

**Повысьте устойчивость, наглядность и безопасность Kubernetes с помощью F5 NGINX**

Если вы готовы внедрить методы, описанные в этой книге, вам нужны инструменты с нужными функциями для поддержки ваших вариантов использования. NGINX может помочь с нашим набором инструментов управления трафиком производственного уровня Kubernetes:

[**NGINX Ingress Controller** –](https://www.nginx.com/products/nginx-ingress-controller/)  Основанный на NGINX Plus, с расширенным управлением трафиком и формированием, мониторингом и видимостью, аутентификацией и SSO, а также функциями шлюза API.

[**NGINX App Protect** –](https://www.nginx.com/products/nginx-app-protect/)  Комплексная защита современных приложений и API, построенная на ведущих на рынке технологиях безопасности F5 и интегрированная с NGINX Ingress Controller и NGINX Plus. Два независимых модуля обеспечивают гибкость развертывания и оптимальное использование ресурсов

* [**NGINX App Protect WAF** –](https://www.nginx.com/products/nginx-app-protect/web-application-firewall/)  Надежная и легкая защита от OWASP Top 10 и более поздних версий, а также соответствие требованиям PCI DSS
* [**NGINX App Protect DoS** –](https://www.nginx.com/products/nginx-app-protect/denial-of-service/)  Основанное на поведении, адаптивное и согласованное обнаружение DOS и смягчение последствий в облаках и архитектурах

[**NGINX Service Mesh** –](https://www.nginx.com/products/nginx-service-mesh/)  Легкий, готовый к использованию и удобный для разработчиков, с использованием NGINX Plus в качестве корпоративной коляски.

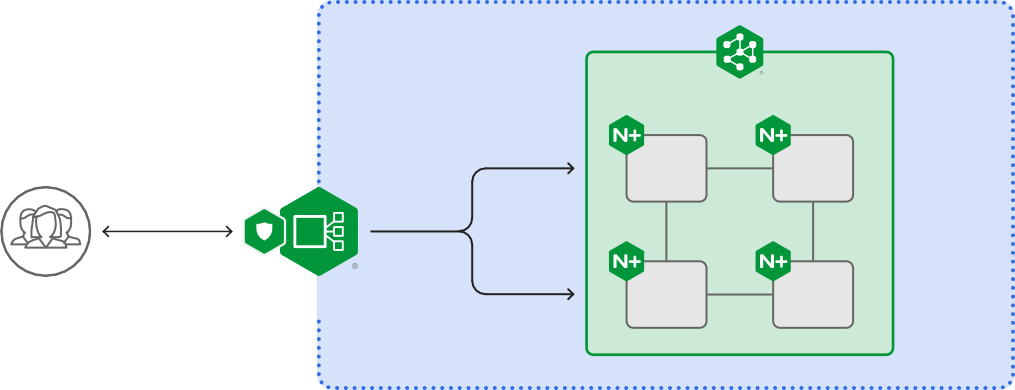
**Kubernetes Environment**

**NGINX Сервисная сетка**

**Страница Сервис A**

**Контроллер входа NGINX с защитой приложения NGINX**

**Страница Сервис B**



Для начала запросите бесплатную 30-дневную пробную версию NGINX Ingress Controller с приложением NGINX для защиты WAF и DoS и загрузите всегда бесплатный сервис NGINX Mesh. Узнайте больше сегодня на nginx.com.

©2022 F5, Inc. All rights reserved. F5, the F5 logo, NGINX, the NGINX logo, NGINX App Protect, NGINX App Protect DoS, NGINX App Protect WAF, NGINX Ingress Controller, NGINX Plus, and NGINX Service Mesh are trademarks of F5, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at [f5.com](https://www.f5.com/). Any other products, services, or company names referenced herein may be trademarks of their respective owners with no endorsement or affiliation, expressed or implied, claimed by F5, Inc.



##### ВТОРОЕ ИЗДАНИЕ

NGINX Cookbook

***Advanced Recipes for High-Performance***

***Load Balancing***

***Перевод: @trigantail***

***Derek DeJonghe***



**NGINX Cookbook**

by Derek DeJonghe

Авторское право © 2022 O'Reilly Media, Inc. Все права защищены. Напечатано в Соединенных Штатах Америки.

Опубликовано O'Reilly Media, Inc., 1005 Gravenstein Highway North, Севастополь, Калифорния 95472.

Книги О'Рейли можно приобрести для образовательных, деловых или рекламных целей. Онлайн-издания также доступны для большинства изданий (http://oreilly.com ). Для получения дополнительной информации свяжитесь с нашим корпоративным/институциональным отделом продаж: 800-998-9938 или corporate@oreilly.com .

**Acquisitions Editor:** Melissa Duffield **Development Editor:** Gary O’Brien **Production Editor:** Christopher Faucher **Copyeditor:** Kim Cofer

**Proofreader:** JM Olejarz

**Indexer:** Potomac Indexing, LLC **Interior Designer:** David Futato **Cover Designer:** Karen Montgomery **Illustrator:** Kate Dullea

November 2020: First Edition May 2022: Second Edition

**Revision History for the First Edition**

2022-05-16: First Release

See [*http://oreilly.com/catalog/errata.csp?isbn=9781492078487*](http://oreilly.com/catalog/errata.csp?isbn=9781492078487) for release details.

Логотип O'Reilly является зарегистрированной торговой маркой O'Reilly Media, Inc. Кулинарная книга NGINX, изображение на обложке и связанный с ней фирменный стиль являются товарными знаками O'Reilly Media, Inc.

Взгляды, выраженные в этой работе, принадлежат автору и не отражают точку зрения издателя. Несмотря на то, что издатель и автор приложили добросовестные усилия для обеспечения точности информации и инструкций, содержащихся в этой работе, издатель и автор снимают с себя всякую ответственность за ошибки или упущения, включая, помимо прочего, ответственность за ущерб, возникший в результате использования этой работы или доверия к ней. Используйте информацию и инструкции, содержащиеся в данной работе, на свой страх и риск. Если на какие-либо образцы кода или другие технологии, содержащиеся или описываемые в этой работе, распространяются лицензии с открытым исходным кодом или права интеллектуальной собственности других лиц, вы несете ответственность за то, чтобы их использование соответствовало таким лицензиям и / или правам.

This work is part of a collaboration between O’Reilly and NGINX. See our [statement of editorial](https://oreil.ly/editorial-independence) [independence](https://oreil.ly/editorial-independence).

978-1-0098-12626-1 [LSI]

# Cодержание

[Foreword. xi](#_bookmark0)

[Preface. xiii](#_bookmark1)

1. [Basics. 1](#_bookmark2)
   1. [Introduction](#_bookmark3) 1
   2. [Installing NGINX on Debian/Ubuntu](#_bookmark5) 1
   3. [Installing NGINX on RedHat/CentOS](#_bookmark8) 2
   4. [Installing NGINX Plus](#_bookmark10) 3
   5. [Verifying Your Installation](#_bookmark12) 3
   6. [Key Files, Directories, and Commands](#_bookmark18) 4
   7. [Serving Static Content](#_bookmark24) 6
   8. [Graceful Reload](#_bookmark31) 7
2. [High-Performance Load Balancing. 9](#_bookmark35)
   1. [Introduction](#_bookmark36) 9
   2. [HTTP Load Balancing](#_bookmark42) 10
   3. [TCP Load Balancing](#_bookmark46) 11
   4. [UDP Load Balancing](#_bookmark51) 13
   5. [Load-Balancing Methods](#_bookmark58) 14
   6. [Sticky Cookie with NGINX Plus](#_bookmark67) 16
   7. [Sticky Learn with NGINX Plus](#_bookmark70) 17
   8. [Sticky Routing with NGINX Plus](#_bookmark74) 18
   9. [Connection Draining with NGINX Plus](#_bookmark77) 19
   10. [Passive Health Checks](#_bookmark81) 20
   11. [Active Health Checks with NGINX Plus](#_bookmark85) 21
   12. [Slow Start with NGINX Plus](#_bookmark91) 23
3. [Traffic Management. 25](#_bookmark96)
   1. [Introduction](#_bookmark97) 25
   2. [A/B Testing](#_bookmark99) 25
   3. [Using the GeoIP Module and Database](#_bookmark103) 27
   4. [Restricting Access Based on Country](#_bookmark110) 29
   5. [Finding the Original Client](#_bookmark112) 30
   6. [Limiting Connections](#_bookmark118) 31
   7. [Limiting Rate](#_bookmark123) 32
   8. [Limiting Bandwidth](#_bookmark128) 34
4. [Massively Scalable Content Caching. 35](#_bookmark133)
   1. [Introduction](#_bookmark134) 35
   2. [Caching Zones](#_bookmark137) 35
   3. [Cache Locking](#_bookmark141) 36
   4. [Caching Hash Keys](#_bookmark144) 37
   5. [Cache Bypass](#_bookmark148) 38
   6. [Cache Performance](#_bookmark152) 39
   7. [Cache Purging with NGINX Plus](#_bookmark155) 39
   8. [Cache Slicing](#_bookmark159) 40
5. [Programmability and Automation. 43](#_bookmark166)
   1. [Introduction](#_bookmark167) 43
   2. [NGINX Plus API](#_bookmark168) 43
   3. [Using the Key-Value Store with NGINX Plus](#_bookmark174) 47
   4. [Using the NJS Module to Expose JavaScript Functionality Within NGINX](#_bookmark181) 49
   5. [Extending NGINX with a Common Programming Language](#_bookmark189) 52
   6. [Installing with Chef](#_bookmark198) 54
   7. [Installing with Ansible](#_bookmark203) 55
   8. [Automating Configurations with Consul Templating](#_bookmark208) 57
6. [Authentication. 59](#_bookmark213)
   1. [Introduction](#_bookmark214) 59
   2. [HTTP Basic Authentication](#_bookmark216) 59
   3. [Authentication Subrequests](#_bookmark224) 61
   4. [Validating JWTs with NGINX Plus](#_bookmark229) 62
   5. [Creating JSON Web Keys](#_bookmark235) 63
   6. [Validate JSON Web Tokens with NGINX Plus](#_bookmark238) 64
   7. [Automatically Obtaining and Caching JSON Web Key Sets with NGINX](#_bookmark242)

[Plus](#_bookmark242) 65

* 1. [Authenticate Users via Existing OpenID Connect SSO with NGINX Plus](#_bookmark245) 66

1. [Security Controls. 69](#_bookmark249)
   1. [Introduction](#_bookmark250) 69
   2. [Access Based on IP Address](#_bookmark252) 69
   3. [Allowing Cross-Origin Resource Sharing](#_bookmark256) 70
   4. [Client-Side Encryption](#_bookmark261) 72
   5. [Advanced Client-Side Encryption](#_bookmark265) 73
   6. [Upstream Encryption](#_bookmark271) 75
   7. [Securing a Location](#_bookmark277) 75
   8. [Generating a Secure Link with a Secret](#_bookmark281) 76
   9. [Securing a Location with an Expire Date](#_bookmark284) 77
   10. [Generating an Expiring Link](#_bookmark287) 78
   11. [HTTPS Redirects](#_bookmark292) 80
   12. [Redirecting to HTTPS Where SSL/TLS Is Terminated Before NGINX](#_bookmark294) 80
   13. [HTTP Strict Transport Security](#_bookmark297) 81
   14. [Satisfying Any Number of Security Methods](#_bookmark300) 82
   15. [NGINX Plus Dynamic Application Layer DDoS Mitigation](#_bookmark303) 83
   16. [Installing and Configuring NGINX Plus with the NGINX App Protect](#_bookmark308)

[WAF Module](#_bookmark308) 84

8. [HTTP/2. 89](#_bookmark316)

* 1. [Introduction](#_bookmark317) 89
  2. [Basic Configuration](#_bookmark319) 89
  3. [gRPC](#_bookmark322) 90
  4. [HTTP/2 Server Push](#_bookmark330) 92

1. [Sophisticated Media Streaming. 95](#_bookmark334)
   1. [Introduction](#_bookmark335) 95
   2. [Serving MP4 and FLV](#_bookmark337) 95
   3. [Streaming with HLS with NGINX Plus](#_bookmark339) 96
   4. [Streaming with HDS with NGINX Plus](#_bookmark343) 97
   5. [Bandwidth Limits with NGINX Plus](#_bookmark347) 98
2. [Cloud Deployments. 99](#_bookmark352)
   1. [Introduction](#_bookmark353) 99
   2. [Auto-Provisioning on AWS](#_bookmark355) 99
   3. [Routing to NGINX Nodes Without an AWS ELB](#_bookmark362) 101
   4. [The NLB Sandwich](#_bookmark367) 102
   5. [Deploying from the AWS Marketplace](#_bookmark372) 104
   6. [Creating an NGINX Virtual Machine Image on Azure](#_bookmark378) 105
   7. [Load Balancing Over NGINX Scale Sets on Azure](#_bookmark381) 107
   8. [Deploying Through the Azure Marketplace](#_bookmark384) 108
   9. [Deploying to Google Compute Engine](#_bookmark387) 109
   10. [Creating a Google Compute Image](#_bookmark389) 109
   11. [Creating a Google App Engine Proxy](#_bookmark391) 110
3. [Containers/Microservices. 113](#_bookmark398)
   1. [Introduction](#_bookmark399) 113
   2. [Using NGINX as an API Gateway](#_bookmark403) 114
   3. [Using DNS SRV Records with NGINX Plus](#_bookmark418) 118
   4. [Using the Official NGINX Image](#_bookmark422) 119
   5. [Creating an NGINX Dockerfile](#_bookmark427) 120
   6. [Building an NGINX Plus Docker Image](#_bookmark429) 122
   7. [Using Environment Variables in NGINX](#_bookmark433) 124
   8. [Kubernetes Ingress Controller](#_bookmark437) 125
   9. [Prometheus Exporter Module](#_bookmark447) 127
   10. [NGINX Service Mesh mTLS](#_bookmark454) 128
4. [High-Availability Deployment Modes. 131](#_bookmark463)
   1. [Introduction](#_bookmark464) 131
   2. [NGINX Plus HA Mode](#_bookmark468) 131
   3. [Load-Balancing Load Balancers with DNS](#_bookmark472) 132
   4. [Load Balancing on EC2](#_bookmark477) 132
   5. [NGINX Plus Configuration Synchronization](#_bookmark483) 133
   6. [State Sharing with NGINX Plus and Zone Sync](#_bookmark493) 136
5. [Advanced Activity Monitoring. 139](#_bookmark498)
   1. [Introduction](#_bookmark499) 139
   2. [Enable NGINX Open Source Stub Status](#_bookmark501) 139
   3. [Enabling the NGINX Plus Monitoring Dashboard](#_bookmark505) 140
   4. [Collecting Metrics Using the NGINX Plus API](#_bookmark513) 143
6. [Debugging and Troubleshooting with Access Logs, Error Logs, and Request Tracing. 147](#_bookmark519)
   1. [Introduction](#_bookmark520) 147
   2. [Configuring Access Logs](#_bookmark522) 147
   3. [Configuring Error Logs](#_bookmark530) 149
   4. [Forwarding to Syslog](#_bookmark533) 150
   5. [Request Tracing](#_bookmark538) 151
   6. [OpenTracing for NGINX](#_bookmark542) 152
7. [Performance Tuning. 155](#_bookmark547)
   1. [Introduction](#_bookmark548) 155
   2. [Automating Tests with Load Drivers](#_bookmark550) 155
   3. [Keeping Connections Open to Clients](#_bookmark553) 156
   4. [Keeping Connections Open Upstream](#_bookmark556) 157
   5. [Buffering Responses](#_bookmark562) 158
   6. [Buffering Access Logs](#_bookmark565) 159
   7. [OS Tuning](#_bookmark569) 159
8. [Introduction to NGINX Instance Manager. 161](#_bookmark574)
   1. [Introduction](#_bookmark575) 161
   2. [Setup Overview](#_bookmark577) 161
   3. [Agent Installation](#_bookmark584) 163
   4. [Automating NGINX Discovery, Configuration, and Monitoring with the](#_bookmark588)

[API](#_bookmark588) 165

1. [Introduction to NGINX Controller. 167](#_bookmark593)
   1. [Introduction](#_bookmark594) 167
   2. [Setup Overview](#_bookmark596) 167
   3. [Connecting NGINX Plus with Controller](#_bookmark605) 169
   4. [Driving NGINX Controller with the API](#_bookmark607) 170
   5. [Enable WAF Through Controller App Security](#_bookmark610) 171
2. [Practical Ops Tips and Conclusion. 175](#_bookmark616)
   1. [Introduction](#_bookmark617) 175
   2. [Using Includes for Clean Configs](#_bookmark618) 175
   3. [Debugging Configs](#_bookmark621) 176

[Conclusion. 179](#_bookmark628)

[Index. 181](#_bookmark629)

# Предисловие

Добро пожаловать на выпуск 2022 года NGINX Cookbook. О'Рейли публикует *NGINX Cookbook* уже семь лет, и мы продолжаем совершенствовать контент. По данным W3Techs, NGINX сегодня является самым популярным веб-сервером в мире и обслуживает почти 50% из 1000 самых загруженных сайтов. С тех пор как мы выпустили первую версию NGINX в 2002 году, продукт эволюционировал и превратился в швейцарский армейский нож в лучшем смысле этого слова. Мы спроектировали NGINX для обеспечения гибкости и масштабируемости, что позволило расширить наши возможности за пределы веб-сервиса, а также за счет балансировки нагрузки, обратного прокси-сервера и API-шлюза. Мы считаем свидетельством ценности NGINX тот факт, что многие сервисы балансировки нагрузки, предлагаемые крупными общедоступными облаками и CDN, на самом деле основаны на коде NGINX.

NGINX также продолжает расширяться в новых областях; наш брандмауэр веб-приложений теперь применяет машинное обучение для адаптации к новым угрозам по мере их возникновения. Контроллер входа NGINX для Kubernetes изначально интегрирован с NGINX и предоставляет ключевые возможности для управления трафиком как с востока на запад, так и с севера на юг, что является критическим требованием в мире Kubernetes. Все это не имело бы значения, если бы NGINX не продолжал обеспечивать скорость, отказоустойчивость и маневренность — все фундаментальные требования современных распределенных приложений.

*NGINX Cookbook* это пошаговое руководство по NGINX от людей, которые лучше всего знают код. Независимо от того, используете ли вы NGINX с открытым исходным кодом для небольшого проекта или NGINX Plus для критически важного корпоративного развертывания в нескольких регионах, и независимо от того, работаете ли вы локально или в облаке, кулинарная книга покажет вам, как получить максимальную отдачу от NGINX. В нем представлены десятки простых в использовании рецептов, описывающих, как установить NGINX, настроить его практически для любого варианта использования, защитить и масштабировать, а также устранить распространенные неполадки.

Издание 2022 года обновляет многие разделы, охватывая новые функциональные возможности NGINX и добавляя совершенно новые разделы и главы, посвященные развертыванию собственных облачных вычислений, а также новым инструментам и шаблонам развертывания для этой смены парадигмы. Мир технологий и приложений быстро меняется, и NGINX меняется вместе с ним, чтобы продолжать вносить свой вклад в ваш успех.

Первоначальным видением NGINX было высокомасштабируемое, надежное, быстрое и безопасное веб-обслуживание. Все, что мы делаем сегодня, построено на этом оригинальном видении и разработано для того, чтобы помочь нашему сообществу уверенно развертывать нужные им приложения в любой среде, в любом масштабе. Пожалуйста, наслаждайтесь этим последним изданием и не стесняйтесь сообщать нам, что вы думаете (на нашем канале Slack или онлайн). Мы слушаем.

*—Christopher Guzikowski Product Marketing, NGINX, Inc.*

# Предисловие

*NGINX Cookbook* цель состоит в том, чтобы предоставить простые для понимания примеры реальных проблем при доставке приложений. На протяжении всей этой книги вы познакомитесь со многими функциями NGINX и с тем, как их использовать. Это руководство является довольно полным и затрагивает большинство основных возможностей NGINX.

На протяжении всей этой книги будут встречаться ссылки как на бесплатное программное обеспечение NGINX с открытым исходным кодом, так и на коммерческие продукты NGINX Inc.: NGINX Plus и NGINX Controller. Функции и директивы, которые доступны только в рамках платной подписки на NGINX Plus, будут обозначены как таковые. Поскольку NGINX Plus является контроллером доставки приложений (ADC) и предоставляет множество расширенных функций, важно выделить эти функции, чтобы получить полное представление о возможностях платформы.

Книга начнется с объяснения процесса установки NGINX и NGINX Plus, а также некоторых основных шагов по началу работы для читателей, впервые знакомящихся с NGINX. Далее разделы перейдут к балансировке нагрузки во всех формах, сопровождаемой главами об управлении трафиком, кэшировании и автоматизации. Глава 6, “Аутентификация”, охватывает много вопросов, но она важна, поскольку NGINX часто является первой точкой входа веб-трафика в ваше приложение и первой линией защиты прикладного уровня от веб-атак и уязвимостей. Существует ряд глав, в которых рассматриваются передовые темы, такие как HTTP / 2, потоковая передача мультимедиа, облако, сервисная сетка и контейнерные среды, а также более традиционные операционные темы, такие как мониторинг, отладка, производительность и советы по эксплуатации. Книга завершается введением в NGINX Instance Manager и NGINX Controller, платформу управления, ориентированную на приложения.

Я лично использую NGINX как мультитул и верю, что эта книга позволит вам сделать то же самое. Это программное обеспечение, в которое я верю и с которым мне нравится работать. Я рад поделиться с вами этими знаниями и надеюсь, что, читая эту книгу, вы соотнесете рецепты с вашими реальными сценариями и будете использовать эти решения.

## Условные обозначения, используемые в этой книге

В этой книге используются следующие типографские соглашения:

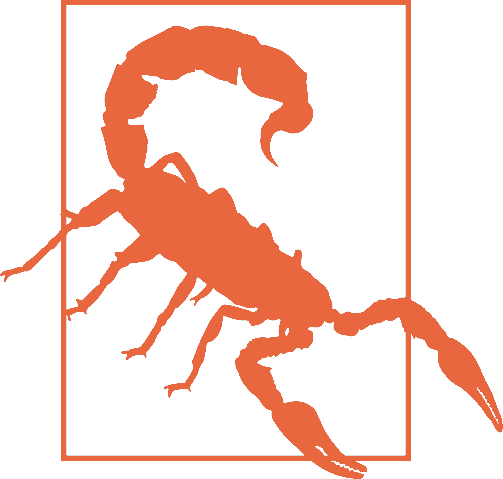
*Курсивный*

Указывает новые термины, URL-адреса, адреса электронной почты, имена файлов и расширения файлов.

Постоянная ширина

Используется для списков программ, а также внутри абзацев для ссылки на программные элементы, такие как имена переменных или функций, базы данных, типы данных, переменные окружения, инструкции и ключевые слова.

 Этот элемент обозначает общее замечание.

 Этот элемент указывает на предупреждение или предостережение.

## O’Reilly Онлайн-обучение

Вот уже более 40 лет O'Reilly Media предоставляет технологии и бизнес‐тренинги, знания и инсайт, помогающие компаниям добиться успеха.

## Наша уникальная сеть экспертов и новаторов делится своими знаниями и экспертизой с помощью книг, статей и нашей платформы онлайн-обучения. Платформа онлайн-обучения O'Reilly предоставляет вам доступ по запросу к живым учебным курсам, путям углубленного обучения, интерактивным средам программирования и обширной коллекции текстов и видео от O'Reilly и более 200 других издателей. Для получения дополнительной информации посетите http://oreilly.com .

## Как связаться с нами

## Пожалуйста, направляйте комментарии и вопросы, касающиеся этой книги, издателю:

O’Reilly Media, Inc.

1005 Gravenstein Highway North Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)

707-829-0515 (international or local) 707-829-0104 (fax)

У нас есть веб-страница для этой книги, где мы перечисляем ошибки, примеры и любую дополнительную информацию. Вы можете получить доступ к этой странице через [*https://oreil.ly/NGINX-cookbook-2e*](https://oreil.ly/NGINX-cookbook-2e).

Электронная почта bookquestions@oreilly.com чтобы прокомментировать или задать технические вопросы по поводу этой книги.

## Для получения новостей и информации о наших книгах и курсах посетите https://oreilly.com .

## Найдите нас в LinkedIn: https://linkedin.com/company/oreilly-media

## Следите за нами в Twitter: https://twitter.com/oreillymedia

## Смотрите нас на YouTube: <https://youtube.com/oreillymedia>

## Подтверждения

Я хотел бы поблагодарить Хуссейна Нассера и Гонсало Хосуэ Спину за их полезные и подробные обзоры этого названия. Я также хотел бы поблагодарить свою жену за ее поддержку в процессе написания.

**CHAPTER 1**

# Основы

## Введение

Чтобы начать работу с NGINX с открытым исходным кодом или NGINX Plus, вам сначала нужно установить его в систему и изучить некоторые основы. В этой главе вы узнаете, как установить NGINX, где находятся основные конфигурационные файлы и какие команды предназначены для администрирования. Вы также узнаете, как подтвердить свою установку и отправлять запросы на сервер по умолчанию.

## Установите NGINX на Debian/Ubuntu

### Проблема

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

### Решение

Обновите информацию о пакетах для настроенных исходных текстов и установите некоторые пакеты, которые помогут в настройке официального репозитория пакетов NGINX:

apt-get update

apt install -y curl gnupg2 ca-certificates lsb-release **\**

debian-archive-keyring

Загрузите и сохраните ключ подписи NGINX:

curl https://nginx.org/keys/nginx\_signing.key | gpg --dearmor **\**

| tee /usr/share/keyrings/nginx-archive-keyring.gpg >/dev/null

Используйте lsb\_release чтобы задать переменные, определяющие названия ОС и релизов, затем создайте

*apt* source file:

OS=**$(**lsb\_release -is | tr '[:upper:]' '[:lower:]'**)** RELEASE=**$(**lsb\_release -cs**)**

echo "deb [signed-by=/usr/share/keyrings/nginx-archive-keyring.gpg] \ [http://nginx.org/packages/${](http://nginx.org/packages/%24)OS} ${RELEASE} nginx" **\**

| tee /etc/apt/sources.list.d/nginx.list

Обновите информацию о пакете еще раз, затем установите NGINX:

apt-get update

apt-get install -y nginx nginx

### Обсуждение

Команды, приведенные в этом разделе, предписывают системе управления пакетами advanced package tool (APT) использовать официальный репозиторий пакетов NGINX. Ключ подписи пакета NGINX GPG был загружен и сохранен в папке файловой системы для использования APT. Предоставление APT ключа подписи позволяет системе APT проверять пакеты из репозитория. Команда lsb\_release использовалась для автоматического определения операционной системы и названия выпуска, так что эти инструкции можно использовать во всех выпускаемых версиях Debian или Ubuntu. Команда apt-get update предписывает системе APT обновить списки своих пакетов из известных репозиториев. После обновления списка пакетов вы можете установить NGINX с открытым исходным кодом из официального репозитория NGINX. После того, как вы установите его, последняя команда запустит NGINX.

Installing NGINX on RedHat/CentOS

### Проблема

### Вам необходимо установить NGINX с открытым исходным кодом на Red Hat или CentOS.

### Решение

Создайте файл с названием */etc/yum.repos.d/nginx.repo* который содержит следующее содержимое:

[nginx] name=nginx repo

[baseurl=http://nginx.org/packages/OS/OSRELEASE/$basearch/](http://nginx.org/packages/OS/OSRELEASE/%24basearch/) gpgcheck=0

enabled=1

Измените файл, заменив OS в середине URL-адреса на rhel или centos, в зависимости от вашего дистрибутива. Замените OSRELEASE на 7 или 8 для версии 7.x или 8.x, соответственно. Затем выполните следующие 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 для этого решения, инструктирует систему управления пакетами YUM использовать официальный репозиторий пакетов с открытым исходным кодом NGINX. Следующие команды устанавливают NGINX с открытым исходным кодом из официального репозитория, инструктируют системы включить NGINX во время загрузки и сообщают ей запустить его сейчас. Брандмауэр командует открыть порт 80 для протокола TCP, который является портом по умолчанию для HTTP. Последняя команда перезагружает брандмауэр для фиксации изменений.

## Установка NGINX Plus

### Проблема

Вам надо установить NGINX Plus.

### Решение

### Визит http://cs.nginx.com/repo\_setup . В выпадающем меню выберите операционную систему, на которую вы устанавливаете, а затем следуйте инструкциям. 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 это руководство по установке репозитория с инструкциями по установке NGINX Plus. В зависимости от вашей операционной системы и версии эти инструкции немного различаются, но есть одна общая черта. Вы должны получить сертификат и ключ с портала NGINX и предоставить их своей системе, чтобы пройти аутентификацию в репозитории NGINX Plus.

Verifying Your Installation

### Проблема

Вы хотите проверить установку NGINX и проверить версию.

### Решение

Вы можете убедиться, что NGINX установлен, и проверить его версию, используя следующую команду:

$ nginx -v

nginx version: nginx/1.21.3

Как показано в этом примере, в ответе отображается версия.

Вы можете подтвердить, что NGINX запущен, используя следующую команду:

$ 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. Передав его в grep, вы можете выполнять поиск, по определенным словам, в выходных данных. В этом примере используется grep для поиска nginx. Результат показывает два запущенных процесса: главный и рабочий. Если запущен NGINX, вы всегда будете видеть главный и один или несколько рабочих процессов. Обратите внимание, что главный процесс запущен от имени root, поскольку NGINX нуждается в повышенных привилегиях для правильной работы. Инструкции по запуску NGINX приведены в следующем разделе. Чтобы узнать, как запустить NGINX в качестве демона, воспользуйтесь методологиями init.d или systems.

Чтобы убедиться, что NGINX корректно возвращает запросы, используйте свой браузер для отправки запроса на ваш компьютер или используйте curl. При отправке запроса используйте IP-адрес компьютера или имя хоста. Если он установлен локально, вы можете использовать localhost следующим образом:

curl localhost

### Вы увидите HTML-сайт приветствия NGINX по умолчанию.

### Обсуждение

Nginx позволяет вам взаимодействовать с двоичным файлом NGINX для проверки версии, составления списка установленных модулей, тестирования конфигураций и отправки сигналов главному процессу. NGINX должен быть запущен для того, чтобы он мог обслуживать запросы. Команда ps - это верный способ определить, запущен ли NGINX либо как демон, либо на переднем плане. Конфигурация, предоставляемая NGINX по умолчанию, запускает HTTP-сервер статического сайта на порту 80. Вы можете протестировать этот сайт по умолчанию, отправив HTTP-запрос на компьютер по адресу localhost, а также указав IP-адрес и имя хоста.

### 1.3 Ключевые файлы, каталоги и командыProblem

Вам необходимо разобраться в важных каталогах и командах NGINX.

### Решение

##### NGINX файлы и каталоги

Для начала работы с NGINX важно знать следующие файлы, каталоги и команды.

*/etc/nginx/*

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

*/etc/nginx/nginx.conf*

Файл /etc/nginx/nginx.conf является точкой входа в конфигурацию по умолчанию, используемой службой NGINX. Этот конфигурационный файл устанавливает глобальные настройки для таких вещей, как рабочие процессы, настройка, ведение журнала, загрузка динамических модулей и ссылки на другие конфигурационные файлы NGINX. В конфигурации по умолчанию файл /etc/nginx/ nginx.conf содержит http-блок верхнего уровня, или контекст, который включает в себя 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, заканчивающиеся на .conf, включаются в http-блок верхнего уровня из файла /etc/nginx/nginx.conf. Рекомендуется использовать инструкции include и организовывать конфигурацию таким образом, чтобы ваши конфигурационные файлы были краткими. В некоторых репозиториях пакетов эта папка называется sites-enabled, а файлы конфигурации связаны из папки с именем site-available; это соглашение устарело.*/var/log/nginx/*

##### Каталог /var/log/nginx/ является расположением журнала по умолчанию для NGINX. В этом каталоге вы найдете доступ.файл журнала и файл журнала ошибок. Журнал доступа содержит запись для каждого запроса, который обслуживает NGINX. Файл журнала ошибок содержит события ошибок и информацию об отладке, если модуль отладки включен.

##### NGINX Комманды

nginx -h

Показывает меню справки NGINX.

nginx -v

Показывает версию NGINX.

nginx -V

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

nginx -t

Проверяет конфигурацию NGINX.

nginx -T

Проверяет конфигурацию NGINX и выводит проверенную конфигурацию на экран. Эта команда полезна при обращении за поддержкой.

nginx -s signal

Флаг -s посылает сигнал главному процессу NGINX. Вы можете посылать такие сигналы, как остановка, завершение работы, перезагрузка и повторное открытие. Сигнал остановки немедленно прерывает процесс NGINX. Сигнал quit останавливает процесс NGINX после того, как он завершит обработку запросов в полете. Сигнал перезагрузки перезагружает конфигурацию. Сигнал повторного открытия предписывает NGINX повторно открыть файлы журналов.Discussion

Разобравшись с этими ключевыми файлами, каталогами и командами, вы сможете приступить к работе с NGINX. Используя эти знания, вы можете изменить файлы конфигурации по умолчанию и протестировать свои изменения с помощью команды nginx -t. Если ваш тест прошел успешно, вы также знаете, как дать указание NGINX перезагрузить свою конфигурацию с помощью команды nginx -s reload.

### Предоставление статического контента

### Проблема

### Вам нужно обслуживать статический контент с помощью NGINX.

### Решение

Перезапишите конфигурацию HTTP-сервера по умолчанию, расположенную в /etc/nginx/conf.d/ default.conf, следующим примером 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

Эта конфигурация обслуживает статические файлы по протоколу HTTP на порту 80 из каталога /usr/ share/nginx/html/. Первая строка в этой конфигурации определяет новый серверный блок. Это определяет новый контекст для прослушивания NGINX. Вторая строка предписывает NGINX прослушивать порт 80, а параметр default\_server предписывает NGINX использовать этот сервер в качестве контекста по умолчанию для порта 80. Директива listen также может принимать диапазон портов. Директива server\_name определяет имя хоста или имена запросов, которые должны быть направлены на этот сервер. Если бы конфигурация не определяла этот контекст как default\_server, NGINX направлял бы запросы на этот сервер только в том случае, если заголовок HTTP host соответствовал значению, указанному в директиве server\_name. С установленным контекстом default\_server вы можете опустить директиву server\_name, если у вас еще нет доменного имени для использования.

Блок location определяет конфигурацию на основе пути в URL-адресе. Путь или часть URL-адреса после домена называется единым идентификатором ресурса (URI). NGINX наилучшим образом сопоставит запрошенный URI с блоком местоположения. В примере используется / для сопоставления всех запросов. Директива root показывает NGINX, где искать статические файлы при обслуживании контента для данного контекста. URI запроса добавляется к значению корневой директивы при поиске запрошенного файла. Если бы мы указали префикс URI для директивы location, это было бы включено в добавленный путь, если только мы не использовали директиву alias, а не root. Директива location способна соответствовать широкому спектру выражений. Перейдите по ссылке в разделе "Также смотрите" для получения дополнительной информации. Наконец, директива index предоставляет NGINX файл по умолчанию или список файлов для проверки в случае, если в URI не указан дальнейший путь.

Also See

[Документация по директиве определения местоположения NGINX HTTP](https://oreil.ly/-TNyO)

### Плавная перезагрузка

### Проблема

### Вам нужно перезагрузить свою конфигурацию, не отбрасывая пакеты.

### Решение

Используйте метод перезагрузки NGINX для достижения плавной перезагрузки конфигурации без остановки сервера:

nginx -s reload

### Этот пример перезагружает систему NGINX, используя двоичный файл NGINX для отправки сигнала главному процессу.

### Обсуждение

Перезагрузка конфигурации NGINX без остановки сервера обеспечивает возможность изменять конфигурации "на лету" без отбрасывания каких-либо пакетов. В динамичной среде с высоким временем безотказной работы в какой-то момент вам потребуется изменить конфигурацию балансировки нагрузки. NGINX позволяет вам делать это, сохраняя балансировщик нагрузки в режиме онлайн. Эта функция открывает 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 в соответствии с потребностями среды.

**CHAPTER 2**

# Высокопроизводительная балансировка нагрузки

**2.0 Введение**

Современный пользовательский опыт в Интернете требует высокой производительности и времени безотказной работы. Для достижения этой цели запускается несколько копий одной и той же системы, и нагрузка распределяется между ними. По мере увеличения нагрузки другая копия системы может быть подключена к сети. Этот архитектурный прием называется горизонтальным масштабированием. Инфраструктура, основанная на программном обеспечении, приобретает все большую популярность благодаря своей гибкости, открывая огромный мир возможностей. Независимо от того, является ли вариант использования таким же небольшим, как набор из двух копий системы для обеспечения высокой доступности, или таким большим, как тысячи по всему миру, существует потребность в решении для балансировки нагрузки, которое было бы таким же динамичным, как и инфраструктура. NGINX удовлетворяет эту потребность несколькими способами, такими как балансировка нагрузки HTTP, TCP и протокола пользовательских дейтаграмм (UDP), которые мы рассмотрим в этой главе.

При балансировке нагрузки важно, чтобы воздействие на опыт клиента было исключительно положительным. Многие современные веб-архитектуры используют уровни приложений без сохранения состояния, хранящие состояние в общей памяти или базах данных. Однако это реальность не для всех. Состояние сеанса чрезвычайно ценно и обширно в интерактивных приложениях. Это состояние может храниться локально на сервере приложений по ряду причин; например, в приложениях, для которых обрабатываемые данные настолько велики, что сетевые издержки слишком велики с точки зрения производительности. Когда состояние сохраняется локально на сервере приложений, для удобства пользователя чрезвычайно важно, чтобы последующие запросы продолжали доставляться на тот же сервер. Другим аспектом ситуации является то, что серверы не должны освобождаться до завершения сеанса. Для масштабной работы с приложениями с отслеживанием состояния требуется интеллектуальный балансировщик нагрузки. NGINX Plus предлагает несколько способов решения этой проблемы путем отслеживания файлов cookie или маршрутизации. В этой главе рассматривается сохранение сеанса, поскольку это относится к балансировке нагрузки с помощью NGINX и NGINX Plus.

