без балансировщика нагрузки перед NGINX.

Решение

Используйте DNS-сервис Amazon Route53 для маршрутизации в несколько активных узлов NGINX или настройте проверку работоспособности и отработку отказа между активно-пассивным набором узлов NGINX.

Обсуждение

DNS в течение долгого времени выполнял балансировку нагрузки между серверами; переход в облако не меняет этого. Сервис Route53 от Amazon предоставляет DNS-службу с большим количеством расширенных функций, доступных через API. Доступны все типичные приемы DNS, такие как несколько IP-адресов в одной записи A и взвешенные записи A.

При работе нескольких активных узлов NGINX вам понадобится использовать одну из этих функций записи A, чтобы распределить нагрузку по всем узлам. Алгоритм циклического перебора используется, когда для одной записи A указаны несколько IP-адресов. Взвешенное распределение может использоваться для неравномерного распределения нагрузки путем определения весов для каждого IP-адреса сервера в записи A.

Одной из наиболее интересных особенностей Route53 является возможность проверки работоспособности. Вы можете настроить Route53 для мониторинга работоспособности конечной точки, установив TCP-соединение или отправив запрос через HTTP или HTTPS. Проверку работоспособности легко настроить с помощью параметров для IP-адреса, имени хоста, порта, пути URI, интервалов, мониторинга и географии. С помощью этих проверок работоспособности Route53 может вывести IP из ротации, если он начинает отказывать. Вы также можете настроить Route53 на аварийное переключение на вторичную запись в случае сбоя, что приведет к настройке «активный–пассивный» с высокой доступностью.

Route53 имеет геологическую функцию маршрутизации, которая позволит вам направлять своих клиентов к ближайшему к ним узлу NGINX с наименьшей задержкой. При маршрутизации по географии ваш клиент направляется в ближайшее работоспособное физическое местоположение. При запуске нескольких наборов инфраструктуры в конфигурации «активный–активный» вы можете автоматически переключаться на другое геологическое местоположение с помощью проверок работоспособности.

При использовании Route53 DNS для маршрутизации трафика в узлы NGINX в группе автоматического масштабирования вам понадобится автоматизировать создание и удаление записей DNS. Чтобы автоматизировать добавление и удаление машин NGINX в Route53 по мере масштабирования узлов NGINX, можно использовать стадии жизненного цикла автоматического масштабирования от Amazon для запуска сценариев в самом NGINX или скрипты, работающие независимо на Amazon Lambda. Эти скрипты будут использовать интерфейс командной строки или набор средств разработки Amazon для взаимодействия с API Amazon Route53, чтобы добавлять или удалять IP-адреса машины NGINX и настроенные проверки работоспособности при загрузке или до ее завершения.

См. также:
https://docs.nginx.com/nginx/deployment-guides/amazon-web-services/route-53-global-server-load-balancing/

10.3. NLB-сэндвич

Задача

Вам необходимо автоматически масштабировать слой NGINX с открытым исходным кодом и равномерно и легко распределять нагрузку между серверами приложений.

Решение

Создайте балансировщик сетевой нагрузки (NLB). Во время создания NLB через консоль вам будет предложено организовать новую целевую группу. Если вы не сделаете это через консоль, вам нужно будет создать этот ресурс и подключить его к слушателю на NLB. Вы организовываете группу автоматического масштабирования с конфигурацией запуска, которая обеспечивает экземпляр EC2 с установленным NGINX с откры-

тым исходным кодом. Группа автоматического масштабирования имеет конфигурацию для связи с целевой группой, которая автоматически регистрирует любой экземпляр в группе автоматического масштабирования в целевой группе, настроенной при первой загрузке. На целевую группу ссылается слушатель в NLB. Поместите ваши вышестоящие приложения за другим балансировщиком сетевой нагрузки и целевой группой, а затем настройте NGINX для проксирования в NLB приложения.

Обсуждение

Этот распространенный шаблон называется *NLB-сэндвичем* (рис. 10.1). Он помещает NGINX с открытым исходным кодом в группу автоматического масштабирования позади NLB, а группу автоматического масштабирования приложения – позади другого NLB. Причина наличия NLB между каждым уровнем заключается в том, что балансировщик сетевой нагрузки очень хорошо работает с группами автоматического масштабирования; они автоматически регистрируют новые узлы и удаляют те, которые были прерваны, а также выполняют проверки работоспособности и передают трафик только исправным узлам. Возможно, потребуется создать второй внутренний NLB для уровня NGINX с открытым исходным кодом, потому что это позволяет сервисам в вашем приложении вызывать другие сервисы через группу автоматического масштабирования NGINX, не выходя из сети и не возвращаясь через общедоступный NLB. Это ставит NGINX в центр всего сетевого трафика вашего приложения, что делает его основой маршрутизации трафика вашего приложения. Этот паттерн раньше называли *сэндвичем с эластичным балансировщиком нагрузки* (ELB); однако балансировщик сетевой нагрузки предпочтительнее при работе с NGINX, потому что он является балансировщиком нагрузки уровня 4, тогда как ELB и балансировщик нагрузки приложения находятся на уровне 7. Балансировщики нагрузки уровня 7 преобразуют запрос через прокси-протокол и резервируются с использованием NGINX. Этот шаблон необходим только для NGINX с открытым исходным кодом, поскольку набор функций, предоставляемый NLB, доступен в NGINX Plus.

10.4. Развертывание из AWS Marketplace

Задача

Вам нужно с легкостью запускать NGINX Plus в AWS по лицензии с оплатой по факту потребления.

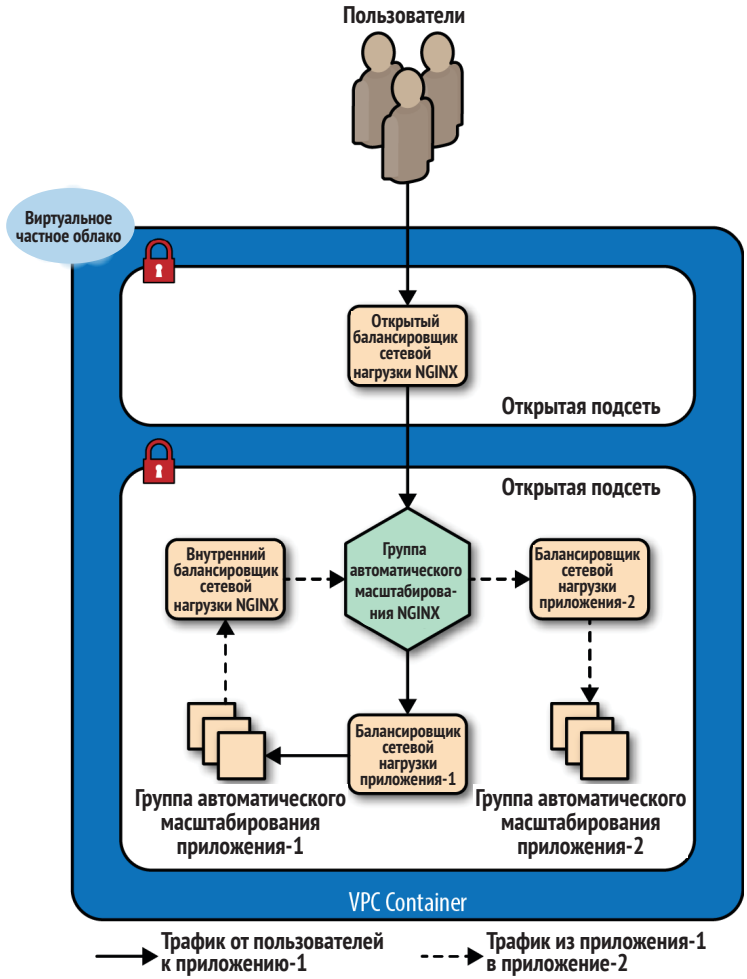

Рис. 10.1. Здесь изображен NGINX в виде шаблона NLB-сэндвич с внутренним балансировщиком сетевой нагрузки для использования внутренними приложениями. Пользователь отправляет запрос в приложение-1, которое отправляет запрос в приложение-2 через NGINX, чтобы выполнить запрос пользователя

Решение

Выполните развертывание через AWS Marketplace. Посетите сайт AWS Marketplace (https://aws.amazon.com/marketplace) и введите в поисковую строку «NGINX Plus» (рис. 10.2). Выберите образ машины Amazon на базе выбранного вами дистрибутива Linux; ознакомьтесь с деталями, условиями и ценами; затем нажмите на кнопку **Continue** (Продолжить). На

следующей странице вы сможете принять условия и развернуть NGINX Plus одним щелчком мыши или принять условия и использовать образ машины Amazon.

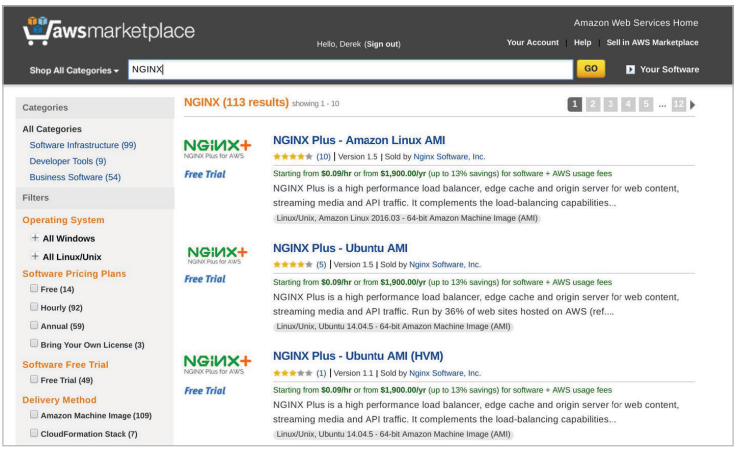

Рис. 10.2. Поиск NGINX на сайте AWS Marketplace

Обсуждение

Решение AWS Marketplace для развертывания NGINX Plus обеспечивает простоту использования и лицензию с оплатой по факту потребления. Вам не только не нужно ничего устанавливать, но у вас также есть лицензия без лишней нервотрепки, такой как получение заказа на покупку для годичной лицензии. Это решение позволяет вам опробовать NGINX Plus без каких-либо обязательств. Вы также можете использовать образ машины NGINX Plus Marketplace в качестве емкости переполнения. Обычной практикой является приобретение лицензий на ожидаемую рабочую нагрузку и использование образа машины Marketplace в группе автоматического масштабирования в качестве емкости переполнения. Эта стратегия гарантирует, что вы платите только за то, что используете.

10.5. Создание образа виртуальной машины NGINX в Azure

Задача

Вам нужно создать образ виртуальной машины (ВМ) собственного сервера NGINX, настроенного так, как вы считаете нужным, чтобы быст-

ро создать больше серверов или использовать их в наборах масштабирования.

Решение

Создайте виртуальную машину из базовой операционной системы по вашему выбору. После того как она загрузится, войдите в систему и установите NGINX или NGINX Plus по вашему выбору либо из исходного кода, либо через инструмент управления пакетами для дистрибутива, который вы используете. Настройте NGINX по своему усмотрению и создайте новый образ виртуальной машины. Чтобы создать образ виртуальной машины, для начала нужно сделать ее универсальной. Для этого необходимо удалить пользователя, подготовленного Azure, подключиться к нему по протоколу SSH и выполнить приведенную ниже команду:

```
$ sudo waagent -deprovision + user -force
```

Эта команда удаляет пользователя, которого подготовил Azure при создании виртуальной машины. Опция `-force` просто пропускает шаг подтверждения. После того как вы установили NGINX или NGINX Plus и удалили подготовленного пользователя, вы можете выйти из сеанса.