Важно убедиться, что приложение, обслуживаемое NGINX, исправно. По ряду причин восходящие запросы могут начать завершаться сбоем. Это может быть связано с подключением к сети, сбоем сервера или сбоем приложения, и это лишь некоторые из них. Прокси-серверы и средства балансировки нагрузки должны быть достаточно умными, чтобы обнаруживать сбой вышестоящих серверов (серверов, находящихся за балансировщиком нагрузки или прокси-сервером), и прекращать передачу трафика на них; в противном случае клиент будет ждать, только чтобы получить тайм-аут. Способ уменьшить ухудшение качества обслуживания при сбое сервера заключается в том, чтобы прокси-сервер проверял работоспособность вышестоящих серверов. NGINX предлагает два различных типа проверок работоспособности: пассивную, доступную в NGINX с открытым исходным кодом; и активную, доступную только в NGINX Plus. Активные проверки работоспособности через регулярные промежутки времени позволят установить соединение или запрос к вышестоящему серверу и могут подтвердить правильность ответа. Пассивные проверки работоспособности отслеживают подключение или ответы вышестоящего сервера по мере того, как клиенты отправляют запрос или подключение. Возможно, вы захотите использовать пассивные проверки работоспособности, чтобы снизить нагрузку на ваши вышестоящие серверы, и, возможно, вы захотите использовать активные проверки работоспособности, чтобы определить сбой вышестоящего сервера до того, как клиенту будет сообщено о сбое. В конце этой главы рассматривается мониторинг работоспособности вышестоящих серверов приложений, для которых выполняется балансировка нагрузки.

HTTP Load Balancing

### Проблема

### Вам нужно распределить нагрузку между двумя или более HTTP-серверами.

### Решение

Используйте NGINX’s HTTP модуль чтобы сбалансировать нагрузку по HTTP серверу используя upstream

блок:

upstream backend {

server 10.10.12.45:80 weight=1; server app.example.com:80 weight=2; server spare.example.com:80 backup;

}

server {

location / {

proxy\_pass http://backend;

}

}

Эта конфигурация балансирует нагрузку между двумя HTTP-серверами на порту 80 и определяет один из них в качестве резервного, который используется, когда два основных недоступны. Параметр weight указывает NGINX передавать в два раза больше запросов второму серверу, а параметр weight по умолчанию равен 1.

### Рассуждение

HTTP upstream модуль управляет балансировкой нагрузки для HTTP. Этот модуль определяет пул адресатов — любую комбинацию сокетов Unix, IP-адресов и записей DNS (службы доменных имен) или их сочетание. Вышестоящий модуль также определяет, как любой отдельный запрос назначается любому из вышестоящих серверов.

Каждое вышестоящее назначение определяется в вышестоящем пуле директивой сервера. В директиве сервера указывается сокет Unix, IP-адрес или полное доменное имя (FQDN), а также ряд необязательных параметров. Необязательные параметры дают больше контроля над маршрутизацией запросов. Эти параметры включают вес сервера в алгоритме балансировки; находится ли сервер в режиме ожидания, доступен или недоступен; и как определить, недоступен ли сервер. NGINX Plus предоставляет ряд других удобных параметров, таких как ограничения на подключение к серверу, расширенный контроль разрешения DNS и возможность медленно увеличивать количество подключений к серверу после его запуска.TCP Load Balancing

### Проблема

### Вам необходимо распределить нагрузку между двумя или более TCP-серверами.

### Решение

Используйте NGINX’s stream модуль чтоб сбалансировать нагрузку TCP сервера используя upstream

block:

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;

}

}

server block в этом примере NGINX дает команду прослушивать TCP-порт 3306 и балансировать нагрузку между двумя репликами чтения базы данных MySQL, а также указывает другой сервер в качестве резервной копии, на который будет передаваться трафик, если основные серверы не работают.

Эта конфигурация не должна добавляться в папку conf.d, поскольку эта папка включена в блок http; вместо этого вам следует создать другую папку с именем stream.conf.d,

Откройте stream block в файле nginx.conf и включите новую папку для потоковых конфигураций. Ниже приводится пример.

В конфигурационном файле /etc/nginx/nginx.conf:

user nginx; worker\_processes auto; pid /run/nginx.pid;

stream {

include /etc/nginx/stream.conf.d/\*.conf;

}

Файл с именем /etc/nginx/stream.conf.d/mysql\_reads.conf может содержать следующую конфигурацию:

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;

}

### Обсуждение

Основное различие между контекстом http и stream заключается в том, что они работают на разных уровнях модели OSI. Контекст http работает на прикладном уровне, 7, а stream работает на транспортном уровне, 4. Это не означает, что контекст stream не может стать зависимым от приложения с помощью какого-либо умного сценария, но что контекст http специально разработан для полного понимания протокола HTTP и контекста stream посредством по умолчанию просто маршрутизирует пакеты и балансирует нагрузку.

Балансировка нагрузки TCP определяется потоковым модулем NGINX. Модуль stream, как и модуль HTTP, позволяет вам определять вышестоящие пулы серверов и настраивать сервер прослушивания. При настройке сервера для прослушивания данного порта необходимо определить порт, который он будет прослушивать, или, необязательно, адрес и порт. Оттуда необходимо настроить пункт назначения, будь то прямой обратный прокси-сервер на другой адрес или вышестоящий пул ресурсов.

Для настройки доступен ряд опций, изменяющих свойства обратного прокси-сервера TCP-соединения. Некоторые из них включают ограничения на проверку SSL/TLS, тайм-ауты и сохранение времени ожидания. Некоторые из значений этих параметров прокси-сервера могут быть (или содержать) переменными, такими как скорость загрузки и имя, используемое для проверки сертификата SSL/TLS.

Восходящий поток для балансировки нагрузки TCP во многом похож на восходящий поток для HTTP, поскольку он определяет исходящие ресурсы как серверы, настроенные с использованием сокета Unix, IP или полного доменного имени, а также вес сервера, максимальное количество подключений, средства распознавания DNS, периоды нарастания подключения и, если сервер активен, выключен или находится в режиме резервного копирования.

NGINX Plus предлагает еще больше возможностей для балансировки нагрузки TCP. Эти расширенные функции можно найти на протяжении всей этой книги. Проверки работоспособности для всей балансировки нагрузки будут рассмотрены далее в этой главе.

## UDP Балансированная загрузка

### Проблема

### Вам необходимо распределить нагрузку между двумя или более UDP-серверами.

### Решение

Используйте NGINX’s stream модуль для балансировки нагрузки на UDP-серверы с использованием восходящего

блока, определенного как udp:

stream {

upstream ntp {

server ntp1.example.com:123 weight=2; server ntp2.example.com:123;

}

server {

listen 123 udp; proxy\_pass ntp;

}

}

Этот раздел конфигурации балансирует нагрузку между двумя вышестоящими серверами network time protocol (NTP), использующими протокол UDP. Указать балансировку нагрузки UDP так же просто, как использовать параметр udp в директиве listen.

Если служба, с помощью которой вы выполняете балансировку нагрузки, требует отправки нескольких пакетов туда и обратно между клиентом и сервером, вы можете указать параметр reuseport. Примерами таких типов сервисов являются OpenVPN, передача голоса по интернет-протоколу (VoIP), решения для виртуальных рабочих столов и безопасность транспортного уровня дейтаграмм (DTLS). Ниже приведен пример использования NGINX для обработки подключений OpenVPN и прокси-передачи их службе OpenVPN, запущенной локально:

stream {

server {

listen 1195 udp reuseport; proxy\_pass 127.0.0.1:1194;

}

}

### Обсуждение

Вы можете спросить: “Зачем мне нужен балансировщик нагрузки, когда я могу иметь несколько хостов в записи DNS A или service record (запись SRV)?” Ответ заключается в том, что существуют не только альтернативные алгоритмы балансировки, с помощью которых мы можем балансировать, но и мы можем балансировать нагрузку на сами DNS-серверы. Сервисы UDP составляют большую часть сервисов, от которых мы зависим в сетевых системах, таких как DNS, NTP, QUIC, HTTP/3 и VoIP. Балансировка нагрузки UDP может быть менее распространена для некоторых, но она столь же полезна в мире масштабирования.

Вы можете найти балансировку нагрузки UDP в stream модуле, точно так же, как TCP, и настраивайте его в основном таким же образом. Основное отличие заключается в том, что директива listen указывает, что открытый сокет предназначен для работы с дейтаграммами. При работе с дейтаграммами существуют некоторые другие директивы, которые могут применяться там, где их не было бы в TCP, такие как директива proxy\_response, которая указывает NGINX, сколько ожидаемых ответов может быть отправлено с вышестоящего сервера. По умолчанию это значение не ограничено до тех пор, пока не будет достигнуто ограничение proxy\_timeout. Директива proxy\_timeout устанавливает время между двумя последовательными операциями чтения или записи в клиентских или прокси-серверных соединениях перед закрытием соединения.

Параметр reuseport указывает NGINX создать отдельный прослушивающий сокет для каждого рабочего процесса. Это позволяет ядру распределять входящие соединения между рабочими процессами для обработки нескольких пакетов, отправляемых между клиентом и сервером. Функция повторного использования работает только на ядрах Linux версии 3.9 и выше, DragonFly BSD и FreeBSD 12 и выше.

Load-Balancing Methods

### Проблема

### Циклическая балансировка нагрузки не подходит для вашего варианта использования, поскольку у вас разнородные рабочие нагрузки или пулы серверов.

### Решение

Используйте один из методов балансировки нагрузки NGINX, таких как наименьшее количество подключений, наименьшее время, общий хэш, случайный или IP-хэш. В этом примере алгоритм балансировки нагрузки для восходящего пула серверной части задает наименьшее количество подключений:

upstream backend { least\_conn;

server backend.example.com;

server backend1.example.com;

}

### Все алгоритмы балансировки нагрузки, за исключением generic hash, random и least time, являются автономными директивами, такими как предыдущий пример. Параметры этих директив объясняются в следующем обсуждении.

### Обсуждение

Не все запросы или пакеты имеют одинаковый вес. Учитывая это, циклический поиск или даже взвешенный циклический поиск, использованный в предыдущих примерах, не будет соответствовать потребностям всех приложений или потоку трафика. NGINX предоставляет ряд алгоритмов балансировки нагрузки, которые вы можете использовать в соответствии с конкретными вариантами использования. В дополнение к возможности выбора этих алгоритмов или методов балансировки нагрузки, вы также можете настроить их. Следующие методы балансировки нагрузки доступны для вышестоящих пулов HTTP, TCP и UDP:

*Round robin*

Это метод балансировки нагрузки по умолчанию, который распределяет запросы в порядке списка серверов в вышестоящем пуле. Вы также можете учитывать вес для взвешенного циклического анализа, который вы можете использовать, если пропускная способность вышестоящих серверов различается. Чем выше целочисленное значение веса, тем более предпочтительным будет сервер в циклическом режиме. Алгоритм, лежащий в основе веса, - это просто статистическая вероятность средневзвешенного значения.*Least connections*

Этот метод балансирует нагрузку путем проксирования текущего запроса на вышестоящий сервер с наименьшим количеством открытых подключений. Наименьшее количество подключений, как и циклический поиск, также учитывает весовые коэффициенты при принятии решения о том, на какой сервер отправлять соединение. Имя директивы - это least\_conn.

*Least time*

Доступный только в NGINX Plus, параметр "Наименьшее время" сродни "наименьшему количеству подключений" в том смысле, что он проксируется к вышестоящему серверу с наименьшим количеством текущих подключений, но отдает предпочтение серверам с наименьшим средним временем отклика. Этот метод является одним из самых сложных алгоритмов балансировки нагрузки и соответствует потребностям высокопроизводительных веб-приложений. Этот алгоритм повышает эффективность при наименьшем количестве подключений, поскольку небольшое количество подключений не обязательно означает самый быстрый отклик. При использовании этого алгоритма важно принимать во внимание статистическую дисперсию времени запроса услуг. Некоторые запросы, естественно, могут требовать большей обработки и, следовательно, иметь более длительное время запроса, увеличивая диапазон статистики. Длительное время выполнения запросов не всегда означает меньшую производительность или перегруженность сервера. Однако запросы, требующие дополнительной обработки, могут быть кандидатами для асинхронных рабочих процессов. Параметр, определяющий header или last\_byte должно быть указано для этой директивы. Когда header указан заголовок, то

используется время для получения заголовка ответа. Когда last\_byte указано, используется время для получения полного ответа. Имя директивы - это least\_time.

*Generic hash*

Администратор определяет хэш с заданным текстом, переменными запроса или среды выполнения, или и тем, и другим. NGINX распределяет нагрузку между серверами, создавая хэш для текущего запроса и размещая его на вышестоящих серверах. Этот метод очень полезен, когда вам нужно больше контролировать, куда отправляются запросы, или для определения того, на каком вышестоящем сервере, скорее всего, будут кэшироваться данные. Обратите внимание, что при добавлении или удалении сервера из пула хэшированные запросы будут перераспределены. Этот алгоритм имеет необязательный параметр, consistent, чтобы свести к минимуму эффект перераспределения. Имя директивы - это hash.

*Random*

Этот метод используется для указания NGINX выбрать случайный сервер из группы, принимая во внимание вес сервера. Необязательный параметр two [method] предписывает NGINX случайным образом выбрать два сервера, а затем использовать предоставленный метод балансировки нагрузки для балансировки между этими двумя. По умолчанию least\_conn Использует метод если two передается без метода. Имя директивы для случайной балансировки нагрузки таково - random.

*IP hash*

This Этот метод работает только для HTTP. IP-хэш использует IP-адрес клиента в качестве хэша. Немного отличающийся от использования удаленной переменной в универсальном хэше, этот алгоритм использует первые три октета IPv4-адреса или весь IPv6-адрес. Этот метод гарантирует, что клиенты подключаются к одному и тому же вышестоящему серверу до тех пор, пока этот сервер доступен, что чрезвычайно полезно, когда состояние сеанса вызывает беспокойство и не обрабатывается общей памятью приложения. Этот метод также учитывает параметр weight при распределении хэша. Имя директивы - ip\_hash.

Sticky Cookie with NGINX Plus

### Проблема

### Вам нужно привязать нисходящий клиент к восходящему серверу с помощью NGINX Plus.

### Решение

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

upstream backend {

server backend1.example.com; server backend2.example.com; sticky cookie

affinity expires=1h

domain=.example.com httponly

secure path=/;

}

### Эта конфигурация создает и отслеживает файл cookie, который связывает нисходящий клиент с вышестоящим сервером. В этом примере файл cookie называется affinity и установлен для example.com , истекает через час, не может быть использован на стороне клиента, может быть отправлен только по протоколу HTTPS и действителен для всех путей.

### Обсуждение

Использование параметра cookie в директиве sticky создает файл cookie по первому запросу, который содержит информацию о вышестоящем сервере. NGINX Plus отслеживает этот файл cookie, позволяя ему продолжать направлять последующие запросы на тот же сервер. Первым позиционным параметром параметра cookie является имя файла cookie, который будет создан и отслеживаться. Другие параметры обеспечивают дополнительный контроль, информируя браузер о соответствующем использовании — например, о времени истечения срока действия, домене, пути и о том, может ли файл cookie использоваться на стороне клиента или может быть передан по незащищенным протоколам.

Sticky Learn with NGINX Plus

### Проблема

### Вам нужно привязать нисходящий клиент к вышестоящему серверу, используя существующий файл cookie с помощью NGINX Plus.

### Solution

Используйте sticky learn директиву для обнаружения и отслеживания файлов cookie, которые создаются вышестоящим 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 инструктирует искать и отслеживать сеансы, ища файл cookie с именем COOKIENAME в заголовках ответов и просматривая существующие сеансы, ища тот же файл cookie в заголовках запросов. Эта привязка к сеансу хранится в зоне общей памяти объемом 2 МБ, которая позволяет отслеживать приблизительно 16 000 сеансов. Название файла cookie всегда будет зависеть от конкретного приложения. Обычно используемые имена файлов cookie, такие как jsessionid или phpsessionid, обычно устанавливаются по умолчанию в приложении или конфигурации сервера приложений.

### Обсуждение

Когда приложения создают свои собственные файлы cookie состояния сеанса, NGINX Plus может обнаруживать их в ответах на запросы и отслеживать их. Этот тип отслеживания файлов cookie выполняется при наличии директивы sticky: параметр learn. Общая память для отслеживания файлов cookie задается параметром zone с указанием имени и размера. NGINX Plus получает указание искать файлы cookie в ответе от вышестоящего сервера с помощью указания параметра create и выполняет поиск ранее зарегистрированного соответствия сервера с помощью параметра lookup. Значения этих параметров являются переменными, предоставляемыми модулем HTTP.

Sticky Routing with NGINX Plus

### Проблема

### Вам нужен детальный контроль над тем, как ваши постоянные сеансы перенаправляются на вышестоящий сервер с помощью NGINX Plus.

### Решение

Используйте директиву sticky с параметром route, чтобы использовать переменные о запросе для маршрутизации:

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 предпринята попытка извлечь идентификатор сеанса Java, во-первых, из файла cookie, сопоставив значение файла cookie идентификатора сеанса Java с переменной с помощью первого блока map, а во-вторых, просмотрев URI запроса для параметра с именем jsessionid, сопоставив значение переменной с использованием второго блока map. Директиве sticky с параметром route передается любое количество переменных. Для маршрута используется первое ненулевое или непустое значение. Если используется файл cookie jsessionid, запрос направляется на серверную часть 1; если используется параметр URL, запрос направляется на серверную часть 2. Хотя этот пример основан на общем идентификаторе сеанса Java, то же самое относится и к другим технологиям сеанса, таким как phpsessionid, или к любому гарантированному уникальному идентификатору, который ваше приложение генерирует для идентификатора сеанса.

### Обсуждение

Иногда, используя немного более детализированный контроль, вы можете захотеть перенаправить трафик на определенный сервер. Параметр route для директивы sticky создан для достижения этой цели. Sticky route обеспечивает вам лучший контроль, фактическое отслеживание и липкость, в отличие от общего алгоритма балансировки нагрузки хэша. Клиент сначала перенаправляется на вышестоящий сервер на основе указанного маршрута, а затем последующие запросы будут содержать информацию о маршруте в файле cookie или URI. Sticky route принимает ряд позиционных параметров, которые вычисляются. Первая непустая переменная используется для маршрутизации к серверу. Блоки карты можно использовать для выборочного анализа переменных и сохранения их как других переменных, которые будут использоваться в маршруте. По сути, директива sticky route создает сеанс в зоне общей памяти NGINX Plus для отслеживания любого идентификатора сеанса клиента, который вы указываете вышестоящему серверу, последовательно доставляя запросы с этим идентификатором сеанса на тот же вышестоящий сервер, что и его исходный запрос.

### 2.1 Подключение Drain с помощью NGINX PlusProblem

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

### Решение

Используйте параметр drain через API NGINX Plus, более подробно описанный в главе 5, чтобы дать указание NGINX прекратить отправку новых подключений, которые еще не отслеживаются:

$ curl -X POST -d '{"drain":true}' \ ['http://nginx.local/api/3/http/upstreams/backend/servers/0'](http://nginx.local/api/3/http/upstreams/backend/servers/0%27)

{

**"id"**:0, **"server"**:"172.17.0.3:80",

* 1. **Connection Draining with NGINX Plus | 19**

**"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 сохраняется локально на сервере, соединения и постоянные сеансы должны быть отключены до удаления сервера из пула. Удаление подключений - это процесс, при котором срок действия сеансов на сервере изначально истекает перед удалением сервера из вышестоящего пула. Вы можете настроить слив для конкретного сервера, добавив параметр drain в директиву сервера. Когда установлен параметр drain, NGINX Plus прекращает отправку новых сеансов на этот сервер, но позволяет текущим сеансам продолжать обслуживаться в течение всего их сеанса. Вы также можете переключить эту конфигурацию, добавив параметр drain в директиву вышестоящего сервера, а затем перезагрузив конфигурацию NGINX Plus.

### 2.9 Пассивные проверки работоспособности

### Проблема

### Вам необходимо пассивно проверять работоспособность вышестоящих серверов.

### Решение

Используйте проверки работоспособности NGINX с балансировкой нагрузки, чтобы гарантировать, что используются только исправные 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;

}

Эта конфигурация пассивно отслеживает работоспособность вышестоящего сервера, отслеживая ответы клиентских запросов, направленных вышестоящему серверу. В примере для директивы max\_fails задано значение три, а для fail\_timeout - 3 секунды. Эти параметры директивы работают одинаково как на потоковых, так и на HTTP-серверах.

### Обсуждение

### Пассивная проверка работоспособности доступна в версии NGINX с открытым исходным кодом и настраивается с использованием тех же параметров сервера для балансировки нагрузки HTTP, TCP и UDP. Пассивный мониторинг отслеживает сбойные соединения или время ожидания, когда они проходят через NGINX по запросу клиента. Пассивные проверки работоспособности включены по умолчанию; упомянутые здесь параметры позволяют вам настроить их поведение. Значение max\_fails по умолчанию равно 1, а значение fail\_timeout по умолчанию равно 10 секундам. Мониторинг работоспособности важен при всех типах балансировки нагрузки не только с точки зрения удобства работы пользователей, но и для обеспечения непрерывности бизнеса. NGINX пассивно отслеживает вышестоящие серверы HTTP, TCP и UDP, чтобы убедиться в их исправности и производительности.

Active Health Checks with NGINX Plus

### Проблема

### Вам необходимо активно проверять работоспособность ваших вышестоящих серверов с помощью NGINX Plus.

### Решение

Для 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!";

}

}

* 1. **Active Health Checks with NGINX Plus | 21**

Эта конфигурация проверки работоспособности для HTTP-серверов проверяет работоспособность вышестоящих серверов, отправляя HTTP-запрос GET к URI “/” каждые 2 секунды. Метод HTTP не может быть определен для проверки работоспособности; выполняются только запросы GET, так как другие методы могут изменять состояние серверных систем. Вышестоящие серверы должны пройти пять последовательных проверок работоспособности, чтобы считаться исправными. Они считаются нездоровыми, если не проходят две проверки подряд. Ответ от вышестоящего сервера должен соответствовать определенному блоку сопоставления, который определяет код состояния как 200, значение типа содержимого заголовка как 'text/html' и строку "Добро пожаловать в nginx!" в теле ответа. Блок HTTP match содержит три директивы: status, header и body. Все три из этих директив также имеют флаги сравнения.

Потоковые проверки работоспособности служб TCP/UDP очень похожи:

stream {

# ...

server {

listen 1234;

proxy\_pass stream\_backend; health\_check interval=10s

passes=2 fails=3;

health\_check\_timeout 5s;

}

# ...

}

### В этом примере TCP-сервер настроен на прослушивание порта 1234 и прокси-соединение с вышестоящим набором серверов, для которых он активно проверяет работоспособность. Директива stream health\_check принимает все те же параметры, что и в HTTP, за исключением uri, а в версии stream есть параметр для переключения протокола проверки на udp. В этом примере интервал установлен равным 10 секундам, требуется два прохода, чтобы считаться исправными, и требуется три неудачных, чтобы считаться неработоспособными. Проверка работоспособности активного потока также позволяет проверить ответ от вышестоящего сервера. Однако блок сопоставления для потоковых серверов содержит всего две директивы: send и expect. Направление отправки ‐ это необработанные данные, которые должны быть отправлены, а ожидание - это точный ответ или соответствующее регулярное выражение.

### Обсуждение

В NGINX Plus для мониторинга исходных серверов можно использовать пассивную или активную проверку работоспособности. Эти проверки работоспособности могут измерять нечто большее, чем просто код ответа. В NGINX Plus активные проверки работоспособности HTTP отслеживаются на основе ряда критериев приемлемости ответа от вышестоящего сервера. Вы можете настроить мониторинг активной проверки работоспособности для определения того, как часто проверяются вышестоящие серверы, сколько раз сервер должен пройти эту проверку, чтобы считаться работоспособным, сколько раз он может выйти из строя, прежде чем будет признан неработоспособным, и каким должен быть ожидаемый результат. Для большего

### сложная логика, директива require для блока match позволяет использовать переменные, значение которых не должно быть пустым или нулевым. Параметр match указывает на блок match, который определяет критерии приемлемости для ответа. Блок сопоставления 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 с NGINX Plus

### Проблема

### Прежде чем приступить к полной производственной загрузке, ваше приложение должно быть расширено.

### Решение

Используйте параметр slow\_start в директиве сервера, чтобы постепенно увеличивать количество подключений в течение указанного времени по мере повторного подключения сервера к вышестоящему пулу балансировки нагрузки:

upstream {

zone backend 64k;

server server1.example.com slow\_start=20s; server server2.example.com slow\_start=15s;

}

### Конфигурации серверных директив будут медленно увеличивать трафик на вышестоящие серверы после того, как они будут повторно добавлены в пул. сервер1 будет медленно увеличивать количество подключений в течение 20 секунд, а сервер2 - в течение 15 секунд.

### Обсуждение

*Медленный запуск - это концепция медленного увеличения количества запросов, передаваемых через прокси на сервер в течение определенного периода времени. Медленный запуск позволяет приложению разогреться, заполняя кэши, инициируя подключения к базе данных, не перегружаясь подключениями сразу после запуска. Эта функция вступает в силу, когда сервер, который не прошел проверку работоспособности, начинает проходить снова и повторно попадает в пул балансировки нагрузки, и доступна только в NGINX Plus. Медленный запуск нельзя использовать с методами hash, ip\_hash или случайной балансировки нагрузки*

**CHAPTER 3**

**Управление дорожным движением**

**3.0 Введение**

## NGINX и NGINX Plus также классифицируются как контроллеры веб-трафика. Вы можете использовать NGINX для разумной маршрутизации трафика и управления потоком на основе многих атрибутов. В этой главе описывается способность NGINX разделять клиентские запросы на основе процентного соотношения; использовать географическое местоположение клиентов; и управлять потоком трафика в виде скорости, соединения и ограничения полосы пропускания. Читая эту главу, имейте в виду, что вы можете смешивать и сочетать эти функции, открывая бесчисленные возможности.

## A/B Тестирование

### Проблема

### Вам необходимо разделить клиентов между двумя или более версиями файла или приложения, чтобы протестировать принятие или взаимодействие.

### Решение

Используйте модуль split\_clients, чтобы перенаправить определенный процент ваших клиентов в другой восходящий пул:

split\_clients "${remote\_addr}AAA" $variant { 20.0% "backendv2";

\* "backendv1";

}

Директива split\_clients хэширует строку, предоставленную вами в качестве первого параметра, и делит этот хэш на указанные проценты, чтобы сопоставить значение переменной, предоставленной в качестве второго параметра. Добавление “AAA” к первому параметру является

чтобы продемонстрировать, что это объединенная строка, которая может включать в себя множество переменных, как упомянуто в универсальном алгоритме балансировки нагрузки хэша. Третий параметр - это объект, содержащий пары ключ-значение, где ключ - это процентный вес, а значение - это значение, которое должно быть присвоено. Ключом может быть либо процент, либо звездочка. Звездочкой обозначена остальная часть целого после того, как будут взяты все проценты. Значением переменной $variant будет backend v2 для 20% клиентских IP-адресов и backend v1 для оставшихся 80%.

В этом примере серверная часть v1 и серверная часть v2 представляют пулы вышестоящих серверов и могут использоваться с директивой proxy\_pass как таковой:

location / {

proxy\_pass http://$variant

}

Используя переменную $variant, наш трафик будет распределен между двумя разными пулами серверов приложений.

Чтобы продемонстрировать широкий спектр возможных применений split\_clients, ниже приведен пример разделения между двумя версиями статического сайта:

http {

split\_clients "${remote\_addr}" $site\_root\_folder { 33.3% "/var/www/sitev2/";

\* "/var/www/sitev1/";

}

server {

listen 80 \_;

root $site\_root\_folder; location / {

index index.html;

}

}

}

### Обсуждение

Этот тип A/B‐тестирования полезен при тестировании различных типов маркетинговых и интерфейсных функций для определения коэффициента конверсии на сайтах электронной коммерции. Обычно приложения используют тип развертывания, называемый canary release. При таком типе развертывания трафик медленно переключается на новую версию путем постепенного увеличения процента пользователей, перенаправляемых на новую версию. Разделение ваших клиентов между разными версиями вашего приложения может быть полезно при развертывании новых версий кода, чтобы ограничить радиус действия в случае ошибки. Еще более распространенным является сине-зеленый стиль развертывания, при котором пользователи переходят на новую версию, а старая версия по-прежнему доступна, пока выполняется проверка развертывания. Какой бы ни была причина разделения клиентов между двумя разными наборами приложений, NGINX упрощает это благодаря модулю split\_clients.

### Использование модуля GeoIP и базы данных

### Проблема

### Вам необходимо установить базу данных GeoIP и включить ее встроенные переменные в NGINX, чтобы использовать физическое местоположение ваших клиентов в журнале NGINX, прокси-запросах или маршрутизации запросов.

### Решение

Официальный репозиторий пакетов с открытым исходным кодом NGINX, настроенный в главе 2 при установке NGINX, предоставляет пакет с именем nginx-module-geoip. При использовании репозитория пакетов NGINX Plus этот пакет называется nginx-plus-module- geoip. В следующих примерах показано, как установить пакет динамического модуля NGINX GeoIP, а также как загрузить базы данных стран и городов GeoIP:

RHEL/CentOS NGINX с открытым исходным кодом:

# yum install nginx-module-geoip

Debian/Ubuntu NGINX Open Source:

# 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

Установите the GeoIP базы данных по странам и городам и распакуйте их:

# mkdir /etc/nginx/geoip # cd /etc/nginx/geoip

# wget ["http://geolite.maxmind.com/\](http://geolite.maxmind.com/\) download/geoip/database/GeoLiteCountry/GeoIP.dat.gz" # gunzip GeoIP.dat.gz

# wget ["http://geolite.maxmind.com/\](http://geolite.maxmind.com/\) download/geoip/database/GeoLiteCity.dat.gz" # gunzip GeoLiteCity.dat.gz

Этот набор команд создает каталог geoip в каталоге /etc/nginx, перемещается в этот новый каталог и загружает и распаковывает пакеты.

Имея базу данных GeoIP для стран и городов на локальном диске, теперь вы можете дать указание модулю NGINX GeoIP использовать их для предоставления встроенных 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. Директива load\_module действительна только в основном контексте. Директива geoip\_country указывает путь к файлу GeoIP.dat, содержащему базу данных, сопоставляющую IP-адреса с кодами стран, и действительна только в контексте http.

### Обсуждение

Чтобы использовать эту функциональность, у вас должен быть установлен модуль NGINX GeoIP и локальная база данных стран и городов GeoIP. Установка и извлечение этих предварительных условий были продемонстрированы в этом разделе.

Директивы geoip\_country и geoip\_city предоставляют ряд встроенных переменных, доступных в этом модуле. Директива geoip\_country включает переменные, которые позволяют вам различать страну происхождения вашего клиента. Эти переменные включают в себя $geoip\_country\_code, $geoip\_country\_code3 и $geoip\_country\_name. Переменная country code возвращает двухбуквенный код страны, а переменная с цифрой 3 в конце возвращает трехбуквенный код страны. Переменная country name возвращает полное название страны.

Директива geoip\_city позволяет использовать довольно много переменных. Директива geoip\_city включает 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 $geoip\_city, $geoip\_latitude, $geoip\_longitude, $geoip\_city\_continent\_code и $geoip\_postal\_code, все из которых описывают возвращаемое ими значение. $geoip\_region и

$geoip\_region\_name описывает регион, территорию, штат, провинцию, федеральную землю и тому подобное. Регион - это двухбуквенный код, где название региона - это полное название.

$geoip\_area\_code, действительный только в США, возвращает трехзначный телефонный код региона.

С помощью этих переменных вы можете регистрировать информацию о вашем клиенте. При желании вы могли бы передать эту информацию своему приложению в виде заголовка или переменной или использовать NGINX для маршрутизации вашего трафика определенным образом.

### Ограничение доступа в зависимости от страны

### Проблема

### Вам необходимо ограничить доступ из определенных стран в соответствии с требованиями контракта или приложения.

### Решение

Сопоставьте коды стран, которые вы хотите заблокировать или разрешить, с переменной:

load\_module "/usr/lib64/nginx/modules/ngx\_http\_geoip\_module.so";

http {

map $geoip\_country\_code $country\_access { "US" 0;

default 1;

}

# ...

}

Это сопоставление установит новой переменной $country\_access значение 1 или 0. Если IP-адрес клиента исходит из США, переменной будет присвоено значение 0. Для любой другой страны переменной будет присвоено значение 1.

Теперь, в рамках нашего серверного блока, мы будем использовать оператор if, чтобы запретить доступ любому лицу, не являющемуся гражданином США:

server {

if ($country\_access = '1') { return 403;

}

# ...

}

### Этому оператору if будет присвоено значение True, если переменной $country\_access присвоено значение 1. При значении True сервер вернет 403 неавторизованный запрос. В противном случае сервер работает в обычном режиме. Так что этот блок if существует только для того, чтобы отказать людям, которые не из США.

### Обсуждение

Это короткий, но простой пример того, как разрешить доступ только из нескольких стран. Этот пример можно пояснить в соответствии с вашими потребностями. Вы можете использовать

* 1. **Restricting Access Based on Country | 29**

эта же практика позволяет разрешать или блокировать на основе любой из встроенных переменных, доступных из модуля GeoIP.

Finding the Original Client

### Проблема

### Вам нужно найти исходный IP-адрес клиента, потому что перед сервером NGINX есть прокси-серверы.

### Решение

Используйте директиву geoip\_proxy для определения диапазона IP-адресов вашего 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 междоменной маршрутизации (CIDR), в котором находятся наши прокси-серверы, и предписывает NGINX использовать заголовок X-Forwarded-For для поиска IP-адреса клиента. Директива geoip\_proxy\_recursive предписывает NGINX рекурсивно просматривать заголовок X-Forwarded-For для последнего известного IP-адреса клиента.

 Заголовок с именем Forwarded стал стандартным заголовком для добавления информации о прокси-сервере для проксируемых запросов. Заголовок, используемый модулем NGINX GeoIP, является X-Forwarded-For и не может быть истолкован иначе на момент написания статьи. Хотя X-Forwarded-For не является официальным стандартом, он по-прежнему очень широко используется, принимается и устанавливается большинством прокси-серверов.

### Обсуждение

Вы можете обнаружить, что если вы используете прокси-сервер перед NGINX, NGINX будет получать IP-адрес прокси-сервера, а не клиента. Для этого вы можете использовать директиву geoip\_proxy, чтобы указать NGINX использовать заголовок X-Forwarded-For при открытии соединений из заданного диапазона. Директива geoip\_proxy принимает адрес или диапазон CIDR. Когда есть несколько прокси-серверов, передающих трафик перед NGINX,

вы можете использовать директиву geoip\_proxy\_recursive для рекурсивного поиска по адресам X-Forwarded-For, чтобы найти исходящий клиент. Вы захотите использовать что-то подобное при использовании средств балансировки нагрузки, таких как Amazon Web Services Elastic Load Balancing (AWS ELB), балансировщик нагрузки Google или балансировщик нагрузки Microsoft Azure перед NGINX.

Limiting Connections

### Проблема

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

### Решение

Создайте зону общей памяти для хранения показателей подключений и используйте директиву limit\_conn для ограничения открытых подключений:

http {

limit\_conn\_zone $binary\_remote\_addr zone=limitbyaddr:10m; limit\_conn\_status 429;

# ...

server {

# ...

limit\_conn limitbyaddr 40; # ...

}

}

### Эта конфигурация создает зону общей памяти с именем limit by addr. Используемый заранее определенный ключ ‐ это IP-адрес клиента в двоичной форме. Размер зоны общей памяти установлен равным 10 МБ. Директива limit\_conn принимает два параметра: имя зоны limit\_conn\_zone и количество разрешенных подключений. Параметр limit\_conn\_status устанавливает ответ, когда количество подключений ограничено 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 подключений на основе ключа может быть использовано для защиты от злоупотреблений и справедливого распределения ваших ресурсов между всеми вашими клиентами. Важно быть осторожным с вашим предопределенным ключом. Использование IP-адреса, как в предыдущем примере, может быть опасным, если многие пользователи находятся в одной сети, исходящей с одного и того же IP-адреса, например, при преобразовании сетевых адресов (NAT). Вся группа клиентов будет ограничена. Директива limit\_conn\_zone действительна только в

http-контекст. Вы можете использовать любое количество переменных, доступных NGINX в контексте http, чтобы создать строку для ограничения. Использование переменной, которая может идентифицировать пользователя на уровне приложения, такой как файл cookie сеанса, может быть более чистым решением в зависимости от варианта использования. Значение limit\_conn\_status по умолчанию равно 503, служба недоступна. Возможно, вам покажется предпочтительным использовать 429, поскольку услуга доступна, а ответы уровня 500 указывают на ошибку сервера, тогда как ответы уровня 400 указывают на ошибку клиента.

Тестирование ограничений может быть сложным делом. Часто бывает трудно смоделировать реальный трафик в альтернативной среде для тестирования. В этом случае вы можете установить директиву limit\_req\_dry\_run в значение on, а затем использовать переменную $limit\_req\_status в вашем журнале доступа. Переменная $limit\_req\_sta tus будет иметь значение либо ПЕРЕДАНО, ОТЛОЖЕНО, ОТКЛОНЕНО, DELAYED\_DRY\_RUN, либо REJECTED\_DRY\_RUN. Включив "сухой запуск", вы сможете анализировать журналы текущего трафика и при необходимости корректировать свои лимиты перед включением, что гарантирует правильность настройки лимитов.

Limiting Rate

### Проблема

### Вам необходимо ограничить частоту запросов предопределенным ключом, таким как IP-адрес клиента.

### Решение

Используйте модуль ограничения скорости для ограничения частоты запросов:

http {

limit\_req\_zone $binary\_remote\_addr zone=limitbyaddr:10m rate=3r/s;

limit\_req\_status 429; # ...

server {

# ...

limit\_req zone=limitbyaddr; # ...

}

}

В этом примере конфигурации создается зона общей памяти с именем limit by addr. Используемый предопределенный ключ - это IP-адрес клиента в двоичной форме. Размер зоны общей памяти установлен равным 10 МБ. Зона устанавливает скорость с помощью аргумента ключевого слова. Директива limit\_req принимает обязательный аргумент ключевого слова: zone. zone указывает директиву, в отношении которой следует использовать зону ограничения запросов к общей памяти. Запросы, превышающие выраженния скорости возвращается HTTP-кодом 429, как определено limit\_req\_

status статусе. Я советую установить статус в диапазоне 400 уровней, так как по умолчанию используется значение 503, что подразумевает проблему с сервером, в то время как на самом деле проблема связана с клиентом.

Используйте необязательные аргументы ключевого слова в директиве limit\_req, чтобы включить двухэтапное ограничение скорости:

server {

location / {

limit\_req zone=limitbyaddr burst=12 delay=9;

}

}

### В некоторых случаях клиенту нужно будет сделать много запросов одновременно, а затем на некоторое время снизить свою ставку, прежде чем делать больше. Вы можете использовать ключевое слово argument burst, чтобы позволить клиенту превысить свой лимит скорости, но не отклонять запросы. Запросы с превышенной скоростью будут обрабатываться с задержкой, чтобы привести ограничение скорости в соответствие с настроенным значением. Набор аргументов ключевого слова изменяет это поведение: delay и nodelay. Аргумент nodelay не принимает значения и просто позволяет клиенту использовать расширяемое значение сразу; однако все запросы будут отклоняться до тех пор, пока не пройдет достаточно времени, чтобы удовлетворить лимит скорости. В этом примере, если бы мы использовали nodelay, клиент мог бы обработать 12 запросов в первую секунду, но ему пришлось бы ждать 4 секунды после первоначального запроса, чтобы выполнить другой. Аргумент ключевого слова delay определяет, сколько запросов может быть выполнено заранее без регулирования. В этом случае клиент может сделать девять запросов заранее без задержки, следующие три будут заблокированы, а все остальные в течение 4 секунд будут отклонены.

### Обсуждение

Модуль ограничения скорости является очень мощным средством защиты от неправомерных быстрых запросов, обеспечивая при этом качественное обслуживание для всех. Существует много причин для ограничения частоты запросов, одной из которых является безопасность. Вы можете предотвратить атаку методом грубой силы, установив очень строгое ограничение на своей странице входа в систему. Вы можете установить разумный лимит на все запросы, тем самым отключив планы злоумышленников, которые могут попытаться отказать в обслуживании вашего приложения или растратить ресурсы впустую. Конфигурация модуля ограничения скорости во многом похожа на модуль ограничения подключения, описанный в рецепте 3.5, и применяются многие из тех же требований. Вы можете указать скорость, с которой количество запросов ограничено в запросах в секунду или запросах в минуту. Когда лимит скорости достигнут, инцидент регистрируется. Также есть директива, отсутствующая в примере, limit\_req\_log\_level, которая по умолчанию имеет значение error, но может быть установлена в info, notice или warn. В NGINX Plus ограничение скорости теперь зависит от кластера (пример зональной синхронизации смотрите в рецепте 12.5).

Тестирование ограничений может быть сложным делом. Часто бывает трудно смоделировать живой трафик в альтернативной среде для тестирования. В этом случае вы можете установить директиву limit\_req\_dry\_run в значение on, а затем использовать переменную $limit\_req\_status в вашем журнале доступа. Переменная $limit\_req\_status будет иметь значение ПЕРЕДАНО, ОТКЛОНЕНО или REJECTED\_DRY\_RUN. С сухим ходом

включив его, вы сможете анализировать журналы текущего трафика и при необходимости корректировать свои лимиты перед включением, что обеспечит вам уверенность в правильности настройки лимита.

Limiting Bandwidth

### Проблема

### Вам необходимо ограничить пропускную способность загрузки для каждого клиента для ваших ресурсов.

### Решение

Используйте директивы NGINX limit\_rate и limit\_rate\_after для ограничения ширины полосы пропускания ответа клиенту:

location /download/ { limit\_rate\_after 10m; limit\_rate 1m;

}

### Конфигурация этого блока определения местоположения определяет, что для URI с префиксом down‐load скорость, с которой ответ будет передаваться клиенту, будет ограничена после 10 МБ до скорости 1 МБ в секунду. Ограничение пропускной способности указано для каждого соединения, поэтому вы можете установить ограничение на подключение, а также ограничение на пропускную способность, где это применимо.

### Обсуждение

Ограничение пропускной способности для определенных подключений позволяет NGINX распределять свою пропускную способность для загрузки между всеми клиентами указанным вами способом. Эти две директивы делают все это: limit\_rate\_after и limit\_rate. Директива limit\_rate\_after может быть установлена практически в любом контексте: http, сервер, местоположение и if, когда if находится в пределах местоположения. Директива limit\_rate применима в тех же контекстах, что и limit\_rate\_after; однако в качестве альтернативы она может быть задана переменной с именем

$limit\_rate.

Директива limit\_rate\_after указывает, что скорость соединения не должна быть ограничена до тех пор, пока не будет передан указанный объем данных. Директива limit\_rate по умолчанию определяет ограничение скорости для данного контекста в байтах в секунду. Однако вы можете указать m для мегабайт или g для гигабайт. Обе директивы по умолчанию имеют значение 0. Значение 0 означает, что скорость загрузки вообще не ограничивается. Этот модуль позволяет вам программно изменять лимит тарифов для клиентов.

**CHAPTER 4**

**Масштабируемое кэширование контента**

**4.0 Введение**

## Кэширование ускоряет обслуживание контента за счет сохранения ответов на запросы для повторного обслуживания в будущем. Кэширование содержимого снижает нагрузку на вышестоящие серверы, кэшируя полный ответ вместо повторного выполнения вычислений и запросов по одному и тому же запросу. Кэширование повышает производительность и снижает нагрузку, а это означает, что вы можете работать быстрее при меньшем количестве ресурсов. Масштабирование и распределение серверов кэширования в стратегически важных местах может оказать существенное влияние на работу пользователей. Оптимально размещать контент поближе к потребителю для достижения наилучшей производительности. Вы также можете кэшировать свой контент рядом с вашими пользователями. Такова схема сетей доставки контента, или CDN. С помощью NGINX вы можете кэшировать свой контент везде, где вы можете разместить сервер NGINX, что эффективно позволяет вам создавать свой собственный CDN. С помощью кэширования NGINX вы также можете пассивно кэшировать и обслуживать кэшированные ответы в случае сбоя восходящего потока. Функции кэширования доступны только в контексте http.

### Зоны кэширования

### Проблема

### Вам нужно кэшировать содержимое и определить, где хранится кэш.

### Решение

Используйте директиву proxy\_cache\_path, чтобы определить зоны общего кэша памяти и местоположение для содержимого:

proxy\_cache\_path /var/nginx/cache

keys\_zone=CACHE:60m levels=1:2

inactive=3h max\_size=20g;

proxy\_cache CACHE;

**35**

### Пример определения кэша создает каталог для кэшированных ответов в файловой системе по адресу /var/nginx/cache и создает разделяемое пространство памяти с именем CACHE объемом 60 МБ. В этом примере задаются уровни структуры каталогов, определяется освобождение кэшированных ответов после того, как они не были запрошены в течение 3 часов, и определяется максимальный размер кэша в 20 ГБ. Директива proxy\_cache сообщает конкретному контексту об использовании зоны кэширования. proxy\_cache\_path действителен в httpcontext, а директива proxy\_cache действительна в контекстах http, сервера и местоположения.

### Обсуждение

## Чтобы настроить кэширование в NGINX, необходимо объявить путь и зону, которые будут использоваться. Зона кэширования в NGINX создается с помощью директивы proxy\_cache\_path. proxy\_cache\_path определяет местоположение для хранения кэшированной информации и разделяемое пространство памяти для хранения активных ключей и метаданных ответа. Необязательные параметры этой директивы обеспечивают больший контроль над тем, как поддерживается кэш и осуществляется доступ к нему. Параметр levels определяет, как создается файловая структура. Это значение, разделенное двоеточием, которое объявляет длину имен подкаталогов максимум с тремя уровнями. Кэши NGINX основаны на ключе кэша, который представляет собой хэшированное значение. Затем NGINX сохраняет результат в предоставленной файловой структуре, используя ключ кэша в качестве пути к файлу и разбивая каталоги на основе значения levels. Параметр inactive позволяет контролировать продолжительность времени, в течение которого элемент кэша будет размещен после его последнего использования. Размер кэша также настраивается с помощью параметра max\_size. Другие параметры относятся к процессу загрузки кэша, который загружает ключи кэша в зону общей памяти из файлов, кэшированных на диске.

### Блокировка кэша

### Проблема

### Вы же не хотите, чтобы NGINX отправлял прокси-запросы, которые в данный момент записываются в кэш на вышестоящий сервер.

### Решение

Используйте директиву proxy\_cache\_lock, чтобы гарантировать, что одновременно в кэш может быть записан только один запрос, при этом последующие запросы будут ожидать записи ответа:

proxy\_cache\_lock on; proxy\_cache\_lock\_age 10s; proxy\_cache\_lock\_timeout 3s;

### Обсуждение

## Директива proxy\_cache\_lock предписывает NGINX удерживать запросы, предназначенные для кэшированного элемента, которые в данный момент заполняются. Прокси-запрос, заполняющий кэш, ограничен количеством времени, которое у него есть до того, как другой запрос попытается заполнить элемент, определенный директивой proxy\_cache\_lock\_age, значение которой по умолчанию равно 5 секундам. NGINX также может разрешить отправку запросов, ожидавших определенное количество времени, на прокси-сервер, который не будет пытаться заполнить кэш с помощью директивы proxy\_cache\_lock\_timeout, которая также по умолчанию равна 5 секундам. Вы можете думать о proxy\_cache\_lock\_age и proxy\_cache\_lock\_timeout как “Вы занимаете слишком много времени, я заполню кэш за вас” и “Вы заставляете меня ждать слишком долго, я собираюсь получить то, что мне нужно, и позволить вам заполнить кэш в свое свободное время" соответственно.

### Кэширование хэш-ключей

### Проблема

### Вам нужно контролировать, как кэшируется и извлекается ваш контент.

### Решение

Используйте директиву proxy\_cache\_key вместе с переменными, чтобы определить, что представляет собой попадание в кэш или промах:

proxy\_cache\_key "$host$request\_uri $cookie\_user";

### Этот хэш-ключ кэша даст указание NGINX кэшировать страницы на основе запрашиваемого хоста и URI, а также файла cookie, который определяет пользователя. С помощью этого вы можете кэшировать динамические страницы, не обслуживая контент, который был сгенерирован для другого пользователя.

### Обсуждение

Ключом proxy\_cache\_key по умолчанию, который подойдет для большинства вариантов использования, является "$scheme$proxy\_host$request\_uri". Используемые переменные включают схему, HTTP или HTTPS, прокси-сервер, на который отправляется запрос, и URI запроса. В целом, это отражает URL-адрес, по которому NGINX проксирует запрос. Вы можете обнаружить, что существует множество других факторов, определяющих уникальный запрос для каждого приложения, таких как аргументы запроса, заголовки, идентификаторы сеанса и т.д., для которых вы захотите создать свой собственный хэш—ключ.1

1 Любая комбинация текста или переменных, доступных NGINX, может быть использована для формирования ключа кэша. Список переменных доступен в NGINX.

* 1. **Caching Hash Keys | 37**

Выбор хорошего хэш-ключа очень важен и должен быть продуман с учетом особенностей приложения. Выбор ключа кэша для статического содержимого, как правило, довольно прост; достаточно использовать имя хоста и URI. Выбор ключа кэширования для достаточно динамичного контента, такого как страницы для приложения dashboard, требует больше знаний о том, как пользователи взаимодействуют с приложением, и о степени различий между пользовательскими впечатлениями. Из соображений безопасности вы, возможно, не захотите передавать кэшированные данные от одного пользователя другому без полного понимания контекста. Директива proxy\_cache\_key настраивает строку, которая будет хэшироваться для ключа кэша. Ключ proxy\_cache\_key может быть установлен в контексте блоков http, сервера и местоположения, обеспечивая гибкий контроль над тем, как кэшируются запросы.

Cache Bypass

### Проблема

### Вам нужна возможность обойти кэширование.

### Решение

Используйте директиву proxy\_cache\_bypass с непустым или ненулевым значением. Один из способов сделать это динамически - установить переменной значение, отличное от пустой строки или 0, в блоке местоположения, который вы не хотите кэшировать:

proxy\_cache\_bypass $http\_cache\_bypass;

### Конфигурация сообщает NGINX об обходе кэша, если заголовку HTTP-запроса с именем cache\_bypass присвоено любое значение, отличное от 0. В этом примере заголовок используется в качестве переменной, чтобы определить, следует ли обходить кэширование — клиенту нужно будет специально задать этот заголовок для своего запроса.

### Обсуждение

Существует ряд сценариев, которые требуют, чтобы запрос не кэшировался. Для этого NGINX предоставляет директиву proxy\_cache\_bypass, так что, когда значение непустое или ненулевое, запрос будет отправлен на вышестоящий сервер, а не извлечен из кэша. Различные потребности и сценарии обхода кэша будут продиктованы вариантом использования вашего приложения. Методы обхода кэша могут быть такими же простыми, как использование заголовка запроса или ответа, или такими же сложными, как совместная работа нескольких блоков карты.

Возможно, вы захотите обойти кэш по многим причинам, таким как устранение неполадок или отладка. Воспроизведение проблем может быть затруднено, если вы постоянно извлекаете кэшированные страницы или если ваш ключ кэша привязан к идентификатору пользователя. Имея возможность обойти

кэш имеет жизненно важное значение. Опции включают, но не ограничиваются ими, обход кэша при задании определенного файла cookie, заголовка или аргумента запроса. Вы также можете отключить кэш

полностью для заданного контекста, такого как блок местоположения, отключив proxy\_cache;.

Cache Performance

### Проблема

### Вам необходимо повысить производительность за счет кэширования на стороне клиента.

### Решение

Используйте заголовки управления кэшем на стороне клиента:

location ~\* \.(css|js)$ { expires 1y;

add\_header Cache-Control "public";

}

### Этот блок расположения указывает, что клиент может кэшировать содержимое файлов CSS и Java‐Script. Директива expires указывает клиенту, что его кэшированный ресурс больше не будет действителен по истечении одного года. Директива add\_header добавляет к ответу элемент управления кэшем заголовка HTTP-ответа со значением public, что позволяет любому кэширующему серверу на этом пути кэшировать ресурс. Если мы укажем private, только клиенту будет разрешено кэшировать значение.

### Обсуждение

Производительность кэша зависит от многих факторов, в том числе от скорости диска. В конфигурации NGINX есть много вещей, которые вы можете сделать, чтобы повысить производительность кэша. Один из вариантов - установить заголовки ответа таким образом, чтобы клиент фактически кэшировал ответ и вообще не отправлял запрос в NGINX, а просто обслуживал его из своего собственного кэша.

### 4.4 Очистка кэша с помощью NGINX Plus

### Проблема

### Вам нужно сделать недействительным объект из кэша.

### Решение

Используйте функцию очистки 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 кэш для конкретного объекта будет очищен, если он запрошен с помощью метода PURGE. Ниже приведен пример curl очистки кэша файла с именем main.js:

$ curl -XPURGE localhost/main.js

### Обсуждение

Распространенным способом обработки статических файлов является добавление хэша файла в имя файла. Это гарантирует, что при развертывании нового кода и содержимого ваш CDN распознает его как новый файл, поскольку URI изменился. Однако это не совсем работает для динамического контента, для которого вы установили ключи кэша, которые не соответствуют этой модели. В каждом сценарии кэширования у вас должен быть способ очистить кэш. NGINX Plus предоставил простой метод очистки кэшированных ответов. Директива proxy\_cache\_purge, при передаче ненулевого или непустого значения, удалит кэшированные элементы, соответствующие запросу. Простой способ настроить очистку - это сопоставить метод запроса для очистки. Однако вы можете захотеть использовать это в сочетании с модулем geo\_ip или простой аутентификацией, чтобы гарантировать, что никто не сможет удалить ваши драгоценные элементы кэша. NGINX также разрешил использование \*, которое удалит элементы кэша, соответствующие общему префиксу URI. Чтобы использовать подстановочные знаки, вам нужно будет настроить вашу директиву proxy\_cache\_path с аргументом purger=on.

Cache Slicing

### Проблема

### Вам необходимо повысить эффективность кэширования, разбив файл на фрагменты.

### Решение

Используйте NGINX slice директива и ее встроенные переменные для разделения результата кэширования 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

### Эта конфигурация определяет зону кэширования и включает ее для сервера. Директива slice затем используется для указания NGINX разбить ответ на сегменты файла размером 1 МБ. Файлы кэша хранятся в соответствии с директивой proxy\_cache\_key. Обратите внимание на использование встроенной переменной с именем slice\_range. Эта же переменная используется в качестве заголовка при отправке запроса к источнику, и версия HTTP этого запроса обновлена до HTTP/1.1, поскольку 1.0 не поддерживает запросы в диапазоне байт. Срок действия кэша устанавливается для кодов ответов 200 или 206 на 1 час, а затем определяются местоположение и источники.

### Модуль Cache Slice был разработан для доставки видео HTML5, который использует запросы в байтовом диапазоне для псевдопотока контента в браузер. По умолчанию NGINX способен обслуживать запросы в диапазоне байт из своего кэша. Если для некэшированного содержимого выполняется запрос диапазона байт, NGINX запрашивает весь файл из источника. Когда вы используете модуль Cache Slice, NGINX запрашивает только необходимые сегменты из источника. Запросы диапазона, размер которых превышает размер фрагмента, включая весь файл целиком, запускают подзапросы для каждого из требуемых сегментов, а затем эти сегменты кэшируются. Когда все сегменты кэшируются, ответ собирается и отправляется клиенту, что позволяет NGINX более эффективно кэшировать и обслуживать контент, запрошенный в диапазонах.

### Модуль среза кэша следует использовать только для больших файлов, которые не изменяются. NGINX проверяет ETag каждый раз, когда получает сегмент из источника. Если ETag в источнике изменяется, NGINX прерывает заполнение кэша сегмента, поскольку кэш больше недействителен. Если содержимое действительно изменилось и файл стал меньше, или ваш origin может справиться с скачками загрузки в процессе заполнения кэша, лучше использовать модуль блокировки кэша, описанный в блоге, перечисленном в следующем разделе "Также смотрите". Этот модуль не собран по умолчанию и должен быть включен конфигурацией --with-http\_slice\_module при сборке NGINX.

**CHAPTER 5**

**Программируемость и автоматизация**

## Введение

## Программируемость относится к способности взаимодействовать с чем‐либо посредством программирования. API для NGINX Plus предоставляет именно это: возможность взаимодействовать с конфигурацией и поведением NGINX Plus через HTTP-интерфейс. Этот API предоставляет возможность перенастраивать NGINX Plus путем добавления или удаления вышестоящих серверов с помощью HTTP-запросов. Функция хранения ключей и значений в NGINX Plus обеспечивает другой уровень динамической настройки - вы можете использовать HTTP—вызовы для ввода информации, которую NGINX Plus может использовать для динамической маршрутизации трафика или управления им. В этой главе мы коснемся API NGINX Plus и модуля хранилища ключей и значений, предоставляемого этим же API.

## Инструменты управления конфигурацией автоматизируют установку и конфигурирование серверов, что является бесценной утилитой в эпоху облачных технологий. Разработчикам крупномасштабных веб-приложений больше не нужно настраивать серверы вручную; вместо этого они могут использовать один из множества доступных инструментов управления конфигурацией. С помощью этих инструментов инженерам нужно всего лишь один раз написать конфигурации и код, чтобы создать множество серверов с одинаковой конфигурацией повторяемым, тестируемым и модульным способом. В этой главе рассматриваются несколько наиболее популярных доступных инструментов управления конфигурацией и способы их использования для установки NGINX и создания шаблона базовой конфигурации. Эти примеры чрезвычайно просты, но демонстрируют, как запустить сервер NGINX на каждой платформе.

## NGINX Plus API

### Проблема

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

### Решение

Настройте API NGINX Plus, чтобы включить добавление и удаление серверов с помощью вызовов API:

upstream backend {

zone http\_backend 64k;

}

server {

# ...

location /api {

api [write=on];

# Directives limiting access to the API # See chapter 7

}

location = /dashboard.html {

root /usr/share/nginx/html;

}

}

Эта конфигурация NGINX Plus создает вышестоящий сервер с зоной общей памяти, включает API в блоке /api location и предоставляет местоположение для панели мониторинга NGINX Plus.

Вы можете 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/'](http://nginx.local/api/3/http/upstreams/backend/servers/%27)

{

**"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**

}

Вызов curl в этом примере отправляет запрос NGINX Plus на добавление нового сервера в исходную конфигурацию серверной части. HTTP-метод - это POST, в качестве тела передается объект JSON, и возвращается ответ JSON. Ответ в формате JSON показывает конфигурацию объекта сервера — обратите внимание, что был сгенерирован новый идентификатор, а для других параметров конфигурации были установлены значения по умолчанию.

API NGINX Plus является RESTful; следовательно, в URI запроса есть параметры.

Формат URI следующий:

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

Вы можете использовать API NGINX Plus для составления списка серверов в вышестоящем пуле:

$ curl ['http://nginx.local/api/3/http/upstreams/backend/servers/'](http://nginx.local/api/3/http/upstreams/backend/servers/%27) [

{

"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

}

]

Вызов curl в этом примере отправляет запрос NGINX Plus на получение списка всех серверов в вышестоящем пуле с именем backend. В настоящее время у нас есть только один сервер, который мы добавили в предыдущем вызове curl к API. Запрос вернет объект вышестоящего сервера, содержащий все настраиваемые параметры для сервера.

Используйте API NGINX Plus для удаления подключений с вышестоящего сервера, подготавливая его к плавному удалению из вышестоящего пула, как показано в следующем коде. Вы можете найти подробную информацию о сливе соединения в [Recipe 2.8](#_bookmark77).

$ curl -X PATCH -d '{"drain":true}' \ ['http://nginx.local/api/3/http/upstreams/backend/servers/0'](http://nginx.local/api/3/http/upstreams/backend/servers/0%27)

{

"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

}

В этом curl мы указываем, что метод запроса ‐ PATCH, передаем тело JSON, инструктирующее его отключать соединения для сервера, и указываем идентификатор сервера, добавляя его к URI. Мы нашли идентификатор сервера, перечислив серверы в вышестоящем пуле в предыдущей команде curl.

* 1. **NGINX Plus API | 45**

NGINX Plus начнет отключать соединения. Этот процесс может занять столько времени, сколько длится сеанс работы приложения. Чтобы проверить, сколько активных подключений обслуживается сервером, который вы начали отключать, воспользуйтесь следующим вызовом и найдите атрибут active для удаляемого сервера:

$ curl ['http://nginx.local/api/3/http/upstreams/backend'](http://nginx.local/api/3/http/upstreams/backend%27)

{

"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 завершения всех подключений используйте NGINX Plus API, чтобы полностью удалить сервер из вышестоящего пула:

$ curl -X DELETE \ ['http://nginx.local/api/3/http/upstreams/backend/servers/0'](http://nginx.local/api/3/http/upstreams/backend/servers/0%27)

[]

### Команда curl отправляет запрос метода УДАЛЕНИЯ к тому же URI, который использовался для обновления состояния сервера. Метод DELETE предписывает NGINX удалить сервер. Этот

### 

### Вызов API возвращает все серверы и их идентификаторы, которые все еще остаются в пуле. Поскольку мы начали с пустого пула, добавили только один сервер через API, осушили его, а затем удалили, теперь у нас снова пустой пул.

### Обсуждение

### Эксклюзивный API NGINX Plus позволяет динамическим серверам приложений добавлять себя в конфигурацию NGINX и удалять из нее "на лету". Когда серверы подключаются к сети, они могут зарегистрироваться в пуле, и NGINX начнет отправлять нагрузку на вновь добавленные серверы. Когда сервер необходимо удалить, сервер может запросить NGINX Plus для удаления своих подключений, а затем удалить себя из вышестоящего пула до его завершения. Это позволяет инфраструктуре, благодаря некоторой автоматизации, масштабироваться без вмешательства человека.

### 5.2 Использование хранилища ключей и значений в NGINX Plus

### Проблема

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

### Решение

В этом разделе будет использован пример динамического списка блокировок в качестве решения по управлению трафиком.

Настройте хранилище ключей и значений с поддержкой кластера и API, а затем добавьте ключи и значения:

keyval\_zone zone=blocklist:1M;

keyval $remote\_addr $blocked zone=blocklist;

server {

# ...

location / {

if ($blocked) {

return 403 'Forbidden';

}

return 200 'OK';

}

}

server {

# ...

# Directives limiting access to the API # See chapter 6

location /api { api write=on;

}

}

* 1. **Using the Key-Value Store with NGINX Plus | 47**

Эта конфигурация NGINX Plus использует каталог keyval\_zone для создания зоны общей памяти хранилища ключей и значений с именем blocklist и устанавливает ограничение памяти в 1 МБ. Затем директива keyval сопоставляет значение ключа, соответствующее первому параметру

$remote\_addr к новой переменной с именем $заблокирован из зоны. Эта новая переменная затем используется для определения того, должен ли NGINX Plus обработать запрос или вернуть запрещенный код 403.

После запуска сервера NGINX Plus с такой конфигурацией вы можете отключить локальный компьютер и ожидать получения ответа 200 OK:

$ curl ['http://127.0.0.1/'](http://127.0.0.1/%27) OK

Теперь добавьте IP-адрес локального компьютера в хранилище значений ключей со значением 1:

$ curl -X POST -d '{"127.0.0.1":"1"}' \

['http://127.0.0.1/api/3/http/keyvals/blocklist'](http://127.0.0.1/api/3/http/keyvals/blocklist%27)

Эта команда curl отправляет HTTP POST-запрос с объектом JSON, содержащим объект ключ-значение, который должен быть отправлен в зону общей памяти blocklist. URI API хранилища ключей и значений отформатирован следующим образом:

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

IP-адрес локального компьютера теперь добавлен в зону значений ключей с именем blocklist со значением 1. В следующем запросе NGINX Plus просматривает $remote\_addr в зоне ключ-значение, находит запись и сопоставляет значение переменной $blocked. Затем эта переменная вычисляется в операторе if. Когда переменная имеет значение, if принимает значение True, а NGINX Plus возвращает код возврата 403 Forbidden:$ curl ['http://127.0.0.1/'](http://127.0.0.1/%27) Forbidden

Вы можете обновить или удалить ключ, отправив запрос на метод исправления:

$ curl -X PATCH -d '{"127.0.0.1":null}' \ ['http://127.0.0.1/api/3/http/keyvals/blocklist'](http://127.0.0.1/api/3/http/keyvals/blocklist%27)

### NGINX Plus удаляет ключ, если значение равно null, и запросы снова возвращают значение 200 OK.

### Обсуждение

Хранилище ключей и значений, эксклюзивная функция NGINX Plus, позволяет приложениям вводить информацию в NGINX Plus. В приведенном примере переменная $remote\_addr используется для создания динамического списка блокировок. Вы можете заполнить хранилище ключ-значение любым

### ключ, который NGINX Plus может иметь в качестве переменной — например, файл cookie сеанса — и предоставить NGINX Plus внешнее значение. В NGINX Plus R16 хранилище значений ключей стало зависимым от кластера, что означает, что вы должны предоставить обновление значений ключей только одному серверу NGINX Plus, и все они получат информацию.

### В NGINX Plus R19 хранилище значений ключей включило параметр type, который позволяет индексировать определенные типы ключей. По умолчанию тип имеет значение string, где ip и префикс также являются параметрами. Тип string не создает индекс, и все запросы ключей должны быть точными совпадениями, в то время как prefix допускает частичные совпадения ключей при условии, что префикс ключа совпадает. Тип ip позволяет использовать обозначение CIDR. В нашем примере, если бы мы указали type=ip в качестве параметра для нашей зоны, мы могли бы предоставить весь диапазон CIDR для блокировки, например 192.168.0.0/16, чтобы заблокировать весь блок RFC 1918 private range, или 127.0.0.1/32 для localhost, что дало бы тот же эффект, что и продемонстрировано в примере.

### 5.3 Использование модуля NJS для предоставления функциональности JavaScript в NGINX

### Проблема

### Вам нужен NGINX для выполнения пользовательской логики в запросах или ответах.

### Решение

Включите использование JavaScript, установив модуль NJS для NGINX. Следующие шаги по установке пакета предполагают, что вы добавили официальные репозитории NGINX для своего дистрибутива Linux, как показано в [Chapter 1](#_bookmark2).

Debian/Ubuntu:

apt-get install nginx-module-njs

Debian/Ubuntu and NGINX Plus:

apt-get install nginx-plus-module-njs

RHEL/CentOS:

yum install nginx-module-njs

RHEL/CentOS and NGINX Plus:

yum install nginx-plus-module-njs

Если у вас еще нет каталога для файлов JavaScript в конфигурации NGINX, создайте его:

mkdir -p /etc/nginx/njs

Создайте файл JavaScript с именем /etc/nginx/njs/jwt.js со следующим содержанием:

function jwt(data) {

var parts = data.split('.').slice(0,2)

.map(v=>Buffer.from(v, 'base64url').toString())

.map(JSON.parse);

return { headers:parts[0], payload: parts[1] };

}

function jwt\_payload\_subject(r) {

return jwt(r.headersIn.Authorization.slice(7)).payload.sub;

}

function jwt\_payload\_issuer(r) {

return jwt(r.headersIn.Authorization.slice(7)).payload.iss;

}

export default {jwt\_payload\_subject, jwt\_payload\_issuer}

Приведенный пример JavaScript определяет функцию, которая декодирует веб-токены JSON (JWTS). Кроме того, определены две функции, которые используют декодер JWT для возврата определенных ключей внутри JWT. Эти функции экспортируются, чтобы стать доступными для NGINX. Эти функции возвращают общие ключи, найденные в JWT: субъект и эмитент. Часть кода .slice(7) предназначена для удаления первых семи символов значения заголовка авторизации. С появлением JWT значением типа является Bearer. Носитель состоит из 6 символов, и нам также нужно удалить разделитель пробелом, вот почему мы вырезаем первые 7 символов. Существуют некоторые службы аутентификации, которые не предоставляют тип, такие как AWS Cognito, поскольку такие службы изменяют количество фрагментов или полностью удаляют его, так что функция jwt получает только значение токена.

В рамках базовой конфигурации NGINX загрузите модуль JS. Импортируйте и используйте JavaScript в http-блоке:

load\_module /etc/nginx/modules/ngx\_http\_js\_module.so;

http {

js\_path "/etc/nginx/njs/"; js\_import main from jwt.js;

js\_set $jwt\_payload\_subject main.jwt\_payload\_subject; js\_set $jwt\_payload\_issuer main.jwt\_payload\_issuer;

...

}

Предоставленная конфигурация NGINX динамически загружает модуль JS и импортирует файл JavaScript, который мы определили ранее. Директивы NGINX используются для присвоения переменным NGINX возвращаемых значений функций JavaScript.

Используйте переменные для доказательства логики JavaScript. Определите сервер, который возвращает переменные, заданные JavaScript:

server {

listen 80 default\_server;

listen [::]:80 default\_server; server\_name \_;

location / {

return 200 "$jwt\_payload\_subject $jwt\_payload\_issuer";

}

}

Предоставленная конфигурация создаст сервер, который возвращает значения субъекта и эмитента, предоставленные клиентом через заголовок авторизации. Эти значения декодируются определенным кодом JavaScript.

Чтобы проверить, что код работает, вы сделаете запрос к серверу с заданным JWT. Ниже приведен формат JSON JWT, который вы можете использовать для проверки работоспособности кода:

{

"iss": "nginx",

"sub": "alice", "foo": 123, "bar": "qq", "zyx": false

}

Сделайте запрос к серверу с заданным JWT, чтобы убедиться, что код JavaScript выполняется и возвращает правильные значения:

curl ['http://localhost/'](http://localhost/%27) -H \

"Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1\ NiIsImV4cCI6MTU4NDcyMzA4NX0.eyJpc3MiOiJuZ2lueCIsInN1YiI6Im\ FsaWNlIiwiZm9vIjoxMjMsImJhciI6InFxIiwie\ nl4IjpmYWxzZX0.Kftl23Rvv9dIso1RuZ8uHaJ83BkKmMtTwch09rJtwgk"

alice nginx

### Обсуждение

NGINX предоставил модуль, который предоставляет стандартную функциональность JavaScript во время обработки запросов и ответов. Этот модуль позволяет вам встраивать бизнес-логику в ваш прокси-уровень. JavaScript был выбран из-за его широкого использования.

Модуль NJS предоставляет возможность вводить логику во время запросов, поступающих в NGINX, а также ответов, поступающих от NGINX. Вы можете проверять запросы и манипулировать ими по мере их прохождения через прокси-сервер, как показано в этом разделе путем декодирования веб-токена JSON. Модуль NJS также способен манипулировать ответами от вышестоящих служб путем потоковой передачи данных ответа через логику Java‐Script. NJS дополнительно позволяет потоковым службам стать осведомленными о прикладном уровне, как показано в разделе "Также смотрите".

### 5.3 Расширение NGINX с помощью общего языка программирования

### Проблема

### Вам нужен NGINX для выполнения некоторого пользовательского расширения с использованием общего языка программирования.

### Решение

Прежде чем готовиться к написанию пользовательского модуля NGINX с нуля на C, сначала оцените, подойдет ли один из других модулей языка программирования для вашего варианта использования. Язык программирования Си чрезвычайно мощный и производительный. Однако существует множество других языков, доступных в качестве модулей, которые могут обеспечить необходимую настройку. NGINX представила NGINX JavaScript (NJS), который предоставляет возможности Java‐скрипта в конфигурации NGINX простым включением модуля. Также доступны модули Lua и Perl.

С помощью этих языковых модулей вы либо импортируете файл, содержащий код, либо определяете блок кода непосредственно в конфигурации.

Чтобы использовать Lua, установите модуль Lua и следующую конфигурацию NGINX, чтобы определить встроенный скрипт Lua:

load\_module modules/ndk\_http\_module.so; load\_module modules/ngx\_http\_lua\_module.so;

events {} http {

server {

listen 8080; location / {

default\_type text/html; content\_by\_lua\_block {

ngx.say("hello, world")

}

}

}

}

Модуль Lua предоставляет свой собственный NGINX API через объект, определенный модулем с именем ngx. Подобно объекту запроса в NJS, следующий объект имеет атрибуты и методы для описания запроса и манипулирования ответом.

При установленном модуле Perl в этом примере Perl будет использоваться для установки переменной NGINX из среды выполнения:

load\_module modules/ngx\_http\_perl\_module.so; events {}

http {

perl\_set $app\_endpoint 'sub { return $ENV{"APP\_DNS\_ENDPOINT"}; }'; server {

listen 8080; location / {

proxy\_pass http://$app\_endpoint

}

}

}

}

### Предыдущий пример демонстрирует, что эти языковые модули предоставляют больше функциональных возможностей, чем просто возвращают ответ. Директива perl\_set устанавливает переменную NGINX для данных, возвращаемых из Perl-скрипта. Этот ограниченный пример просто возвращает системную переменную окружения, которая используется в качестве конечной точки для прокси-запросов.

### Обсуждение

### Возможности, предоставляемые расширяемостью NGINX, безграничны. NGINX расширяется пользовательским кодом с помощью модулей C, которые могут быть скомпилированы в NGINX при сборке из исходного кода или динамически загружены в конфигурацию. Существующие модули, предоставляющие функциональность и синтаксис JavaScript (js), Lua и Perl, уже доступны. Во многих случаях, если только пользовательская функциональность NGINX не распространяется на других, этих ранее существовавших модулей может быть достаточно. Многие скрипты, созданные для этих модулей, уже существуют в сообществе с открытым исходным кодом.

### Это решение продемонстрировало базовое использование языков сценариев Lua и Perl, доступных в NGINX и NGINX Plus. Независимо от того, требуется ли ответить, установить переменную, создать подзапрос или определить сложную перезапись, эти модули NGINX предоставляют такую возможность.

### 5.3 Установка с помощью Chef

### Проблема

### Вам необходимо установить и настроить NGINX с помощью Chef, чтобы управлять конфигурациями NGINX в виде кода и согласовывать их с остальными вашими конфигурациями Chef.

### Решение

Установите кулинарную книгу NGINX, которую ведут шеф-повара, из супермаркета Chef:

knife supermarket install nginx

Эта кулинарная книга основана на ресурсах, что означает, что она предоставляет ресурсы шеф-повара, которые вы можете использовать в своей собственной кулинарной книге. Создайте кулинарную книгу для вашего варианта использования NGINX. Эта кулинарная книга будет включать в себя поваренную книгу nginx, установленную из супермаркета в качестве зависимости. Как только зависимость будет установлена, вы сможете использовать предоставленные ресурсы. Создайте рецепт для установки NGINX:

nginx\_install 'nginx' do source 'repo'

end

Где source установлено значение repo, NGINX устанавливается из репозиториев, поддерживаемых NGINX Inc., которые предоставляют самые последние версии.