Подключите интерфейс командной строки Azure к своей учетной записи Azure с помощью команды входа в Azure, а затем убедитесь, что вы используете режим диспетчера ресурсов Azure. Теперь выключите виртуальную машину и высвободите ресурсы:

```
$ azure vm deallocate -g <ResourceGroupName> \
                        -n <VirtualMachineName>
```

После этого вы сможете пометить ее как универсальную с помощью команды `azure vm generalize`:

```
$ azure vm generalize -g <ResourceGroupName> \
                        -n <VirtualMachineName>
```

После того как ваша виртуальная машина станет универсальной, вы можете создать образ. Приведенная ниже команда создаст образ, а также сгенерирует шаблон диспетчера ресурсов Azure (ARM), который вы будете использовать для загрузки этого образа:

```
$ azure vm capture <ResourceGroupName> <VirtualMachineName> \
     <ImageNamePrefix> -t <TemplateName>.json
```

Командная строка выдаст сообщение о том, что ваш образ создан, что он сохраняет шаблон ARM в указанном вами месте и запрос завершен. Вы можете использовать этот шаблон для создания другой виртуальной машины из вновь созданного образа. Однако, чтобы использовать этот шаблон, созданный Azure, необходимо сначала создать новый сетевой интерфейс:

```
$ azure network nic create <ResourceGroupName> \
    <NetworkInterfaceName> \
    <Region> \
    --subnet-name <SubnetName> \
    --subnet-vnet-name <VirtualNetworkName>
```

Эта команда выводит подробную информацию о недавно созданном сетевом интерфейсе. Первая строка выходных данных будет идентификатором сетевого интерфейса, который вам понадобится для использования шаблона ARM, созданного Azure. Получив идентификатор, вы можете создать развертывание с помощью шаблона ARM:

```
$ azure group deployment create <ResourceGroupName> \
    <DeploymentName> \
    -f <TemplateName>.json
```

Вам будет предложено ввести несколько переменных ввода, таких как `vmName`, `adminUserName`, `adminPassword` и `networkInterfaceId`. Введите имя виртуальной машины, а также имя пользователя и пароль администратора. Используйте идентификатор сетевого интерфейса, полученный из последней команды, в качестве входных данных для приглашения `networkInterfaceId`. Эти переменные будут переданы в качестве параметров в шаблон ARM и использованы для создания новой виртуальной машины из созданного вами пользовательского образа NGINX или NGINX Plus.

После ввода необходимых параметров Azure приступит к созданию новой виртуальной машины из вашего собственного образа.

Обсуждение

Создание пользовательского образа в Azure позволяет делать копии предварительно настроенного сервера NGINX или NGINX Plus по желанию. Шаблон ARM позволяет быстро и надежно развертывать этот же сервер снова и снова по мере необходимости. Используя путь к образу виртуальной машины, который можно найти в шаблоне, можно созда-

вать различные наборы инфраструктуры, такие как наборы масштабирования виртуальных машин или другие виртуальные машины с различными конфигурациями.

См. также:
https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest
https://docs.microsoft.com/en-us/cli/azure/authenticate-azure-cli?view=azure-cli-latest
https://docs.microsoft.com/en-us/azure/virtual-machines/linux/capture-image?toc=%2Fazure%2Fvirtual-machines%2Flinux%2Ftoc.json

10.6. Балансировка нагрузки поверх наборов масштабирования NGINX в Azure

Задача

Вам необходимо масштабировать узлы NGINX за балансировщиком нагрузки Azure, чтобы добиться высокой доступности и динамического использования ресурсов.

Решение

Создайте балансировщик нагрузки Azure, который является общедоступным или внутренним. Разверните образ виртуальной машины NGINX, созданный в предыдущем разделе, или образ NGINX Plus из Marketplace, описанный в рецепте 10.7, в набор для масштабирования виртуальных машин Azure (VMSS). После развертывания балансировщика нагрузки и VMSS настройте внутренний пул балансировщика нагрузки в VMSS. Установите правила балансировки нагрузки для портов и протоколов, на которые вы хотите принимать трафик, и направьте их во внутренний пул.

Обсуждение

Обычно NGINX масштабируется для достижения высокой доступности или обработки пиковых нагрузок без чрезмерного выделения ресурсов. В Azure этого можно достичь с помощью VMSS. Использование балансировщика нагрузки Azure обеспечивает простоту управления для добавления и удаления узлов NGINX в пул ресурсов при масштабировании. С помощью балансировщиков нагрузки Azure можно проверить работоспособность своих внутренних пулов и передавать трафик только на исправные узлы. Вы можете запустить внутренние балансировщики

ровщики нагрузки Azure перед NGINX, где вы хотите разрешить доступ только через внутреннюю сеть. Можно использовать NGINX, чтобы передать запрос внутреннему балансировщику нагрузки, находящемуся в приложении VMSS, с помощью балансировщика нагрузки для простоты регистрации и ее отмены в пуле.

10.7. Развертывание через Azure Marketplace

Задача

Вам нужно с легкостью запустить NGINX Plus в Azure с лицензией с оплатой по факту использования.

Решение

Разверните образ виртуальной машины NGINX Plus через Azure Marketplace:

1. На информационной панели Azure выберите значок **New** (Новый) и используйте строку поиска, чтобы ввести туда слово «NGINX». Появятся результаты поиска.
2. Из списка выберите образ виртуальной машины NGINX Plus, опубликованный NGINX, Inc.
3. При появлении запроса на выбор модели развертывания выберите параметр **Resource Manager** (Диспетчер ресурсов) и нажмите кнопку **Create** (Создать).
4. Затем вам будет предложено заполнить форму, указав имя своей виртуальной машины, тип диска, имя пользователя и пароль по умолчанию или открытый ключ SSH, для какой подписки вы хотите выставить счет, группу ресурсов, которую вы хотите использовать, и местоположение.
5. Как только эта форма будет заполнена, можете нажать OK. Ваша форма будет подтверждена.
6. При появлении запроса выберите размер виртуальной машины и нажмите кнопку **Select** (Выбрать).
7. На следующей панели у вас есть возможность выбрать дополнительные конфигурации, которые будут использоваться по умолчанию в зависимости от группы ресурсов, выбранной ранее. После изменения этих параметров и их принятия нажмите кнопку **OK**.

8. На следующем экране просмотрите сводку. У вас есть возможность загрузить эту конфигурацию как шаблон ARM, чтобы иметь возможность быстрее создавать эти ресурсы с помощью шаблона JSON.
9. После того как вы проверите и загрузите свой шаблон, можно нажать OK, чтобы перейти к приобретению. На экране вы увидите уведомление о расходах, которые вы понесете в связи с использованием этой виртуальной машины. Нажмите кнопку **Purchase** (Купить), и ваш NGINX Plus начнет загружаться.

Обсуждение

Azure и NGINX упростили создание виртуальной машины NGINX Plus в Azure с помощью всего лишь нескольких форм конфигурации. Azure Marketplace – отличный способ получить NGINX Plus по требованию с лицензией с оплатой по факту использования. С помощью этой модели можно опробовать функции NGINX Plus или использовать его для емкости переполнения по требованию ваших уже лицензированных серверов NGINX Plus.

10.8. Развертывание в Google Compute Engine

Задача

Вам необходимо создать сервер NGINX в Google Compute Engine, чтобы осуществлять балансировку нагрузки или выступать в качестве посредника (proxy) для остальных своих ресурсов в Google Compute или App Engine.

Решение

Запустите новую виртуальную машину в Google Compute Engine. Выберите имя для своей виртуальной машины, зоны, тип машины и загрузочного диска. Настройте управление идентификацией и доступом, брандмауэр и любые дополнительные конфигурации, которые вам нужны. Создайте виртуальную машину.

После ее создания войдите в систему через SSH или через Google Cloud Shell. Установите NGINX или NGINX Plus с помощью менеджера пакетов для данного типа ОС. Настройте NGINX, как считаете нужным, и перезагрузите.

Кроме того, вы можете установить и настроить NGINX с помощью сценария запуска Google Compute Engine, который является дополнительным параметром конфигурации при создании виртуальной машины.

Обсуждение

Google Compute Engine предлагает настраиваемые виртуальные машины в любой момент. Запуск виртуальной машины требует небольших усилий и открывает целый мир возможностей. Google Compute Engine предлагает организацию сетей и вычисления в виртуальной облачной среде. Имея экземпляр Google Compute, у вас есть все возможности сервера NGINX, где бы и когда бы он вам ни понадобился.

10.9. Создание образа Google Compute

Задача

Вам нужно создать образ Google Compute, чтобы быстро сделать экземпляр виртуальной машины или шаблон экземпляра для группы экземпляров.

Решение

Создайте виртуальную машину, как описано в рецепте 10.8. После установки и настройки NGINX на экземпляре виртуальной машины установите для состояния автоматического удаления загрузочного диска значение `false`. Чтобы установить состояние автоматического удаления диска, отредактируйте виртуальную машину. На странице **Edit** (Правка) под конфигурацией диска установлен флажок **Delete boot disk when instance is deleted** (Удалить загрузочный диск при удалении экземпляра). Снимите этот флажок и сохраните конфигурацию виртуальной машины. Как только состояние автоматического удаления экземпляра будет установлено в `false`, удалите экземпляр. При появлении запроса не устанавливайте флажок, предлагающий удалить загрузочный диск. После выполнения этих задач у вас останется неприкрепленный загрузочный диск с установленным NGINX.

После того как ваш экземпляр удален и у вас есть неприкрепленный загрузочный диск, вы можете создать образ Google Compute. В разделе **Image** (Образ) консоли Google Compute Engine выберите **Create Image** (Создать образ). Вам будет предложено указать имя образа, семейство,

описание, тип шифрования и источник. Тип источника, который вам нужно использовать, – это диск; и для исходного диска выберите неприкрепленный загрузочный диск NGINX. Выберите **Create** (Создать), и Google Compute Cloud создаст образ из вашего диска.

Обсуждение

Можно использовать образы Google Cloud для создания виртуальных машин с загрузочным диском, идентичным только что созданному серверу. Ценность создания образов заключается в том, чтобы гарантировать, что каждый экземпляр этого образа идентичен. При установке пакетов во время загрузки в динамической среде, если только вы не используете блокировку версий с закрытыми репозиториями, вы рискуете не проверить версию пакета и обновления до запуска в производственной среде. Используя образы машины, вы можете проверить, что каждый пакет, работающий на этой машине, соответствует вашим требованиям, что повышает надежность предложения услуги.

См. также:
https://cloud.google.com/compute/docs/images/create-delete-deprecate-private-images

10.10. Создание прокси-сервера для Google App Engine

Задача

Вам нужно создать прокси-сервер для Google App Engine, чтобы переключать контексты между приложениями или обслуживать HTTPS в настраиваемом домене.

Решение

Используйте NGINX в Google Compute Cloud. Создайте виртуальную машину в Google Compute Engine или образ виртуальной машины с установленным NGINX и сделайте шаблон экземпляра с этим образом в качестве загрузочного диска. Если вы создали шаблон экземпляра, то нужна и группа экземпляров, которая использует этот шаблон.

Выполните настройку NGINX. Убедитесь, что вы используете протокол HTTPS, потому что Google App Engine является общедоступным, и вы должны убедиться, что не отключаете HTTPS на вашем экземпля-

ре NGINX и не позволяете информации перемещаться между NGINX и Google App Engine без защиты. Поскольку App Engine предоставляет только одну конечную точку DNS, вы будете использовать директиву `proxy_pass` вместо блоков `upstream` в версии NGINX с открытым исходным кодом.

Убедитесь, что конечная точка задана в качестве переменной в NGINX, а затем используйте эту переменную в директиве `proxy_pass`, чтобы NGINX выполнял DNS-разрешение при каждом запросе. Чтобы NGINX мог выполнять любое разрешение помощью DNS, вам также необходимо использовать директиву `resolver` и указать свой любимый DNS-резолвер. Google делает IP-адрес 8.8.8.8 доступным для общего пользования. Если вы используете NGINX Plus, можно использовать флаг `resolve` в директиве `server` в блоке `upstream`, активные подключения и другие преимущества восходящего модуля при создании прокси-сервера для Google App Engine.

Вы можете сохранить файлы конфигурации NGINX в Google Storage, а затем использовать сценарий запуска для своего экземпляра, чтобы извлечь конфигурацию во время загрузки. Это позволит вам изменить конфигурацию без необходимости записывать новый образ. Тем не менее это увеличит время запуска вашего сервера NGINX.