Используйте ресурс nginx\_config в рецепте для переопределения основных конфигураций NGINX:

nginx\_config 'nginx' do default\_site\_enabled true keepalive\_timeout 65 worker\_processes 'auto' action :create

notifies :reload, 'nginx\_service[nginx]', :delayed end

Используй nginx\_site ресурс в рецепте для настройки серверного блока NGINX:

nginx\_site 'test\_site' do mode '0644'

variables( 'server' => {

'listen' => [ '\*:80' ],

'server\_name' => [ 'test.example.com' ],

'access\_log' => '/var/log/nginx/test\_site.access.log', 'locations' => {

'/' => {

'root' => '/var/www/nginx-default', 'index' => 'index.html index.htm',

},

},

}

)

action :create

notifies :reload, 'nginx\_service[nginx]', :delayed end

### Обсуждение

### Chef - это инструмент управления конфигурацией, написанный на Ruby. Он может быть запущен как в режиме клиент‐сервер, так и в автономной конфигурации. У шеф-повара очень большое сообщество под названием "Супермаркет", в котором много общедоступных кулинарных книг. Общедоступные кулинарные книги из супермаркета можно устанавливать и обслуживать с помощью утилиты командной строки под названием Knife. Шеф-повар чрезвычайно способный, и то, что мы продемонстрировали, - всего лишь небольшой пример.

### Общедоступная кулинарная книга NGINX в супермаркете чрезвычайно гибка и предоставляет опции для простой установки NGINX из менеджера пакетов или из исходного кода, а также возможность компилировать и устанавливать множество различных модулей, а также создавать шаблоны базовых конфигураций. В этом разделе мы установили NGINX из репозиториев, поддерживаемых NGINX Inc., и настроили серверный блок для размещения HTML–файла в качестве базового примера. Вы можете предоставить свои собственные шаблоны ресурсу nginx\_site, чтобы получить дополнительный контроль над вашей конфигурацией NGINX.

### 5.3 Установка с помощью Ansible

### Проблема

### Вам необходимо установить и настроить NGINX с помощью Ansible, чтобы управлять конфигурациями NGINX в виде кода и согласовывать их с остальными вашими конфигурациями Ansible.

### Решение

Установите коллекцию Ansible NGINX из Ansible Galaxy:

ansible-galaxy collection install nginxinc.nginx\_core

Создайте playbook, который использует коллекцию и роль nginx для установки NGINX:

* hosts: all collections:
  + nginxinc.nginx\_core tasks:
  + name: Install NGINX include\_role:

name: nginx

* + Чтобы настроить NGINX, добавьте задачу и используйте роль nginx\_config, а также предоставьте переопределения переменных шаблону по умолчанию в соответствии с вашим вариантом использования:
  + name: Configure NGINX include\_role:

name: nginx\_config vars:

nginx\_config\_http\_template\_enable: true nginx\_config\_http\_template:

* + - template\_file: http/default.conf.j2 deployment\_location: /etc/nginx/conf.d/default.conf config:

servers:

* + - * core:

listen:

* + - * + port: 80 server\_name: localhost

log:

access:

* + - * + path: /var/log/nginx/access.log format: main

sub\_filter: sub\_filters:

* + - * + string: server\_hostname replacement: $hostname

once: false locations:

- location: / core:

root: /usr/share/nginx/html index: index.html

nginx\_config\_html\_demo\_template\_enable: true nginx\_config\_html\_demo\_template:

* + - template\_file: www/index.html.j2 html\_file\_name: index.html html\_file\_location: /usr/share/nginx/html web\_server\_name: Ansible NGINX collection

### Обсуждение

### Ansible - это широко используемый и мощный инструмент управления конфигурацией, написанный на Python. Конфигурация задач выполняется на YAML, и вы используете язык шаблонов Jinja2 для создания шаблонов файлов. Ansible предлагает сервер с именем Ansible Tower по модели подписки. Однако он обычно используется с локальных компьютеров или для создания серверов непосредственно на клиенте или в автономной модели. Ansible подключит SSH к серверам и запустит конфигурацию. Как и в случае с другими инструментами управления конфигурацией, существует большое сообщество общедоступных ролей. Ансибль называет это галактикой Ансибль. Вы можете найти очень сложные роли для использования в своих сборниках пьес.

### Это решение использовало набор общедоступных ролей, поддерживаемых NGINX Inc., для установки NGINX и создания образца конфигурации. Конфигурация, используемая в этом примере, создает демонстрационный HTML‐файл NGINX и размещает его по адресу /usr/share/nginx/html/ index.html . Файл конфигурации NGINX также является шаблоном для создания серверного блока, прослушивающего на localhost:80, с одним блоком местоположения, настроенным для обслуживания демонстрационного файла. Предоставляемые шаблоны конфигурации NGINX чрезвычайно всеобъемлющи; однако роль nginx\_config позволяет вам предоставлять свои собственные шаблоны для полного контроля, используя преимущества предварительно созданной и поддерживаемой конфигурации Ansible.

### 5.3 Автоматизация конфигураций с помощью шаблонов Consul

### Проблема

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

### Решение

Используйте демон consul-template и файл шаблона для создания шаблона файла конфигурации NGINX по вашему выбору:

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

}

Этот пример представляет собой файл шаблона Consul, который создает шаблон для вышестоящего блока конфигурации. Шаблон будет проходить по узлам в Consul, идентифицированным как app.backend. Для

для каждого узла в Consul шаблон создаст серверную директиву с IP-адресом этого узла.

Демон consul-template запускается через командную строку и может использоваться для перезагрузки NGINX при каждом изменении шаблона файла конфигурации:

# consul-template -consul-addr consul.example.internal -template \

./upstream.template:/etc/nginx/conf.d/upstream.conf:"nginx -s reload"

### Эта команда предписывает демону consul-template подключиться к кластеру Consul по адресу consul.example.internal и использовать файл с именем upstream.template в текущем рабочем каталоге для создания шаблона файла и вывода сгенерированного содержимого в /etc/nginx/conf.d/upstream.conf, затем чтобы перезагружать NGINX каждый раз, когда изменяется файл шаблона. Флаг -template принимает строку файла шаблона, расположение выходных данных и команду, которая будет запущена после завершения процесса создания шаблона. Эти три переменные разделены двоеточиями. Если в выполняемой команде есть пробелы, обязательно заключите ее в двойные кавычки. Флаг -consul сообщает демону, к какому кластеру Consul подключаться.

### Обсуждение

### Consul - это мощный инструмент обнаружения служб и хранилище конфигураций. Consul хранит информацию об узлах, а также пары ключ-значение в структуре, подобной каталогу, и обеспечивает взаимодействие с RESTful API. Consul также предоставляет DNS-интерфейс для каждого клиента, позволяющий выполнять поиск доменных имен узлов, подключенных к кластеру. Отдельным проектом, использующим кластеры Consul, является демон consul-template; этот инструмент создает шаблоны файлов в ответ на изменения в узлах Consul, службах или парах ключ-значение. Это делает Consul очень эффективным выбором для автоматизации NGINX. С помощью consul-template вы также можете поручить демону выполнить команду после внесения изменений в шаблон. С помощью этого вы можете перезагрузить конфигурацию NGINX и позволить вашей конфигурации NGINX оживать вместе с вашей средой. С Consul и consul-template ваша конфигурация NGINX может быть такой же динамичной, как и ваша среда. Информация об инфраструктуре, конфигурации и приложении хранится централизованно, и consul-template может подписываться и повторно создавать шаблоны по мере необходимости на основе событий. С помощью этой технологии NGINX может динамически перенастраиваться в ответ на добавление и удаление серверов, служб, версий приложений и т.д.

**CHAPTER 6**

# Идентификация

## Вводные

NGINX способен аутентифицировать клиентов. Аутентификация клиентских запросов с помощью NGINX разгружает работу и предоставляет возможность предотвращать попадание не прошедших проверку подлинности запросов на серверы ваших приложений. Модули, доступные для NGINX с открытым исходным кодом, включают базовую аутентификацию и подзапросы аутентификации. Эксклюзивный модуль NGINX Plus для проверки веб-токенов JSON (JWTS) обеспечивает интеграцию со сторонними поставщиками аутентификации, использующими стандарт аутентификации OpenID Connect.

### Базовая аутентификация по протоколу HTTP

### Проблема

### Вам необходимо защитить свое приложение или контент с помощью базовой аутентификации по протоколу HTTP.

### Решение

Сгенерируйте файл в следующем формате, где пароль зашифрован или хэширован с помощью одного из разрешенных форматов:

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

Имя пользователя - это первое поле, пароль - второе поле, а разделителем является двоеточие. Существует необязательное третье поле, которое вы можете использовать для комментирования каждого пользователя. NGINX может понимать несколько различных форматов паролей, одним из которых является пароль, зашифрованный с помощью функции C crypt(). Эта функция доступна для

командной строки с помощью команды openssl passwd. С установленным openssl вы можете создать зашифрованные строки паролей, используя следующую команду:

$ openssl passwd MyPassword1234

Результатом будет строка, которую NGINX может использовать в вашем файле паролей.

Используйте директивы auth\_basic и auth\_basic\_user\_file в вашей конфигурации NGINX, чтобы включить базовую аутентификацию:

location / {

auth\_basic "Private site"; auth\_basic\_user\_file conf.d/passwd;

}

Вы можете использовать директивы auth\_basic в контекстах http, сервера или местоположения. Директива auth\_basic принимает строковый параметр, который отображается во всплывающем окне базовой аутентификации при входе пользователя, не прошедшего проверку подлинности. Файл auth\_basic\_user\_file указывает путь к пользовательскому файлу.

Чтобы протестировать свою конфигурацию, вы можете использовать curl с флагом -u или --user для создания

Заголовок авторизации для запроса:

$ curl --user myuser:MyPassword1234 https://localhost

### Обсуждение

## Вы можете сгенерировать базовые пароли для аутентификации несколькими способами и в нескольких различных форматах с разной степенью безопасности. Команда htpasswd из Apache также может генерировать пароли. Как команды openssl, так и htpasswd могут генерировать пароли с помощью алгоритма apr1, который также понятен NGINX. Пароль также может быть в засоленном формате SHA-1, который используют Lightweight Directory Access Protocol (LDAP) и Dovecot. NGINX поддерживает больше форматов и алгоритмов хэширования; однако многие из них считаются небезопасными, поскольку их можно легко обойти с помощью атак методом перебора.

## Вы можете использовать базовую аутентификацию для защиты контекста всего хоста NGINX, определенных виртуальных серверов или даже только определенных блоков расположения. Базовая аутентификация не заменит аутентификацию пользователя в веб‐приложениях, но она может помочь обеспечить безопасность личной информации. Под капотом базовая аутентификация выполняется сервером, возвращающим неавторизованный HTTP-код 401 с заголовком ответа WWW- Authenticate. Этот заголовок будет иметь значение Basic realm="ваша строка". Этот ответ приводит к тому, что браузер запрашивает имя пользователя и пароль. Имя пользователя и пароль объединяются и разделяются двоеточием, затем кодируются в формате base64, а затем отправляются в заголовке запроса с именем Authorization. В заголовке запроса на авторизацию будет указана строка в кодировке Basic и user:password. Сервер декодирует заголовок и проверяет его на соответствие предоставленному файлу auth\_basic\_user\_file. Потому что

## строка имени пользователя и пароля просто закодирована в base64, рекомендуется использовать HTTPS с базовой аутентификацией.

### Подзапросы аутентификации

### Проблема

### У вас есть сторонняя система аутентификации, для которой вы хотели бы аутентифицировать запросы.

### Решение

Используйте http\_auth\_request\_module, чтобы отправить запрос в службу аутентификации для проверки подлинности перед отправкой 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 локальным внутренним местоположением. Директива auth\_request\_set позволяет вам устанавливать переменные из подзапроса аутентификации.

### Обсуждение

http\_auth\_request\_module обеспечивает аутентификацию при каждом запросе, обрабатываемом сервером NGINX. Модуль будет использовать подзапрос, чтобы определить, разрешено ли выполнение запроса. Подзапрос - это когда NGINX передает запрос в альтернативное внутреннее местоположение и наблюдает за его ответом, прежде чем направить запрос по назначению. Местоположение аутентификации передает исходный запрос, включая текст и заголовки, на сервер аутентификации. Код состояния HTTP подзапроса - это то, что определяет, предоставлен доступ или нет. Если подзапрос возвращается с кодом состояния HTTP 200, аутентификация проходит успешно и запрос выполняется. Если подзапрос возвращает HTTP 401 или 403, то же самое будет возвращено и для исходного запроса.

Если ваша служба аутентификации не запрашивает тело запроса, вы можете удалить тело запроса с помощью директивы proxy\_pass\_request\_body, как показано на рисунке. Этот

6.2 Подзапросы аутентификации

практика сократит размер запроса и время его выполнения. Поскольку тело ответа отбрасывается, заголовок Content-Length должен быть установлен в пустую строку. Если вашей службе аутентификации необходимо знать URI, к которому обращается запрос, вы захотите поместить это значение в пользовательский заголовок, который проверяет ваша служба аутентификации. Если есть вещи, которые вы хотите сохранить из подзапроса к службе аутентификации, например заголовки ответов или другую информацию, вы можете использовать директиву auth\_request\_set для создания новых переменных из данных ответа.

Validating JWTs with NGINX Plus

### Проблема

### Вам необходимо проверить JWT, прежде чем запрос будет обработан с помощью NGINX Plus.

### Решение

Используйте модуль аутентификации HTTP JWT от NGINX Plus для проверки подписи токена и встраивания утверждений и заголовков JWT в качестве переменных NGINX:

location /api/ {

auth\_jwt "api"; auth\_jwt\_key\_file conf/keys.json;

}

### Эта конфигурация позволяет выполнять проверку JWTS для данного местоположения. В директорию auth\_jwt передается строка, которая используется в качестве области аутентификации. Конфигурация auth\_jwt принимает необязательный параметр токена переменной, которая содержит JWT. По умолчанию заголовок аутентификации используется в соответствии со стандартом JWT. Директива auth\_jwt также может использоваться для отмены эффектов требуемой аутентификации JWT из унаследованных конфигураций. Чтобы отключить аутентификацию, передайте ключевое слово в директиву auth\_jwt и ничего больше. Чтобы отменить унаследованные требования аутентификации, передайте ключевое слово off директиве auth\_jwt, ничего больше не добавляя. Файл auth\_jwt\_key\_file принимает один параметр. Этот параметр представляет собой путь к файлу ключа в стандартном формате JSON Web Key (JWK).

### Обсуждение

NGINX Plus способен проверять типы токенов веб-подписи JSON, в отличие от типа веб-шифрования JSON, где весь токен зашифрован. NGINX Plus способен проверять подписи, подписанные с помощью алгоритмов HS256, RS 256 и 256256. Наличие NGINX Plus для проверки токена может сэкономить время и ресурсы, необходимые для выполнения подзапроса к службе аутентификации. NGINX Plus расшифровывает заголовок JWT и полезную нагрузку и записывает стандартные заголовки и утверждения в

### встроенные переменные для вашего использования. Директива auth\_jwt может использоваться в контекстах http, server, location и limit\_except.

### Создание веб-ключей JSON

### Проблема

### Для использования NGINX Plus вам нужен веб-ключ JSON (JWK).

### Решение

NGINX Plus использует формат JWK, указанный в стандарте RFC. Этот стандарт допускает использование массива ключевых объектов в файле JWK.

Ниже приведен пример того, как может выглядеть ключевой файл:

{**"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"

}

]

}

* 1. **Creating JSON Web Keys | 63**

### Показанный файл JWK демонстрирует три начальных типа ключей, указанных в стандарте RFC. Формат этих ключей также является частью стандарта RFC. Атрибутом key является тип ключа. В этом файле показаны три типа ключей: последовательность октетов (oct), эллиптическая кривая (EC) и тип RSA. Атрибут kid - это идентификатор ключа. Другие атрибуты этих ключей указаны в стандарте для этого типа ключей. Для получения дополнительной информации обратитесь к документации RFC по этим стандартам.

### Обсуждение

### Существует множество библиотек, доступных на самых разных языках для создания JWK. Рекомендуется создать ключевую службу, которая является центральным органом управления JWK, для создания и ротации ваших JWK через регулярные промежутки времени. Для повышения безопасности рекомендуется сделать ваши JWKS такими же защищенными, как и ваши сертификаты SSL / TLS. Защитите свой ключевой файл с помощью соответствующих разрешений пользователя и группы. Лучше всего хранить их в памяти на вашем хостинге. Вы можете сделать это, создав файловую систему в памяти, такую как ramfs. Также важно регулярно поворачивать клавиши; вы можете выбрать создание службы ключей, которая создает открытые и закрытые ключи и предлагает их приложению и NGINX через API.

### 6.5 Проверка веб-токенов JSON с помощью NGINX Plus

### Проблема

### Вы хотите проверить веб-токены JSON с помощью NGINX Plus.

### Решение

Используйте модуль JWT, который поставляется с NGINX Plus, для защиты местоположения или сервера и укажите директиве auth\_jwt использовать $cookie\_auth\_token в качестве токена, подлежащего проверке:

location /private/ {

auth\_jwt "Google Oauth" token=$cookie\_auth\_token; auth\_jwt\_key\_file /etc/nginx/google\_certs.jwk;

}

Эта конфигурация предписывает NGINX Plus защитить путь /private/ URI с помощью проверки JWT. Google OAuth 2.0 OpenID Connect использует файл cookie auth\_token, а не токен-носитель по умолчанию. Таким образом, вы должны проинструктировать NGINX искать токен в этом файле cookie, а не в местоположении NGINX Plus по умолчанию. То

Для расположения файла auth\_jwt\_key\_file задан произвольный путь, что является шагом, который мы рассмотрим в [Recipe 6.6](#_bookmark242).

### Обсуждение

### Эта конфигурация демонстрирует, как вы можете проверить JWT OpenID Connect Google OAuth 2.0 с помощью NGINX Plus. Модуль аутентификации NGINX Plus JWT для HTTP способен проверять любой JWT, который соответствует спецификации веб-подписи RFC для JSON, мгновенно позволяя любому органу единого входа, использующему JWTS, проходить проверку на уровне NGINX Plus. Протокол OpenID 1.0 - это слой поверх OAuth

### протокол аутентификации 2.0, добавляющий идентификацию, позволяющий использовать JWTS для подтверждения личности пользователя, отправляющего запрос. С помощью подписи токена NGINX Plus может подтвердить, что токен не был изменен с момента его подписания. Таким образом, Google использует метод асинхронной подписи и позволяет распространять общедоступные JWK, сохраняя при этом свой личный JWK в секрете.

### 6.6 Автоматическое получение и кэширование наборов веб-ключей JSON с помощью NGINX Plus

### Проблема

### Вы хотите, чтобы NGINX Plus автоматически запрашивал набор веб-ключей JSON (JWK) у поставщика и кэшировал его.

### Решение

Используйте зону кэширования и директиву auth\_jwt\_key\_request для автоматического обновления вашего ключа:

proxy\_cache\_path /data/nginx/cache levels=1 keys\_zone=foo:10m;

server {

# ...

location / {

auth\_jwt "closed site"; auth\_jwt\_key\_request /jwks\_uri;

}

location = /jwks\_uri { internal; proxy\_cache foo;

proxy\_pass https://idp.example.com/keys;

}

}

### В этом примере директива auth\_jwt\_key\_request предписывает NGINX Plus извлекать JWK из внутреннего подзапроса. Подзапрос направляется в /jwks\_uri, который будет передавать запрос поставщику удостоверений. Запрос кэшируется по умолчанию на 10 минут, чтобы ограничить накладные расходы.

### Обсуждение

### В NGINX Plus R17 была введена директива auth\_jwt\_key\_request. Эта функция позволяет серверу NGINX Plus динамически обновлять свои JWKS при поступлении запроса. Для извлечения JWKS используется метод подзапроса, что означает, что местоположение, на которое указывает директива, должно быть локальным для сервера NGINX Plus. В приведенном примере местоположение подзапроса было заблокировано, чтобы гарантировать, что будут обслуживаться только внутренние запросы NGINX Plus. Кэш также использовался для обеспечения того, чтобы запрос на получение JWKS выполнялся только так часто, как это необходимо, и не перегружал поставщика удостоверений. Директива auth\_jwt\_key\_request действительна в контекстах http, server, location и limit\_except.

### 6.6 Аутентификация пользователей с помощью существующего единого входа OpenID Connect с помощью NGINX Plus

### Проблема

### Вы хотите интегрировать NGINX Plus с поставщиком удостоверений OpenID Connect (OIDC).

### Решение

Это решение состоит из ряда аспектов конфигурации и небольшого количества JavaScript-кода NGINX. Поставщик удостоверений (IdP) должен поддерживать OpenID Connect 1.0. NGINX Plus будет выступать в качестве ретранслирующей стороны вашего OIDC в потоке кода авторизации.

NGINX Inc. поддерживает общедоступный репозиторий GitHub, содержащий конфигурацию и код в качестве эталонной реализации интеграции OIDC с NGINX Plus. Ссылка на репозиторий в разделе "Также смотрите" содержит обновленные инструкции о том, как настроить эталонную реализацию с помощью вашего собственного IdP.

### Обсуждение

### Это решение просто связано с эталонной реализацией, чтобы гарантировать, что у вас, читатель, есть самое современное решение. Предоставленная ссылка настраивает NGINX Plus в качестве ретранслирующей стороны для потока кода авторизации для OpenID Connect 1.0. Когда в NGINX Plus в этой конфигурации выполняются неаутентифицированные запросы на защищенные ресурсы, NGINX Plus сначала перенаправляет запрос IdP. IdP проводит клиента через свой собственный поток входа в систему и возвращает клиента в NGINX Plus с кодом аутентификации. Затем NGINX Plus напрямую связывается с IdP, чтобы обменять код аутентификации на набор идентификационных токенов. Эти токены проверяются с помощью JWTS и хранятся в хранилище ключей и значений NGINX Plus. Используя хранилище ключ-значение, токены становятся доступными для всех узлов NGINX Plus в конфигурации с высокой степенью доступности (HA). Во время этого процесса NGINX Plus генерирует сеансовый файл cookie для клиента, который используется в качестве ключа для поиска токена в хранилище значений ключей. Затем клиенту отправляется перенаправление с файлом cookie на первоначальный запрошенный ресурс. Последующие запросы проверяются с помощью файла cookie для поиска идентификационного токена в хранилище ключей и значений NGINX Plus.

### Эта возможность обеспечивает интеграцию с большинством основных поставщиков удостоверений, включая единый вход CA (ранее SiteMinder), ForgeRock OpenAM, Keycloak, Okta, OneLogin и Ping Identity. OIDC как стандарт чрезвычайно важен для аутентификации ‐ вышеупомянутые поставщики идентификационных данных являются лишь подмножеством возможных интеграций.

**CHAPTER 7**

**Средства контроля безопасности**

## Введение

* 1. Безопасность обеспечивается поэтапно, и в вашей модели безопасности должно быть несколько уровней, чтобы она была по-настоящему надежной. В этой главе мы рассмотрим множество различных способов защиты ваших веб-приложений с помощью NGINX и NGINX Plus. Вы можете использовать многие из этих методов защиты в сочетании друг с другом, чтобы усилить безопасность. Вы могли заметить, что в этой главе не затрагивается модуль ModSecurity 3.0 NGINX, который превращает NGINX в брандмауэр веб-приложений (WAF). Чтобы узнать больше о возможностях WAF, загрузите [*ModSecurity 3.0 and NGINX: Quick Start*](https://oreil.ly/SULeY)[*Guide*](https://oreil.ly/SULeY). Пожалуйста, обратите внимание, что WAF NGINX ModSecurity для NGINX Plus переходит на режим окончания срока службы (EoL) с 31 марта 2024 года. Для получения дополнительной информации, пожалуйста, прочтите этот [blog](https://oreil.ly/hMbSF).

## Access Based on IP Address

### Проблема

### Вам необходимо контролировать доступ на основе IP-адреса клиента.

### Решение

Используйте модуль HTTP или stream access 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 определения местоположения разрешает доступ с любого адреса IPv4 в 10.0.0.0/20, за исключением 10.0.0.1, разрешает доступ с адресов IPv6 в подсети 2001:0db8::/32 и возвращает 403 для запросов, исходящих с любого другого адреса. Директивы allow и deny действительны в контекстах http, сервера и местоположения, а также в контексте потока и сервера для TCP/UDP. Правила проверяются последовательно до тех пор, пока не будет найдено совпадение для удаленного адреса.

### Обсуждение

## Защита ценных ресурсов и сервисов в Интернете должна осуществляться на нескольких уровнях. Функциональность NGINX предоставляет возможность быть одним из таких уровней. Директива deny блокирует доступ к заданному 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 междоменной маршрутизации (CIDR), ключевое слово all и сокет Unix. Как правило, при защите ресурса можно разрешить блокировку внутренних IP-адресов и запретить доступ ко всем.

### Обеспечение совместного использования ресурсов разных источников

### Проблема

### Вы обслуживаете ресурсы из другого домена, и вам необходимо разрешить совместное использование ресурсов разных источников (CORS), чтобы браузеры могли использовать эти ресурсы.

### Решение

Измените заголовки на основе метода запроса, чтобы включить 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 много чего происходит, что было сокращено с помощью карты для группировки методов GET и POST вместе. Метод запроса ПАРАМЕТРОВ возвращает клиенту предполетный запрос о правилах CORS этого сервера. Методы OPTIONS, GET и POST разрешены в CORS. Установка заголовка Access-Control-Allow-Origin позволяет использовать контент, передаваемый с этого сервера, также на страницах источников, соответствующих этому заголовку. Предполетный запрос может быть кэширован на клиенте в течение 1 728 000 секунд или 20 дней.

### Обсуждение

Такие ресурсы, как JavaScript, создают CORS, когда запрашиваемый ими ресурс относится к домену, отличному от их собственного. Когда запрос считается исходящим из другого источника, браузер обязан подчиняться правилам CORS. Браузер не будет использовать ресурс, если у него нет заголовков, которые конкретно разрешают его использование. Чтобы разрешить использование наших ресурсов другими поддоменами, мы должны установить заголовки CORS, что можно сделать с помощью директивы add\_header. Если запрос представляет собой 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 предполетный запрос, чтобы проверить условия сервера, которым он будет подчиняться для этого ресурса. Если вы не зададите эти заголовки соответствующим образом, браузер выдаст сообщение об ошибке при попытке использовать этот ресурс.

### Шифрование на стороне клиента

### Проблема

### Вам необходимо зашифровать трафик между вашим сервером NGINX и клиентом.

### Решение

Используйте один из модулей SSL для шифрования трафика, таких как ngx\_http\_ssl\_module или

ngx\_stream\_ssl\_module:

http { # All directives used below are also valid in stream server {

listen 8443 ssl;

ssl\_certificate /etc/nginx/ssl/example.crt; ssl\_certificate\_key /etc/nginx/ssl/example.key;

}

}

### Эта конфигурация настраивает сервер на прослушивание порта, зашифрованного с помощью SSL/TLS, 8443. Директива ssl\_certificate определяет сертификат и необязательную цепочку, которая передается клиенту. Директива ssl\_certificate\_key определяет ключ, используемый NGINX для расшифровки запросов и шифрования ответов. Для ряда конфигураций согласования SSL/TLS по умолчанию установлены предустановки безопасности на дату выпуска версии NGINX.

### Обсуждение

### Защищенные транспортные уровни являются наиболее распространенным способом шифрования передаваемой информации. На момент написания этой статьи протокол TLS является предпочтительным по сравнению с протоколом SSL. Это потому, что версии SSL с 1 по 3 теперь считаются небезопасными. Хотя название протокола может быть другим, TLS по-прежнему устанавливает уровень защищенных сокетов. NGINX позволяет вашему сервису защищать информацию между вами и вашими клиентами, что, в свою очередь, защищает клиента и ваш бизнес. При использовании сертификата, подписанного центром сертификации, вам необходимо объединить сертификат с цепочкой центров сертификации. Когда вы объединяете свой сертификат и цепочку, ваш сертификат должен находиться над файлом объединенной цепочки. Если ваш центр сертификации предоставил несколько файлов в качестве промежуточных сертификатов для цепочки, существует порядок, в котором они распределены по уровням. Для получения информации о заказе обратитесь к документации поставщика сертификатов.

### Расширенное шифрование на стороне клиента

### Проблема

### У вас есть расширенные потребности в настройке клиент-серверного шифрования.

### Решение

Модули http и stream SSL для NGINX позволяют полностью контролировать принятое подтверждение SSL/TLS. Сертификаты и ключи могут быть предоставлены NGINX в виде пути к файлу или значения переменной. NGINX предоставляет клиенту принятый список протоколов, шифров и типов ключей в соответствии со своей конфигурацией. Оговаривается самый высокий стандарт между клиентом и сервером NGINX. NGINX может кэшировать результат согласования SSL/TLS между клиентом и сервером в течение определенного периода времени.

Ниже намеренно демонстрируется сразу несколько вариантов, чтобы проиллюстрировать доступную сложность согласования клиент-сервер:

http { # All directives used below are also valid in stream server {

listen 8443 ssl;

# Set accepted protocol and cipher ssl\_protocols TLSv1.2 TLSv1.3; ssl\_ciphers HIGH:!aNULL:!MD5;

# RSA certificate chain loaded from file ssl\_certificate /etc/nginx/ssl/example.crt; # RSA encryption key loaded from file

ssl\_certificate\_key /etc/nginx/ssl/example.pem;

# Elliptic curve cert from variable value ssl\_certificate $ecdsa\_cert;

# Elliptic curve key as file path variable ssl\_certificate\_key data:$ecdsa\_key\_path;

# Client-Server negotiation caching ssl\_session\_cache shared:SSL:10m; ssl\_session\_timeout 10m;

}

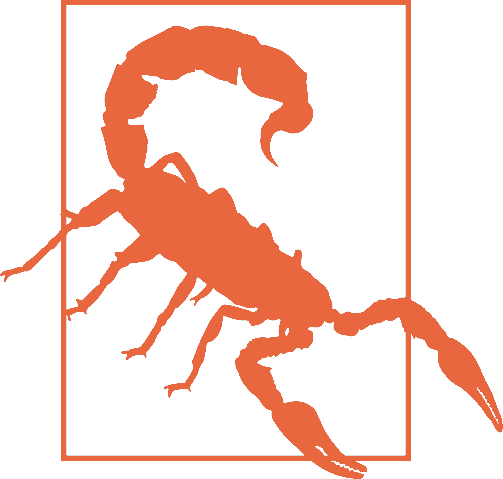
}

Сервер принимает версии протокола SSL TLSv1.2 и TLSv1.3. Принятые шифры имеют значение HIGH, что является макросом для самого высокого стандарта; явные отклонения демонстрируются для aNULL и MD5 путем обозначения !.

Используются два набора пар сертификат-ключ. Значения, передаваемые директивам NGINX, демонстрируют различные способы предоставления значений ключа сертификата NGINX. Переменная интерпретируется как путь к файлу. При наличии префикса data: значение переменной интерпретируется как прямое значение. Для обеспечения

* 1. **Advanced Client-Side Encryption | 73**

обратная совместимость с клиентом. Результатом согласования будет самый строгий стандарт, доступный клиенту и принятый сервером.

Если ключ SSL/TLS отображается как переменная прямого значения, он потенциально может быть занесен в журнал или открыт конфигурацией. Убедитесь, что у вас есть строгий контроль изменений и доступа, если вы предоставляете значение ключа в качестве переменной.

### Кэш сеанса SSL и тайм-аут позволяют рабочим процессам NGINX кэшировать и сохранять параметры сеанса в течение заданного промежутка времени. Рабочие процессы NGINX совместно используют этот кэш между собой как процессы в рамках одного экземпляра, но не между машинами. Существует множество других параметров кэширования сеансов, которые могут повысить производительность или безопасность всех типов вариантов использования. Вы можете использовать параметры кэширования сеансов в сочетании друг с другом. Однако указание параметра без значения по умолчанию приведет к отключению встроенного кэша сеансов по умолчанию.

### Обсуждение

В этом расширенном примере NGINX предоставляет клиенту опции SSL/TLS версии 1.2 или 1.3, высоко оцененные алгоритмы шифрования и возможность использовать ключи в формате RSA или криптографии с эллиптической кривой (ECC). Результатом переговоров является самый надежный из протоколов, шифров и форматов ключей, на которые способен клиент. Конфигурация предписывает NGINX кэшировать согласование в течение 10 минут с выделением доступной памяти в размере 10 МБ.

В ходе тестирования было обнаружено, что сертификаты ECC работают быстрее, чем сертификаты RSA эквивалентной силы. Размер ключа меньше, что приводит к возможности обслуживать больше соединений SSL/TLS и более быстрым рукопожатиям. NGINX позволяет вам настроить несколько сертификатов и ключей, а затем предоставить оптимальный сертификат для клиентского браузера. Это позволяет вам использовать преимущества новой технологии, но при этом по-прежнему обслуживать старых клиентов.

 В этом примере NGINX шифрует трафик между собой и клиентом. Однако подключение к вышестоящим серверам также может быть зашифровано. Согласование между NGINX и вышестоящим сервером продемонстрировано в [Recipe 7.5](#_bookmark271).

### 7.5 Восходящее шифрование

### Проблема

### Вам необходимо зашифровать трафик между NGINX и вышестоящей службой и установить конкретные правила согласования для соответствия нормативным требованиям, иначе вышестоящая служба находится за пределами вашей защищенной сети.

### Решение

Используйте директивы SSL модуля HTTP proxy для указания правил SSL:

location / {

proxy\_pass https://upstream.example.com; proxy\_ssl\_verify on; proxy\_ssl\_verify\_depth 2; proxy\_ssl\_protocols TLSv1.2;

}

### Эти директивы прокси-сервера устанавливают определенные правила SSL, которым должен подчиняться NGINX. Настроенные директивы гарантируют, что NGINX проверяет, что сертификат и цепочка в вышестоящей службе действительны на глубину до двух сертификатов. Директива proxy\_ssl\_protocols указывает, что NGINX будет использовать только TLS версии 1.2. По умолчанию NGINX не проверяет вышестоящие сертификаты и принимает все версии TLS.

### Обсуждение

Директивы конфигурации для модуля HTTP proxy обширны, и если вам нужно зашифровать восходящий трафик, вы должны, по крайней мере, включить проверку. Вы можете прокси-сервер по протоколу HTTPS, просто изменив протокол на значение, передаваемое директиве proxy\_pass. Однако это не проверяет вышестоящий сертификат. Другие директивы, такие как proxy\_ssl\_certificate и proxy\_ssl\_certificate\_key, позволяют заблокировать восходящее шифрование для повышения безопасности. Вы также можете указать proxy\_ssl\_crl или список отзыва сертификатов, в котором перечислены сертификаты, которые больше не считаются действительными. Эти директивы SSL‐прокси помогают упростить каналы связи вашей системы в вашей собственной сети или через общедоступный Интернет..

### 7.5 Обеспечение безопасности местоположения

### Проблема

Вам нужно обезопасить блок местоположения с помощью секрета.

### Решение

Используйте модуль защищенной ссылки и директиву secure\_link\_secret, чтобы ограничить доступ к ресурсам пользователям, имеющим защищенную ссылку:

location /resources { secure\_link\_secret mySecret;

if ($secure\_link = "") { return 403; }

rewrite ^ /secured/$secure\_link;

}

location /secured/ { internal;

root /var/www;

}

### Эта конфигурация создает внутренний и общедоступный блок расположения. Общедоступный блок определения местоположения /ресурсов вернет значение 403 Forbidden, если URI запроса не содержит хэш-строку md5, которая может быть проверена с помощью секрета, предоставленного директиве secure\_link\_secret. Переменная $secure\_link является пустой строкой, если хэш в URI не проверен.

### Обсуждение

Защита ресурсов с помощью секрета - отличный способ обеспечить защиту ваших файлов. Секрет используется в сочетании с URI. Затем эта строка хэшируется в формате md5, и шестнадцатеричный код этого хэша md5 используется в URL-адресе. Хэш помещается в ссылку и вычисляется NGINX. NGINX знает путь к запрашиваемому файлу, потому что он находится в URI после хэша. NGINX также знает ваш секрет, поскольку он предоставляется с помощью директивы secure\_link\_secret. NGINX способен быстро проверить хэш md5 и сохранить URI в переменной $secure\_link. Если хэш не может быть проверен, переменной присваивается значение пустой строки. Важно отметить, что аргумент, передаваемый secure\_link\_secret, должен быть статической строкой; он не может быть переменной.

### 7.5 Создание защищенной ссылки с секретным

### Проблема

### Вам нужно сгенерировать защищенную ссылку из вашего приложения, используя секрет.

### Решение

Модуль secure link в NGINX принимает шестнадцатеричный дайджест хэшированной строки md5, где строка представляет собой объединение пути URI и секрета. Опираясь на последний раздел, [Recipe 7.6](#_bookmark277), мы создадим защищенную ссылку, которая будет работать с

предыдущий пример конфигурации, в котором указан файл, присутствующий в /var/www/secured/ index.html . Чтобы сгенерировать шестнадцатеричный дайджест хэша md5, мы можем использовать команду Unix openssl:

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

Здесь мы показываем URI, который мы защищаем, index.html , связанный с нашим секретом,

mySecret. Эта строка передается команде openssl для вывода шестнадцатеричного дайджеста md5.

Ниже приведен пример того же хэш-дайджеста, созданного в Python с использованием библиотеки hashlib, которая включена в стандартную библиотеку Python:

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

Теперь, когда у нас есть этот хэш-дайджест, мы можем использовать его в URL-адресе. Нашим примером будет www.example.com отправка запроса на получение файла /var/www/secured/index.html через наше местоположение /ресурсы. Наш полный URL-адрес будет следующим:

[www.example.com/resources/a53bee08a4bf0bbea978ddf736363a12/\](http://www.example.com/resources/a53bee08a4bf0bbea978ddf736363a12/\) index.html

### Обсуждение

Создание дайджеста может быть выполнено многими способами на многих языках. Что следует помнить: URI-путь идет перед secret, в строке нет возврата каретки, и используйте шестнадцатеричный дайджест хэша md5.

### 7.5 Обеспечение сохранности местоположения с истекшим сроком годности

### Проблема

### Вам нужно закрепить местоположение с помощью ссылки, срок действия которой истекает в какой-то момент в будущем и которая является специфичной для клиента.

### Решение

Используйте другие директивы, включенные в модуль защищенной ссылки, чтобы установить время истечения срока действия и использовать переменные в вашей защищенной ссылке:

location /resources { root /var/www;

secure\_link $arg\_md5,$arg\_expires;

secure\_link\_md5 "$secure\_link\_expires$uri$remote\_addrmySecret"; if ($secure\_link = "") { return 403; }

if ($secure\_link = "0") { return 410; }

}

The secure\_link directive takes two parameters separated with a comma. The first parameter is the variable that holds the md5 hash. This example uses an HTTP argument of md5. The second parameter is a variable that holds the time in which the link expires in Unix epoch time format. The secure\_link\_md5 directive takes a single parameter that declares the format of the string that is used to construct the md5 hash. Like the other configuration, if the hash does not validate, the $secure\_link variable is set to an empty string. However, with this usage, if the hash matches but the time has expired, the $secure\_link variable will be set to 0.

### Discussion

This usage of securing a link is more flexible and looks cleaner than the secure\_link\_secret shown in [Recipe 7.6](#_bookmark277). With these directives, you can use any number of variables that are available to NGINX in the hashed string. Using user- specific variables in the hash string will strengthen your security, as users won’t be able to trade links to secured resources. It’s recommended to use a variable like

$remote\_addr or $http\_x\_forwarded\_for, or a session cookie header generated by the application. The arguments to secure\_link can come from any variable you prefer, and they can be named whatever best fits. The conditions are: Do you have access? Are you accessing it within the time frame? If you don’t have access: Forbid‐ den. If you have access but you’re late: Gone. The HTTP 410, Gone, works great for expired links because the condition is to be considered permanent.

## Generating an Expiring Link

### Problem

You need to generate a link that expires.

### Solution

Generate a timestamp for the expire time in the Unix epoch format. On a Unix system, you can test by using the date as demonstrated in the following:

$ date -d "2030-12-31 00:00" +%s --utc 1924905600

Next, you’ll need to concatenate your hash string to match the string configured with the secure\_link\_md5 directive. In this case, our string to be used will be 1924905600/ resources/index.html127.0.0.1 mySecret. The md5 hash is a bit different than just a hex digest. It’s an md5 hash in binary format, base64-encoded, with plus signs (+) translated to hyphens (-), slashes (/) translated to underscores (\_), and equal signs (=) removed. The following is an example on a Unix system:

$ echo -n '1924905600/resources/index.html127.0.0.1 mySecret' \

| openssl md5 -binary \

| openssl base64 \

| tr +/ -\_ \

| tr -d = sqysOw5kMvQBL3j9ODCyoQ

Now that we have our hash, we can use it as an argument along with the expire date:

/resources/index.html?md5=sqysOw5kMvQBL3j9ODCyoQ&expires=1924905600

The following is a more practical example in Python utilizing a relative time for the expiration, setting the link to expire one hour from generation. At the time of writing this example works with Python 2.7 and 3.x utilizing the Python Standard Library:

**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)

### Discussion

With this pattern, we’re able to generate a secure link in a special format that can be used in URLs. The secret provides security through use of a variable that is never sent to the client. You’re able to use as many other variables as you need to in order to secure the location. md5 hashing and base64 encoding are common, lightweight, and available in nearly every language.

## HTTPS Redirects

### Problem

You need to redirect unencrypted requests to HTTPS.

### Solution

Use a rewrite to send all HTTP traffic to HTTPS:

server {

listen 80 default\_server;

listen [::]:80 default\_server; server\_name \_;

return 301 https://$host$request\_uri;

}

This configuration listens on port 80 as the default server for both IPv4 and IPv6 and for any hostname. The return statement returns a 301 permanent redirect to the HTTPS server at the same host and request URI.

### Discussion

It’s important to always redirect to HTTPS where appropriate. You may find that you do not need to redirect all requests but only those with sensitive information being passed between client and server. In that case, you may want to put the return statement in particular locations only, such as */login*.

## Redirecting to HTTPS Where SSL/TLS Is Terminated Before NGINX

### Problem

You need to redirect to HTTPS, however, you’ve terminated SSL/TLS at a layer before NGINX.

### Solution

Use the common X-Forwarded-Proto header to determine if you need to redirect:

server {

listen 80 default\_server;

listen [::]:80 default\_server; server\_name \_;

if ($http\_x\_forwarded\_proto = 'http') { return 301 https://$host$request\_uri;

}

}

This configuration is very much like HTTPS redirects. However, in this configuration we’re only redirecting if the header X-Forwarded-Proto is equal to HTTP.

### Discussion

It’s a common use case that you may terminate SSL/TLS in a layer in front of NGINX. One reason you may do something like this is to save on compute costs. However, you need to make sure that every request is HTTPS, but the layer terminating SSL/TLS does not have the ability to redirect. It can, however, set proxy headers. This configuration works with layers such as the Amazon Web Services Elastic Load Balancer (AWS ELB), which will offload SSL/TLS at no additional cost. This is a handy trick to make sure that your HTTP traffic is secured.

## HTTP Strict Transport Security

### Problem

You need to instruct browsers to never send requests over HTTP.

### Solution

Use the HTTP Strict Transport Security (HSTS) enhancement by setting the Strict- Transport-Security header:

add\_header Strict-Transport-Security max-age=31536000;

This configuration sets the Strict-Transport-Security header to a max age of a year. This will instruct the browser to always do an internal redirect when HTTP requests are attempted to this domain, so that all requests will be made over HTTPS.

### Discussion

For some applications, a single HTTP request trapped by a man-in-the-middle attack could be the end of the company. If a form post containing sensitive information

is sent over HTTP, the HTTPS redirect from NGINX won’t save you; the damage is done. This opt-in security enhancement informs the browser to never make an HTTP request, and therefore the request is never sent unencrypted.

### Also See

[RFC Standard documentation of HTTP Strict Transport Security](https://oreil.ly/oLaZc) [OWASP HTTP Strict Transport Security Cheat Sheet](https://oreil.ly/AVn-g)

## Satisfying Any Number of Security Methods

### Problem

You need to provide multiple ways to pass security to a closed site.

### Solution

Use the satisfy directive to instruct NGINX that you want to satisfy any or all of the security methods used:

location / {

satisfy any;

allow 192.168.1.0/24; deny all;

auth\_basic "closed site"; auth\_basic\_user\_file conf/htpasswd;

}

This configuration tells NGINX that the user requesting the location / needs to satisfy one of the security methods: either the request needs to originate from the *192.168.1.0/24* CIDR block or it must be able to supply a username and password that can be found in the *conf/htpasswd* file. The satisfy directive takes one of two options: any or all.

### Discussion

The satisfy directive is a great way to offer multiple ways to authenticate to your web application. By specifying any to the satisfy directive, the user must meet one of the security challenges. By specifying all to the satisfy directive, the user must meet all of the security challenges. This directive can be used in conjunction with the http\_access\_module detailed in [Recipe 7.1](#_bookmark252), the http\_auth\_basic\_module detailed in [Recipe 6.1](#_bookmark216), the http\_auth\_request\_module detailed in [Recipe 6.2](#_bookmark224), and the http\_auth\_jwt\_module detailed in [Recipe 6.3](#_bookmark229). Security is only truly secure if it’s done

in multiple layers. The satisfy directive will help you achieve this for locations and servers that require deep security rules.

## NGINX Plus Dynamic Application Layer DDoS Mitigation

### Problem

You need a dynamic Distributed Denial of Service (DDoS) mitigation solution.

### Solution

Use NGINX Plus to build a cluster-aware rate limit and automatic blocklist:

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/ {

* 1. **NGINX Plus Dynamic Application Layer DDoS Mitigation | 83**

api write=on;

# directives to control access to the API

}

}

### Discussion

This solution uses a synchronized rate limit by use of a synchronized key-value store to dynamically respond to DDoS attacks and mitigate their effects. The sync parameter provided to the limit\_req\_zone and keyval\_zone directives synchronizes the shared memory zone with other machines in the active-active NGINX Plus cluster. This example identifies clients that send more than 100 requests per second, regardless of which NGINX Plus node receives the request. When a client exceeds the rate limit, its IP address is added to a “sin bin” key-value store by making a call to the NGINX Plus API. The sin bin is synchronized across the cluster. Further requests from clients in the sin bin are subject to a very low bandwidth limit, regardless of which NGINX Plus node receives them. Limiting bandwidth is preferable to rejecting requests outright because it does not clearly signal to the client that DDoS mitigation is in effect. After 10 minutes, the client is automatically removed from the sin bin.

## Installing and Configuring NGINX Plus with the NGINX App Protect WAF Module

### Problem

You need to install and configure the NGINX App Protect WAF module.

### Solution

Follow the [NGINX App Protect WAF administration guide](https://oreil.ly/jEOqg) for your platform. Make sure not to skip the portion about installing NGINX App Protect WAF signatures from the separate repository.

Ensure that the NGINX App Protect WAF module is dynamically loaded by NGINX Plus using the load\_module directive in the main context, and enabled by using the app\_protect\_\* directives:

user nginx; worker\_processes auto;

load\_module modules/ngx\_http\_app\_protect\_module.so; # ... Other main context directives

http {

app\_protect\_enable on;

app\_protect\_policy\_file "/etc/nginx/AppProtectTransparentPolicy.json"; app\_protect\_security\_log\_enable on;

app\_protect\_security\_log "/etc/nginx/log-default.json" syslog:server=127.0.0.1:515;

# ... Other http context directives

}

In this example, the app\_protect\_enable directive set to on enabled the module for the current context. This directive, and all of the following, are valid within the http context, as well as the server and location contexts with HTTP. The app\_protect\_policy\_file directive points to an NGINX App Protect WAF policy file, which we will define next; if not defined, the default policy is used. Security log‐ ging is configured next, and requires a remote logging server. For the example, we’ve configured it to the local Syslog listener. The app\_protect\_security\_log directive takes two parameters; the first is a JSON file that defines the logging settings, and the second is a log stream destination. The log settings file will be shown later in this section.

Build an NGINX App Protect WAF policy file, naming it */etc/nginx/AppProtectTrans‐ parentPolicy.json*:

{

**"policy"**: {

**"name"**: "transparent\_policy",

**"template"**: { **"name"**: "POLICY\_TEMPLATE\_NGINX\_BASE" },

**"applicationLanguage"**: "utf-8",

**"enforcementMode"**: "transparent"

}

}

This policy file configures the default NGINX App Protect WAF policy by use of a template, setting the policy name to transparent\_policy, and setting the enforce mentMode to transparent, which means NGINX Plus will log but not block. Trans‐ parent mode is great to test out new policies before putting them into effect.

Enable blocking by changing the enforcementMode to blocking. This policy file can be named */etc/nginx/AppProtectBlockingPolicy.json*. To switch between the files, update the app\_protect\_policy\_file directive in your NGINX Plus configuration:

{

**"policy"**: {

**"name"**: "blocking\_policy",

**"template"**: { **"name"**: "POLICY\_TEMPLATE\_NGINX\_BASE" },

**"applicationLanguage"**: "utf-8",

**"enforcementMode"**: "blocking"

}

}

To enable some of the protection features of NGINX App Protect WAF, enable some violations:

{

**"policy"**: {

**"name"**: "blocking\_policy",

**"template"**: { **"name"**: "POLICY\_TEMPLATE\_NGINX\_BASE" },

**"applicationLanguage"**: "utf-8", **"enforcementMode"**: "blocking", **"blocking-settings"**: {

**"violations"**: [

{

**"name"**: "VIOL\_JSON\_FORMAT",

**"alarm"**: **true**, **"block"**: **true**

},

{

**"name"**: "VIOL\_PARAMETER\_VALUE\_METACHAR",

**"alarm"**: **true**, **"block"**: **false**

}

]

}

}

}

The preceding example demonstrates adding two violations to our policy. Take note that VIOL\_PARAMETER\_VALUE\_METACHAR is not set to block, but only alarm; whereas VIOL\_JSON\_FORMAT is set to block and alarm. This functionality enables the overriding of the default enforcementMode when set to blocking. When enforcementMode is set to transparent, the default enforcement setting takes precedence.

Set up an NGINX Plus logging file, named */etc/nginx/log-default.json*:

{

**"filter"**:{ **"request\_type"**:"all"

},

**"content"**:{ **"format"**:"default", **"max\_request\_size"**:"any", **"max\_message\_size"**:"5k"

}

}

This file was defined in the NGINX Plus configuration by the app\_protect\_secu rity\_log directive and is necessary for NGINX App Protect WAF logging.

### Discussion

This solution demonstrates the basis of configuring NGINX Plus with the NGINX App Protect WAF module. The NGINX App Protect WAF module enables an entire suite of Web Application Firewall (WAF) definitions. These definitions derive from the Advanced F5 Application Security functionality in F5. This comprehensive set of WAF attack signatures has been extensively field-tested and proven. By adding this to an NGINX Plus installation, it renders the best of F5 Application Security with the agility of the NGINX Platform.

Once the module is installed and enabled, most of the configuration is done in a pol‐ icy file. The policy files in this section showed how to enable active blocking, passive monitoring, and transparent mode, as well as explained overrides to this functionality with violations. Violations are only one type of protection offered. Other protections include HTTP Compliance, Evasion Techniques, Attack Signatures, Server Technolo‐ gies, Data Guard, and many more. To retrieve NGINX App Protect WAF logs, it’s necessary to use the NGINX Plus logging format and send the logs to a remote listening service, a file, or /dev/stderr.

If you’re using NGINX Controller ADC, you can enable NGINX App Protect WAF capabilities through the NGINX Controllers App Security component, and visualize the WAF metrics through the web interface.

### Also See

[NGINX App Protect WAF Administration Guides](https://oreil.ly/jEOqg) [NGINX App Protect WAF Configuration Guide](https://oreil.ly/eoeq7) [NGINX Controller App Security guide](https://oreil.ly/pBZyh)

[NGINX App Protect DoS deployment guide](https://oreil.ly/dEtWU)

**CHAPTER 8**

# HTTP/2

## Introduction

HTTP/2 is a major revision to the HTTP protocol. Much of the change to this revision is focused on the transport layer, such as enabling full request and response multiplexing over a single TCP connection. Efficiencies were gained through the use of compression on HTTP header fields, and support for request prioritization was added. Another large addition to the protocol was the ability for the server to push messages to the client. This chapter details the basic configuration for enabling HTTP/2 in NGINX as well as configuring Google’s open source remote procedure call (gRPC) and HTTP/2 server push support.

## Basic Configuration

### Problem

You want to take advantage of HTTP/2.

### Solution

Turn on HTTP/2 on your NGINX server:

server {

listen 443 ssl http2 default\_server;

ssl\_certificate server.crt; ssl\_certificate\_key server.key; # ...

}

### Discussion

To turn on HTTP/2, you simply need to add the http2 parameter to the listen directive. The catch, however, is that although the protocol does not require the connection to be wrapped in SSL/TLS, some implementations of HTTP/2 clients only support HTTP/2 over an encrypted connection. Another caveat is that the HTTP/2 specification has blocked a number of TLS 1.2 cipher suites, and therefore will fail the handshake. The ciphers NGINX uses by default are not on the blocklist. The Application-Layer Protocol Negotiation of TLS allows the application layer to negoti‐ ate which protocol should be used over the secure connection to avoid additional round trips. To test that your setup is correct, you can install a plug-in for Chrome and Firefox browsers that indicates when a site is using HTTP/2, or on the command line with the nghttp utility.

### Also See

[HTTP/2 RFC Cipher Suite Black List](https://oreil.ly/T6mD1) [Chrome HTTP/2 and SPDY indicator plug-in](https://oreil.ly/yKyrU) [Firefox HTTP Indicator add-on](https://mzl.la/2A4LT4o)

## gRPC

### Problem

You need to terminate, inspect, route, or load balance gRPC method calls.

### Solution

Use NGINX to proxy gRPC connections:

server {

listen 80 http2;

location / {

grpc\_pass grpc://backend.local:50051;

}

}

In this configuration, NGINX is listening on port 80 for unencrypted HTTP/2 traffic, and proxying that traffic to a machine named backend.local on port 50051. The grpc\_pass directive instructs NGINX to treat the communication as a gRPC call. The grpc:// in front of our backend server location is not necessary; however, it does directly indicate that the backend communication is not encrypted.

To utilize TLS encryption between the client and NGINX, and terminate that encryp‐ tion before passing the calls to the application server, turn on SSL and HTTP/2, as you did in the first section:

server {

listen 443 ssl http2 default\_server;

ssl\_certificate server.crt; ssl\_certificate\_key server.key; location / {

grpc\_pass grpc://backend.local:50051;

}

}

This configuration terminates TLS at NGINX and passes the gRPC communication to the application over unencrypted HTTP/2.

To configure NGINX to encrypt the gRPC communication to the application server, providing end-to-end encrypted traffic, simply modify the grpc\_pass directive to specify grpcs:// before the server information (note the addition of the s denoting secure communication):

grpc\_pass grpcs://backend.local:50051;

You also can use NGINX to route calls to different backend services based on the gRPC URI, which includes the package, service, and method. To do so, utilize the location directive:

location /mypackage.service1 { grpc\_pass grpc://$grpc\_service1;

}

location /mypackage.service2 { grpc\_pass grpc://$grpc\_service2;

}

location / {

root /usr/share/nginx/html; index index.html index.htm;

}

This configuration example uses the location directive to route incoming HTTP/2 traffic between two separate gRPC services, as well as a location to serve static content. Method calls for the mypackage.service1 service are directed to the value of the variable grpc\_service1, which may contain a hostname or IP and optional port. Calls for mypackage.service2 are directed to the value of the variable grpc\_ser vice2. The location / catches any other HTTP request and serves static content. This demonstrates how NGINX is able to serve gRPC and non-gRPC under the same HTTP/2 endpoint and route accordingly.

Load balancing gRPC calls is also similar to non-gRPC HTTP traffic:

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;

}

}

The upstream block works the exact same way for gRPC as it does for other HTTP traffic. The only difference is that the upstream is referenced by grpc\_pass.

### Discussion

NGINX is able to receive, proxy, load balance, route, and terminate encryption for gRPC calls. The gRPC module enables NGINX to set, alter, or drop gRPC call headers, set timeouts for requests, and set upstream SSL/TLS specifications. As gRPC communicates over the HTTP/2 protocol, you can configure NGINX to accept gRPC and non-gRPC web traffic on the same endpoint.

## HTTP/2 Server Push

### Problem

You need to preemptively push content to the client.

### Solution

Use the HTTP/2 server push feature of NGINX:

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;

}

}

### Discussion

To use HTTP/2 server push, your server must be configured for HTTP/2, as is done in [Recipe 7.1](#_bookmark252). From there, you can instruct NGINX to push specific files preemptively with the http2\_push directive. This directive takes one parameter, the full URI path of the file to push to the client.

NGINX can also automatically push resources to clients if proxied applica‐ tions include an HTTP response header named Link. This header is able to instruct NGINX to preload the resources specified. To enable this feature, add http2\_push\_preload on; to the NGINX configuration.

**CHAPTER 9**

# Sophisticated Media Streaming

## Introduction

This chapter covers streaming media with NGINX in MPEG-4 (MP4) or Flash Video (FLV) formats. NGINX is widely used to distribute and stream content to the masses. NGINX supports industry-standard formats and streaming technologies, which will be covered in this chapter. NGINX Plus enables the ability to fragment content on the fly with the HTTP Live Stream (HLS) module, as well as the ability to deliver HTTP Dynamic Streaming (HDS) of already fragmented media. NGINX natively allows for bandwidth limits, and NGINX Plus’s advanced features offer bitrate limiting, enabling your content to be delivered in the most efficient manner while reserving the servers’ resources to reach the most users.

## Serving MP4 and FLV

### Problem

You need to stream digital media, originating in MP4 or FLV.

### Solution

Designate an HTTP location block to serve *.mp4* or *.flv* videos. NGINX will stream the media using progressive downloads or HTTP pseudostreaming and support seeking:

http {

server {

# ...

location /videos/ {

mp4;

}

location ~ \.flv$ { flv;

}

}

}

The example location block tells NGINX that files in the *videos* directory are in MP4 format type and can be streamed with progressive download support. The second location block instructs NGINX that any files ending in *.flv* are in FLV format and can be streamed with HTTP pseudostreaming support.

### Discussion

Streaming video or audio files in NGINX is as simple as a single directive. Progressive download enables the client to initiate playback of the media before the file has finished downloading. NGINX provides support for seeking to an undownloaded portion of the media in both formats.

## Streaming with HLS with NGINX Plus

### Problem

You need to support HTTP Live Streaming (HLS) for H.264/AAC-encoded content packaged in MP4 files.

### Solution

Utilize NGINX Plus’s HLS module with real-time segmentation, packetization, and multiplexing, with control over fragmentation buffering and more, like forwarding HLS arguments:

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;

}

This location block directs NGINX to stream HLS media out of the */var/www/video* directory, fragmenting the media into 4-second segments. The number of HLS buf‐ fers is set to 10 with a size of 10 MB. The initial MP4 buffer size is set to 1 MB with a maximum of 5 MB.

### Discussion

The HLS module available in NGINX Plus provides the ability to transmultiplex MP4 media files on the fly. There are many directives that give you control over how your media is fragmented and buffered. The location block must be configured to serve the media as an HLS stream with the HLS handler. The HLS fragmentation is set in number of seconds, instructing NGINX to fragment the media by time length. The amount of buffered data is set with the hls\_buffers directive specifying the number of buffers and the size. The client is allowed to start playback of the media after a cer‐ tain amount of buffering has accrued, specified by the hls\_mp4\_buffer\_size. How‐ ever, a larger buffer may be necessary because metadata about the video may exceed the initial buffer size. This amount is capped by the hls\_mp4\_max\_buffer\_size. These buffering variables allow NGINX to optimize the end-user experience. Choos‐ ing the right values for these directives requires knowing the target audience and your media. For instance, if the bulk of your media is large video files, and your target audience has high bandwidth, you may opt for a larger max buffer size and longer length fragmentation. This will allow for the metadata about the content to be downloaded initially without error and for your users to receive larger fragments.

## Streaming with HDS with NGINX Plus

### Problem

You need to support Adobe’s HTTP Dynamic Streaming (HDS) files that have already been fragmented and separated from the metadata.

### Solution

Use NGINX Plus’s support for fragmented FLV files via the F4F module to offer Adobe Adaptive Streaming to your users:

location /video/ {

alias /var/www/transformed\_video; f4f;

f4f\_buffer\_size 512k;

}

The example instructs NGINX Plus to serve previously fragmented media from a location on disk to the client using the NGINX Plus F4F module. The buffer size for the index file (.*f4x*) is set to 512 KB.

* 1. **Streaming with HDS with NGINX Plus | 97**

### Discussion

The NGINX Plus F4F module enables NGINX to serve previously fragmented media to end users. The configuration of such is as simple as using the f4f handler inside of an HTTP location block. The f4f\_buffer\_size directive configures the buffer size for the index file of this type of media.

## Bandwidth Limits with NGINX Plus

### Problem

You need to limit bandwidth to downstream media streaming clients without affect‐ ing the viewing experience.

### Solution

Utilize NGINX Plus’s bitrate-limiting support for MP4 media files:

location /video/ { mp4;

mp4\_limit\_rate\_after 15s; mp4\_limit\_rate 1.2;

}

This configuration allows the downstream client to download for 15 seconds before applying a bitrate limit. After 15 seconds, the client is allowed to download media at a rate of 120% of the bitrate, which enables the client to always download faster than they play.

### Discussion

NGINX Plus’s bitrate limiting allows your streaming server to limit bandwidth dynamically, based on the media being served, allowing clients to download just as much as they need to ensure a seamless user experience. The MP4 handler described in [Recipe 9.1](#_bookmark337) designates this location block to stream MP4 media formats. The rate- limiting directives, such as mp4\_limit\_rate\_after, tell NGINX to only rate-limit traffic after a specified amount of time, in seconds. The other directive involved in MP4 rate limiting is mp4\_limit\_rate, which specifies the bitrate at which clients are allowed to download in relation to the bitrate of the media. A value of 1 provided to the mp4\_limit\_rate directive specifies that NGINX is to limit bandwidth (1-to-1) to the bitrate of the media. Providing a value of more than 1 to the mp4\_limit\_rate directive will allow users to download faster than they watch so they can buffer the media and watch seamlessly while they download.

**CHAPTER 10**

# Cloud Deployments

## Introduction

The advent of cloud providers has changed the landscape of web application hosting. A process such as provisioning a new machine used to take anywhere from hours to months; now, you can create one with as little as a click or an API call. These cloud providers lease their virtual machines, called infrastructure as a service (IaaS), or managed software solutions such as databases, through a pay-per-usage model, which means you pay only for what you use. This enables engineers to build up entire environments for testing and tear them down when they’re no longer needed. These cloud providers also enable applications to scale horizontally based on performance need at a moment’s notice. This chapter covers basic NGINX and NGINX Plus deployments on a couple of the major cloud-provider platforms.

## Auto-Provisioning on AWS

### Problem

You need to automate the configuration of NGINX servers on Amazon Web Services (AWS) for machines to be able to automatically provision themselves.

### Solution

Utilize EC2 UserData as well as a prebaked Amazon Machine Image (AMI). Create an Amazon Machine Image with NGINX and any supporting software packages installed. Utilize Amazon Elastic Compute Cloud (EC2) UserData to configure any environment-specific configurations at runtime.

### Discussion

There are three patterns of thought when provisioning on AWS:

*Provision at boot*

Start from a common Linux image, then run configuration management or shell scripts at boot time to configure the server. This pattern is slow to start and can be prone to errors.

*Fully baked AMIs*

Fully configure the server, then burn an AMI to use. This pattern boots very fast and accurately. However, it’s less flexible to the environment around it, and maintaining many images can be complex.

*Partially baked AMIs*

It’s a mix of both worlds. Partially baked is where software requirements are installed and burned into an AMI, and environment configuration is done at boot time. This pattern is flexible compared to a fully baked pattern, and fast compared to a provision-at-boot solution.

Whether you choose to partially or fully bake your AMIs, you’ll want to automate that process. To construct an AMI build pipeline, it’s suggested to use a couple of tools:

*Configuration management*

Configuration management tools define the state of the server in code, such as what version of NGINX is to be run and what user it’s to run as, what DNS resolver to use, and who to proxy upstream to. This configuration management code can be source controlled and versioned like a software project. Some popular configuration management tools are Chef and Ansible, which were described in [Chapter 5](#_bookmark166).

*Packer from HashiCorp*

Packer is used to automate running your configuration management on virtually any virtualization or cloud platform and to burn a machine image if the run is successful. Packer basically builds a virtual machine (VM) on the platform of your choosing, then will SSH into the VM, run any provisioning you specify, and burn an image. You can utilize Packer to run the configuration management tool and reliably burn a machine image to your specification.

To provision environmental configurations at boot time, you can utilize the Amazon EC2 UserData to run commands the first time the instance is booted. If you’re using the partially baked method, you can utilize this to configure environment-based items at boot time. Examples of environment-based configurations might be what server names to listen for, resolver to use, domain name to proxy to, or upstream server pool to start with. UserData is a base64-encoded string that is downloaded at the first boot and run. UserData can be as simple as an environment file accessed by other bootstrapping processes in your AMI, or it can be a script written in any

language that exists on the AMI. It’s common for UserData to be a bash script that specifies variables, or downloads variables, to pass to configuration management. Configuration management ensures the system is configured correctly, templates configuration files based on environment variables, and reloads services. After UserData runs, your NGINX machine should be completely configured in a very reliable way.

## Routing to NGINX Nodes Without an AWS ELB

### Problem

You need to route traffic to multiple active NGINX nodes or create an active-passive failover set to achieve high availability without a load balancer in front of NGINX.

### Solution

Use the Amazon Route 53 DNS service to route to multiple active NGINX nodes or configure health checks and failover between an active-passive set of NGINX nodes.

### Discussion

DNS has balanced load between servers for a long time; moving to the cloud doesn’t change that. The Route 53 service from Amazon provides a DNS service with many advanced features, all available through an API. All the typical DNS tricks are available, such as multiple IP addresses on a single A record and weighted A records. When running multiple active NGINX nodes, you’ll want to use one of these A-record features to spread load across all nodes. The round-robin algorithm is used when multiple IP addresses are listed for a single A record. A weighted distribution can be used to distribute load unevenly by defining weights for each server IP address in an A record.

One of the more interesting features of Route 53 is its ability to health check. You can configure Route 53 to monitor the health of an endpoint by establishing a TCP connection or by making a request with HTTP or HTTPS. The health check is highly configurable with options for the IP, hostname, port, URI path, interval rates, monitoring, and geography. With these health checks, Route 53 can take an IP out of rotation if it begins to fail. You could also configure Route 53 to failover to a secondary record in case of a failure, which would achieve an active-passive, highly available setup.

Route 53 has a geological-based routing feature that will enable you to route your clients to the closest NGINX node to them, for the least latency. When routing by geography, your client is directed to the closest healthy physical location. When

running multiple sets of infrastructure in an active-active configuration, you can automatically failover to another geological location through the use of health checks.

When using Route 53 DNS to route your traffic to NGINX nodes in an Auto Scaling group, you’ll want to automate the creation and removal of DNS records. To automate adding and removing NGINX machines to Route 53 as your NGINX nodes scale, you can use Amazon’s Auto Scaling lifecycle hooks to trigger scripts within the NGINX box itself or scripts running independently on Amazon Lambda. These scripts would use the Amazon Command Line Interface (CLI) or software development kit (SDK) to interface with the Amazon Route 53 API to add or remove the NGINX machine IP and configured health check as it boots or before it is terminated.

### Also See

[“Global Server Load Balancing with Amazon Route 53 and NGINX Plus”](https://oreil.ly/IlhWg)

## The NLB Sandwich

### Problem

You need to autoscale your NGINX Open Source layer and distribute load evenly and easily between application servers.

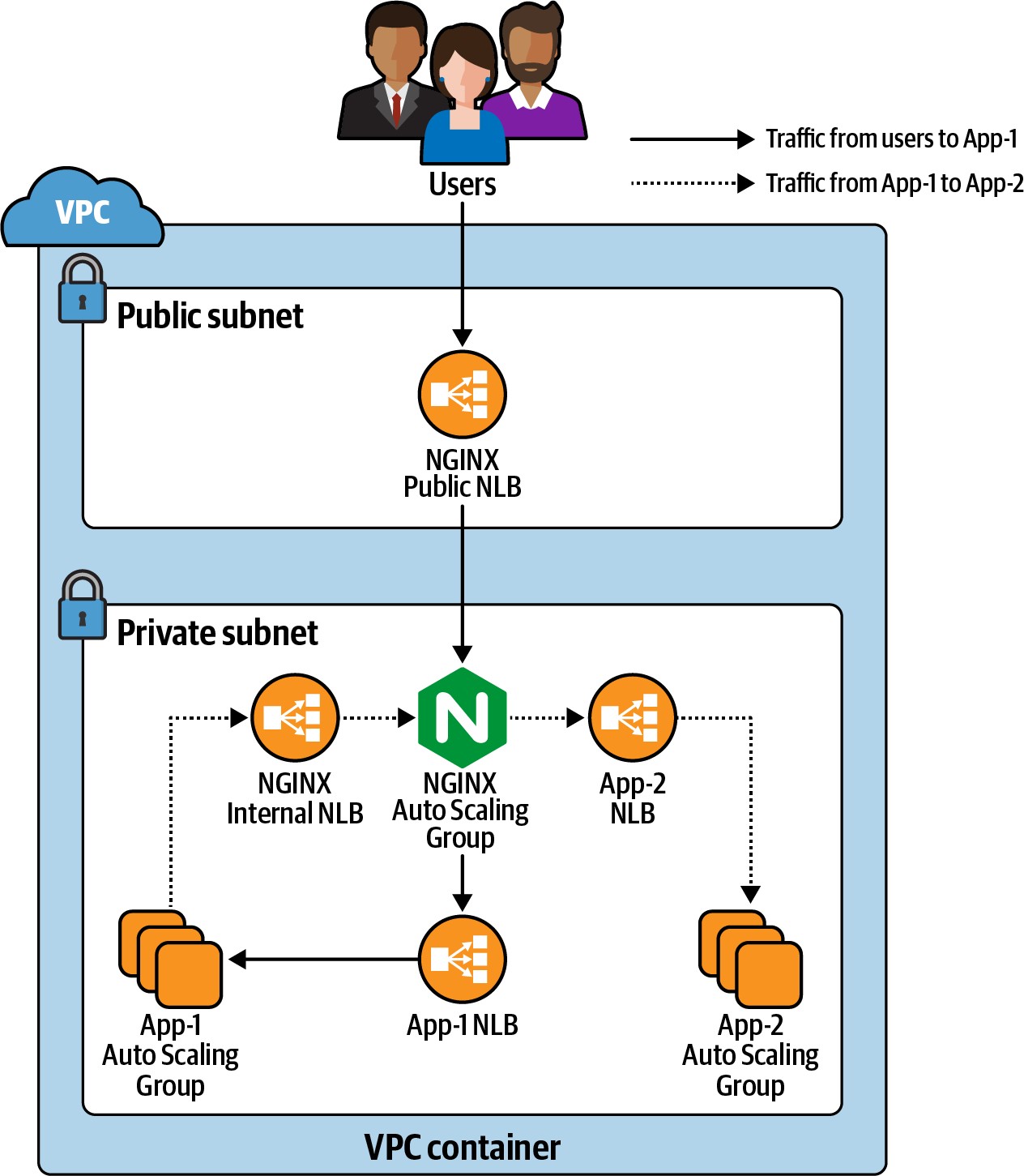
### Solution

Create a network load balancer (NLB). During creation of the NLB through the console, you are prompted to create a new target group. If you do not do this through the console, you will need to create this resource and attach it to a listener on the NLB. You create an Auto Scaling group with a launch configuration that provisions an EC2 instance with NGINX Open Source installed. The Auto Scaling group has a configuration to link to the target group, which automatically registers any instance in the Auto Scaling group to the target group configured on first boot. The target group is referenced by a listener on the NLB. Place your upstream applications behind another network load balancer and target group and then configure NGINX to proxy to the application NLB.

### Discussion

This common pattern is called the NLB sandwich (see [Figure 10-1](#_bookmark370)), putting NGINX Open Source in an Auto Scaling group behind an NLB and the application Auto Scaling group behind another NLB. The reason for having NLBs between every layer is because the NLB works so well with Auto Scaling groups; they automatically

register new nodes and remove those being terminated as well as run health checks and pass traffic to only healthy nodes.



*Figure 10-1. This image depicts NGINX in an NLB sandwich pattern with an internal NLB for internal applications to utilize. A user makes a request to App-1, and App-1 makes a request to App-2 through NGINX to fulfill the user’s request.*

It might be necessary to build a second, internal NLB for the NGINX Open Source layer because it allows services within your application to call out to other services through the NGINX Auto Scaling group without leaving the network and re-entering through the public NLB. This puts NGINX in the middle of all network traffic within your application, making it the heart of your application’s traffic routing. This pattern used to be called the *elastic load balancer sandwich*; however, the NLB is preferred when working with NGINX because the NLB is a Layer 4 load balancer, whereas

ELBs and ALBs are Layer 7 load balancers. Layer 7 load balancers transform the request via the PROXY Protocol and are redundant with the use of NGINX. This pattern is needed only for NGINX Open Source because the feature set provided by the NLB is available in NGINX Plus.

You can use this pattern within other cloud providers as well. The concept is the same if you’re using Azure load balancers and scale sets, or GCP load balancers and Auto Scaling groups. The core value of this pattern is to use cloud native services to provide automatic registration and load balancing of scaling application servers, and NGINX Open Source for proxy logic.

## Deploying from the AWS Marketplace

### Problem

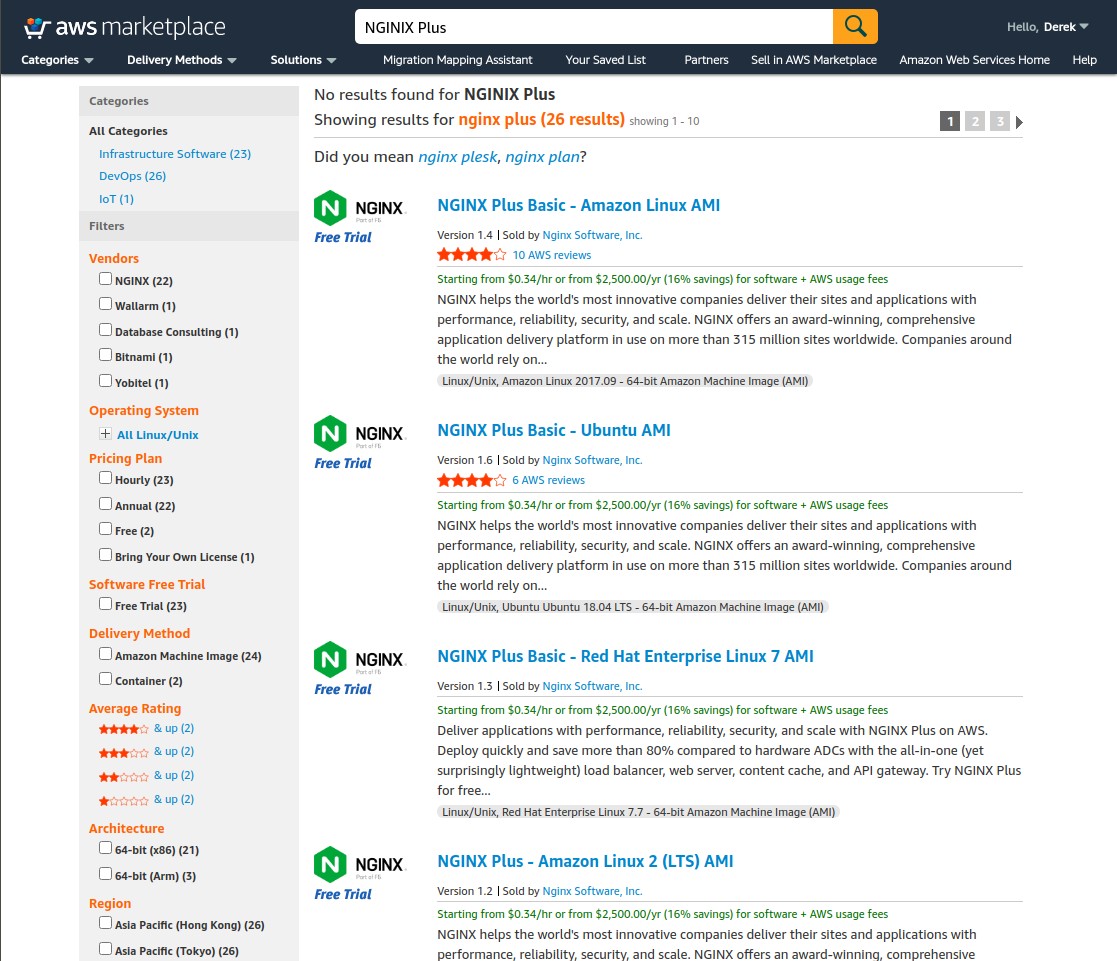
You need to run NGINX Plus in AWS with ease, with a pay-as-you-go license.