Обсуждение

Вы хотите запустить NGINX перед Google App Engine, если используете собственный домен, и сделать свое приложение доступным через HTTPS. В настоящее время Google App Engine не позволяет загружать собственные сертификаты SSL. Поэтому, если вы хотите обслуживать свое приложение в домене, отличном от appspot.com, с шифрованием, вам необходимо создать прокси-сервер с помощью NGINX для прослушивания в своем пользовательском домене. NGINX будет шифровать связь между собой и вашими клиентами, а также между собой и Google App Engine.

Еще одна причина, по которой вам может потребоваться запустить NGINX перед Google App Engine, – это размещение большого количества приложений App Engine в одном домене и использование NGINX для переключения контекста на основе URI. Микросервисы – это популярная архитектура, и прокси-сервер, такой как NGINX, обычно выполняет маршрутизацию трафика. Google App Engine упрощает развертывание приложений, и вместе с NGINX у вас есть полноценная платформа доставки приложений.

Глава 11

Контейнеры/Микросервисы

11.0. Введение

Контейнеры предлагают уровень абстракции на уровне приложения, сдвигая необходимый процесс установки пакетов и зависимостей из процесса развертывания в процесс сборки. Это важно по той причине, что инженеры теперь снаряжают элементы кода, которые исполняются и снабжаются единообразным способом вне зависимости от имеющейся среды. Предложение контейнеров как исполняемых элементов снижает значение риска путаницы зависимостей и настроек между средами. Принимая это во внимание, организации получают большой импульс для развертывания своих приложений в платформах контейнеров. При запуске приложения в платформе обычно размещают в контейнерах все, что позволяет имеющийся стек, включая вашего посредника и балансировщик нагрузки. NGINX и NGINX Plus запросто помещаются в контейнер и доставляются ими. Они также содержат множество функций, которые делают доставку приложений в контейнерах более подвижной. Данная глава сосредоточена на сборке образов контейнеров NGINX и NGINX Plus, свойствах, которые делают более простой работу в средах с контейнерами, а также развертывании ваших образов в Kubernetes и OpenShift.

11.1. Записи DNS SRV

Задача

Вы бы хотели использовать свою имеющуюся реализацию записей DNS SRV в качестве источника для вышестоящих серверов с помощью NGINX Plus.

Решение

Определите соответствующую директиву службы со значением `http` для вышестоящего сервера, чтобы указать NGINX на необходимость применения имеющихся записей SRV в качестве пула балансировки нагрузки:

```
http {
    resolver 10.0.0.2;

    upstream backend {
        zone backends 64k;
        server api.example.internal service=http resolve;
    }
}
```

Данная функциональность присутствует исключительно в NGINX Plus. Эта настройка инструктирует NGINX Plus выполнять разрешение DNS с сервера DNS `10.0.0.2` и настраивать пул вышестоящих серверов в отдельной директиве `server`. Эта директива определяется с параметром **`resolve`** и указывает на необходимость периодически повторно разрезолвить имя домена. Параметр `service=http` и значение сообщают NGINX, что это некая запись SRV, которая содержит список IP-адресов и портов, и обязуют выполнять балансировку нагрузки по ним, если они были настроены с помощью директивы `server`.

Обсуждение

Динамичная инфраструктура становится все более популярной благодаря спросу и внедрению инфраструктур на облачной основе. Среды с автоматическим масштабированием масштабируются горизонтально, увеличивая или уменьшая общее число серверов в имеющемся пуле для соответствия текущей нагрузке. Горизонтальное масштабирование требует балансировщика нагрузки, который способен добавлять или удалять ресурсы из имеющегося пула. Обладая некоей записью SRV, вы снимаете с себя ответственность за отслеживание текущего списка серверов и передаете ее DNS. Такой тип настройки чрезвычайно заманчив для сред с контейнерами, так как у вас могут быть контейнеры с изменяемыми номерами портов, причем, возможно, даже с одним и тем же IP-адресом. Важно также отметить, что полезная нагрузка записи UDP DNS ограничена примерно 512 байтами.

11.2. Использование официального образа NGINX

Задача

Вам требуется быстро подготовить к работе образ NGINX из Docker Hub.

Решение

Воспользуйтесь имеющимся в Docker Hub образом NGINX. Он содержит установленные по умолчанию настройки. Вам необходимо либо смонтировать локальный каталог настроек, либо создать файл Dockerfile и встроить свою настройку в сборку образа, чтобы изменить конфигурацию. В нашем случае мы монтируем том, в котором настройки по умолчанию NGINX обслуживают статическое содержимое, чтобы продемонстрировать его возможности при помощи отдельной команды:

```
$ docker run --name my-nginx -p 80:80 \
    -v /path/to/content:/usr/share/nginx/html:ro -d nginx
```

Команда `docker` извлекает образ `nginx:latest` из Docker Hub, если не обнаруживает его локально. Затем эта команда запускает данный образ NGINX в качестве контейнера Docker, отображая `localhost:80` в порт `80` самого контейнера NGINX. Она также монтирует имеющийся локальный каталог */path/to/content/* в качестве тома контейнера в */usr/share/nginx/html/* с доступом только для чтения. Установленная по умолчанию конфигурация NGINX будет обслуживать этот каталог в качестве статического содержимого. При отображении из вашей локальной машины в контейнер первым идет значение порта или каталога локальной машины, а порт или каталог самого контейнера идут вторыми.

Обсуждение

NGINX сделал официальный образ Docker, доступный через Docker Hub. Этот официальный образ Docker позволяет легко начать работу в Docker с вашей любимой платформой доставки приложений, NGINX. В данном разделе мы имели возможность запустить NGINX в контейнере при помощи одной единственной команды! Основная ветвь официального образа Docker NGINX, которой мы воспользовались в данном примере, собрана из образа Docker Debian Jessie. Однако вы также мо-

жете выбирать официальные образы, собранные из Alpine Linux. Файл Dockerfile и исходные коды для этих официальных образов доступны в GitHub. Вы можете расширить данный официальный образ, создав собственный файл Dockerfile и определив свой официальный образ в соответствующей команде FROM.

См. также:
https://hub.docker.com/_/nginx/
https://github.com/nginxinc/docker-nginx/

11.3. Создание Dockerfile NGINX

Задача

Для сборки образа Docker вам требуется создать файл Dockerfile NGINX.

Решение

Начните с FROM для предпочитаемого вами дистрибутива образа Docker. Для установки NGINX воспользуйтесь командой RUN. Чтобы добавить свои файлы настроек, примените команду ADD. Чтобы указать Docker открыть заданные порты, прибегните к помощи команды EXPOSE или сделайте это вручную при запуске своего образа в качестве контейнера. После того как ваш образ создаст экземпляр контейнера, для запуска NGINX воспользуйтесь CMD. Вам требуется запустить NGINX в приоритетном режиме. Для этого вам понадобится запустить NGINX при помощи `-g "daemon off;"` или добавить `daemon off;` в свои настройки. В данном примере мы будем использовать `daemon off;` в файле настроек внутри своего основного контекста. Вам также понадобится изменить свои настройки NGINX, чтобы выполнять регистрацию для журналов доступа в */dev/stdout* и */dev/stderr* для журналов ошибок; таким образом, ваши журналы попадут в руки демона Docker, что сделает их более доступными для вас на основе драйвера журналов, который вы выбрали при помощи Docker:

Файл Dockerfile:

```
FROM centos:7

# Устанавливаем репозиторий epel, чтобы получить nginx и установить nginx.
```

```
RUN yum -y install epel-release && \
yum -y install nginx

# Добавляем локальные файлы конфигурации в образ.
ADD /nginx-conf /etc/nginx

EXPOSE 80 443

CMD ["nginx"]
```

Получаемая структура каталогов выглядит так:

```
├── Dockerfile
└── nginx-conf
    ├── conf.d
    │   └── default.conf
    ├── fastcgi.conf
    ├── fastcgi_params
    ├── koi-utf
    ├── koi-win
    ├── mime.types
    ├── nginx.conf
    ├── scgi_params
    ├── uwsgi_params
    └── win-utf
```

Я решил разместить настройки NGINX целиком внутри данного каталога Docker для простоты доступа ко всем его настройкам при помощи всего одной строки в соответствующем файле Dockerfile, чтобы добавить все свои настройки NGINX.

Обсуждение

Вы можете найти полезным создание своего собственного файла Dockerfile, когда вам требуется полный контроль над установкой пакетов и их обновлением. Обычно держат собственный репозиторий образов, с тем чтобы знать, что ваш базовый образ надежен и проверен вашей командой, прежде чем запускать его в реальную эксплуатацию.

11.4. Сборка образа NGINX Plus

Задача

Вам требуется собрать образ Docker NGINX Plus для запуска NGINX Plus в среде с контейнерами.

Решение

Для создания образа Docker NGINX Plus воспользуйтесь этим файлом Dockerfile. Вам потребуется выгрузить свои сертификаты репозитория NGINX Plus и держать их в том же каталоге, что и данный Dockerfile с именами *nginx-repo.crt* и *nginx-repo.key* соответственно. С их помощью этот Dockerfile сделает всю остальную работу, установив NGINX Plus для вашего использования и связав журналы доступа и ошибок NGINX со сборщиком журналов Docker.

```
FROM debian:stretch-slim

LABEL maintainer="NGINX <docker-maint@nginx.com>"

# Скачиваем сертификат и ключ с портала клиента
# (https://cs.nginx.com) и копируем в контекст сборки.

COPY nginx-repo.crt /etc/ssl/nginx/
COPY nginx-repo.key /etc/ssl/nginx/
# Устанавливаем NGINX Plus.
RUN set -x \
  && APT_PKG="Acquire::https::plus-pkgs.nginx.com::" \
  && REPO_URL="https://plus-pkgs.nginx.com/debian" \
  && apt-get update && apt-get upgrade -y \
  && apt-get install \
    --no-install-recommends --no-install-suggests\
    -y apt-transport-https ca-certificates gnupg1 \
  && \
  NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62;\
  found=''; \
  for server in \
    ha.pool.sks-keyservers.net \
    hkp://keyserver.ubuntu.com:80 \
    hkp://p80.pool.sks-keyservers.net:80 \
```

```
    pgp.mit.edu \
  ; do \
   echo "Fetching GPG key $NGINX_GPGKEY from $server"; \
   apt-key adv --keyserver "$server" --keyserver-options \
      timeout=10 --recv-keys "$NGINX_GPGKEY" \
    && found=yes \
    && break;\
  done;\
  test -z "$found" && echo >&2 \
    "error: failed to fetch GPG key $NGINX_GPGKEY" && exit 1; \
  echo "${APT_PKG}Verify-Peer "true";"\
    >> /etc/apt/apt.conf.d/90nginx \
  && echo \
    "${APT_PKG}Verify-Host "true";">>\
    /etc/apt/apt.conf.d/90nginx \
  && echo "${APT_PKG}SslCert \
    "/etc/ssl/nginx/nginx-repo.crt";" >> \
    /etc/apt/apt.conf.d/90nginx \
  && echo "${APT_PKG}SslKey \
    "/etc/ssl/nginx/nginx-repo.key";" >> \
    /etc/apt/apt.conf.d/90nginx \
  && printf \
    "deb ${REPO_URL} stretch nginx-plus" \
    > /etc/apt/sources.list.d/nginx-plus.list \
  && apt-get update && apt-get install -y nginx-plus \
  && apt-get remove --purge --auto-remove -y gnupg1 \
  && rm -rf /var/lib/apt/lists/*

# Перенаправляем журналы запросов сборщику журналов Docker.
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
  && ln -sf /dev/stderr /var/log/nginx/error.log

EXPOSE 80
STOPSIGNAL SIGTERM
CMD ["nginx", "-g", "daemon off;"]
```

Чтобы собрать этот файл Dockerfile в образ Docker, выполните приведенную ниже команду в каталоге, который содержит данный Dockerfile, а также ваши сертификат и ключ репозитория NGINX Plus:

```
$ docker build --no-cache -t nginxplus .
```

Данная команда `docker build` использует флаг `--no-cache`, чтобы гарантировать, что вне зависимости от того, когда вы это создадите, из репозитория NGINX Plus будут вытянуты для обновления свежие пакеты NGINX Plus. Если для вас будет допустимо применение той же самой версии NGINX Plus, которую мы собирали ранее, вы можете опустить флаг `--no-cache`. В этом примере наш новый образ Docker снабжается тегом `nginxplus`.