### Solution

Deploy through the AWS Marketplace. Visit the [AWS Marketplace](https://oreil.ly/SZrQr) and search “NGINX Plus” (see [Figure 10-2](#_bookmark377)). Select the AMI that is based on the Linux distri‐ bution of your choice; review the details, terms, and pricing; then click the Continue link. On the next page you’ll be able to accept the terms and deploy NGINX Plus with a single click, or accept the terms and use the AMI.

### Discussion

The AWS Marketplace solution to deploying NGINX Plus provides ease of use and a pay-as-you-go license. Not only do you have nothing to install, but you also have a license without jumping through hoops like getting a purchase order for a year license. This solution enables you to try NGINX Plus without commitment. You can also use the NGINX Plus Marketplace AMI as overflow capacity. It’s a common prac‐ tice to purchase your expected workload worth of licenses and use the Marketplace AMI in an Auto Scaling group as overflow capacity. This strategy ensures you only pay for as much licensing as you use.



*Figure 10-2. Searching for NGINX Plus on the AWS Marketplace*

## Creating an NGINX Virtual Machine Image on Azure

### Problem

You need to create a virtual machine (VM) image of your own NGINX server configured as you see fit to quickly create more servers or to use in scale sets.

### Solution

Create a VM from a Linux-based operating system of your choice. Once the VM is booted, log in and install NGINX or NGINX Plus in your preferred way, either from source or through the package management tool for the distribution you’re running. Configure NGINX as desired and create a new VM image. To create a VM image, you must first generalize the VM. To generalize your VM, you need to remove the user that Azure provisioned, connect to it over SSH, and run the following command:

$ sudo waagent -deprovision+user -force

* 1. **Creating an NGINX Virtual Machine Image on Azure | 105**

This command deprovisions the user that Azure provisioned when creating the VM. The -force option simply skips a confirmation step. After you’ve installed NGINX or NGINX Plus and removed the provisioned user, you can exit your session.

Connect your Azure CLI to your Azure account using the Azure login command, then ensure you’re using the Azure Resource Manager mode. Now deallocate your VM:

$ azure vm deallocate -g <ResourceGroupName> \

-n <VirtualMachineName>

Once the VM is deallocated, you will be able to generalize it with the azure vm generalize command:

$ azure vm generalize -g <ResourceGroupName> \

-n <VirtualMachineName>

After your VM is generalized, you can create an image. The following command will create an image and also generate an Azure Resources Manager (ARM) template for you to use to boot this image:

$ azure vm capture <ResourceGroupName> <VirtualMachineName> \

<ImageNamePrefix> -t <TemplateName>.json

The command line will produce output saying that your image has been created, that it’s saving an ARM template to the location you specified, and that the request is complete. You can use this ARM template to create another VM from the newly created image. However, to use this template Azure has created, you must first create a new network interface:

$ azure network nic create <ResourceGroupName> \

<NetworkInterfaceName> \

<Region> \

--subnet-name <SubnetName> \

--subnet-vnet-name <VirtualNetworkName>

This command output will detail information about the newly created network interface. The first line of the output data will be the network interface ID, which you will need to utilize the ARM template created by Azure. Once you have the ID, you can create a deployment with the ARM template:

$ azure group deployment create <ResourceGroupName> \

<DeploymentName> \

-f <TemplateName>.json

You will be prompted for multiple input variables such as vmName, adminUserName, adminPassword, and networkInterfaceId. Enter a name for the VM and the admin username and password. Use the network interface ID harvested from the last com‐ mand as the input for the networkInterfaceId prompt. These variables will be passed as parameters to the ARM template and used to create a new VM from the

custom NGINX or NGINX Plus image you’ve created. After entering the necessary parameters, Azure will begin to create a new VM from your custom image.

### Discussion

Creating a custom image in Azure enables you to create copies of your preconfigured NGINX or NGINX Plus server at will. An Azure ARM template enables you to quickly and reliably deploy this same server time and time again as needed. With the VM image path that can be found in the template, you can create different sets of infrastructure such as VM scaling sets or other VMs with different configurations.

### Also See

[“How to install the Azure CLI”](https://oreil.ly/cKEyU) [“Sign in with Azure CLI”](https://oreil.ly/Dh2uI)

[“How to create a managed image of a virtual machine or VHD”](https://oreil.ly/VSw5h)

## Load Balancing Over NGINX Scale Sets on Azure

### Problem

You need to scale NGINX nodes behind an Azure load balancer to achieve high availability and dynamic resource usage.

### Solution

Create an Azure load balancer that is either public-facing or internal. Deploy the NGINX VM image created in the prior section, or the NGINX Plus image from the Marketplace described in [Recipe 10.7](#_bookmark384), into an Azure virtual machine scale set (VMSS). Once your load balancer and VMSS are deployed, configure a backend pool on the load balancer to the VMSS. Set up load-balancing rules for the ports and protocols you’d like to accept traffic on, and direct them to the backend pool.

### Discussion

It’s common to scale NGINX to achieve high availability or to handle peak loads without overprovisioning resources. In Azure you achieve this with VMSS. Using the Azure load balancer provides ease of management for adding and removing NGINX nodes to the pool of resources when scaling. With Azure load balancers, you’re able to check the health of your backend pools and only pass traffic to healthy nodes. You can run internal Azure load balancers in front of NGINX where you want to enable access only over an internal network. You may use NGINX to proxy to an internal load balancer fronting an application inside of a VMSS, using the load balancer for the ease of registering and deregistering from the pool.

* 1. **Load Balancing Over NGINX Scale Sets on Azure | 107**

## Deploying Through the Azure Marketplace

### Problem

You need to run NGINX Plus in Azure with ease and a pay-as-you-go license.

### Solution

Deploy an NGINX Plus VM image through the Azure Marketplace:

* + 1. From the Azure dashboard, select the New icon, and use the search bar to search for “NGINX.” Search results will appear.
    2. From the list, select the NGINX Plus Virtual Machine Image published by NGINX Inc.
    3. When prompted to choose your deployment model, select the Resource Manager option, and click the Create button.
    4. You will then be prompted to fill out a form to specify the name of your VM, the disk type, the default username and password or SSH key-pair public key, which subscription to bill under, the resource group you’d like to use, and the location.
    5. Once this form is filled out, you can click OK. Your form will be validated.
    6. When prompted, select a VM size, and click the Select button.
    7. On the next panel, you have the option to select optional configurations, which will be the default based on your resource group choice made previously. After altering these options and accepting them, click OK.
    8. On the next screen, review the summary. You have the option of downloading this configuration as an ARM template so that you can create these resources again more quickly via a JSON template.
    9. Once you’ve reviewed and downloaded your template, you can click OK to move to the purchasing screen. This screen will notify you of the costs you’re about to incur from this VM usage. Click Purchase and your NGINX Plus box will begin to boot.

### Discussion

Azure and NGINX have made it easy to create an NGINX Plus VM in Azure through just a few configuration forms. The Azure Marketplace is a great way to get NGINX Plus on demand with a pay-as-you-go license. With this model, you can try out the features of NGINX Plus or use it for on-demand overflow capacity of your already licensed NGINX Plus servers.

## Deploying to Google Compute Engine

### Problem

You need to create an NGINX server in Google Compute Engine to load balance or proxy for the rest of your resources in Google Compute or App Engine.

### Solution

Start a new VM in Google Compute Engine. Select a name for your VM, zone, machine type, and boot disk. Configure identity and access management, firewall, and any advanced configuration you’d like. Create the VM.

Once the VM has been created, log in via SSH or through the Google Cloud Shell. Install NGINX or NGINX Plus through the package manager for the given OS type. Configure NGINX as you see fit and reload.

Alternatively, you can install and configure NGINX through the Google Compute Engine startup script, which is an advanced configuration option when creating a VM.

### Discussion

Google Compute Engine offers highly configurable VMs at a moment’s notice. Start‐ ing a VM takes little effort and enables a world of possibilities. Google Compute Engine offers networking and compute in a virtualized cloud environment. With a Google Compute instance, you have the full capabilities of an NGINX server wherever and whenever you need it.

## Creating a Google Compute Image

### Problem

You need to create a Google Compute Image to quickly instantiate a VM or create an instance template for an instance group.

### Solution

Create a VM as described in [Recipe 10.8](#_bookmark387). After installing and configuring NGINX on your VM instance, set the auto-delete state of the boot disk to false. To set the auto-delete state of the disk, edit the VM. On the Edit page under the disk configuration is a checkbox labeled “Delete boot disk when instance is deleted.” Deselect this checkbox and save the VM configuration. Once the auto-delete state of the instance is set to false, delete the instance. When prompted, do not select the

checkbox that offers to delete the boot disk. By performing these tasks, you will be left with an unattached boot disk with NGINX installed.

After your instance is deleted and you have an unattached boot disk, you can create a Google Compute Image. From the Image section of the Google Compute Engine console, select Create Image. You will be prompted for an image name, family, description, encryption type, and the source. The source type you need to use is disk; for the source disk, select the unattached NGINX boot disk. Select Create, and Google Compute Cloud will create an image from your disk.

### Discussion

You can utilize Google Cloud Images to create VMs with a boot disk identical to the server you’ve just created. The value in creating images is being able to ensure that every instance of this image is identical. When installing packages at boot time in a dynamic environment, unless using version locking with private repositories, you run the risk of package version and updates not being validated before being run in a production environment. With machine images, you can validate that every package running on this machine is exactly as you tested, strengthening the reliability of your service offering.

### Also See

[“Create, delete, and deprecate custom images”](https://oreil.ly/JtnYE)

## Creating a Google App Engine Proxy

### Problem

You need to create a proxy for Google App Engine to context switch between applica‐ tions or serve HTTPS under a custom domain.

### Solution

Utilize NGINX in Google Compute Cloud. Create a VM in Google Compute Engine, or create a virtual machine image with NGINX installed and create an instance template with this image as your boot disk. If you’ve created an instance template, follow up by creating an instance group that utilizes that template.

Configure NGINX to proxy to your Google App Engine endpoint. Make sure to proxy to HTTPS because Google App Engine is public, and you’ll want to ensure you do not terminate HTTPS at your NGINX instance and allow information to travel between NGINX and Google App Engine unsecured. Because App Engine provides just a single DNS endpoint, you’ll be using the proxy\_pass directive rather than

upstream blocks in the open source version of NGINX because the open source version is not able to resolve DNS names of upstream servers. When proxying to Google App Engine, make sure to set the endpoint as a variable in NGINX, then use that variable in the proxy\_pass directive to ensure NGINX does DNS resolution on every request. For NGINX to do any DNS resolution, you’ll need to also utilize the resolver directive and point to your favorite DNS resolver. Google makes the IP address 8.8.8.8 available for public use. If you’re using NGINX Plus, you’ll be able to use the resolve flag on the server directive within the upstream block, keepalive connections, and other benefits of the upstream module when proxying to Google App Engine.

You may choose to store your NGINX configuration files in Google Storage, then use the startup script for your instance to pull down the configuration at boot time. This will allow you to change your configuration without having to burn a new image. However, it will add to the startup time of your NGINX server.

### Discussion

You want to run NGINX in front of Google App Engine if you’re using your own domain and want to make your application available via HTTPS. At this time, Google App Engine does not allow you to upload your own SSL certificates. Therefore, if you’d like to serve your app under a domain other than appspot.com with encryption, you’ll need to create a proxy with NGINX to listen at your custom domain. NGINX will encrypt communication between itself and your clients, as well as between itself and Google App Engine.

Another reason you may want to run NGINX in front of Google App Engine is to host many App Engine apps under the same domain and use NGINX to do URI- based context switching. Microservices are a popular architecture, and it’s common for a proxy like NGINX to conduct the traffic routing. Google App Engine makes it easy to deploy applications, and in conjunction with NGINX, you have a full-fledged application delivery platform.

**CHAPTER 11**

# Containers/Microservices

## Introduction

Containers offer a layer of abstraction at the application layer, shifting the installation of packages and dependencies from the deploy to the build process. This is important because engineers are now shipping units of code that run and deploy in a uniform way regardless of the environment. Promoting containers as runnable units reduces the risk of dependency and configuration snafus between environments. Given this, there has been a large drive for organizations to deploy their applications on con‐ tainer platforms. When running applications on a container platform, it’s common to containerize as much of the stack as possible, including your proxy or load balancer. NGINX and NGINX Plus containerize and ship with ease. They also include many features that make delivering containerized applications fluid. This chapter focuses on building NGINX and NGINX Plus container images, features that make working in a containerized environment easier, and deploying your image on Kubernetes and OpenShift.

When containerizing, it’s often common to decompose services into smaller applica‐ tions. When doing so, they’re tied back together by an API gateway. The first section in this chapter provides a common case scenario of using NGINX as an API gateway to secure, validate, authenticate, and route requests to the appropriate service. The last section, while more of a security topic, discusses NGINX Service Mesh, an infrastructure pattern used in containerized environments to optimize and secure connections between such aforementioned services.

A couple of architecture considerations about running NGINX or NGINX Plus in a container should be called out. When containerizing a service, to make use of the Docker log driver, logs must be output to */dev/stdout* and error logs directed

to */dev/stderr*. By doing so, the logs are streamed to the Docker log driver, which is able to route them to consolidated logging servers natively.

Load-balancing methods are also of consideration when using NGINX Plus in a containerized environment. The least\_time load-balancing method was designed with containerized networking overlays in mind. By favoring low response time, NGINX Plus will pass the incoming request to the upstream server with the fastest average response time. When all servers are adequately load balanced and performing equally, NGINX Plus can optimize by network latency, preferring servers in closest network proximity.

## Using NGINX as an API Gateway

### Problem

You need an API gateway to validate, authenticate, manipulate, and route incoming requests for your use case.

### Solution

Use NGINX or NGINX Plus as an API gateway. An API gateway provides an entry point to one or more application programming interfaces (APIs). NGINX fits this role very well. This section will highlight some core concepts and reference other sections within this book for more detail on specifics. It’s also important to note that NGINX has published an entire ebook on this topic: [*Deploying NGINX as an API*](https://oreil.ly/75l-m)[*Gateway* by Liam Crilly](https://oreil.ly/75l-m).

Start by defining a server block for your API gateway within its own file. A name such as */etc/nginx/api\_gateway.conf* will do.

server {

listen 443 ssl;

server\_name api.company.com; # SSL Settings Chapter 7

default\_type application/json;

}

Add some basic error-handling responses to your server definition:

proxy\_intercept\_errors on;

error\_page 400 = @400;

location @400 { return 400 '{"status":400,"message":"Bad request"}\n'; }

error\_page 401 = @401;

location @401 { return 401 '{"status":401,"message":"Unauthorized"}\n'; }

error\_page 403 = @403;

location @403 { return 403 '{"status":403,"message":"Forbidden"}\n'; }

error\_page 404 = @404;

location @404 { return 404 '{"status":404,"message":"Resource not found"}\n'; }

The preceding section of NGINX configuration can be added directly to the server block in */etc/nginx/api\_gateway.conf* or a separate file, and imported via an include directive. The include directive is covered in [Recipe 18.1](#_bookmark618).

Use an include directive to import this server configuration into the main *nginx.conf*

file within the http context:

include /etc/nginx/api\_gateway.conf;

You now need to define your upstream service endpoints. [Chapter 2](#_bookmark35) covers load balancing, which discusses the upstream block. As a reminder, upstream is valid within the http context, and not within server context. The following must be included or set outside of the server block:

upstream service\_1 { server 10.0.0.12:80;

server 10.0.0.13:80;

}

upstream service\_2 { server 10.0.0.14:80;

server 10.0.0.15:80;

}

Depending on the use case, you may want to declare your services inline, as an included file, or included per services. A case also exists where services should be defined as proxy location endpoints; in this case it’s suggested to define the endpoint as a variable for use throughout. [Chapter 5, “Programmability and Automation”](#_bookmark166), discusses ways to automate adding and removing machines from upstream blocks.

Build an internally routable location within the server block for each service:

location = /\_service\_1 { internal;

# Config common to service

proxy\_pass http://service\_1/$request\_uri;

}

location = /\_service\_2 { internal;

# Config common to service

proxy\_pass http://service\_2/$request\_uri;

}

By defining internal routable locations for these services, configuration that is com‐ mon to the service can be defined once, rather than repeatedly.

From here, we need to build up location blocks that define specific URI paths for a given service. These blocks will validate and route the request appropriately. An API gateway can be as simple as routing requests based on path, and as detailed as defining specific rules for every single accepted API URI. In the latter, you’ll want to devise a file structure for organization and use NGINX includes to import your configuration files. This concept is discussed in [Recipe 18.1](#_bookmark618).

Create a new directory for the API gateway:

mkdir /etc/nginx/api\_conf.d/

Build a specification of a service use case by defining location blocks within a file at a path that makes sense for your configuration structure. Use the rewrite directive to direct the request to the prior configured location block that proxies the request to a service. The rewrite directive used in the following example instructs NGINX to reprocess the request with an altered URI. The example defines rules specific to an API resource, restricts HTTP methods, then uses the rewrite directive to send the request to the prior defined internal common proxy location for the service:

location /api/service\_1/object { limit\_except GET PUT { deny all; } rewrite ^ /\_service\_1 last;

}

location /api/service\_1/object/[^/]\*$ { limit\_except GET POST { deny all; } rewrite ^ /\_service\_1 last;

}

Repeat this step for each service. Employ logical separation by means of file and directory structures to organize effectively for the use case. Use any and all informa‐ tion provided in this book to configure API location blocks to be as specific and restrictive as possible.

If separate files were used for the preceding location or upstream blocks, ensure they’re included in your server context:

server {

listen 443 ssl;

server\_name api.company.com; # SSL Settings Chapter 7

default\_type application/json;

include api\_conf.d/\*.conf;

}

Enable authentication to protect private resources by using one of the many methods discussed in [Chapter 6](#_bookmark213), or something as simple as preshared API keys as follows (note the map directive is only valid in the http context):

map $http\_apikey $api\_client\_name { default "";

"j7UqLLB+yRv2VTCXXDZ1M/N4" "client\_one"; "6B2kbyrrTiIN8S8JhSAxb63R" "client\_two"; "KcVgIDSY4Nm46m3tXVY3vbgA" "client\_three";

}

Protect backend services from attack with NGINX by employing learnings from [Chapter 2](#_bookmark35) to limit usage. In the http context, define one or many request limit shared memory zones:

limit\_req\_zone $http\_apikey zone=limitbyapikey:10m rate=100r/s;

limit\_req\_status 429;

Protect a given context with rate limits and authentication:

location /api/service\_2/object { limit\_req zone=limitbyapikey;

# Consider writing these if's to a file # and using an include were needed.

if ($http\_apikey = "") { return 401;

}

if ($api\_client\_name = "") { return 403;

}

limit\_except GET PUT { deny all; } rewrite ^ /\_service\_2 last;

}

Test out some calls to your API gateway:

curl -H "apikey: 6B2kbyrrTiIN8S8JhSAxb63R" **\**

https://api.company.com/api/service\_2/object

### Discussion

API gateways provide an entry point to an application programming interface (API). That sounds vague and basic, so let’s dig in. Integration points happen at many different layers. Any two independent services that need to communicate (integrate) should hold an API version contract. Such version contracts define the compatibility of the services. An API gateway enforces such contracts—authenticating, authorizing, transforming, and routing requests between services.

This section demonstrated how NGINX can function as an API gateway by validat‐ ing, authenticating, and directing incoming requests to specific services and limiting

their usage. This tactic is popular in microservice architectures, where a single API offering is split among different services.

Employ all of your learnings thus far to construct an NGINX server configuration to the exact specifications for your use case. By weaving together the core concepts demonstrated in this text, you have the ability to authenticate and authorize the use of URI paths, route or rewrite requests based on any factor, limit usage, and define what is and is not accepted as a valid request. There will never be a single solution to an API gateway, as each is intimately and infinitely definable to the use case it provides.

An API gateway provides an ultimate collaboration space between operations and application teams to form a true DevOps organization. Application development defines validity parameters of a given request. Delivery of such a request is typically managed by what is considered IT (networking, infrastructure, security, and middle‐ ware teams). An API gateway acts as an interface between those two layers. The construction of an API gateway requires input from all sides. Configuration of such should be kept in some sort of source control. Many modern-day source-control repositories have the concept of code owners. This concept allows you to require specific users’ approval for certain files. In this way, teams can collaborate but verify changes specific to a given department.

Something to keep in mind when working with API gateways is the URI path. In the example configuration, the entire URI path is passed to the upstream servers. This means the service\_1 example needs to have handlers at the /api/service\_1/\* path. To perform path-based routing in this way, it’s best that the application doesn’t have conflicting routes with another application.

If conflicting routes do apply, there are a few things you can do. Edit the code to resolve the conflicts, or add a URI prefix configuration to one or both applications to move one of them to another context. In the case of off-the-shelf software that can’t be edited, you can rewrite the requests URI upstream. However, if the application returns links in the body, you’ll need to use regular expressions (regex) to rewrite the body of the request before providing it to the client—this should be avoided.

### Also See

[*Deploying NGINX as an API Gateway* ebook](https://oreil.ly/75l-m)

## Using DNS SRV Records with NGINX Plus

### Problem

You’d like to use your existing DNS SRV record implementation as the source for upstream servers with NGINX Plus.

### Solution

Specify the service directive with a value of http on an upstream server to instruct NGINX to utilize the SRV record as a load-balancing pool:

http {

resolver 10.0.0.2 valid=30s;

upstream backend { zone backends 64k;

server api.example.internal service=http resolve;

}

}

This feature is an NGINX Plus exclusive. The configuration instructs NGINX Plus to resolve DNS from a DNS server at 10.0.0.2 and set up an upstream server pool with a single server directive. This server directive specified with the resolve parameter is instructed to periodically re-resolve the domain name base on the DNS record TTL, or the valid override parameter of the resolver directive. The service=http parameter and value tells NGINX that this is an SRV record containing a list of IPs and ports, and to load balance over them as if they were configured with the server directive.

### Discussion

Dynamic infrastructure is becoming ever more popular with the demand and adop‐ tion of cloud-based infrastructure. Auto Scaling environments scale horizontally, increasing and decreasing the number of servers in the pool to match the demand of the load. Scaling horizontally demands a load balancer that can add and remove resources from the pool. With an SRV record, you offload the responsibility of keeping the list of servers to DNS. This type of configuration is extremely enticing for containerized environments because you may have containers running applications on variable port numbers, possibly at the same IP address. It’s important to note that UDP DNS record payload is limited to about 512 bytes.

## Using the Official NGINX Image

### Problem

You need to get up and running quickly with the NGINX image from Docker Hub.

### Solution

Use the NGINX image from Docker Hub. This image contains a default configura‐ tion. You’ll need to either mount a local configuration directory or create a Dockerfile and ADD in your configuration to the image build to alter the configuration. Here

* 1. **Using the Official NGINX Image | 119**

we mount a volume where NGINX’s default configuration serves static content to demonstrate its capabilities by using a single command:

$ docker run --name my-nginx -p 80:80 **\**

-v /path/to/content:/usr/share/nginx/html:ro -d nginx

The docker command pulls the nginx:latest image from Docker Hub if it’s not found locally. The command then runs this NGINX image as a Docker container, mapping localhost:80 to port 80 of the NGINX container. It also mounts the local directory */path/to/content/* as a container volume at */usr/share/nginx/html/* as read only. The default NGINX configuration will serve this directory as static content. When specifying mapping from your local machine to a container, the local machine port or directory comes first, and the container port or directory comes second.

### Discussion

NGINX has made an official Docker image available via Docker Hub. This official Docker image makes it easy to get up and going very quickly in Docker with your favorite application delivery platform, NGINX. In this section, we were able to get NGINX up and running in a container with a single command! The official NGINX Docker image mainline that we used in this example is built from the Debian Jessie Docker image. However, you can choose official images based on Alpine Linux. The Dockerfile and source for these official images are available on GitHub. You can extend the official image by building your own Dockerfile and specifying the official image in the FROM command. You can also mount an NGINX configuration directory as a Docker volume to override the NGINX configuration without modifying the official image.

### Also See

[Official NGINX Docker image, NGINX](https://oreil.ly/8zvNE) [Docker repo on GitHub](https://oreil.ly/oUpJ9)

## Creating an NGINX Dockerfile

### Problem

You need to create an NGINX Dockerfile in order to create a Docker image.

### Solution

Start with a FROM line of your favorite distribution’s Docker image. Use the RUN com‐ mand to install NGINX. Use the ADD command to add your NGINX configuration files. Use the EXPOSE command to instruct Docker to expose given ports, or do this manually when you run the image as a container. Use CMD to start NGINX when the

image is instantiated as a container. You’ll need to run NGINX in the foreground. To do this, you’ll need to start NGINX with -g "daemon off;" or add daemon off; to your configuration. This example will use the latter with daemon off; in the configuration file within the main context. You will also want to alter your NGINX configuration to log to */dev/stdout* for access logs and */dev/stderr* for error logs; doing so will put your logs into the hands of the Docker daemon, which will make them more easily available, based on the log driver you’ve chosen to use with Docker:

**FROM** centos:7

*# Install epel repo to get nginx and install nginx*

**RUN** yum -y install epel-release && **\**

yum -y install nginx

*# add local configuration files into the image*

**ADD** /nginx-conf /etc/nginx

**EXPOSE** 80 443

**CMD** ["nginx"]

The directory structure looks as follows:

.

├── 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

I chose to host the entire NGINX configuration within this Docker directory for ease of access to all of the configurations with only one line in the Dockerfile to add all my NGINX configurations.

### Discussion

You will find it useful to create your own Dockerfile when you require full control over the packages installed and updates. It’s common to keep your own repository of images so that you know your base image is reliable and tested by your team before running it in production.

## Building an NGINX Plus Docker Image

### Problem

You need to build an NGINX Plus Docker image to run NGINX Plus in a container‐ ized environment.

### Solution

Use this Dockerfile to build an NGINX Plus Docker image. You’ll need to download your NGINX Plus repository certificates, named *nginx-repo.crt* and *nginx-repo.key*, respectively, and keep them in the directory with this Dockerfile. With that, this Dockerfile will do the rest of the work of installing NGINX Plus for your use and linking NGINX access and error logs to the Docker log collector.

**FROM** debian:stretch-slim

**LABEL** maintainer="NGINX [<docker-maint@nginx.com>"](mailto:docker-maint@nginx.com)

*# Download certificate and key from the customer portal # (https://cs.nginx.com) and copy to the build context*

**COPY** nginx-repo.crt /etc/ssl/nginx/

**COPY** nginx-repo.key /etc/ssl/nginx/

*# Install 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/\*

*# Forward request logs to Docker log collector*

**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;"]

To build this Dockerfile into a Docker image, run the following in the directory that contains the Dockerfile and your NGINX Plus repository certificate and key:

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

This docker build command uses the flag --no-cache to ensure that whenever you build this, the NGINX Plus packages are pulled fresh from the NGINX Plus repository for updates. If it’s acceptable to use the same version on NGINX Plus as the prior build, you can omit the --no-cache flag. In this example, the new Docker image is tagged nginxplus.

### Discussion

By creating your own Docker image for NGINX Plus, you can configure your NGINX Plus container however you see fit and drop it into any Docker environment. This opens up all of the power and advanced features of NGINX Plus to your containerized environment. This Dockerfile does not use the Dockerfile property ADD to add in your configuration; you will need to add in your configuration manually.

### Also See

[“Deploying NGINX and NGINX Plus with Docker”](https://oreil.ly/AJSVc)

## Using Environment Variables in NGINX

### Problem

You need to use environment variables inside your NGINX configuration in order to use the same container image for different environments.

### Solution

Use the ngx\_http\_perl\_module to set variables in NGINX from your environment:

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;

}

}

}

To use perl\_set you must have the ngx\_http\_perl\_module installed; you can do so by loading the module dynamically or statically if building from source. NGINX by default wipes environment variables from its environment; you need to declare any variables you do not want removed with the env directive. The perl\_set directive takes two parameters: the variable name you’d like to set and a Perl string that renders the result.

The following is a Dockerfile that loads the ngx\_http\_perl\_module dynamically, installing this module from the package management utility. When installing mod‐ ules from the package utility for CentOS, they’re placed in the */usr/lib64/nginx/mod‐ ules/* directory, and configuration files that dynamically load these modules are placed in the */usr/share/nginx/modules/* directory. This is why in the preceding configuration snippet we include all configuration files at that path:

**FROM** centos:7

*# Install epel repo to get nginx and install nginx*

**RUN** yum -y install epel-release && **\**

yum -y install nginx nginx-mod-http-perl

*# add local configuration files into the image*

**ADD** /nginx-conf /etc/nginx

**EXPOSE** 80 443

**CMD** ["nginx"]

### Discussion

A typical practice when using Docker is to utilize environment variables to change the way the container operates. You can use environment variables in your NGINX configuration so that your NGINX Dockerfile can be used in multiple, diverse envi‐ ronments.

## Kubernetes Ingress Controller

### Problem

You are deploying your application on Kubernetes and need an ingress controller.

### Solution

Ensure that you have access to the ingress controller image. For NGINX, you can use the *nginx/nginx-ingress* image from Docker Hub. For NGINX Plus, you will need to build your own image and host it in your private Docker registry. You can find instructions on building and pushing your own NGINX Plus Kubernetes Ingress Controller in the [NGINX documentation](https://oreil.ly/Ut2lo).

Visit the Kubernetes Ingress Controller Deployments folder in the [kubernetes-ingress](https://oreil.ly/KxF7i) [repository on GitHub](https://oreil.ly/KxF7i). The commands that follow will be run from within this directory of a local copy of the repository.

Create a namespace and a service account for the ingress controller; both are named

nginx-ingress:

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

Create a secret with a TLS certificate and key for the ingress controller:

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

This certificate and key are self-signed and created by NGINX Inc. for testing and example purposes. It’s recommended to use your own because this key is publicly available.

Optionally, you can create a config map for customizing NGINX configuration (the config map provided is blank; however, you can read more about customization of [ConfigMaps](https://oreil.ly/MgcTH) and [Annotations](https://oreil.ly/sVL1Z)):

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

If Role-Based Access Control (RBAC) is enabled in your cluster, create a cluster role and bind it to the service account. You must be a cluster administrator to perform this step:

$ kubectl apply -f rbac/rbac.yaml

Now deploy the ingress controller. Two example deployments are made available in this repository: a Deployment and a DaemonSet. Use a Deployment if you plan to dynamically change the number of ingress controller replicas. Use a DaemonSet to deploy an ingress controller on every node or a subset of nodes.

If you plan to use the NGINX Plus Deployment manifests, you must alter the YAML file and specify your own registry and image.

For NGINX Deployment:

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

For NGINX Plus Deployment:

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

For NGINX DaemonSet:

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

For NGINX Plus DaemonSet:

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

Validate that the ingress controller is running:

$ kubectl get pods --namespace=nginx-ingress

If you created a DaemonSet, ports 80 and 443 of the ingress controller are mapped to the same ports on the node where the container is running. To access the ingress controller, use those ports and the IP address of any of the nodes on which the ingress controller is running. If you deployed a Deployment, continue with the next steps.

For the Deployment methods, there are two options for accessing the ingress control‐ ler pods. You can instruct Kubernetes to randomly assign a node port that maps to the ingress controller pod. This is a service with the type NodePort. The other option is to create a service with the type LoadBalancer. When creating a service of type LoadBalancer, Kubernetes builds a load balancer for the given cloud platform, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Compute.

To create a service of type NodePort, use the following:

$ kubectl create -f service/nodeport.yaml

To statically configure the port that is opened for the pod, alter the YAML and add the attribute nodePort: {port} to the configuration of each port being opened.

To create a service of type LoadBalancer for Google Cloud Compute or Azure, use this code:

$ kubectl create -f service/loadbalancer.yaml

To create a service of type LoadBalancer for Amazon Web Services:

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

On AWS, Kubernetes creates a classic ELB in TCP mode with the PROXY Protocol enabled. You must configure NGINX to use the PROXY Protocol. To do so, you can add the following to the config map mentioned previously in reference to the file *common/nginx-config.yaml*:

**proxy-protocol**: "True"

**real-ip-header**: "proxy\_protocol"

**set-real-ip-from**: "0.0.0.0/0"

Then, update the config map:

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

You can now address the pod by its NodePort or by making a request to the load balancer created on its behalf.

### Discussion

As of this writing, Kubernetes is the leading platform in container orchestration and management. The ingress controller is the edge pod that routes traffic to the rest of your application. NGINX fits this role perfectly and makes it simple to configure with its annotations. The NGINX Ingress project offers an NGINX Open Source ingress controller out of the box from a Docker Hub image, and NGINX Plus through a few steps to add your repository certificate and key. Enabling your Kubernetes cluster with an NGINX Ingress controller provides all the same features of NGINX but with the added features of Kubernetes networking and DNS to route traffic.

## Prometheus Exporter Module

### Problem

You are deploying NGINX into an environment using Prometheus monitoring and need NGINX statistics.

### Solution

Use the NGINX Prometheus Exporter to harvest NGINX or NGINX Plus statistics and ship them to Prometheus.

The NGINX Prometheus Exporter Module is written in GoLang and distributed as a [binary on GitHub](https://oreil.ly/TmUEo) and can be found as a prebuilt [Docker Image on Docker Hub](https://oreil.ly/mC_i9).

By default, the exporter will be started for NGINX and will only harvest the [stub](https://oreil.ly/rnEgI)

[\_status](https://oreil.ly/rnEgI) information. To run the exporter for NGINX Open Source, ensure stub status is enabled (if it’s not, there is more information on how to do so in [Recipe](#_bookmark501) [13.1](#_bookmark501)). Then use the following Docker command:

docker run -p 9113:9113 nginx/nginx-prometheus-exporter:0.8.0 **\**

-nginx.scrape-uri http://{nginxEndpoint}:8080/stub\_status

To use the exporter with NGINX Plus, a flag must be used to switch the exporter’s context because much more data can be collected from the NGINX Plus API. You can learn how to turn on the NGINX Plus API in [Recipe 13.2](#_bookmark505). Use the following Docker command to run the exporter for an NGINX Plus environment:

docker run -p 9113:9113 nginx/nginx-prometheus-exporter:0.8.0 **\**

-nginx.plus -nginx.scrape-uri http://{nginxPlusEndpoint}:8080/api

### Discussion

Prometheus is an extremely common metric monitoring solution that is very preva‐ lent in the Kubernetes ecosystem. The NGINX Prometheus Exporter Module is a fairly simple component; however, it enables prebuilt integration between NGINX and common monitoring platforms. With NGINX, the stub status does not provide a vast amount of data, but important data to provide insight into the amount of work an NGINX node is handling. The NGINX Plus API enables many more statistics about the NGINX Plus server, all of which the exporter ships to Prometheus. With either case, the information gleaned is valuable monitoring data, and the work to ship this data to Prometheus is already done; you just need to wire it up and take advantage of the insight provided by NGINX statistics.

### Also See

[NGINX Prometheus Exporter GitHub](https://oreil.ly/WaUDA) [NGINX stub\_status module documentation](https://oreil.ly/vtP6k) [NGINX Plus API module documentation](https://oreil.ly/K6Rif) [NGINX Plus Monitoring Dashboard](https://oreil.ly/55IGt)

## NGINX Service Mesh mTLS

### Problem

You would like to enable mutual TLS (mTLS) authentication between services using NGINX Secure Service Mesh.

### Solution

Use the nginx-meshcli tool to enable mTLS in permissive mode while validating:

nginx-meshctl deploy ... --mtls-mode permissive

Configure proper Public Key Infrastructure (PKI) by creating a YAML configuration file specifying a root certificate authority from disk. SPIRE will use this configuration for its upstream authorities:

**apiVersion**: v1 **upstreamAuthority**: disk **config**:

**cert\_file\_path**: /path/to/rootCA.crt

**key\_file\_path**: /path/to/rootCA.key

If you’re deploying on AWS, use the AWS Secrets Manager plug-in to pull the certificate authority certificate and key from AWS Secrets Manager:

**apiVersion**: "v1" **upstreamAuthority**: "awssecret" **config**:

**region**: "us-east-1"

**cert\_file\_arn**: "arn:aws:secretsmanager:us-east-1:123456789012:secret:

/certificate-authority/test-certificate"

**key\_file\_arn**: "arn:aws:secretsmanager:us-east-1:123456789012:secret:

/certificate-authority/test-key"

Deploy the mTLS configuration to inform SPIRE of the upstream authorities using

nginx-meshcli:

nginx-meshctl deploy ... --mtls-upstream-ca-conf /path/to/upstream\_authority.yaml

Verify your deployment:

kubectl get pods -n nginx-mesh

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | READY | STATUS | RESTARTS | AGE |
| ... |  |  |  |  |
| spire-agent-sv2tv | 1/1 | Running | 0 | 2h |
| spire-server-0 | 2/2 | Running | 0 | 2h |
| ... |  |  |  |  |

Once you’ve validated your mTLS configuration, and that your applications are working as expected, it’s recommended to enable the strict mTLS mode:

nginx-meshctl deploy ... --mtls-mode strict

### Discussion

A service mesh is a dedicated infrastructure layer designed to handle communication between distributed applications. The NGINX Service Mesh utilizes a service-level sidecar pattern in Kubernetes. The NGINX sidecar handles interservice communica‐ tion, monitoring, and security. NGINX Service Mesh and the nginx-meshctl tool

**11.9 NGINX Service Mesh mTLS | 129**

make deploying a service mesh for your environment extremely straightforward. Sidecar injection is used to automatically configure an NGINX sidecar for each service, allowing you to enable this functionality and continue focusing on your core service offering rather than the extensive configuration required to implement such an infrastructure layer from scratch.

This section focused specifically on enabling mTLS. mTLS provides authentication between client and server—the client verifies the server’s identity and the server verifies the client’s identity through a connection that is encrypted by both parties. In our service mesh, any given service could also be the client of another service, as the services may communicate between each other.

This section instructed you to enable NGINX Service Mesh in permissive mode, which will allow for communication to fall back to regular communication if miscon‐ figured. At this stage, SPIRE, the service responsible for signing and distributing the certificates and keys, defaulted to generating a self-signed certificate. The con‐ figuration was then strengthened by providing a specific certificate authority (CA) certificate and key pair. This section showed two ways of providing the CA key pair: one directly from disk, then another through AWS Secrets Manager. If you happen to be using AWS Certificate Manager for a private CA, you could also instruct SPIRE to use AWS ACM to sign service certificates. Once configured, you validated your configuration, then enabled mTLS in strict mode, which requires all interservice communication to be authenticated through mTLS.

### Also See

[“The mTLS Architecture in NGINX Service Mesh”](https://oreil.ly/CWEU8) [“Secure Mesh Traffic using mTLS”](https://oreil.ly/VJDge)

[*The Enterprise Path to Service Mesh Architectures* (O’Reilly)](https://oreil.ly/FAeeA)

**CHAPTER 12**

# High-Availability Deployment Modes

## Introduction

Fault-tolerant architecture separates systems into identical, independent stacks. Load balancers like NGINX are employed to distribute load, ensuring that what’s provi‐ sioned is utilized. The core concepts of high availability are load balancing over multiple active nodes or an active-passive failover. Highly available applications have no single points of failure; every component must use one of these concepts, includ‐ ing the load balancers themselves. For us, that means NGINX. NGINX is designed to work in either configuration: multiple active or active-passive failover. This chapter details techniques on how to run multiple NGINX servers to ensure high availability in your load-balancing tier.

## NGINX Plus HA Mode

### Problem

You need a highly available (HA) load-balancing solution.

### Solution

Use NGINX Plus’s HA mode with keepalived by installing the nginx-ha-keepalived

package from the NGINX Plus repository.

### Discussion

The nginx-ha-keepalived package is based on keepalived and manages a virtual IP address exposed to the client. Another process is run on the NGINX server that ensures that NGINX Plus and the keepalived process are running. Keepalived

is a process that utilizes the Virtual Router Redundancy Protocol (VRRP), sending small messages, often referred to as heartbeats, to the backup server. If the backup server does not receive the heartbeat for three consecutive periods, the backup server initiates the failover, moving the virtual IP address to itself and becoming the pri‐ mary. The failover capabilities of nginx-ha-keepalived can be configured to identify custom failure situations.

## Load-Balancing Load Balancers with DNS

### Problem

You need to distribute load between two or more NGINX servers.

### Solution

Use DNS to round robin across NGINX servers by adding multiple IP addresses to a DNS A record.

### Discussion

When running multiple load balancers, you can distribute load via DNS. The A record allows for multiple IP addresses to be listed under a single FQDN. DNS will automatically round robin across all the IPs listed. DNS also offers weighted round robin with weighted records, which works in the same way as weighted round robin in NGINX as described in [Chapter 2](#_bookmark35). These techniques work great. However, a pitfall can be removing the record when an NGINX server encounters a failure. There are DNS providers—Amazon Route 53 for one, and Dyn DNS for another—that offer health checks and failover with their DNS offering, which alleviates these issues. If you are using DNS to load balance over NGINX, when an NGINX server is marked for removal, it’s best to follow the same protocols that NGINX does when removing an upstream server. First, stop sending new connections to it by removing its IP from the DNS record, then allow connections to drain before stopping or shutting down the service.

## Load Balancing on EC2

### Problem

You’re using NGINX on AWS, and the NGINX Plus HA does not support Amazon IPs.

### Solution

Put NGINX behind an AWS NLB by configuring an Auto Scaling group of NGINX servers and linking the Auto Scaling group to a target group, and then attach the target group to the NLB. Alternatively, you can place NGINX servers into the target group manually by using the AWS console, command-line interface, or API.

### Discussion

The HA solution from NGINX Plus based on keepalived will not work on AWS because it does not support the floating virtual IP address, since EC2 IP addresses work in a different way. This does not mean that NGINX can’t be HA in the AWS cloud; in fact, the opposite is true. The AWS NLB is a product offering from Ama‐ zon that will natively load balance over multiple, physically separated data centers called *availability zones*, provide active health checks, and provide a DNS CNAME endpoint. A common solution for HA NGINX on AWS is to put an NGINX layer behind the NLB. NGINX servers can be automatically added to and removed from the target group as needed. The NLB is not a replacement for NGINX; there are many things NGINX offers that the NLB does not, such as multiple load-balancing methods, rate limiting, caching, and Layer 7 routing. The AWS ALB does perform Layer 7 load balancing based on the URI path and host header, but it does not by itself offer features that NGINX does, such as WAF caching, bandwidth limiting, HTTP/2 server push, and more. In the event that the NLB does not fit your need, there are many other options. One option is the DNS solution: Route 53 from AWS offers health checks and DNS failover.

## NGINX Plus Configuration Synchronization

### Problem

You’re running an HA NGINX Plus tier and need to synchronize configuration across servers.

### Solution

Use the NGINX Plus exclusive configuration synchronization feature. To configure this feature, follow these steps.

Install the nginx-sync package from the NGINX Plus package repository. For RHEL or CentOS:

$ sudo yum install nginx-sync

For Ubuntu or Debian:

$ sudo apt-get install nginx-sync

Grant the primary machine SSH access as root to the peer machines. Generate an SSH authentication key pair for root, and retrieve the public key:

$ sudo ssh-keygen -t rsa -b 2048

$ sudo cat /root/.ssh/id\_rsa.pub

ssh-rsa AAAAB3Nz4rFgt...vgaD root@node1

Get the IP address of the primary node:

$ 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

The ip addr command will dump information about interfaces on the machine. Dis‐ regard the loopback interface, which is normally the first. Look for the IP address fol‐ lowing inet for the primary interface. In this example, the IP address is 192.168.1.2.

Distribute the public key to the root user’s *authorized\_keys* file on each peer node, and specify to authorize only from the primary IP address:

$ sudo echo ‘from=”192.168.1.2" ssh-rsa AAAAB3Nz4rFgt...vgaD \ root@node1' >> /root/.ssh/authorized\_keys

Add the following line to */etc/ssh/sshd\_config* and reload sshd on all nodes:

$ sudo echo 'PermitRootLogin without-password' >> \

/etc/ssh/sshd\_config

$ sudo service sshd reload

Verify that the root user on the primary node can ssh to each of the peer nodes without a password:

$ sudo ssh [root@node2.example.com](mailto:root@node2.example.com)

Create the configuration file */etc/nginx-sync.conf* on the primary machine with the following configuration:

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

This example configuration demonstrates the three common configuration parame‐ ters for this feature: NODES, CONFIGPATHS, and EXCLUDE. The NODES parameter is set to a string of hostnames or IP addresses separated by spaces; these are the peer nodes to which the primary will push its configuration changes. The CONFIGPATHS parameter denotes which files or directories should be synchronized. Lastly, you can use the EXCLUDE parameter to exclude configuration files from synchronization. In our exam‐ ple, the primary pushes configuration changes of the main NGINX configuration file and includes the directory */etc/nginx/nginx.conf* and */etc/nginx/conf.d* to peer nodes named node2.example.com, node3.example.com, and node4.example.com. If the synchronization process finds a file named *default.conf*, it will not be pushed to the peers, because it’s configured as an EXCLUDE.

There are advanced configuration parameters to configure the location of the NGINX binary, RSYNC binary, SSH binary, diff binary, lockfile location, and backup direc‐ tory. There is also a parameter that utilizes sed to template the given files. For more information about the advanced parameters, see [“Synchronizing NGINX Configura‐](https://oreil.ly/bsEjm) [tion in a Cluster”](https://oreil.ly/bsEjm).

Test your configuration:

$ nginx-sync.sh -h *# display usage info*

$ nginx-sync.sh -c node2.example.com *# compare config to node2*

$ nginx-sync.sh -C *# compare primary config to all peers*

$ nginx-sync.sh *# sync the config & reload NGINX on peers*

### Discussion

This NGINX Plus exclusive feature enables you to manage multiple NGINX Plus servers in an HA configuration by updating only the primary node and synchroniz‐ ing the configuration to all other peer nodes. By automating the synchronization of the configuration, you limit the risk of mistakes when transferring configurations. The *nginx-sync.sh* application provides some safeguards to prevent sending bad con‐ figurations to the peers. They include testing the configuration on the primary, creat‐ ing backups of the configuration on the peers, and validating the configuration on the peer before reloading. Although it’s preferable to synchronize your configuration by using configuration management tools or Docker, the NGINX Plus configuration synchronization feature is valuable if you have yet to make the big leap to managing environments in this way.

## State Sharing with NGINX Plus and Zone Sync

### Problem

You need NGINX Plus to synchronize its shared memory zones across a fleet of highly available servers.

### Solution

Configure zone synchronization, then use the sync parameter when configuring an NGINX Plus shared memory zone:

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;

}

}

}

### Discussion

The zone\_sync module is an NGINX Plus exclusive feature that enables NGINX Plus to truly cluster. As shown in the configuration, you must set up a stream server configured as the zone\_sync. In the example, this is the server listening on port 9000. NGINX Plus communicates with the rest of the servers defined by the zone\_sync\_server directive. You can set this directive to a domain name that resolves to multiple IP addresses for dynamic clusters, or statically define a series

of zone\_sync\_server directives to avoid single points of failure. You should restrict access to the zone sync server; there are specific SSL/TLS directives for this module for machine authentication. The benefit of configuring NGINX Plus into a cluster is that you can synchronize shared memory zones for rate limiting, sticky-learn sessions, and the key-value store. The example provided shows the sync parameter tacked on to the end of a sticky learn directive. In this example, a user is bound to an upstream server based on a cookie named session. Without the zone\_sync module, if a user makes a request to a different NGINX Plus server, they could lose their session. With the zone\_sync module, all of the NGINX Plus servers are aware of the session and which upstream server it’s bound to.

**CHAPTER 13**

# Advanced Activity Monitoring

## Introduction

To ensure that your application is running at optimal performance and precision, you need insight into the monitoring metrics about its activity. NGINX Plus offers an advanced monitoring dashboard and a JSON feed to provide in-depth monitoring about all requests that come through the heart of your application. The NGINX Plus activity monitoring provides insight into requests, upstream server pools, caching, health, and more. This chapter details the power and possibilities of the NGINX Plus dashboard, the NGINX Plus API, and the open source stub status module.

## Enable NGINX Open Source Stub Status

### Problem

You need to enable basic monitoring for NGINX.

### Solution

Enable the stub\_status module in a location block within an NGINX HTTP server:

location /stub\_status { stub\_status;

allow 127.0.0.1; deny all;

# Set IP restrictions as appropriate

}

Test your configuration by making a request for the status:

$ curl localhost/stub\_status Active connections: 1

server accepts handled requests 1 1 1

Reading: 0 Writing: 1 Waiting: 0

### Discussion

The stub\_status module enables some basic monitoring of the NGINX OSS server. The information that is returned provides insight into the number of active connec‐ tions, as well as the total connections accepted, connections handled, and requests served. The current number of connections being read, written, or in a waiting state is also shown. The information provided is global and is not specific to the parent server where the stub\_status directive is defined. This means that you can host the status on a protected server. For security reasons we blocked all access to the monitoring feature except local traffic. This module provides active connection counts as embedded variables for use in logs and elsewhere. These vari‐ ables are $connections\_active, $connections\_reading, $connections\_writing, and $connections\_waiting.

## Enabling the NGINX Plus Monitoring Dashboard

### Problem

You require in-depth metrics about the traffic flowing through your NGINX Plus server.

### Solution

Utilize the real-time activity monitoring dashboard:

server {

# ...

location /api {

api [write=on];

# Directives limiting access to the API # See chapter 7

}

location = /dashboard.html {

root /usr/share/nginx/html;

}

}

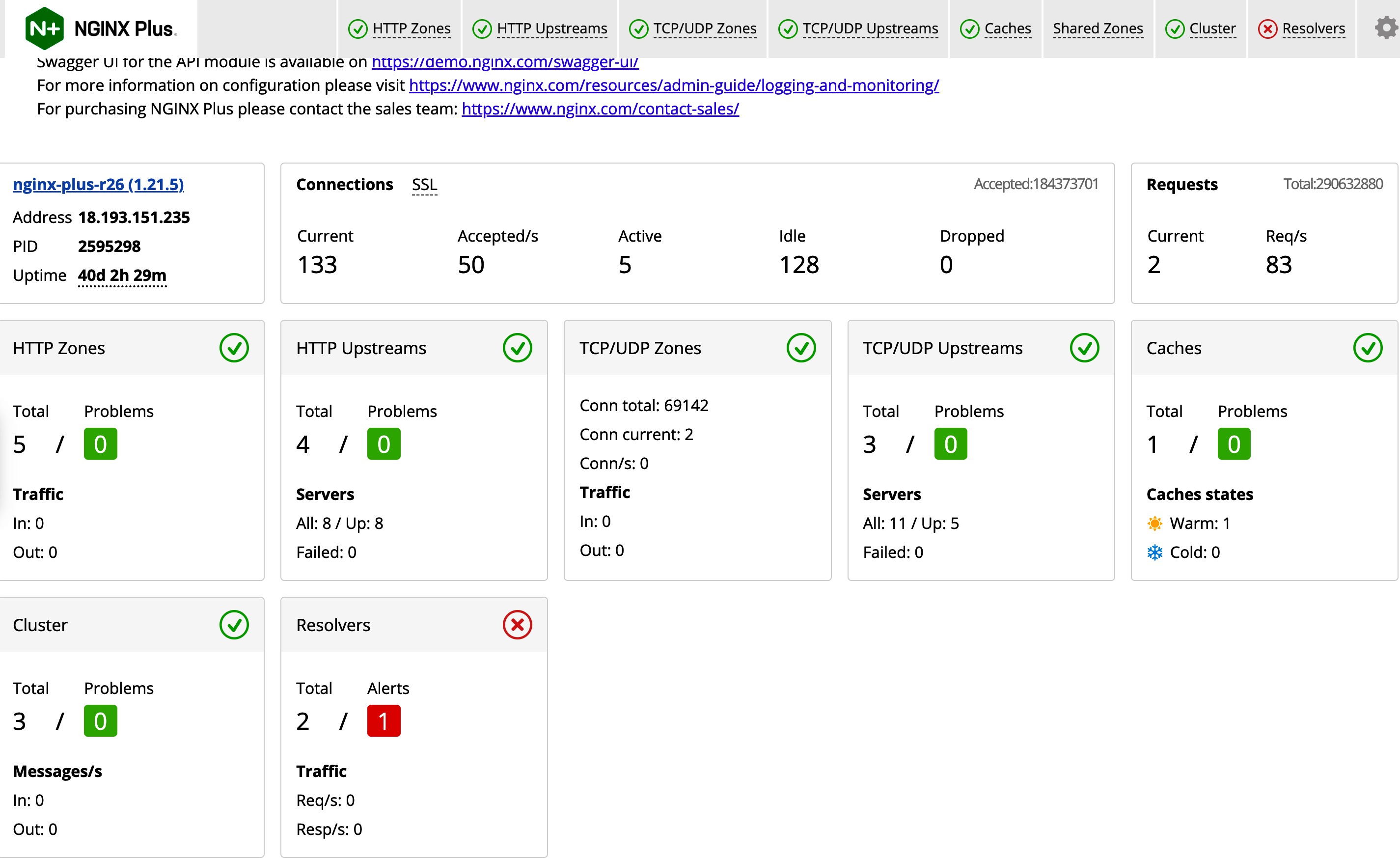
The NGINX Plus configuration serves the NGINX Plus status monitoring dashboard. This configuration sets up an HTTP server to serve the API and the status dashboard. The dashboard is served as static content out of the */usr/share/nginx/html* directory. The dashboard makes requests to the API at */api/* in order to retrieve and display the status in real time.

### Discussion

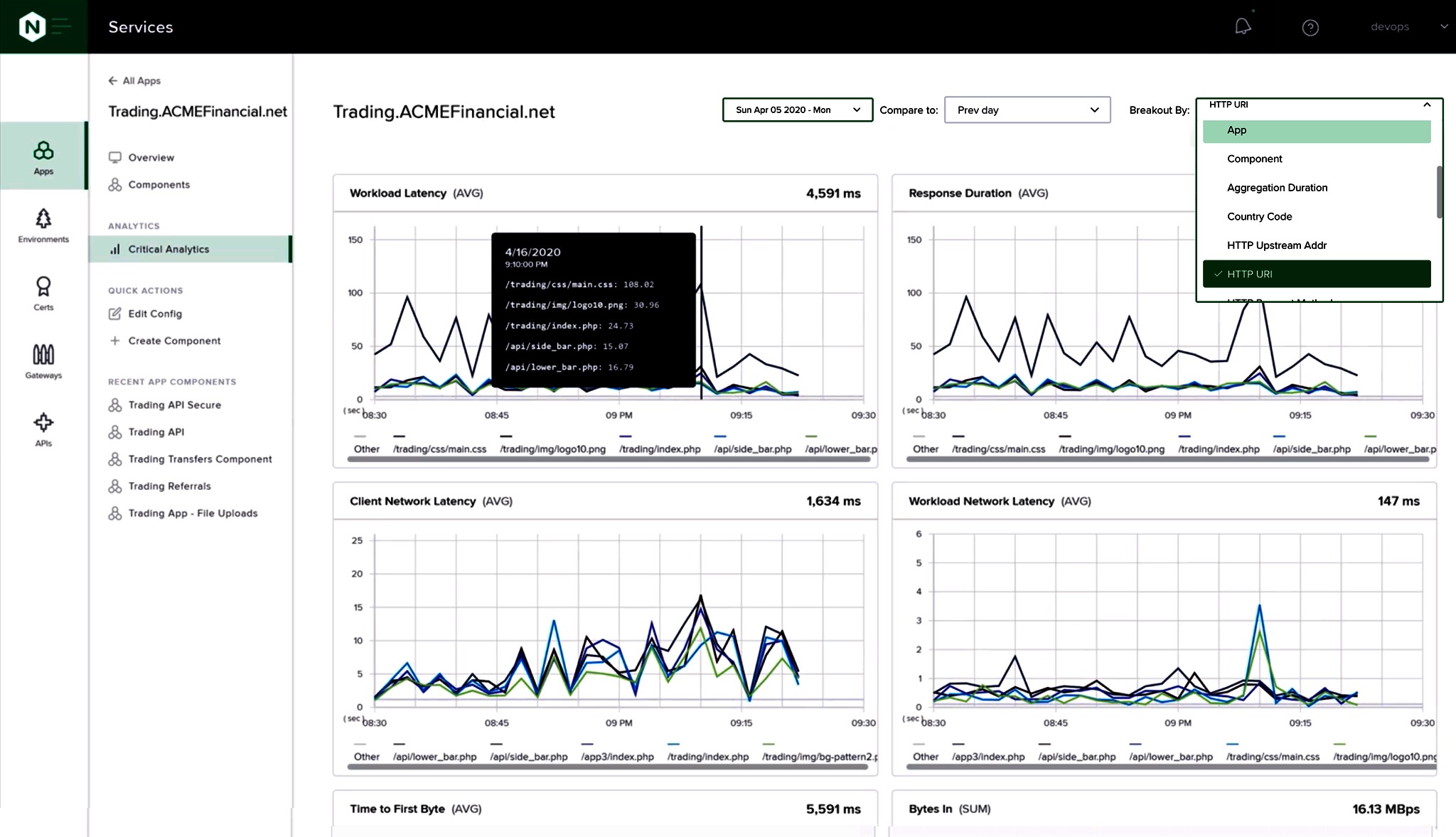
NGINX Plus provides an advanced status monitoring dashboard. This status dash‐ board provides a detailed status of the NGINX system, such as number of active connections, uptime, upstream server pool information, and more. For a glimpse of the console, see [Figure 13-1](#_bookmark511).

NGINX Controller ADC provides an app-centric view of monitoring a fleet of NGINX Plus servers across different locations. The Infrastructure tab provides information and metrics about servers, environments, and applications in a single interface. For a glimpse at the console, see [Figure 13-2](#_bookmark512).

The landing page of the status dashboard provides an overview of the entire system. Clicking into the HTTP Zones tab lists details about all HTTP servers configured in the NGINX configuration, detailing the number of responses from 1XX to 5XX and an overall total as well as requests per second and the current traffic throughput. The HTTP Upstreams tab details upstream server status: if the server were in a failed state, how many requests it has served, and a total of how many responses have been served by status code, as well as other statistics such as how many health checks it has passed or failed. The TCP/UDP Zones tab details the amount of traffic flowing through the TCP or UDP streams and the number of connections. The TCP/UDP Upstreams tab shows information about how much each of the upstream servers in the TCP/UDP upstream pools is serving, as well as health check pass and fail details and response times. The Caches tab displays information about the amount of space utilized for cache; the amount of traffic served, written, and bypassed; as well as the hit ratio. The NGINX status dashboard is invaluable in monitoring the heart of your applications and traffic flow.



*Figure 13-1. The NGINX Plus status dashboard*



*Figure 13-2. An NGINX Controller app-centric analytics dashboard*

### Also See

[NGINX Plus Status Dashboard Demo](https://oreil.ly/20j1Q) [F5 NGINX Controller Product Page](https://oreil.ly/ty2Iu)

## Collecting Metrics Using the NGINX Plus API

### Problem

You need API access to the detail metrics provided by the NGINX Plus status dashboard.

### Solution

Utilize the RESTful API to collect metrics. The examples pipe the output through

json\_pp to make them easier to read:

$ curl "demo.nginx.com/api/3/" | json\_pp [

"nginx", "processes", "connections", "ssl",

"slabs",

"http", "stream"

]

The curl call requests the top level of the API, which displays other portions of the API.

To get information about the NGINX Plus server, use the /api/{version}/nginx

URI:

$ 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

}

To limit information returned by the API, use arguments:

$ curl "demo.nginx.com/api/3/nginx?fields=version,build" **\**

| json\_pp

{

"build" : "nginx-plus-r16",

"version" : "1.15.2"

}

You can request connection statistics from the /api/{version}/connections URI:

$ curl "demo.nginx.com/api/3/connections" | json\_pp

{

"active" : 3,

"idle" : 34,

"dropped" : 0,

"accepted" : 33614951

}

You can collect request statistics from the /api/{version}/http/requests URI:

$ curl "demo.nginx.com/api/3/http/requests" | json\_pp

{

"total" : 52107833,

"current" : 2

}

You can retrieve statistics about a particular server zone using the /api/{ver sion}/http/server\_zones/{httpServerZoneName} URI:

$ 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

}

The API can return any bit of data you can see on the dashboard. It has depth and follows a logical pattern. You can find links to resources at the end of this recipe.

### Discussion

The NGINX Plus API can return statistics about many parts of the NGINX Plus server. You can gather information about the NGINX Plus server and its processes, connections, and slabs. You can also find information about http and stream servers running within NGINX, including servers, upstreams, upstream servers, and key-value stores, as well as information and statistics about HTTP cache zones. This provides you or third-party metric aggregators with an in-depth view of how your NGINX Plus server is performing.

With the NGINX Instance Manager API, you can query metrics from multiple NGINX Plus servers at once. The metrics provided by Controller offer a different, application-centric view of your metrics.

### Also See

[NGINX HTTP API module documentation](https://oreil.ly/qyKjs) [NGINX API REST UI](https://demo.nginx.com/swagger-ui)

[“Using the Metrics API” tutorial](https://oreil.ly/uH5Ul)

**CHAPTER 14**

# Debugging and Troubleshooting with Access Logs, Error Logs, and Request Tracing

## Introduction

Logging is the basis of understanding your application. With NGINX you have great control over logging information meaningful to you and your application. NGINX allows you to divide access logs into different files and formats for different contexts and to change the log level of error logging to get a deeper understanding of what’s happening. The capability of streaming logs to a centralized server comes innately to NGINX through its Syslog logging capabilities. NGINX and NGINX Plus also enable tracing of requests as they move through a system. In this chapter, we discuss access and error logs, streaming over the Syslog protocol, and tracing requests end to end with request identifiers generated by NGINX and OpenTracing.

## Configuring Access Logs

### Problem

You need to configure access log formats to add embedded variables to your request logs.

### Solution

Configure an access log format:

http {

log\_format geoproxy

'[$time\_local] $remote\_addr ' '$realip\_remote\_addr $remote\_user '

'$proxy\_protocol\_server\_addr $proxy\_protocol\_server\_port ' '$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"';

# ...

}

This log format configuration is named geoproxy and uses a number of embedded variables to demonstrate the power of NGINX logging. This configuration shows the local time on the server when the request was made, the IP address that opened the connection, and the IP of the client, as NGINX understands it per geoip\_proxy or realip\_header instructions.

The variables prefixed with $proxy\_protocol\_server\_ provide information about the server from the PROXY Protocol header, when the proxy\_protocol parameter is used on the listen directive of the server. $remote\_user shows the username of the user, authenticated by basic authentication, followed by the request method and protocol, as well as the scheme, such as HTTP or HTTPS. The server name match is logged as well as the request URI and the return status code.

Statistics logged include the processing time in milliseconds and the size of the body sent to the client. Information about the country, region, and city are logged. The HTTP header X-Forwarded-For is included to show if the request is being forwarded by another proxy. The upstream module enables some embedded variables that we’ve used that show the status returned from the upstream server and how long the upstream request takes to return. Lastly, we’ve logged some information about where the client was referred from and what browser the client is using.

The log\_format directive is only valid within the HTTP context. An optional escape parameter can specify what type of escaping is done on the string; default, json, and none are escape values. The none value disables escaping. For default escaping, characters ", \, and other characters with values less than 32 or above 126 are escaped as \xXX. If the variable value is not found, a hyphen (-) will be logged. For json escaping, all characters not allowed in JSON strings will be escaped: characters " and

\ are escaped as \" and \\, characters with values less than 32 are escaped as \n, \r,

\t, \b, \f, or \u00XX.

This log configuration renders a log entry that looks like the following:

[25/Nov/2016:16:20:42 +0000] 10.0.1.16 192.168.0.122 Derek

GET HTTP/1.1 http [www.example.com](http://www.example.com/) / 200 0.001 370 USA MI "Ann Arbor" - 200 0.001 "-" "curl/7.47.0"

To use this log format, use the access\_log directive, providing a logfile path and the format name geoproxy as parameters:

server {

access\_log /var/log/nginx/access.log geoproxy; # ...

}

The access\_log directive takes a logfile path and the format name as parameters. This directive is valid in many contexts and in each context can have a different log path and/or log format. Named parameters such as buffer, flush, and gzip configure how often logs are written to the logfile and if the file is gzipped or not. A parameter named if exists and takes a condition; if the condition evaluates to 0 or an empty string, the access will not be logged.

### Discussion

The log module in NGINX allows you to configure log formats for many different scenarios to log to numerous logfiles as you see fit. You may find it useful to configure a different log format for each context, where you use different modules and employ those modules’ embedded variables, or a single, catchall format that provides all the information you could ever want. It’s also possible to log in JSON or XML, provided you construct the format string in that manner. These logs will aid you in understanding your traffic patterns, client usage, who your clients are, and where they’re coming from. Access logs can also aid you in finding lag in responses and issues with upstream servers or particular URIs. Access logs can be used to parse and play back traffic patterns in test environments to mimic real user interaction. There are limitless possibilities for logs when troubleshooting, debugging, or analyzing your application or market.

## Configuring Error Logs

### Problem

You need to configure error logging to better understand issues with your NGINX server.

### Solution

Use the error\_log directive to define the log path and the log level:

error\_log /var/log/nginx/error.log warn;

The error\_log directive requires a path; however, the log level is optional, and defaults to error. This directive is valid in every context except for if statements. The log levels available are debug, info, notice, warn, error, crit, alert, or emerg. The order in which these log levels were introduced is also the order of severity from least to most. The debug log level is only available if NGINX is configured with the

--with-debug flag.

### Discussion

The error log is the first place to look when configuration files are not working cor‐ rectly. The log is also a great place to find errors produced by application servers like FastCGI. You can use the error log to debug connections down to the worker, mem‐ ory allocation, client IP, and server. The error log cannot be formatted. However, it follows a specific format of date, followed by the level, then the message.

## Forwarding to Syslog

### Problem

You need to forward your logs to a Syslog listener to aggregate logs to a centralized service.

### Solution

Use the error\_log and access\_log directives to send your logs to a Syslog listener:

error\_log syslog:server=10.0.1.42 debug;

access\_log syslog:server=10.0.1.42,tag=nginx,severity=info geoproxy;

The syslog parameter for the error\_log and access\_log directives is followed by a colon and a number of options. These options include the required server flag that denotes the IP, DNS name, or Unix socket to connect to, as well as optional flags such as facility, severity, tag, and nohostname. The server option takes a port number, along with IP addresses or DNS names. However, it defaults to UDP 514. The facility option refers to the facility attribute of the log message, one of the 23 attributes defined in the RFC standard for Syslog; the default value is local7. The tag option tags the message with a value. This value defaults to nginx. severity defaults to info and denotes the severity of the message being sent. The nohostname flag disables the adding of the hostname field into the Syslog message header and does not take a value.

### Discussion

Syslog is a standard protocol for sending log messages and collecting those logs on a single server or collection of servers. Sending logs to a centralized location helps in debugging when you’ve got multiple instances of the same service running on multiple hosts. This is called aggregating logs. Aggregating logs allows you to view logs together in one place without having to jump from server to server, and mentally stitch together logfiles by timestamp. A common log aggregation stack is Elasticsearch, Logstash, and Kibana, also known as the ELK Stack. NGINX makes streaming these logs to your Syslog listener easy with the access\_log and error\_log directives.

## Request Tracing

### Problem

You need to correlate NGINX logs with application logs to have an end-to-end understanding of a request.

### Solution

Use the request identifying variable and pass it to your application to log as well:

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 the header X-Request-ID to the response to the client add\_header X-Request-ID $request\_id;

location / {

proxy\_pass http://backend;

# Send the header X-Request-ID to the application proxy\_set\_header X-Request-ID $request\_id; access\_log /var/log/nginx/access\_trace.log trace;

}

}

In this example configuration, a log\_format named trace is set up, and the variable

$request\_id is used in the log. This $request\_id variable is also passed to the upstream application by use of the proxy\_set\_header directive to add the request ID to a header when making the upstream request. The request ID is also passed

back to the client through use of the add\_header directive, setting the request ID in a response header.

### Discussion

Made available in NGINX Plus R10 and NGINX version 1.11.0, the $request\_id provides a randomly generated string of 32 hexadecimal characters that can be used to uniquely identify requests. By passing this identifier to the client as well as to the application, you can correlate your logs with the requests you make. From the frontend client, you will receive this unique string as a response header and can use it to search your logs for the entries that correspond. You will need to instruct your application to capture and log this header in its application logs to create a true end-to-end relationship between the logs. With this advancement, NGINX makes it possible to trace requests through your application stack.

## OpenTracing for NGINX

### Problem

You have a tracing server that supports OpenTracing and want to integrate NGINX or NGINX Plus.

### Solution

Ensure you have an OpenTrace-compatible server available, and that the correct client is installed on the NGINX or NGINX Plus node.

A plug-in configuration file for the specific OpenTrace-compatible server will be needed. This solution will demonstrate Jaeger and Zipkin.

A Jaeger plug-in configuration example named */etc/jaeger/jaeger-config.json* is as follows:

{

**"service\_name"**: "nginx",

**"sampler"**: { **"type"**: "const", **"param"**: 1

},

**"reporter"**: {

**"localAgentHostPort"**: "Jaeger-server-IP-address:6831"

}

}

A Zipkin plug-in configuration example named */etc/zipkin/zipkin-config.json* is as follows:

{

**"service\_name"**: "nginx",

**"collector\_host"**: "Zipkin-server-IP-address",

**"collector\_port"**: 9411

}

If NGINX Plus is being used, install the OpenTracing module from the NGINX Plus repository by following the [NGINX Plus OpenTracing Admin Guide](https://oreil.ly/coRQC).

If open source NGINX is being used, visit the [NGINX OpenTracing Module Releases](https://oreil.ly/9LYf3) [page](https://oreil.ly/9LYf3) to find a prebuilt dynamic module compatible with your system, or compile the module along with NGINX from source. Alternatively, in a Docker environment, an image named opentracing/nginx-opentracing is available on Docker Hub, and can be used to jump-start your testing.

When using a dynamically loaded module, which includes the NGINX Plus installa‐ tion, ensure that you load it within your NGINX configuration by adding the follow‐ ing load\_module directive, to tell NGINX where on the filesystem to find the module. As a reminder, the load\_module directive is valid only in the main (top-level) context.

load\_module modules/ngx\_http\_opentracing\_module.so;

Once an OpenTrace-compatible server is listening, a client is installed on the NGINX node and plug-in configuration is in place, and the NGINX module is loaded, NGINX can be configured to start tracing requests. The following code provides examples of loading a tracer and configuring NGINX to tag requests. The directive to load a tracer plug-in is included, with a default location for the Jaeger and Zipkin plug-ins and the configuration files provided previously. Uncomment the appropriate vendor example for the use case.

# Load a vendor tracer

#opentracing\_load\_tracer /usr/local/libjaegertracing\_plugin.so # /etc/jaeger/jaeger-config.json;

#opentracing\_load\_tracer /usr/local/lib/libzipkin\_opentracing\_plugin.so # /etc/zipkin/zipkin-config.json;

# Enable tracing for all requests opentracing on;

# Set additional tags that capture the value of NGINX variables opentracing\_tag bytes\_sent $bytes\_sent;

opentracing\_tag http\_user\_agent $http\_user\_agent; opentracing\_tag request\_time $request\_time; opentracing\_tag upstream\_addr $upstream\_addr;

opentracing\_tag upstream\_bytes\_received $upstream\_bytes\_received; opentracing\_tag upstream\_cache\_status $upstream\_cache\_status; opentracing\_tag upstream\_connect\_time $upstream\_connect\_time; opentracing\_tag upstream\_header\_time $upstream\_header\_time; opentracing\_tag upstream\_queue\_time $upstream\_queue\_time; opentracing\_tag upstream\_response\_time $upstream\_response\_time;

server {

listen 9001;

location / {

# The operation name used for OpenTracing Spans defaults # to the name of the 'location' block,

# but uncomment this directive to customize it. #opentracing\_operation\_name $uri;

# Propagate the active Span context upstream,

# so that the trace can be continued by the backend. opentracing\_propagate\_context;

# Example application location service proxy\_pass http://10.0.0.2:8080;

}

}

### Discussion

An OpenTracing setup is by no means trivial, but it does provide enormous value in areas of distributed monitoring of performance and transactions. These tools enable teams to effectively provide root cause, and dependency analysis to pinpoint problem areas with data. It’s natural for NGINX to serve as an API gateway, routing and authorizing requests between applications, and therefore has integral information to tracing requests through a complex system.

NGINX can tag a request with any variable available to itself, which enables the trac‐ ing system user to have a full view of how a request behaves. This example provided a limited sample of using OpenTracing for a proxied request. One can imagine the amount of data that can be harvested from NGINX as the opentracing\_tag directive is valid in the http, server, and location contexts.

### Also See

[OpenTracing NGINX Module GitHub](https://oreil.ly/DpqEY)

[NGINX Plus OpenTracing Dynamic Module Admin Guide](https://oreil.ly/coRQC) [“Tracing a Proxy” plug-in guide from Datadog guide](https://oreil.ly/f4leF) [“OpenTracing for NGINX and NGINX Plus” blog](https://oreil.ly/2WUJW)

[“Enabling OpenTracing with NGINX Ingress Controller for Kubernetes” blog](https://oreil.ly/GlqZL)

**CHAPTER 15**

# Performance Tuning

## Introduction

Tuning NGINX will make an artist of you. Performance tuning of any type of server or application is always dependent on a number of variable items, such as, but not limited to, the environment, use case, requirements, and physical components involved. It’s common to practice bottleneck-driven tuning, meaning to test until you’ve hit a bottleneck, determine the bottleneck, tune for limitations, and repeat until you’ve reached your desired performance requirements. In this chapter, we suggest taking measurements when performance tuning by testing with automated tools and measuring results. This chapter also covers connection tuning for keeping connections open to clients as well as upstream servers, and serving more connec‐ tions by tuning the operating system.

## Automating Tests with Load Drivers

### Problem

You need to automate your tests with a load driver to gain consistency and repeatabil‐ ity in your testing.

### Solution

Use an HTTP load-testing tool such as Apache JMeter, Locust, Gatling, or whatever your team has standardized on. Create a configuration for your load-testing tool that runs a comprehensive test on your web application. Run your test against your service. Review the metrics collected from the run to establish a baseline. Slowly ramp up the emulated user concurrency to mimic typical production usage and identify

points of improvement. Tune NGINX and repeat this process until you achieve your desired results.

### Discussion

Using an automated testing tool to define your test gives you a consistent test to build metrics from when tuning NGINX. You must be able to repeat your test and measure performance gains or losses to conduct science. Running a test before making any tweaks to the NGINX configuration to establish a baseline gives you a basis to work from so that you can measure if your configuration change has improved performance or not. Measuring for each change made will help you identify where your performance enhancements come from.

## Keeping Connections Open to Clients

### Problem

You need to increase the number of requests allowed to be made over a single connection from clients and increase the amount of time that idle connections are allowed to persist.