Обсуждение

Создавая для NGINX Plus свой собственный образ Docker, вы получаете возможность настраивать свой контейнер NGINX Plus так, как считаете нужным, и переносить его в любую среду Docker. Это открывает всю мощность и расширенную функциональность NGINX Plus для вашей среды контейнеров. Данный файл Dockerfile не использует свойство `ADD` для добавления настроек; вам придется добавлять свои настройки вручную.

См. также:
https://www.nginx.com/blog/deploying-nginx-nginx-plus-docker/

11.5. Использование переменных среды в NGINX

Задача

Вам требуется воспользоваться переменными среды внутри своей конфигурации NGINX чтобы применять один и тот же образ контейнера для различных сред.

Решение

Для установки в NGINX переменных из своей среды воспользуйтесь модулем `ngx_http_perl_module`:

```
daemon off;
env APP_DNS;
include /usr/share/nginx/modules/*.conf;
...
http {
    perl_set $upstream_app 'sub { return $ENV{"APP_DNS"}; }';
    server {
```

```
        ...
        location / {
            proxy_pass https://$upstream_app;
        }
    }
}
```

Чтобы использовать `perl_set`, нужно, чтобы у вас был установлен `ngx_http_perl_module`; это можно сделать, загрузив данный модуль динамически или статически при сборке из исходного кода. По умолчанию NGINX стирает переменные среды из своего окружения: вам требуется объявлять все переменные, которые вы не желаете удалять, с помощью директивы `env`. Директива `perl_set` получает два параметра: имя переменной, которую вы бы хотели установить, и строку perl, которая вычисляет необходимый результат.

Приведенный ниже файл Dockerfile, который динамически загружает `ngx_http_perl_module`, устанавливает этот модуль из утилиты управления пакетами. При установке модулей из утилиты пакета для CentOS они помещаются в каталог */usr/lib64/nginx/modules/*, а файлы настройки, которые динамически загружают эти модули, помещаются в каталог */usr/share/nginx/modules/*. Именно по этой причине в предыдущем фрагменте кода мы поместили все файлы настройки в такой путь:

```
FROM centos:7

# Устанавливаем репозиторий epel, чтобы получить nginx и установить nginx.
RUN yum -y install epel-release && \
    yum -y install nginx nginx-mod-http-perl

# Добавляем локальные файлы конфигурации в образ.
ADD /nginx-conf /etc/nginx

EXPOSE 80 443

CMD ["nginx"]
```

Обсуждение

Обычной практикой при использовании Docker является применение переменных окружения для изменения того порядка, в котором работают контейнеры. Можно применять переменные среды в своей

конфигурации NGINX, чтобы ваш файл Dockerfile NGINX можно было использовать во многих разнообразных средах.

11.6. Контроллер Ingress в Kubernetes

Задача

Вы развертываете свое приложение в Kubernetes, и вам требуется контроллер ingress.

Решение

Убедитесь, что у вас имеется доступ к необходимому образу контроллера ingress. В случае с NGINX можно использовать образ *nginx/nginx-ingress* из Docker Hub. В случае с NGINX Plus вам потребуется собрать собственный образ и разместить его в собственном закрытом реестре Docker. Вы можете найти инструкции по сборке и доставке собственного контроллера ingress для NGINX Plus на странице https://github.com/nginxinc/kubernetes-ingress/blob/master/build/README.md.

Зайдите в папку Kubernetes Ingress Controller Deployments (https://github.com/nginxinc/kubernetes-ingress/tree/master/deployments) в репозитории kubernetes-ingress на GitHub. Все нижеприведенные команды будут запускаться из данного каталога локальной копии этого репозитория.

Создайте пространство имен и учетную запись службы для своего контроллера ingress; они оба имеют название `nginx-ingress`.

```
$ kubectl apply -f common/ns-and-sa.yaml
```

Создайте ключ безопасности с помощью сертификата TLS и ключа для контроллера ingress:

```
$ kubectl apply -f common/default-server-secret.yaml
```

Эти сертификат и ключ самостоятельно подписываются и создаются NGINX Inc. для проверок и примеров. Рекомендуется применять свой собственный, поскольку данный ключ открыт для общего доступа.

В качестве варианта можно создать словарь CofigMap для индивидуальной конфигурации NGINX (в данном случае словарь пуст; но вы можете ознакомиться с подробностями пользовательских настроек и аннотацией здесь: https://github.com/nginxinc/kubernetes-ingress/tree/master/examples/customization):

```
$ kubectl apply -f common/nginx-config.yaml
```

Если в вашем кластере включено управление доступом на основе ролей, создайте роль кластера и привяжите ее к соответствующей учетной записи этой службы. Для выполнения данного шага вы должны быть администратором кластера:

```
$ kubectl apply -f rbac/rbac.yaml
```

Теперь разверните свой контроллер ingress. В данном репозитории доступны два примера развертывания: Deployment и DaemonSet. Если вы планируете динамически изменять общее число реплик контроллера ingress, используйте Deployment. Для развертывания контроллера ingress во всех узлах или в подмножестве узлов воспользуйтесь DaemonSet.

Если вы намерены использовать манифесты Deployment NGINX Plus, вам следует исправить соответствующий файл YAML и определить собственный реестр и образ.

Для Deployment NGINX:

```
$ kubectl apply -f deployment/nginx-ingress.yaml
```

Для Deployment NGINX Plus:

```
$ kubectl apply -f deployment/nginx-plus-ingress.yaml
```

Для DaemonSet NGINX:

```
$ kubectl apply -f daemon-set/nginx-ingress.yaml
```

Для DaemonSet NGINX Plus:

```
$ kubectl apply -f daemon-set/nginx-plus-ingress.yaml
```

Убедитесь, что ваш контроллер ingress запущен:

```
$ kubectl get pods --namespace=nginx-ingress
```

Если вы создали DaemonSet, порты контроллера ingress `80` и `443` отображены в те же порты узла, в котором запущен этот контроллер. Для доступа к данному контроллеру ingress используйте эти порыт и IP-адрес любого из узлов, в которых работает контроллер. Если вы развернули Deployment, продолжите, используя приведенную ниже информацию.

Для методов Deployment имеются два варианта доступа к модулям контроллера ingress. Можно указать Kubernetes случайным образом выделять порт узла, который соответствует модулю контроллера ingress.

Такая служба имеет тип `NodePort`. Другой вариант – создание службы с типом `LoadBalancer`. При создании службы типа `LoadBalancer` Kubernetes создает балансировщик нагрузки для конкретной облачной платформы, такой как Amazon Web Services, Microsoft Azure и Google Cloud Compute.

Для создания службы типа **NodePort** используйте приведенную ниже команду:

```
$ kubectl create -f service/nodeport.yaml
```

Чтобы настроить открытый для данного модуля порт статически, измените соответствующий YAML и добавьте атрибу `nodePort: {port}` в настройки всех подлежащих открытию портов.

Для создания службы типа **LoadBalancer** для Google Cloud Compute или Azure воспользуйтесь таким кодом:

```
$ kubectl create -f service/loadbalancer.yaml
```

Для создания службы типа **LoadBalancer** для Amazon Web Services:

```
$ kubectl create -f service/loadbalancer-aws-elb.yaml
```

Для AWS Kubernetes создает классический ELB в режиме TCP с включенным протоколом PROXY. Вам следует настроить NGINX, чтобы использовать этот протокол TCP. Для этого в словарь ConfigMap, упомянутый ранее как файл *common/nginx-confg.yaml*, можно добавить следующее:

```
proxy-protocol: "True"
real-ip-header: "proxy_protocol"
set-real-ip-from: "0.0.0.0/0"
```

Затем обновите словарь:

```
$ kubectl apply -f common/nginx-config.yaml
```

Теперь вы можете выполнять адресацию своего модуля по его `NodePort` или делая запросы к балансировщику нагрузки, созданному от его имени.

Обсуждение

На момент написания данных строк Kubernetes является ведущей платформой для оркестрации и управления контейнерами, пограничн-

ным модулем, который маршрутизирует трафик в остальную часть вашего приложения. NGINX идеально подходит для данной роли и упрощает настройку со своими аннотациями. Проект NGINX-Ingress предлагает контроллер ingress NGINX с открытым исходным кодом сразу после его установки из образа Dockerfile, а также NGINX Plus с несколькими шагами, чтобы добавить ваш ключ и сертификат репозитория. Активация вашего кластера Kubernetes с помощью контроллера ingress NGINX предоставляет те же свойства, которые имеются у NGINX, но с дополнительными функциями организации сети в Kubernetes и DNS для маршрутизации трафика.

11.7. Маршрутизатор OpenShift

Задача

Вам нужно развернуть свое приложение в OpenShift, и вы бы хотели использовать в качестве маршрутизатора NGINX.

Решение

Соберите образ Маршрутизатора и выгрузите его в свой закрытый реестр. Вы можете найти необходимые исходные файлы для этого образа здесь: https://github.com/openshift/origin/tree/master/images/router/nginx. Важно поместить образ вашего Маршрутизатора в имеющийся закрытый реестр до того, как вы удалите установленный по умолчанию Маршрутизатор, поскольку это сделает реестр недоступным.

Выполните вход в кластер OpenShift в качестве администратора:

```
$ oc login -u system:admin
```

Выберите проект `default`:

```
$ oc project default
```

Выполните резервное копирование настроек установленного по умолчанию Маршрутизатора, в случае если вам потребуется его восстановить:

```
$ oc get -o yaml service/router dc/router \
    clusterrolebinding/router-router-role \
    serviceaccount/router > default-router-backup.yaml
```

Удалите Маршрутизатор:

```
$ oc delete -f default-router-backup.yaml
```

Разверните Маршрутизатор NGINX:

```
$ oc adm router router --images={image} --type='' \
    --selector='node-role.kubernetes.io/infra=true'
```

В данном примере значение `{image}` должно указывать на образ Маршрутизатора NGINX в вашем реестре. Параметр `selector` определяет селектор метки для узлов, где будет развернут ваш Маршрутизатор: `node-role.kubernetes.io/infra=true`. Используйте селектор, который актуален в вашей среде.

Проверьте, что ваши модули работают:

```
$ oc get pods
```

Вы должны увидеть модуль Маршрутизатора с именем `router-1-{string}`. По умолчанию страница состояния заглушки NGINX доступна через порт `1936` того узла, где запущен ваш Маршрутизатор (вы можете изменить этот порт с помощью переменной среды `STATS_PORT`). Для доступа к странице вне данного узла вам потребуется добавить запись в правила IPtables для такого узла:

```
$ sudo iptables -I OS_FIREWALL_ALLOW -p tcp -s {ip range} \
    -m tcp --dport 1936 -j ACCEPT
```

Перейдите в своем браузере по адресу *http://{node-ip}:1936/stub_status,* чтобы получить доступ к странице состояния заглушки.

Обсуждение

Маршрутизатор OpenShift является точкой входа для внешних запросов, ограничивающей работающие в OpenShift приложения. Работа этого Маршрутизатора состоит в получении входящих запросов и направлении их в соответствующий модуль приложения. Возможности балансировки нагрузки и маршрутизации NGINX делают его великолепным выбором для использования в качестве Маршрутизатора OpenShift. Переключение с устанавливаемого по умолчанию Маршрутизатора OpenShift на Маршрутизатор NGINX активирует все имеющиеся функции и мощь NGINX в качестве ingress вашего приложения OpenStack.

Глава 12

Режимы развертывания высокой доступности

12.0. Введение

Отказоустойчивая архитектура разделяет системы на идентичные, независимые стеки. Балансировщики нагрузки, подобные NGINX, применяются для распределения нагрузки, гарантируя использование того, что было подготовлено. Базовой концепцией понятия высокой доступности является балансировка нагрузки по множеству активных узлов или отработка отказа «активный–пассивный». Приложения с высокой доступностью не имеют единых точек отказа; все компоненты обязаны применять одну из указанных концепций, включая и сами балансировщики нагрузки. Для нас это означает NGINX. NGINX разработан для работы в одной из конфигураций: множество активных узлов или отработка отказа «активный–пассивный». В данной главе подробно описывается технология запуска нескольких серверов NGINX для обеспечения высокой доступности на уровне балансировки нагрузки.

12.1. Режим высокой доступности NGINX

Задача

Вам необходимо решение для балансировки нагрузки с высокой доступностью.

Решение

Воспользуйтесь режимом высокой доступности (HA, highly available) NGINX Plus с поддержкой активности, установив пакет `nginx-ha-keepalived` из репозитория NGINX Plus.

Обсуждение

Пакет `nginx-ha-keepalived` основан на поддержке активности и управляет виртуальным IP-адресом, открытым клиенту. На самом сервере NGINX запущен еще один процесс, который предоставляет гарантию работы этого NGINX Plus и самого процесса поддержки активности. Процесс поддержки активности – это процесс, который использует протокол VRRP (Virtual Router Redundancy Protocol), отправляя небольшие сообщения, часто именуемые сердцебиениями, в резервный сервер. Если этот сервер не получает такое сердцебиение на протяжении трех последовательных периодов, он инициирует отработку отказа, перемещая виртуальный IP-адрес к себе и превращаясь в хозяина (master). Имеющиеся возможности отработки отказа пакета `nginx-ha-keepalived` можно настраивать для указания индивидуальных ситуаций сбоя.

12.2. Балансировка нагрузки балансировщиками с помощью DNS

Задача

Вам необходимо распределить нагрузку между двумя или более серверами NGINX.

Решение

Воспользуйтесь DNS, чтобы выполнить распределение нагрузки по круговому циклу (round robin) по серверам NGINX, добавляя несколько IP-адресов в запись A DNS.

Обсуждение

При запуске нескольких балансировщиков нагрузки можно распределять нагрузку через DNS. Запись A позволяет перечислять под одним полностью определенным именем домена (FQDN, fully qualified domain name) несколько IP-адресов. DNS будет автоматически прокручивать все IP- адреса каруселью. DNS также предлагает взвешенный цикличеcкий алгоритм со взвешенными записями, которые действуют так же, как и описанный в главе 1 циклический алгоритм в NGINX. Эти методы прекрасно работают. Тем не менее удаление конкретной записи в том случае, когда сервер NGINX испытал сбой, может оказаться ловушкой. Существуют поставщики DNS – Amazon Route53 для одних и Dyn DNS для других, – предлагающие проверку жизнеспособности и отработку

отказов наряду с предложениями DNS, которые помогают решить данные проблемы. Когда вы применяете DNS для балансировки нагрузки по NGINX, в том случае, когда сервер NGINX помечается как подлежащий удалению, лучше всего следовать тем же протоколам, что и NGINX при удалении вышестоящего сервера. Для начала прекратите отправку к нему новых подключений, удалив его IP-адрес из имеющейся записи DNS, затем включите осушение подключений перед остановкой или отключением службы.

12.3. Балансировка нагрузки в EC2

Задача

Вы используете NGINX в AWS, причем ваш режим высокой доступности NGINX Plus не поддерживает IP-адреса Amazon.

Решение

Поместите NGINX позади балансировщика сетевой нагрузки AWS, настроив группу с автоматическим масштабированием из серверов NGINX и привязав ее к целевой группе, а затем подключите эту целевую группу к соответствующему балансировщику сетевой нагрузки. В качестве альтернативы можно поместить серверы NGINX в эту целевую группу вручную при помощи консоли AWS, интерфейса командной строки или API.

Обсуждение

Основанное на поддержке активности (keepaliving) решение с режимом высокой доступности от NGINX Plus не будет работать в AWS, поскольку он не поддерживает плавающие виртуальные IP-адреса, так как IP-адреса EC2 работают совершенно иначе. Это вовсе не означает, что NGINX лишен возможностей режима высокой доступности в AWS; на самом деле совершенно наоборот. Amazon предлагает балансировщик сетевой нагрузки AWS, который будет естественным образом выполнять балансировку нагрузки по множеству физически раздельных центров обработки данных, именуемых *зонами доступности* (AZ, availability zones), предоставит активные проверки жизнеспособности, а также конечную точку DNS CNAME. Распространенным решением для NGINX с режимом высокой доступности в AWS является размещение уровня NGINX позади имеющегося балансировщика сетевой нагрузки.

Серверы NGINX могут по мере необходимости автоматически добавляться в целевую группу и удаляться из нее. Балансировщик сетевой нагрузки не выступает заменой NGINX; существует множество вещей, предлагаемых NGINX, которых нет в балансировщике сетевой нагрузки, например различные методы балансировки, ограничение по частоте, кеширование, а также маршрутизация 7 уровня. Балансировщик нагрузки приложений AWS производит балансировку нагрузки 7 уровня на основе пути URI и заголовка хоста, но сам по себе не предлагает выполнения функций, которые имеет NGINX, таких как кеширование WAF, ограничение полосы пропускания, сервер активной доставки HTTP/2 и многое другое. В случае, когда предоставляемый балансировщик сетевой нагрузки не удовлетворяет вашим потребностям, имеется множество иных вариантов. Одним из вариантов является DNS-сервис, Route53. Данный продукт от AWS предлагает проверки жизнеспособности и отработку отказов.

12.4. Синхронизация конфигурации

Задача

Вы запускаете уровень NGINX Plus с режимом высокой доступности, и вам необходима синхронизация конфигураций серверов.

Решение

Воспользуйтесь эксклюзивной функцией синхронизации конфигураций NGINX Plus. Для настройки этой функциональности следуйте этим этапам:

Из репозитория пакетов NGINX Plus установите пакет `nginx-sync`.

Для RHEL или CentOS:

```
$ sudo yum install nginx-sync
```

Для Ubuntu или Debian:

```
$ sudo apt-get install nginx-sync
```

Предоставьте главной машине SSH-доступ в качестве суперпользователя ко всем одноранговым машинам.

Сгенерируйте пару ключей для аутентификации по протоколу SSH для суперпользователя и получите открытый ключ:

```
$ sudo ssh-keygen -t rsa -b 2048
$ sudo cat /root/.ssh/id_rsa.pub
ssh-rsa AAAAB3Nz4rFgt...vgaD root@node1
```

Получите IP-адрес главного узла:

```
$ ip addr
1: lo: mtu 65536 qdisc noqueue state UNKNOWN group default
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
      valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
2: eth0: mtu 1500 qdisc pfifo_fast state UP group default qlen \
      1000
   link/ether 52:54:00:34:6c:35 brd ff:ff:ff:ff:ff:ff
   inet 192.168.1.2/24 brd 192.168.1.255 scope global eth0
      valid_lft forever preferred_lft forever
   inet6 fe80::5054:ff:fe34:6c35/64 scope link
      valid_lft forever preferred_lft forever
```

Команда `ip addr` выведет дамп информации относительно интерфейсов машины. Игнорируйте интерфейс петли (loopback), который обычно идет первым. Найдите IP-адрес, который следует за `inet` для самого первого интерфейса. В данном примере таким IP-адресом является `192.168.1.2`.

Распространите открытый ключ для файла `authorized_keys` пользователя root во всех одноранговых узлах и задайте авторизацию только с IP-адреса главной машины:

```
$ sudo echo 'from="192.168.1.2" ssh-rsa AAAAB3Nz4rFgt...vgaD \
    root@node1' >> /root/.ssh/authorized_keys
```

Добавьте в */etc/ssh/sshd_confg* приведенную ниже строку и выполните перезагрузку `sshd` на всех узлах:

```
$ sudo echo 'PermitRootLogin without-password' >> \
    /etc/ssh/sshd_config
$ sudo service sshd reload
```

Убедитесь, что пользователь root на главном узле способен подключаться по протоколу SSH ко всем одноранговым узлам без пароля:

```
$ sudo ssh root@node2.example.com
```

На главной машине создайте файл настроек */etc/nginx-sync.conf* со следующей конфигурацией:

```
NODES="node2.example.com node3.example.com node4.example.com"
CONFPATHS="/etc/nginx/nginx.conf /etc/nginx/conf.d"
EXCLUDE="default.conf"
```

Данный пример конфигурации демонстрирует три распространенных параметра для данной функциональности: `NODES`, `CONFIGPATHS` и `EXCLUDE`. Параметр `NODES` устанавливается как строка имен хостов или IP-адресов, разделяемых пробелами; это узлы одноранговой сети (peer), куда ваша главная машина будет помещать изменения в настройках. Параметр `CONFIGPATHS` определяет, какие файлы или каталоги следует синхронизировать. Наконец, вы можете использовать параметр `EXCLUDE` для исключения файлов конфигурации из синхронизации. В нашем примере главная машина выполняет активную доставку изменений настроек в основном файле настроек NGINX и также включает каталоги */etc/nginx/nginx.conf* и */etc/nginx/conf.d* в одноранговые узлы с именами `node2.example.com`, `node3.example.com` и `node4.example.com`. Если процесс синхронизации обнаруживает файл с именем *default.conf*, для него не будет выполняться активная доставка в этой одноранговой сети, так как он настроен в качестве `EXCLUDE`.

Для настройки местоположения бинарных файлов NGINX, RSYNC, SSH, diff, а также расположения lockfile и резервного каталога имеются дополнительные параметры. Существует также параметр, который использует `sed` для выработки шаблона заданных файлов. Для получения дополнительных сведений относительно расширенных параметров посетите https://docs.nginx.com/nginx/admin-guide/high-availability/configuration-sharing/.

Проверьте свою конфигурацию:

```
$ nginx-sync.sh -h # display usage info
$ nginx-sync.sh -c node2.example.com # compare config to node2
$ nginx-sync.sh -C # compare master config to all peers
$ nginx-sync.sh # sync the config & reload NGINX on peers
```

Обсуждение

Эта имеющаяся только в NGINX Plus функциональность позволяет управлять несколькими серверами NGINX Plus в конфигурации с вы-

сокой доступностью через обновление только главного узла и выполнения синхронизации со всеми прочими одноранговыми узлами. Выполняя синхронизацию в автоматическом режиме, вы ограничиваете риск ошибок при передаче настроек. Приложение `nginx-sync.sh` предоставляет меры предосторожности для предотвращения отправки по одноранговой сети плохих конфигураций. Они включают в себя проверку выполненных настроек в главной машине, создание резервных копий имеющейся для одноранговой сети конфигурации и удостоверение полученной в одноранговой сети конфигурации перед перезагрузкой. Хотя более предпочтительной является синхронизация вашей конфигурации при помощи инструментов управления конфигурацией или Docker, синхронизация конфигурации NGINX Plus имеет ценность, когда вам еще только предстоит выполнить большой скачок к управлению средами таким образом.

12.5. Совместное использование состояния с помощью Zone Sync

Задача

Вам необходимо, чтобы NGINX Plus выполнил синхронизацию своих зон общей памяти для парка серверов высокой доступности.

Решение

Воспользуйтесь параметром `sync` при настройке зоны общей памяти NGINX Plus:

```
stream {
    resolver 10.0.0.2 valid=20s;

    server {
        listen 9000;
        zone_sync;
        zone_sync_server nginx-cluster.example.com:9000 resolve;
        # ... Security measures
    }
}
    http {
        upstream my_backend {
            zone my_backend 64k;
```

```
        server backends.example.com resolve;
        sticky learn zone=sessions:1m
                     create=$upstream_cookie_session
                     lookup=$cookie_session
                     sync;
    }

    server {
        listen 80;
        location / {
            proxy_pass http://my_backend;
        }
    }
}
```

Обсуждение

Модуль `zone_sync` является исключительным свойством NGINX Plus, которое в реальности превращает NGINX Plus в кластер. Как показано в приведенной конфигурации, нужно установить сервер `stream`, настроенный в качестве `zone_sync`. В данном примере это сервер, слушающий порт `9000`. NGINX Plus взаимодействует со всеми оставшимися серверами, которые определены в директиве `zone_sync_server`. Для динамических кластеров можно настроить эту директиву на доменное имя, которое преобразуется в несколько IP-адресов, либо статически определить последовательность директив `zone_sync_server`. Вам следует ограничить доступ к зоне синхронизации сервера; имеются особые SSL/TLS директивы для данного модуля в целях выполнения аутентификации машин. Основным преимуществом настройки NGINX Plus в кластер состоит в том, что вы можете синхронизировать зоны общей памяти для ограничения частот, сеансов `sticky learn`, а также хранилища типа ключ/значение. Данный пример демонстрирует применение параметра `sync`, прикрепляемого в конце директивы `sticky learn`. В этом примере пользователь ограничен вышестоящим сервером на базе куки с именем `session`. При отсутствии модуля синхронизации зоны, когда пользователь выполняет запрос к другому серверу NGINX Plus, он может потерять свой сеанс. В случае наличия модуля синхронизации зоны все серверы NGINX Plus осведомлены о таком сеансе и том, с каким вышестоящим сервером он связан.

Глава 13

Расширенный мониторинг активности

13.0. Введение

Чтобы гарантировать оптимальную производительность и точность своего приложения, необходимо ознакомиться с показателями мониторинга его активности. NGINX Plus предлагает расширенную панель мониторинга и формат JSON Feed, чтобы обеспечить всесторонний мониторинг всех запросов, поступающих через сердце вашего приложения. Мониторинг активности NGINX Plus предоставляет информацию о запросах, пулах вышестоящих серверов, кешировании, работоспособности и многое другое. В этой главе подробно описываются мощь и возможности инструментальной панели NGINX Plus, API NGINX Plus и модуль `stub_status` с открытым исходным кодом.

13.1. Активация модуля Stub Status с открытым исходным кодом

Задача

Вам нужно активировать базовый мониторинг для NGINX.

Решение

В блоке location внутри HTTP-сервера NGINX активируйте модуль `stub_status`:

```
location /stub_status {
    stub_status;
```

```
    allow 127.0.0.1;
    deny all;
    # Set IP restrictions as appropriate
}
```

Проверьте свои настройки, выполнив запрос состояния:

```
$ curl localhost/stub_status
Active connections: 1
server accepts handled requests
 1 1 1
Reading: 0 Writing: 1 Waiting: 0
```

Обсуждение

Модуль `stub_status` активирует базовые возможности мониторинга сервера NGINX с открытым исходным кодом. Вся возвращаемая им информация предоставляет сведения относительно значения числа активных соединений, а также общее число принятых и обработанных подключений и обслуженных запросов. Также отображается текущее значение числа подлежащих чтению, записи или пребывающих в состоянии ожидания соединений. Предоставляемая информация является глобальной и неспецифичной для того родительского сервера, в котором определена директива `stub_status`. Это означает, что вы можете размещать это состояние в защищенном сервере. Этот модуль предоставляет счетчики активных соединений в виде встроенных переменных, чтобы использовать их при регистрации и где-либо еще. Это переменные `$connections_active`, `$connections_reading`, `$connections_writing` и `$connections_waiting`.

13.2. Активация инструментальной панели мониторинга NGINX Plus

Задача

Вам необходимы подробные метрики относительно протекающего через ваш сервер NGINX Plus трафика.

Решение

Воспользуйтесь инструментальной панелью мониторинга активности в реальном масштабе времени:

```
server {
# ...
    location /api {
        api [write=on];
        # Директивы, ограничивающие доступ к API
# См. главу 7
    }

    location = /dashboard.html {
        root /usr/share/nginx/html;
    }
}
```

Данная конфигурация NGINX Plus обслуживает инструментальную панель мониторинга состояний NGINX Plus. Эта конфигурация настраивает сервер HTTP для обслуживания соответствующего API и инструментальной панели состояния. Эта инструментальная панель обслуживается в виде выводимого статического содержания каталога */usr/share/nginx/html*. Она выполняет запросы к API в */api/* чтобы выполнять выборку значения состояния и его отображения в реальном масштабе времени.

Обсуждение

NGINX Plus предоставляет продвинутую инструментальную панель мониторинга состояния. Она предоставляет подробное состояние системы NGINX, например число активных подключений, время работы, информацию о пуле вышестоящих серверов и многое другое. Для беглого знакомства с ее консолью посмотрите на рис. 13.1.

Загрузочная страница инструментальной панели состояния дает обзор всей имеющейся системы. Кликнув по вкладке **Server Zones,** вы увидите подробную информацию обо всех настроенных в конфигурации серверах HTTP с детализацией значений числа откликов от 1XX до 5XX и общее итоговое значение, а также число запросов в секунду и текущую пропускную способность трафика. Вкладка **Upstream** выводит подробности состояний вышестоящих серверов, а если какие-либо из них пребывают в состоянии отказа, показывает, сколько запросов было обслужено и общее число обслуженных запросов для состояния кода, а также иные статистические данные, например сколько проверок жизнеспособности были пройдены успешно или завершились провалом. Вкладка **TCP/UDP Zones** отображает подробности объема трафика, про-

ходящего через потоки TCP или UDP, а также количество соединений. На вкладке **TCP/UDP Upstream** показана информация о том, сколько обслуживает каждый из вышестоящих серверов в восходящих пулах TCP/UDP, а также сведения об успешной и неудачной проверке работоспособности и времени отклика. Вкладка **Caches** отображает информацию относительно пространства, используемого для кеширования; объем обслуживаемого трафика, записей и обходов; а также соотношение попаданий в кеш. Данная инструментальная панель NGINX является бесценной при мониторинге всей сердцевины ваших приложений и потока трафика.

См. также:
https://demo.nginx.com/dashboard.html

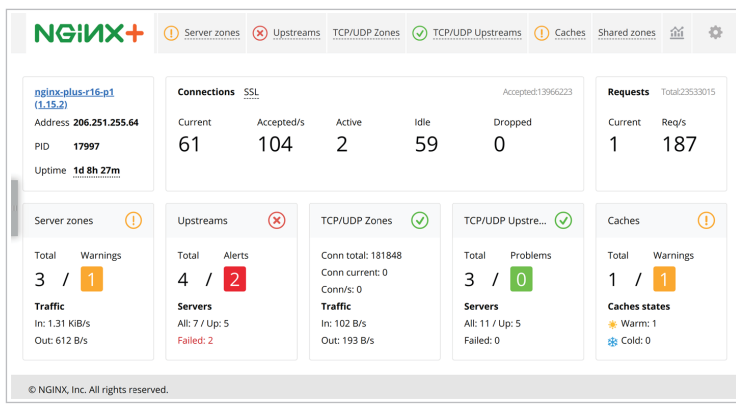

Рис. 13.1. Инструментальная панель состояния NGINX Plus

13.3. Сбор метрик с помощью API NGINX Plus

Задача

Вам необходим API-доступ для детализации измерений, производимых инструментальной панелью состояния NGINX Plus.

Решение

Для сбора метрик воспользуйтесь API RESTful. Вот примеры конвейера, получаемого через `json_pp`, для более простого прочтения вывода:

```
$ curl "demo.nginx.com/api/3/" | json_pp
[
```

```
    "nginx",
    "processes",
    "connections",
    "ssl",
    "slabs",
    "http",
    "stream"
]
```

Вызов с `curl` запрашивает самый верхний уровень API, который отображает другие фрагменты API.

Для получения информации о сервере NGINX Plus воспользуйтесь URI `/api/{version}/nginx`:

```
$ curl "demo.nginx.com/api/3/nginx" | json_pp
{
    "version" : "1.15.2",
    "ppid" : 79909,
    "build" : "nginx-plus-r16",
    "pid" : 77242,
    "address" : "206.251.255.64",
    "timestamp" : "2018-09-29T23:12:20.525Z",
    "load_timestamp" : "2018-09-29T10:00:00.404Z",
    "generation" : 2
}
```

Для ограничения информации, возвращаемой API, используйте аргументы:

```
$ curl "demo.nginx.com/api/3/nginx?fields=version,build" \
  | json_pp
{
    "build" : "nginx-plus-r16",
    "version" : "1.15.2"
}
```

Вы можете запросить статистические данные по соединениям из URI `/api/{version}/connections`:

```
$ curl "demo.nginx.com/api/3/connections" | json_pp
{
```

```
    "active" : 3,
    "idle" : 34,
    "dropped" : 0,
    "accepted" : 33614951
}
```

У вас имеется возможность собирать статистику запросов из URI запросов `/api/{version}/http/`:

```
$ curl "demo.nginx.com/api/3/http/requests" | json_pp
{
    "total" : 52107833,
    "current" : 2
}
```

Для получения выборки статистических данных относительно определенной зоны серверов воспользуйтесь URI `/api/{version}/http/server_zones/{httpServerZoneName}`:

```
$ curl "demo.nginx.com/api/3/http/server_zones/hg.nginx.org" \
  | json_pp
{
    "responses" : {
        "1xx" : 0,
        "5xx" : 0,
        "3xx" : 938,
        "4xx" : 341,
        "total" : 25245,
        "2xx" : 23966
    },
    "requests" : 25252,
    "discarded" : 7,
    "received" : 5758103,
    "processing" : 0,
    "sent" : 359428196
}
```

Данный API способен предоставлять абсолютно все данные, которые вы видите на инструментальной панели. У него есть глубина, и он следует некому логическому шаблону. Ссылки на ресурсы можно найти в конце данного рецепта.

Обсуждение

API NGINX Plus способен возвращать статистические данные о множестве частей вашего сервера NGINX Plus. Вы можете собирать информацию относительно самого сервера NGINX Plus, его процессов, соединений и слоев. Вы также может найти информацию относительно запускаемых внутри NGINX серверов `http` и `stream`, включая серверы, восходящие потоки, вышестоящие серверы и хранилища типа ключ/значение, а также информацию и статистические данные относительно зон кеширования HTTP. Это предоставляет вам или сторонним агрегаторам метрик детальное представление о том, как работает ваш сервер NGINX Plus.

См. также:
https://nginx.org/en/docs/http/ngx_http_api_module.html
https://demo.nginx.com/swagger-ui/

Глава 14

Отладка и устранение неполадок с помощью журналов доступа, журналов ошибок и отслеживания запросов

14.0. Введение

Ведение журналов является основой понимания вашего приложения. Благодаря NGINX вы получаете прекрасный контроль над регистрацией информации, значимой для вас и вашего приложения. NGINX позволяет разделять журналы доступа на разные файлы и форматы для разных контекстов и изменять уровень регистрации ошибок, чтобы глубже понять происходящее. Возможность потоковой передачи журналов на централизованный сервер присуща NGINX благодаря возможностям ведения журналов Syslog. В этой главе мы обсудим журналы доступа и ошибок, потоковую передачу по протоколу Syslog и отслеживание запросов от начала до конца с помощью идентификаторов запросов, сгенерированных NGINX.

14.1. Настройка журналов доступа

Задача

Вам необходимо настроить форматы журналов доступа, чтобы добавить встроенные переменные в журналы запросов.

Решение

Настройте формат журнала доступа:

```
http {
    log_format geoproxy
               '[$time_local] $remote_addr '
               '$realip_remote_addr $remote_user '
               '$request_method $server_protocol '
               '$scheme $server_name $uri $status '
               '$request_time $body_bytes_sent '
               '$geoip_city_country_code3 $geoip_region '
               '"$geoip_city" $http_x_forwarded_for '
               '$upstream_status $upstream_response_time '
               '"$http_referer" "$http_user_agent"';
    ...
}
```

Данная конфигурация формата журнала носит название `geoproxy` и использует ряд встроенных переменных для демонстрации возможностей ведения журналов с NGINX. Эта конфигурация показывает местное время на сервере, когда был сделан запрос, IP-адрес, который открыл соединение, и IP-адрес клиента, как его NGINX понимает согласно инструкциям `geoip_proxy` или `realip_header`. `$remote_user` показывает имя пользователя, прошедшего проверку с помощью базовой аутентификации, за которым следуют метод запроса и протокол, а также схема, такая как HTTP или HTTPS. Регистрируется совпадение `server name`, а также URI запроса и код возвращаемого состояния. Регистрируемая статистика включает в себя время обработки в миллисекундах и размер тела, отправляемого клиенту. Регистрируется информация о стране, регионе и городе. Заголовок HTTP `X-Forwarded-For` включен, чтобы показать, пересылается ли запрос другим прокси. Модуль `upstream` активирует используемые нами встроенные переменные, которые показывают состояние, возвращаемое с вышестоящего сервера, и время, необходимое для возврата запроса. Наконец, мы зарегистрировали информацию о том, откуда клиент обратился и какой браузер использует. Директива `log_format` действительна только в контексте HTTP.

Эта конфигурация журнала отображает запись, которая выглядит следующим образом:

```
[25/Nov/2016:16:20:42 +0000] 10.0.1.16 192.168.0.122 Derek
GET HTTP/1.1 http www.example.com / 200 0.001 370 USA MI
```

```
"Ann Arbor" - 200 0.001 "-" "curl/7.47.0"
```

Чтобы взять этот формат журнала, используйте директиву `access_log`, указав в качестве параметров путь к файлу журнала и имя формата `geoproxy`:

```
server {
    access_log /var/log/nginx/access.log geoproxy;
    ...
}
```

Директива `access_log` принимает в качестве параметров путь к файлу журнала и имя формата. Эта директива действительна во многих контекстах и в каждом контексте может иметь свой путь к журналу и (или) формату журнала.

Обсуждение

Модуль журнала в NGINX позволяет настраивать форматы журналов для множества различных сценариев, чтобы выполнять записи в разные файлы журналов по своему усмотрению.

Может оказаться полезным настроить отдельный формат журнала для каждого контекста, где вы используете разные модули и применяете встроенные переменные этих модулей, или единый общий формат, предоставляющий всю информацию, которая вам может понадобиться. Также возможно отформатировать журнал в формате JSON или XML. Эти журналы помогут вам понять ваши схемы трафика, использование клиентами, а также кто ваши клиенты и откуда они приходят. Журналы доступа также могут помочь вам обнаружить задержки в ответах и проблемы с вышестоящими серверами или определенными URI. Журналы доступа могут использоваться для парсинга и воспроизведения шаблонов трафика в тестовых средах для имитации реального взаимодействия с пользователем. При поиске и устранении неисправностей, отладке или анализе вашего приложения или рынка возможности неограничены.

14.3. Отправка журналов в Syslog

Задача

Вам необходимо пересылать свои журналы слушателю Syslog, чтобы объединить их в централизованную службу.

Решение

Используйте директивы `access_log` и `error_log` для отправки своих журналов слушателю Syslog:

```
error_log syslog:server=10.0.1.42 debug;

access_log syslog:server=10.0.1.42,tag=nginx,severity=info
    geoproxy;
```

За параметром `syslog` для директив `error_log` и `access_log` следует двоеточие и ряд параметров. Эти параметры включают в себя обязательный флаг `server`, который обозначает IP-адрес, DNS-имя или сокет Unix для подключения, а также необязательные флаги, такие как `facility`, `severity`, `tag` и `nohostname`. Опция `server` принимает номер порта, а также IP-адреса или DNS-имена. Однако по умолчанию используется UDP 514. Опция `facility` относится к средству сообщения журнала, определенному как одно из 23, определенных в стандарте RFC для Syslog; значение по умолчанию – `local7`. Опция `tag` помечает сообщение значением. Это значение по умолчанию равно `nginx`.

`severity` по умолчанию имеет значение `info` и обозначает серьезность отправляемого сообщения. Флаг `nohostname` отключает добавление поля имени хоста в заголовок сообщения Syslog и не принимает значения.

Обсуждение

Syslog – это стандартный протокол для отправки сообщений журнала и сбора этих журналов на одном сервере или коллекции серверов. Отправка журналов в централизованное расположение помогает при отладке, когда на нескольких хостах запущено несколько экземпляров одной и той же службы. Это называется агрегированием журналов. Агрегирование журналов позволяет просматривать журналы разом в одном месте без необходимости переходить с сервера на сервер и мысленно объединять файлы журналов по отметке времени. Среди распространенных средств агрегации журналов можно упомянуть ElasticSearch, Logstash и Kibana, также известный как ELK Stack. NGINX упрощает потоковую передачу этих журналов слушателю Syslog с помощью директив `access_log` и `error_log`.

14.4. Трассировка запросов

Задача

Вам необходимо соотнести журналы NGINX с журналами приложений, чтобы иметь полное представление о запросе.

Решение

Используйте идентифицирующую переменную `request` и передайте ее в свое приложение для регистрации:

```
log_format trace '$remote_addr - $remote_user [$time_local] '
                 '"$request" $status $body_bytes_sent '
                 '"$http_referer" "$http_user_agent" '
                 '"$http_x_forwarded_for" $request_id';
upstream backend {
    server 10.0.0.42;
}
server {
    listen 80;
    add_header X-Request-ID $request_id; # Return to client
    location / {
        proxy_pass http://backend;
        proxy_set_header X-Request-ID $request_id; #Pass to app
        access_log /var/log/nginx/access_trace.log trace;
    }
}
```

В этом примере конфигурации мы установили `log_format` с именем `trace`, а в журнале используется переменная `$request_id`, которая также передается в вышестоящее приложение с помощью директивы `proxy_set_header` для добавления идентификатора запроса в заголовок при выполнении запроса в восходящем направлении. Идентификатор запроса также передается обратно клиенту с помощью директивы `add_header`, устанавливающей идентификатор запроса в заголовке ответа.

Обсуждение

Ставшая доступной в NGINX Plus R10 и NGINX версии 1.11.0, переменная `$request_id` предоставляет случайно сгенерированную строку из 32 шестнадцатеричных символов, которые можно использовать для уникальной идентификации запросов. Передав этот идентификатор

как клиенту, так и приложению, вы можете соотнести свои журналы с запросами, которые вы делаете. От клиента веб-интерфейса вы получите эту уникальную строку в качестве заголовка ответа и сможете использовать ее для поиска в ваших журналах записей, которые ей соответствуют. Вам нужно будет указать приложению, чтобы оно записывало и регистрировало этот заголовок в журналах приложения для создания подлинной сквозной связи между журналами. Благодаря этому усовершенствованию NGINX позволяет отслеживать запросы через стек приложений.

Глава 15

Настройка производительности

15.0. Введение

Настройка NGINX сделает из вас художника. Настройка производительности любого типа сервера или приложения всегда зависит от ряда переменных элементов, таких как среда, сценарий использования, требования и задействованные физические компоненты, и это не все. Обычно выполняется настройка, ориентируясь на проблемные места, т. е. выполняется тестирование до тех пор, пока вы не достигнете проблемного места. Вы определяете это место, настраиваете ограничения и повторяете тестирование, до тех пор пока не достигнете желаемых требований к производительности. В этой главе мы предлагаем проводить измерения, когда настройка производительности осуществляется путем тестирования с помощью автоматизированных инструментов и измерения результатов. Здесь также рассказывается о настройке подключений, чтобы они оставались открытыми как для клиентов, так и для вышестоящих серверов, а также для обслуживания большего количества подключений с помощью настройки операционной системы.

15.1. Автоматизация тестов с помощью драйверов нагрузки

Задача

Вам необходимо автоматизировать свои тесты с помощью драйвера нагрузки, чтобы обеспечить согласованность и повторяемость результатов тестирования.

Решение

Используйте инструмент нагрузочного тестирования HTTP, такой как Apache JMeter, Locust, Gatling, или любой другой инструмент, на котором стандартизировалась ваша команда. Создайте конфигурацию для своего инструмента нагрузочного тестирования, который запускает комплексное тестирование в вашем веб-приложении. Запустите тест для своей службы. Просмотрите показатели, собранные во время прогона, чтобы установить базовый уровень. Медленно увеличивайте эмулированный пользовательский параллелизм, чтобы имитировать типичную реальную эксплуатацию, и определите точки улучшения. Настройте NGINX и повторяйте этот процесс, пока не добьетесь желаемых результатов.

Обсуждение

Использование инструмента автоматического тестирования для определения теста дает вам непротиворечивый тест для построения метрик при настройке NGINX. Вы должны быть в состоянии повторить свой тест и измерить прирост производительности или потери для проведения экспериментов. Выполнение теста до внесения каких-либо изменений в конфигурацию NGINX для установления базовой линии дает вам основу для работы, чтобы вы могли оценить, повысило ли изменение в конфигурации производительность или нет. Измерения всех внесенных изменений помогут вам определить, что улучшает производительность.

15.2. Сохраняем подключения открытыми для клиентов

Задача

Вам необходимо увеличить количество запросов, разрешенных к выполнению по одному подключению от клиентов, и количество времени, в течение которого незанятые подключения могут сохраняться.

Решение

Используйте директивы `keepalive_requests` и `keepalive_timeout`, чтобы изменить количество запросов, которые можно сделать через одно подключение и время, в течение которого незанятые подключения могут оставаться открытыми:

```
http {
    keepalive_requests 320;
    keepalive_timeout 300s;
    ...
}
```

По умолчанию директива `keepalive_requests` имеет значение 100, а директива `keepalive_timeout` – 75 секунд.

Обсуждение

Как правило, количество запросов по умолчанию для одного подключения будет соответствовать потребностям клиента, поскольку в наши дни браузерам разрешено открывать несколько подключений к одному серверу на одно полностью определенное доменное имя. Количество параллельных открытых подключений к домену обычно ограничено числом менее 10, поэтому в этом случае будет происходить много запросов по одному подключению. Уловка, обычно используемая сетями доставки содержимого, заключается в создании нескольких доменных имен, указывающих на сервер контента, и чередовании того, какое доменное имя используется в коде, чтобы позволить браузеру открывать больше подключений. Эти оптимизации подключений могут оказаться полезными, если ваше внешнее приложение постоянно опрашивает ваше внутреннее приложение на предмет наличия обновлений, поскольку открытое соединение, которое допускает большее количество запросов и дольше остается открытым, ограничит число подключений, которые необходимо установить.

15.3. Сохраняем подключения открытыми для вышестоящих серверов

Задача

Вам необходимо держать подключения открытыми для вышестоящих серверов для повторного использования, чтобы повысить производительность.

Решение

Используйте директиву `keepalive` в контексте `upstream`, чтобы подключения были открыты для вышестоящих серверов для повторного использования:

```
proxy_http_version 1.1;
proxy_set_header Connection "";

upstream backend {
    server 10.0.0.42;
    server 10.0.2.56;

    keepalive 32;
}
```

Директива `keepalive` в контексте `upstream` активирует кеш подключений, которые остаются открытыми для всех работников NGINX. Она обозначает максимальное количество незанятых подключений, которые должны оставаться открытыми на одного работника. Директивы прокси-модулей, используемые над блоком `upstream`, необходимы для правильной работы директивы `keepalive` для подключений с вышестоящими серверами. Директива `proxy_http_version` указывает прокси-модулю использовать HTTP версии 1.1, что позволяет делать несколько запросов по одному подключению, пока оно открыто. Директива `proxy_set_header` указывает прокси-модулю удалить заголовок по умолчанию `close`, позволяя подключению оставаться открытым.

Обсуждение

Вы хотите, чтобы соединения оставались открытыми для вышестоящих серверов и сэкономили время, необходимое для инициирования подключения, что позволяет рабочему процессу вместо этого перейти непосредственно к выполнению запроса через незанятое подключение. Важно отметить, что количество открытых подключений может превышать количество подключений, указанное в директиве `keepalive`, поскольку открытые подключения и незанятые подключения – это не одно и то же. Количество подключений `keepalive` должно быть достаточно маленьким, чтобы разрешить другие входящие подключения с вышестоящим сервером. Этот маленький трюк с настройкой NGINX может сэкономить циклы и повысить вашу производительность.

15.4. Буферизация ответов

Задача

Вам необходимо выполнить буферизацию ответов между вышестоящими серверами и клиентами в памяти, чтобы избежать записи ответов во временные файлы.

Решение

Настройте параметры прокси-буфера, чтобы позволить NGINX памяти буферизовать тела ответов:

```
server {
    proxy_buffering on;
    proxy_buffer_size 8k;
    proxy_buffers 8 32k;
    proxy_busy_buffer_size 64k;
    ...
}
```

Директива `proxy_buffering` включена (on) либо выключена (off); по умолчанию она включена. Директива `proxy_buffer_size` обозначает размер буфера, используемого для чтения первой части ответа от проксируемого сервера, и по умолчанию он равен 4 или 8 k в зависимости от платформы. Директива `proxy_buffers` принимает два параметра: количество буферов и их размер. По умолчанию в директиве `proxy_buffers` указано 8 буферов размером 4 или 8 k в зависимости от платформы. Директива `proxy_busy_buffer_size` ограничивает размер буферов, которые могут быть заняты, отправляя ответ клиенту, пока ответ не полностью прочитан. Размер занятого буфера по умолчанию удваивает размер прокси-буфера или размер буфера.

Обсуждение

Прокси-буферы могут значительно улучшить производительность прокси в зависимости от типичного размера тел ответов. Настройка этих параметров может иметь неблагоприятные последствия и должна выполняться путем наблюдения за возвращаемым средним размером тела, а также тщательного и многократного тестирования.

Чрезвычайно большие буферы, установленные, когда они не нужны, могут поглотить память вашего NGINX-устройства. Вы можете установить эти параметры для определенных местоположений, которые, как известно, возвращают большие тела ответа для оптимальной производительности.

15.5. Буферизация журналов доступа

Задача

Вам нужно буферизовать журналы, чтобы уменьшить возможность блоков до рабочего процесса NGINX, когда система находится под нагрузкой.

Решение

Установите размер буфера и время сброса ваших журналов доступа:

```
http {
    access_log /var/log/nginx/access.log main buffer=32k
        flush=1m;
}
```

Параметр `buffer` директивы `access_log` обозначает размер буфера памяти, который можно заполнить данными журнала перед записью на диск. Параметр `flush` директивы `access_log` устанавливает максимальное время, в течение которого журнал может оставаться в буфере до его записи на диск.

Обсуждение

Буферизация данных журнала в память может быть небольшим шагом к оптимизации. Однако для сайтов и приложений с большим количеством запросов это может внести существенную корректировку в использование диска и центрального процессора. При использовании параметра `buffer` в директиве `access_log` журналы будут записываться на диск, если следующая запись журнала не помещается в буфер. Если использовать параметр `flush` в сочетании с параметром `buffer`, журналы будут записываться на диск, когда данные в буфере старше указанного времени. При буферизации журналов таким способом, при настройке журнала вы можете увидеть задержки вплоть до количества времени, указанного параметром `flush`.

15.6. Настройка ОС

Задача

Вам нужно настроить свою операционную систему так, чтобы она принимала больше подключений для обработки пиковых нагрузок или сайтов с высокой посещаемостью.

Решение

Проверьте настройки ядра для *net.core.somaxconn*, что является максимальным числом подключений, которые ядро может поставить в очередь для обработки NGINX. Если это число выше 512, вам нужно будет задать параметр `backlog` директивы `listen` в вашей конфигурации

NGINX. Признаком того, что вы должны изучить этот параметр ядра, является то, что ваш журнал ядра явно говорит об этом. NGINX обрабатывает подключения очень быстро, и в большинстве случаев вам не понадобится изменять эту настройку.

Увеличение количества дескрипторов открытых файлов является более распространенной необходимостью. В Linux дескриптор файла открывается для каждого подключения; поэтому NGINX может открыть два, если вы используете его в качестве прокси или балансировщика нагрузки из-за открытого соединения в восходящем направлении. Чтобы обслуживать большое количество подключений, вам может потребоваться увеличить ограничение дескриптора файла для всей системы с помощью параметра ядра `sys.fs.file_max`, или для системного пользователя NGINX работает так же, как в файле */etc/security/limits.conf*. При этом вам также понадобится увеличить количество `worker_connections` и `worker_rlimit_nofile`. Обе эти конфигурации являются директивами в конфигурации NGINX.

Активируйте больше эфемерных портов. Когда NGINX действует как обратный прокси-сервер или балансировщик нагрузки, каждое восходящее соединение открывает временный порт для обратного трафика. В зависимости от конфигурации вашей системы на сервере может не быть открыто максимальное количество временных портов. Чтобы проверить это, посмотрите настройку ядра `net.ipv4.ip_local_port_range`. Настройка представляет собой нижнюю и верхнюю границу диапазона портов. Обычно для этого параметра ядра можно установить значение от 1024 до 65535. 1024 – это то место, где останавливаются зарегистрированные TCP-порты, а 65535 – место, где останавливаются динамические или эфемерные порты. Имейте в виду, что ваша нижняя граница должна быть выше, чем самый высокий открытый сервисный порт прослушивания.

Обсуждение

Настройка операционной системы – одно из первых мест, куда вы смотрите, когда начинаете настраивать большое количество подключений. Есть много оптимизаций, которые вы можете внести в свое ядро для вашего конкретного случая использования. Однако настройку ядра не следует выполнять по прихоти, и нужно измерять изменения их производительности, чтобы убедиться, что эти изменения помогают. Как указывалось ранее, вы будете знать, когда пора начинать настройку ядра из сообщений, зарегистрированных в журнале ядра, или когда NGINX явно регистрирует сообщение в своем журнале ошибок.

Глава 16

Советы по практической эксплуатации и заключение

16.0. Введение

Последняя глава будет охватывать советы по практической эксплуатации, и одновременно это – заключение. В ходе прочтения этой книги мы обсудили множество идей и концепций, относящихся к специалистам по эксплуатации. Тем не менее я подумал, что еще кое-что может быть полезным, чтобы завершить книгу. В этой главе я расскажу о том, как сделать так, чтобы ваши файлы конфигурации были чистыми и лаконичными, а также об отладке файлов конфигурации.

16.1. Использование директивы include для чистых настроек

Задача

Вам необходимо очистить громоздкие файлы конфигурации, чтобы ваши конфигурации логически группировались в модульные наборы конфигурации.

Решение

Используйте директиву `include` для ссылки на файлы конфигурации, каталоги или маски:

```
http {
include config.d/compression.conf;
include sites-enabled/*.conf
}
```

Директива `include` принимает один параметр: либо путь к файлу, либо маску, которая соответствует множеству файлов. Эта директива действует в любом контексте.

Обсуждение

Используя операторы `include`, вы можете сохранить свою конфигурацию NGINX чистой и лаконичной. Вы сможете логически сгруппировать свои конфигурации, чтобы избежать файлов конфигурации на сотни строк. Можно создавать модульные файлы конфигурации, которые могут быть включены в нескольких местах по всей вашей конфигурации, чтобы избежать дублирования. Возьмем в качестве примера файла конфигурации *fastcgi_param*, представленный в большинстве установок NGINX для управления пакетами. Если вы управляете несколькими виртуальными серверами FastCGI в одном блоке NGINX, то можете включить этот файл конфигурации для любого местоположения или контекста, где вам требуются эти параметры FastCGI без необходимости дублировать эту конфигурацию. Еще один пример – конфигурации SSL. Если вы используете несколько серверов, которые требуют одинаковых конфигураций SSL, можете просто написать эту конфигурацию один раз и включать ее там, где это необходимо. Логически группируя свои конфигурации, вы можете быть уверены, что они аккуратны и организованы. Изменение набора файлов конфигурации можно выполнить путем редактирования одного файла, а не путем изменения нескольких наборов блоков конфигурации в нескольких местах в массивном файле конфигурации. Группировка конфигураций в файлы и использование операторов `include` – хорошая практика для вашего душевного равновесия и душевного равновесия ваших коллег.

16.2. Отладка конфигураций

Задача

Вы получаете неожиданные результаты от вашего сервера NGINX.

Решение

Выполните отладку конфигурации и запомните эти советы:

- NGINX обрабатывает запросы в поисках наиболее конкретного сопоставленного правила. Это делает пошаговое выполнение конфигураций вручную немного сложнее, но это самый эффективный способ для работы NGINX. Подробнее о том, как NGINX

обрабатывает запросы, смотрите ссылку на документацию в разделе «См. также» на стр. 173;

- можно включить ведение журнала отладки. Для этого вам необходимо убедиться, что ваш пакет NGINX настроен с флагом `--with-debug`. В большинстве распространенных пакетов он есть; но, если вы создали свой собственный или используете минимальный пакет, можете, по крайней мере, проверить еще раз. Убедившись, что у вас есть отладка, вы можете установить уровень ведения журнала директивы `error_log` на `debug: error_log /var/log/nginx/error.log debug`;
- вы можете активировать отладку для определенных подключений. Директива `debug_connection` действительна в контексте `events` и принимает в качестве параметра IP-адрес или диапазон CIDR. Эту директиву можно объявлять более одного раза для добавления нескольких IP-адресов или диапазонов CIDR для отладки. Это может быть полезно для устранения проблемы в реальной эксплуатации без снижения производительности путем отладки всех подключений;
- можно выполнять отладку только для определенных виртуальных серверов. Поскольку директива `error_log` действительна в контексте main, HTTP, mail, stream, server и location, вы можете установить уровень журнала `debug` только в тех контекстах, которые вам нужны;
- можно активировать дампы ядра и получить от них обратную трассировку. Дампы ядра можно привести в действие через операционную систему или файл конфигурации NGINX. Подробнее об этом можно прочитать в руководстве администратора в разделе «См. также» на стр. 173;
- Вы можете записывать, что происходит в операторах перезаписи, с помощью включенной директивы `rewrite_log`: `rewrite_log on`.

Обсуждение

Платформа NGINX обширна, а ее настройка позволяет вам делать множество удивительных вещей. Тем не менее благодаря способности делать удивительные вещи, также существует возможность навредить самому себе. Выполняя отладку, убедитесь, что вы знаете, как отследить ваш запрос через вашу конфигурацию; и, если у вас есть проблемы, добавьте уровень журнала debug, чтобы справиться с ситуацией. Журнал

отладки довольно многословен, но очень полезен, чтобы определить, что NGINX делает с вашим запросом и где в конфигурации вы ошиблись.

См. также:
http://nginx.org/en/docs/http/request_processing.html
https://docs.nginx.com/nginx/admin-guide/monitoring/debugging/
https://nginx.org/en/docs/http/ngx_http_rewrite_module.html#rewrite_log

16.3. Заключение

В этой книге мы рассказали о высокопроизводительной балансировке нагрузки, безопасности, а также развертывании и обслуживании серверов NGINX и NGINX Plus. Мы продемонстрировали некоторые самые мощные функции платформы доставки приложений NGINX. Компания NGINX Inc. продолжает разрабатывать удивительные функции и остается на шаг впереди.

Эта книга продемонстрировала множество коротких рецептов, которые позволят вам лучше понять некоторые директивы и модули, делающие NGINX сердцем современной сети. Сервер NGINX – это не просто веб-сервер или обратный прокси-сервер, а целая платформа для доставки приложений, полностью способная к аутентификации и позволяющая использовать все среды, в которых она работает. Возможно, теперь вы это знаете.

Сведения об авторе

У **Дерека де Йонге** была страсть к технологиям на протяжении всей жизни. Его опыт и знания в области веб-разработки, системного администрирования и организации сетей дают ему всестороннее понимание современной веб-архитектуры. Дерек возглавляет команду специалистов по SRE (Site Reliability Engineering) и занимается производством самовосстанавливающейся инфраструктуры с автоматическим масштабированием для многочисленных приложений. Он специализируется на облачных средах Linux. Разрабатывая, создавая и поддерживая приложения с высокой доступностью для клиентов, он консультирует крупные организации, когда они отправляются в облако. Дерек и его команда находятся на передовой линии технологической волны, ежедневно разрабатывая передовые облачные технологии. Обладая проверенной репутацией в области отказоустойчивой облачной архитектуры, Дерек помогает компании RightBrain Networks стать одним из самых сильных облачных консалтинговых агентств и поставщиков управляемых услуг в партнерстве с AWS на сегодняшний день.

Предметный указатель

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «Планета Альянс» наложенным платежом, выслав открытку или письмо по почтовому адресу: **115487, г. Москва, 2-й Нагатинский пр-д, д. 6А**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.

Оптовые закупки: тел**. +7(499) 782-38-89**.

Электронный адрес: **books@alians-kniga.ru**.

Дерек де Йонге

NGINX. Книга рецептов

Продвинутые рецепты высокопроизводительной балансировки нагрузки

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод с английского *Беликов Д. А.*
Корректор *Чистякова Л. А.*
Верстка *Паранская Н. В.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×90 1/16. Печать цифровая.
Усл. печ. л. 12,87. Тираж 200 экз.

Веб-сайт издательства: www.dmkpress.com