### Solution

Use the keepalive\_requests and keepalive\_timeout directives to alter the number of requests that can be made over a single connection and change the amount of time idle connections can stay open:

http {

keepalive\_requests 320; keepalive\_timeout 300s; # ...

}

The keepalive\_requests directive defaults to 100, and the keepalive\_timeout

directive defaults to 75 seconds.

### Discussion

Typically, the default number of requests over a single connection will fulfill client needs because browsers these days are allowed to open multiple connections to a single server per FQDN. The number of parallel open connections to a domain is still typically limited to a number less than 10, so in this regard, many requests over a single connection will happen. A trick for HTTP/1.1 commonly employed by content delivery networks is to create multiple domain names pointed to the content server and alternate which domain name is used within the code to enable the browser

to open more connections. You might find these connection optimizations helpful if your frontend application continually polls your backend application for updates, because an open connection that allows a larger number of requests and stays open longer will limit the number of connections that need to be made.

## Keeping Connections Open Upstream

### Problem

You need to keep connections open to upstream servers for reuse to enhance your performance.

### Solution

Use the keepalive directive in the upstream context to keep connections open to upstream servers for reuse:

proxy\_http\_version 1.1; proxy\_set\_header Connection "";

upstream backend { server 10.0.0.42;

server 10.0.2.56;

keepalive 32;

}

The keepalive directive in the upstream context activates a cache of connections that stay open for each NGINX worker. The directive denotes the maximum number of idle connections to keep open per worker. The proxy modules directives used above the upstream block are necessary for the keepalive directive to function properly for upstream server connections. The proxy\_http\_version directive instructs the proxy module to use HTTP version 1.1, which allows for multiple requests to be made in serial over a single connection while it’s open. The proxy\_set\_header directive instructs the proxy module to strip the default header of close, allowing the connec‐ tion to stay open.

### Discussion

You want to keep connections open to upstream servers to save the amount of time it takes to initiate the connection, allowing the worker process to instead move directly to making a request over an idle connection. It’s important to note that the number of open connections can exceed the number of connections specified in the keepalive directive because open connections and idle connections are not the same. The number of keepalive connections should be kept small enough to allow for

other incoming connections to your upstream server. This small NGINX tuning trick can save some cycles and enhance your performance.

## Buffering Responses

### Problem

You need to buffer responses between upstream servers and clients in memory to avoid writing responses to temporary files.

### Solution

Tune proxy buffer settings to allow NGINX the memory to buffer response bodies:

server {

proxy\_buffering on; proxy\_buffer\_size 8k; proxy\_buffers 8 32k; proxy\_busy\_buffer\_size 64k; # ...

}

The proxy\_buffering directive is either on or off; by default it’s on. The proxy\_buffer\_size denotes the size of a buffer used for reading the first part of the response and response headers from the proxied server and defaults to either 4k or 8k, depending on the platform. The proxy\_buffers directive takes two parameters: the number of buffers and the size of the buffers. By default, the proxy\_buffers directive is set to a number of 8 buffers of size either 4k or 8k, depending on the platform. The proxy\_busy\_buffer\_size directive limits the size of buffers that can be busy, sending a response to the client while the response is not fully read. The busy buffer size defaults to double the size of a proxy buffer or the buffer size. If proxy buffering is disabled, the request cannot be sent to the next upstream server in the event of a failure because NGINX has already started sending the request body.

### Discussion

Proxy buffers can greatly enhance your proxy performance, depending on the typical size of your response bodies. Tuning these settings can have adverse effects and should be done by observing the average body size returned and performing thor‐ ough and repeated testing. Extremely large buffers, set when they’re not necessary, can eat up the memory of your NGINX box. You can set these settings for specific locations that are known to return large response bodies for optimal performance.

### Also See

[NGINX proxy\_request\_buffering module documentation](https://oreil.ly/zL1LK)

## Buffering Access Logs

### Problem

You need to buffer logs to reduce the opportunity of blocks to the NGINX worker process when the system is under load.

### Solution

Set the buffer size and flush time of your access logs:

http {

access\_log /var/log/nginx/access.log main buffer=32k flush=1m gzip=1;

}

The buffer parameter of the access\_log directive denotes the size of a memory buffer that can be filled with log data before being written to disk. The flush param‐ eter of the access\_log directive sets the longest amount of time a log can remain in a buffer before being written to disk. When using gzip, the logs are compressed before being written to the log—values of level 1 (fastest, less compression) through 9 (slowest, best compression) are valid.

### Discussion

Buffering log data into memory may be a small step toward optimization. However, for heavily requested sites and applications, this can make a meaningful adjustment to the usage of the disk and CPU. When using the buffer parameter to the access\_log directive, logs will be written out to disk if the next log entry does not fit into the buffer. If using the flush parameter in conjunction with the buffer parameter, logs will be written to disk when the data in the buffer is older than the time specified. When tailing the log, and with buffering enabled, you may see delays up to the amount of time specified by the flush parameter.

## OS Tuning

### Problem

You need to tune your operating system to accept more connections to handle spike loads or highly trafficked sites.

### Solution

Check the kernel setting for *net.core.somaxconn*, which is the maximum number of connections that can be queued by the kernel for NGINX to process. If you set this number over 512, you’ll need to set the backlog parameter of the listen directive in your NGINX configuration to match. A sign that you should look into this kernel setting is if your kernel log explicitly says to do so. NGINX handles connections very quickly, and, for most use cases, you will not need to alter this setting.

Raising the number of open file descriptors is a more common need. In Linux, a file handle is opened for every connection; therefore, NGINX may open two if you’re using it as a proxy or load balancer because of the open connection upstream. To serve a large number of connections, you may need to increase the file descriptor limit system-wide with the kernel option sys.fs.file\_max, or, for the system user, NGINX is running as in the */etc/security/limits.conf* file. When doing so, you’ll also want to bump the number of worker\_connections and worker\_rlimit\_nofile. Both of these configurations are directives in the NGINX configuration.

Enable more ephemeral ports to support more connections. When NGINX acts as a reverse proxy or load balancer, every connection upstream opens a temporary port for return traffic. Depending on your system configuration, the server may not have the maximum number of ephemeral ports open. To check, review the setting for the kernel setting net.ipv4.ip\_local\_port\_range. The setting is a lower- and upper-bound range of ports. It’s typically OK to set this kernel setting from 1024 to 65535. 1024 is where the registered TCP ports stop, and 65535 is where dynamic or ephemeral ports stop. Keep in mind that your lower bound should be higher than the highest open listening service port.

### Discussion

Tuning the operating system is one of the first places you look when you start tuning for a high number of connections. There are many optimizations you can make to your kernel for your particular use case. However, kernel tuning should not be done on a whim, and changes should be measured for their performance to ensure the changes are helping. As stated before, you’ll know when it’s time to start tuning your kernel from messages logged in the kernel log or when NGINX explicitly logs a message in its error log.

**CHAPTER 16**

# Introduction to NGINX Instance Manager

## Introduction

NGINX Instance Manager is a service that works with an agent on your NGINX instances. The service can scan your networks for existing NGINX instances, as well as TLS certificates. By providing an inventory of instances and certificates, you can view your landscape of web servers and proxies from a single interface or API. The API enables you to scan, check, and update your data plane through automation that you can integrate into your existing tools and workflows.

The NGINX Instance Manager agent collects metrics from NGINX or NGINX Plus, and allows for direct configuration checks and updates from a centralized interface. You can use NGINX Instance Manager as a metric collection integration point, which reduces the complexity of adding NGINX stub status or NGINX Plus metric API endpoints to your consolidated metric-gathering platform.

NGINX Instance Manager requires a license; however, there is a free trial available so that you can explore its functionality and value for yourself.

## Setup Overview

### Problem

You would like to set up an NGINX Instance Manager.

### Solution

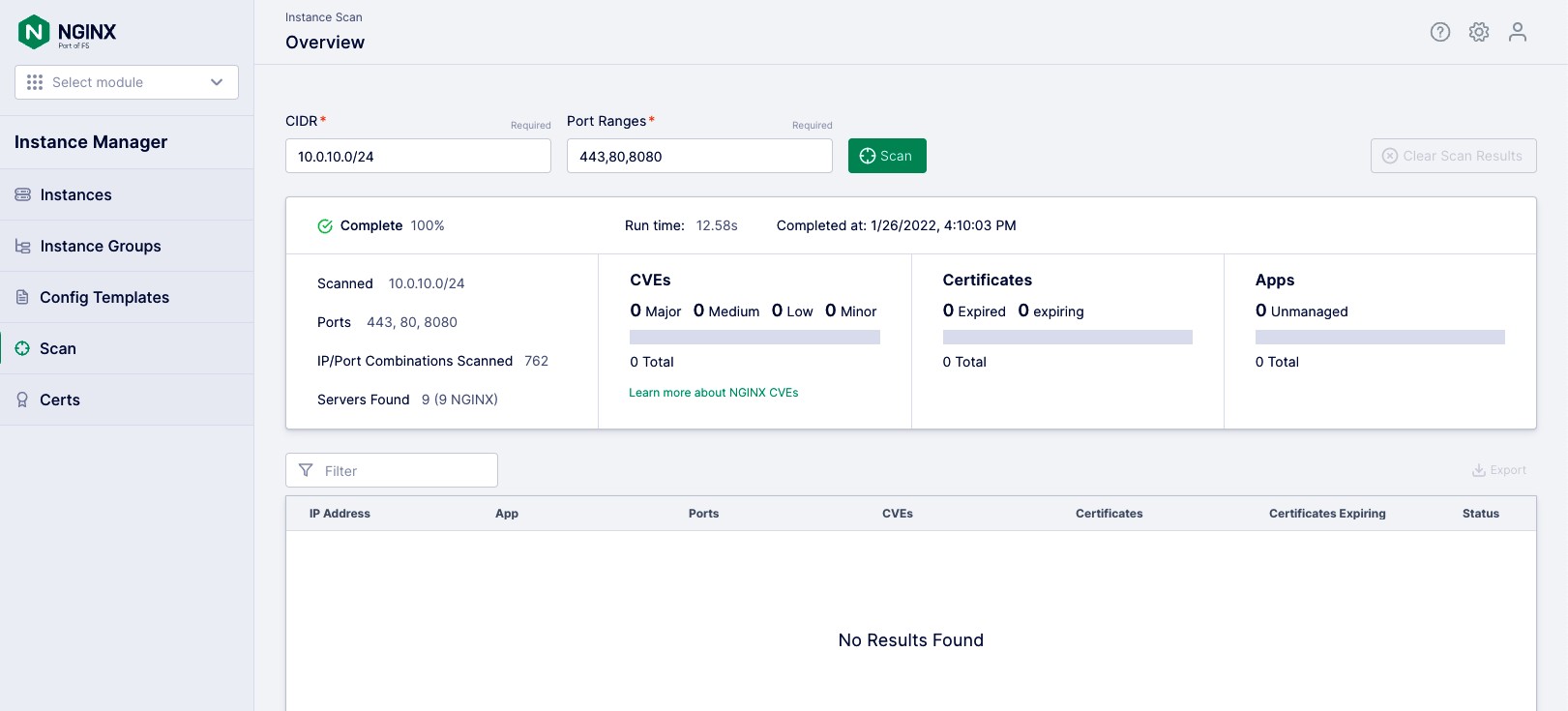
Installing the NGINX Instance Manager server follows the same workflow as other licensed NGINX products. You must first obtain a license file, and either the package or a repository certificate key pair. Once you have the necessary files, you use the

**161**

package manager for your system to install the software. For the most up-to-date and accurate installation information, use the [NGINX Instance Manager Installation](https://oreil.ly/5XUQA) [Guide](https://oreil.ly/5XUQA).

### Discussion

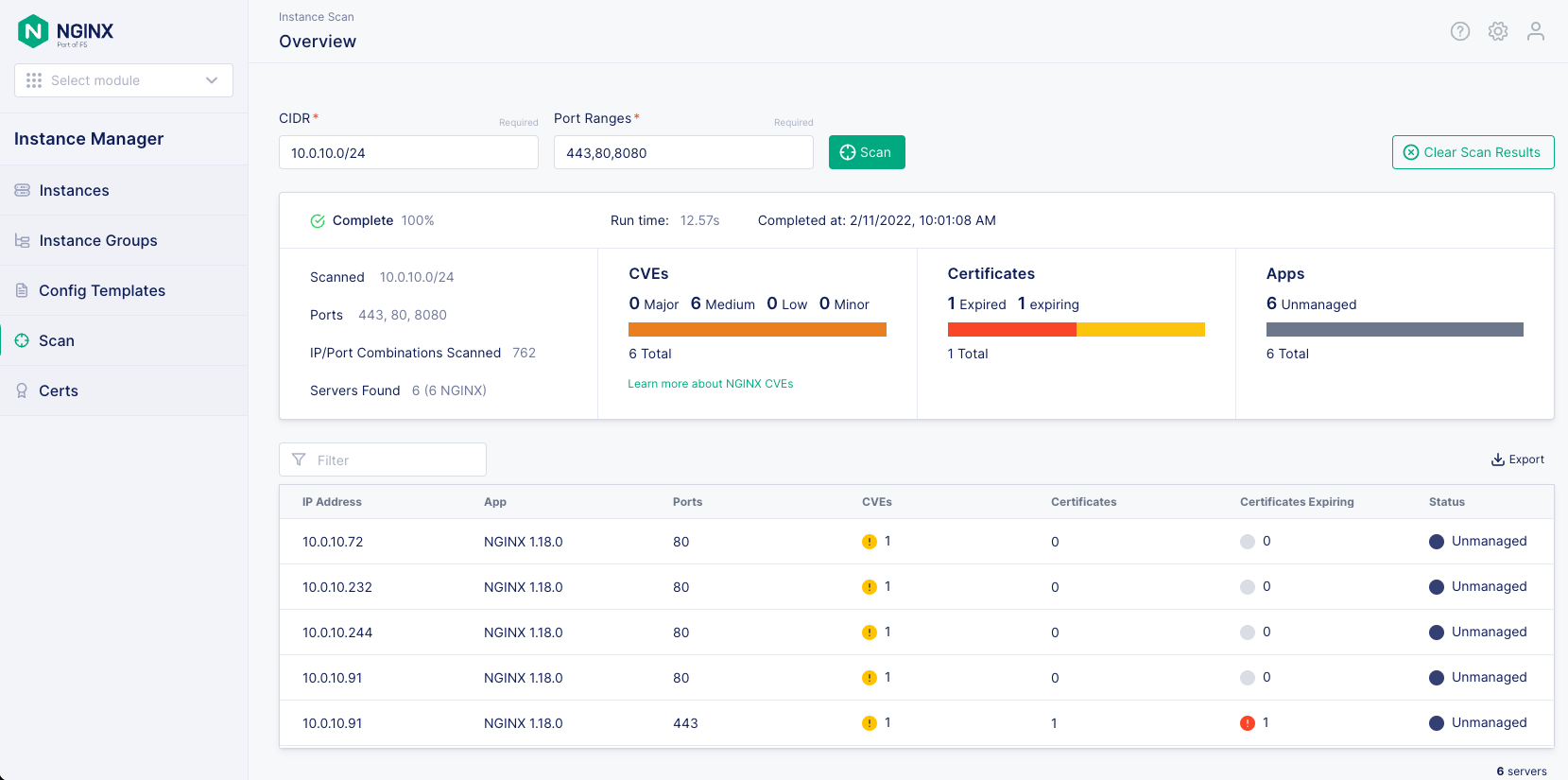
Once you have NGINX Instance Manager installed, browse to its UI. At first you’ll have an empty inventory (see [Figure 16-1](#_bookmark580)).



*Figure 16-1. NGINX Instance Manager UI*

To discover NGINX instances, navigate to the Scan tab (see [Figure 16-2](#_bookmark582)). You can use the scan functionality to discover NGINX and NGINX Plus servers on your network. The scan tool uses NMAP to probe for specific ports that you may run web servers on. If a host is listening on the ports you’re scanning, NGINX Instance Manager attempts to determine if it’s an NGINX server and, if so, what version. This is determined by making a request to the server and inspecting the Server header of the response.

After the scan is complete, you’ll see a table populated with discovered instances. This table will detail information about each instance. When an NGINX instance is discovered, a lookup for any common vulnerabilities and exposures (CVEs) based on the NGINX version will be performed and shown. These results aid in keeping your NGINX servers secure by automating the discovery of necessary patches. A status column informs you which NGINX instances are managed by NGINX Instance Manager and which instances need the agent installed.



*Figure 16-2. NGINX Instance Manager Scan*

On the Certs page, you can scan for hosts configured for TLS. This tool works the same way as the host scan; however, it inspects the certificates in use and provides you information about the certificate, such as the common name and the expiration date. This is a helpful tool to discover where certificates live and where updates may be necessary.

Within Settings, you can view information about NGINX Instance Manager, your license, plus documentation for Instance Manager, the API, and how to configure metric telemetry. This is also where you can configure users and roles in order to control who has access and what they can do.

Before you can manage an instance, you must install the NGINX Instance Manager agent on your NGINX machines. To learn how to install the agent, continue to the next section.

### Also See

[NGINX Instance Manager Installation Guide](https://oreil.ly/3nwIc) [NGINX Instance Manager Admin Guide](https://oreil.ly/1aFpJ)

## Agent Installation

### Problem

You would like to install the NGINX Instance Manager agent on your NGINX or NGINX Plus machines.

### Solution

Installing the NGINX Instance Manager agent is done by downloading and executing the installation script from the NGINX Instance Manager server using the following command:

curl -k https://{NIM hostname or IP}/install/nginx-agent | sudo sh

For a basic installation, these are all the steps required. To set the displayed instance name, location, or instance group membership, you can update the */etc/nginx-agent/ dynamic-agent.conf* file and restart the nginx-agent service:

#

# /etc/nginx-agent/dynamic-agent.conf

# Dynamic configuration file for NGINX Agent.

# The purpose of this file is to track agent configuration

# values that can be dynamically changed via the API and the agent # install script.

# You may edit this file, but API calls that modify the tags on # this system will overwrite the tag values in this file.

# The agent configuration values that API calls can modify are as # follows:

# - tags #

# The agent configuration values that the agent install script can # modify are as follows:

# - instance\_name # - instance\_group # - location # Example Values

instance\_name: nginx1.example.org instance\_group: internal location: cloud1

### Discussion

Once you have the NGINX Instance Manager agent installed and running on your NGINX or NGINX Plus instances, you will see them appear on the inventory page within NGINX Instance Manager. Information about the host will be detailed in a table, such as its hostname, type of NGINX installation, tags, and its connection status.

By selecting an instance, a more detailed view will appear on the righthand side, with the option to edit. By selecting Edit, you will be shown the current NGINX configuration of the instance. You can then use the editor to make changes or analyze the configuration. A green drop-down button in the top right of the editor allows you to switch between configurable files. Once a change is made you can save the configu‐ ration. To reload NGINX from the configuration you’ve saved, you must publish. The separation of the save and publish actions allows you to work on multiple files before NGINX attempts to reload the configuration.

## Automating NGINX Discovery, Configuration, and Monitoring with the API

### Problem

You would like to use the NGINX Instance Manager API to automate discovery, configuration, or monitoring.

### Solution

Your NGINX Instance Manager server hosts an OpenAPI spec via Swagger UI for documentation of the installed versions API. You can find this documentation in the Docs Menu area.

You may find it useful to regularly scan your network for NGINX instances and their certificates. You can use this information to alert about CVEs that need to be patched, or certificates that are about to expire.

To start a scan for NGINX instances or certificates, send a POST to the NGINX Instance Manager endpoint /api/platform/v1/servers/scan with a body that details the CIDR range and ports you would like to scan. The instance scan also allows for a list of port ranges:

curl -X 'POST' **\**

'https://nginx-manager.example.com/api/platform/v1/servers/scan' **\**

-H 'accept: application/json' **\**

-H 'Content-Type: application/json' **\**

-d '{

"cidr": "string", "hostDiscovery": "icmp", "portRanges": [ "10.0.1.0/24"

]

}

You can use a DELETE request to the same API endpoint to cancel a scan. A GET

request will return the status of the current scan.

Use a GET request to /api/platform/v1/scan/servers to return information about known NGINX instances or certificates after the scan is complete.

To list all managed NGINX instances, use a GET request to /api/platform/v1/ instances. The response will provide you with instance IDs that can be used in subsequent requests to list configuration files. Those configuration files can be updated and published through the API. This functionality is useful in automating configuration updates to your NGINX instances. Reference the OpenAPI spec for up-to-date usage.

The NGINX Instance Manager also provides access to metric information of man‐ aged instances. External metric-gathering tools can request metric information directly from NGINX Instance Manager through the API. Limited metric informa‐ tion can be viewed by visiting the Instance Inventory page, selecting an instance, and clicking the View Metrics link.

### Discussion

The operations landscape of today’s web technology depends on automation and monitoring. NGINX Instance Manager’s RESTful API provides a way for you to inte‐ grate NGINX management into your existing automation and DevOps workflows. Metric collection and analysis provides the basis for decision making about config‐ uration changes to optimize your service delivery. You can use NGINX Instance Manager as an integration point for collecting metrics across all of your NGINX instances, which reduces the complexity of metric collection.

### Also See

[NGINX Instance Manager API Overview](https://oreil.ly/AnjIr) [“Using the Metrics API” tutorial](https://oreil.ly/uH5Ul)

**CHAPTER 17**

# Introduction to NGINX Controller

## Introduction

NGINX Controller is an application-centric control plane for your application envi‐ ronments. Controller provides an interface that allows you to view and configure an entire fleet of NGINX Plus servers, no matter their physical location. Controller allows teams to focus less on the raw NGINX Plus configuration, and more on the application they’re using NGINX Plus to deliver.

In this chapter, you’ll read an overview of the NGINX Controller setup, connect an NGINX Plus server instance, and learn about using the NGINX Controller API. NGINX Controller is an enterprise product that requires a license. You can request a free trial from the [F5 NGINX Controller Product Page](https://oreil.ly/Cbrjf).

## Setup Overview

### Problem

You would like to set up an NGINX Controller environment.

### Solution

Use the [official NGINX Controller Installation Guide](https://oreil.ly/oM1PL) for an up-to-date installation process. The following are a few tips, observations, and callouts for items to look out for throughout the setup guide.

NGINX Controller 3.x installs as a Kubernetes stack. It is important to review all of the [technical specifications](https://oreil.ly/OiOCD) before beginning. An external PostgreSQL database is required. The Controller installer is provided as a tarball. Once unpacked, an *install.sh* script will need to be run as a nonroot user.

**167**

Due to the way some OS images are distributed, there may be variance in the package repositories, which can cause some difficulty with the installation. Ubuntu 20.04 seems to be the most consistent in my testing, and is my recommendation for trials and exploration. Remember that NGINX Support is available to assist you in getting NGINX Controller up and running quickly.

There are a number of tools the installer needs installed before it’s able to run correctly. Most of the tools are standard on many operating systems, but the jq tool is not. You will need to ensure all the required tools are installed on your system prior to running any of the installer scripts.

A *helper.sh* script is provided in the installation package that can aid with installa‐ tion, or altering the base configuration once installed. For instance, the argument supportpkg will build a package of debug and log information for you to send to NGINX Support to enable them to have a quick overview of your situation. The prereqs argument will install required packages and set up Kubernetes. To view logs from NGINX Controller, you can use ./helper.sh logs.

When the installer command is run, it will check the system for requirements and install any that it may additionally need. The installer will prompt for database information. Currently PostgreSQL is supported and must be on a remote server. The user information provided must be capable of creating databases. This information can be passed as command-line arguments to the installer.

A time series database volume will need to be supplied. You can use the local disk, an NSF volume, or an AWS EBS volume. If you choose to use an AWS EBS volume, the system will need appropriate AWS IAM permissions to attach the volume to the instance.

An end-user license agreement is presented and must be accepted to move forward. After reading, press q to exit the agreement, then y to accept.

An SMTP server is needed to invite users via email as well as for email notifications. In the event that an SMTP server is not yet available, these settings can be configured later by using the *helper.sh* script. Provide some generic values to these prompts, set the host to localhost, port to 25, and decline authentication and TLS. NGINX Controller will not be able to send email until SMTP is configured.

The FQDN is used when generating agent configuration, and should be set to a domain that can be relied upon. The organization name prompt is a friendly name used for labeling—a team or company name will suffice. When providing values for the administrator user, note that the email is used for system login.

SSL/TLS certificate paths can be provided through command parameters of the installer, or as environment variables. If these are not found, the installer will prompt to generate self-signed certificates.

Once the installation completes, the installer will provide a link to Controller. Follow the link and log in with the administrator credentials.

### Discussion

NGINX Controller provides a single control plane for management of your applica‐ tions. The interface is sectioned into different views: Platform, Infrastructure, Serv‐ ices, and Analytics. By doing so, the view is clean and concise for the specific task at hand.

The Platform view is used for managing Controller and user access. The Infrastruc‐ ture view provides details about the machines running the NGINX Controller agents. The next section will describe adding an NGINX Plus server to Controller by instal‐ ling an agent.

In the Services view, the application-centric attributes of NGINX Controller come to light. Controller organizes your applications, environments, gateways, and APIs to enable you to reorganize and deploy rapidly.

### Also See

[NGINX Controller Installation Guide](https://oreil.ly/QCZmq) [NGINX Controller Tech Specs](https://oreil.ly/T0fiy)

[“Installing and preparing PostgreSQL database for NGINX Controller”](https://oreil.ly/Yb_zm)

## Connecting NGINX Plus with Controller

### Problem

You’ve installed Controller and need to connect an NGINX Plus instance with an agent.

### Solution

If you have not yet installed NGINX Plus, use [Recipe 1.3](#_bookmark10) to get an NGINX Plus node online.

The best way to find documentation for your Controller installation is to visit *https://*

*<Controller-FQDN>/docs/infrastructure/agent*. At this document location, you can find information about the technical specifications needed to run the NGINX Con‐ troller Agent, as well as how to install and manage it.

Installing the Controller Agent to an existing NGINX Plus server is straightforward. You will need to retrieve an installer script from the Controller API on port 8443, and run it with an API key. The Controller UI provides simple copy-and-paste

instructions for your environment. After the install is finished, you must start the Controller Agent by using the service manager for your system.

Once the Controller Agent service is running, you will see an instance running in the Controller Infrastructure view.

### Discussion

In this section, you added an NGINX Plus server to the NGINX Controller as an instance. An inventory of the NGINX Plus systems is now shown in the Infrastruc‐ ture view as well as with a list request to the API. When you have one or more run‐ ning instances in NGINX Controller, you can monitor valuable server and NGINX Plus metrics with the Graphs tab within the Infrastructure view. In the Platform view, under the Agents tab, there’s a setting to enable NGINX config analyzer. When turned on, the Infrastructure view enables an Analysis tab. The Analysis tab provides information relevant to the NGINX Plus installation and its current configuration.

Now that you have a fresh NGINX Plus node with the Controller Agent installed, you may want to take a bootable image of this machine, or build configuration management to support these installations, so that you can replicate the machine. With an instance configured, you’re able to start setting up services, which consist of applications, their environments, and how they’re served.

### Also See

[NGINX Controller Agent Installation Guide](https://oreil.ly/x40XD)

## Driving NGINX Controller with the API

### Problem

You’ve learned how to configure NGINX Controller entities, and want to automate these processes with the API.

### Solution

Ensure you have network connectivity to Controller on the API port, which is by default 8443.

NGINX Controller is 100% driven through its API. The interface simply uses that API to provide point-and-click access and dashboards. Use the API Overview in the documentation on your Controller installation by visiting *https://<Controller- FQDN>/docs/api/overview*. This will teach you the basics of the objects, permissions, and how to authenticate. From there, the API Reference can be found at *https://*

*<Controller-FQDN>/docs/api/api-reference*.

One way to jump-start your automation through using the API is to view already configured entities in the NGINX Controller interface, edit the entity, and view the API Spec. This API Spec will show you the method, path, and payload needed to create that object. With some variable substitution, you’re off to a good start on automating your Controller environment.

### Discussion

For some engineers, the API will be the main interaction they have with NGINX Controller; for others, it will be the web interface. Either is valid and extremely powerful. The addition of showing the API call on the web interface lessens the frustration of digging through API reference and quickens your task automation. An Ansible collection for NGINX Controller exists to aid in Controller automation.

### Also See

[“Getting Started with the Ansible Collection for NGINX Controller”](https://oreil.ly/VCdY7)

## Enable WAF Through Controller App Security

### Problem

You’re using NGINX Controller ADC and would like to enable Web Application Firewall (WAF) capabilities for your applications.

### Solution

If you have not already, follow the [NGINX App Protect WAF Installation Guide](https://oreil.ly/kgpHa) for your platform to install the App Protect module on your NGINX Plus node.

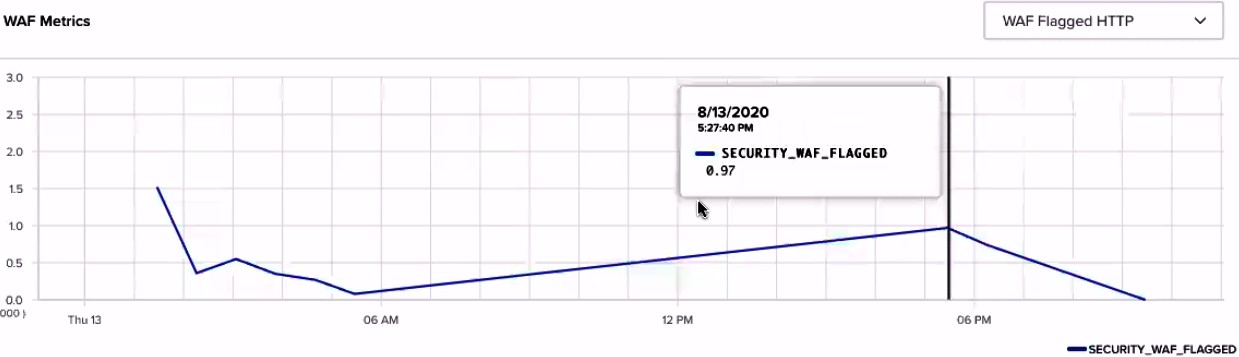
Navigate to the configuration of an existing App Component in NGINX Controller. Within the Security section, locate the WAF header or settings. Enable the WAF and save.

The WAF is now processing requests for the App through the default WAF policy. The default policy is set to alarm on all signatures, but will block for signatures that are considered highly accurate. Accuracy is determined by an algorithm that deter‐ mines the probability of false positives. This means you can immediately start block‐ ing harmful requests while gathering data about security events that are reported by the policy. Both flagged and blocked requests will show up in the NGINX Controller UI, appropriately labeled. NGINX Controller ADC will display WAF statistics and violation events for WAF violations triggered.

Ensure the application is handling some traffic. Test a request that would typically be blocked or flagged by a WAF. The following is an extremely basic SQL Injection request:

curl https://<appComponentEndpoint>/?query=9999999%20UNION%20SELECT%201%2C2

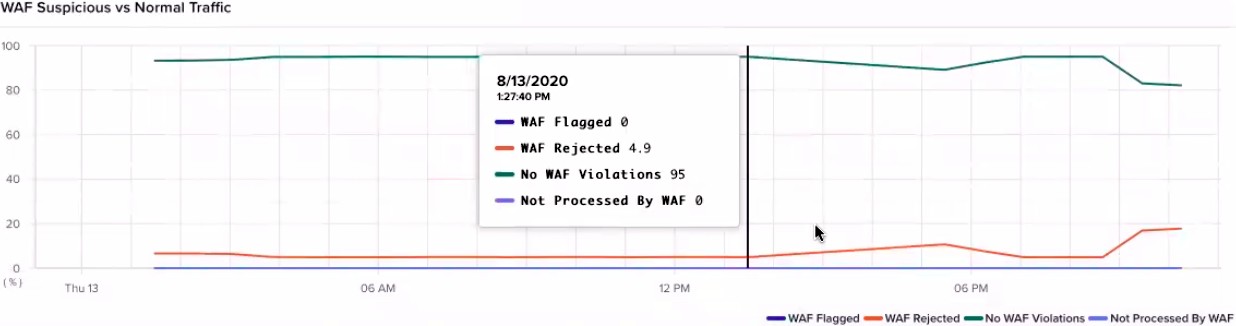
After a request is made that would be considered a security event, NGINX Controller ADC will report security analytics data. Locate these metrics in the Security Analytics section for this App and App Component. In [Figure 17-1](#_bookmark613) you can see how NGINX Controller displays metric information about requests flagged by the WAF.



*Figure 17-1. An NGINX App Security WAF has flagged some requests*

Once some flagged requests start to show up, you can also view the event on the Security Events page. This is where you’ll find detailed information about each request flagged or blocked by NGINX App Security WAF.

Before enabling more strict policies, you should verify that your normal valid appli‐ cation traffic is not getting flagged. If normal behavior is flagged, inspect the reason‐ ing for the individual security event. If normal application traffic is being flagged, you may have vulnerabilities in your application that need to be addressed. In [Figure 17-2](#_bookmark614) all traffic observed by the WAF is displayed to show a pattern between regular traffic and requests that violated WAF rules.



*Figure 17-2. An example of statistics reported by Controller ADC with a WAF enabled*

### Discussion

Controller App Security provides a simplistic WAF experience to secure your applica‐ tions. With the information provided by the monitoring, you can watch for trends over time for security attacks, investigate further by looking at the detail of the event, and decide what action must be taken.

Web Application Firewalls are extremely important in today’s web security architec‐ ture. Applications are constantly being bombarded with attempts to compromise services with common vulnerabilities. By blocking these requests before they reach your application services, you’re not only securing the web application, but reserving resources for legitimate client requests.

A WAF is not only for external clients; you should consider using a WAF on internal traffic as well, so that one compromised service does not affect another.

**CHAPTER 18**

# Practical Ops Tips and Conclusion

## Introduction

This last chapter will cover practical operations tips and is the conclusion to this book. Throughout this book, we’ve discussed many concepts pertinent to operations engineers. However, I thought a few more ideas might be helpful to round things out. In this chapter, I’ll cover making sure your configuration files are clean and concise, as well as debugging configuration files.

## Using Includes for Clean Configs

### Problem

You need to clean up bulky configuration files to keep your configurations logically grouped into modular configuration sets.

### Solution

Use the include directive to reference configuration files, directories, or masks:

http {

include config.d/compression.conf; include sites-enabled/\*.conf

}

The include directive takes a single parameter of either a path to a file or a mask that matches many files. This directive is valid in any context.

### Discussion

By using include statements you can keep your NGINX configuration clean and concise. You’ll be able to logically group your configurations to avoid configuration files that go on for hundreds of lines. You can create modular configuration files that can be included in multiple places throughout your configuration to avoid dupli‐ cation of configurations. Take the example *fastcgi\_param* configuration file provided in most package management installs of NGINX. If you manage multiple FastCGI virtual servers on a single NGINX box, you can include this configuration file for any location or context where you require these parameters for FastCGI without having to duplicate this configuration. Another example is SSL configurations. If you’re running multiple servers that require similar SSL configurations, you can simply write this configuration once and include it wherever needed. By logically grouping your configurations together, you can rest assured that your configurations are neat and organized. Changing a set of configuration files can be done by editing a single file rather than changing multiple sets of configuration blocks in multiple locations within a massive configuration file. Grouping your configurations into files and using include statements is good practice for your sanity and the sanity of your colleagues.

## Debugging Configs

### Problem

You’re getting unexpected results from your NGINX server.

### Solution

Debug your configuration, and remember these tips:

* + - NGINX processes requests looking for the most specific matched rule. This makes stepping through configurations by hand a bit harder, but it’s the most efficient way for NGINX to work. There’s more about how NGINX processes requests in the documentation link in the Also See section.
    - You can turn on debug logging. For debug logging, you’ll need to ensure that your NGINX package is configured with the --with-debug flag. Most of the common packages have it, but if you’ve built your own or are running a mini‐ mal package, you may want to at least double-check. Once you’ve ensured you have debug enabled, you can set the error\_log directive’s log level to debug: error\_log /var/log/nginx/error.log debug.
    - You can enable debugging for particular connections. The debug\_connection directive is valid inside the events context and takes an IP or CIDR range as a parameter. The directive can be declared more than once to add multiple IP

addresses or CIDR ranges to be debugged. This may be helpful to debug an issue in production without degrading performance by debugging all connections.

* + - You can debug for only particular virtual servers. Because the error\_log direc‐ tive is valid in the main http, mail, stream, server, and location contexts, you can set the debug log level in only the contexts you need it.
    - You can enable core dumps and obtain backtraces from them. Core dumps can be enabled through the operating system or through the NGINX configuration file. You can read more about this from the admin guide in the Also See section.
    - You’re able to log what’s happening in rewrite statements with the rewrite\_log

directive on: rewrite\_log on.

### Discussion

The NGINX platform is vast, and the configuration enables you to do many amazing things. However, with the power to do amazing things, there’s also the power to shoot yourself in the foot. When debugging, make sure you know how to trace your request through your configuration; if you have problems, add the debug log level to help. The debug log is quite verbose but very helpful in finding out what NGINX is doing with your request and where in your configuration you’ve gone wrong.

### Also See

[“How nginx processes a request”](https://oreil.ly/gw9Vy) [Debugging NGINX Admin Guide](https://oreil.ly/IfLMF)

[NGINX rewrite\_log module documentation](https://oreil.ly/VYumB)

**Conclusion**

*NGINX Cookbook* has focused on high-performance load balancing, security, and deploying and maintaining NGINX and NGINX Plus servers. The book has demon‐ strated some of the most powerful features of the NGINX application delivery plat‐ form. NGINX Inc. continues to develop amazing features and stay ahead of the curve.

We’ve included many short recipes that enable you to better understand some of the directives and modules that make NGINX the heart of the modern web. The NGINX server is not just a web server, nor just a reverse proxy, but an entire application delivery platform, fully capable of authentication and coming alive with the environments that it’s employed in.

# Index

#### Symbols

\* (asterisk) wildcard symbol, [40](#_bookmark158)

#### A

A/B testing, [25](#_bookmark100)

access control, [29](#_bookmark111)

(see also security controls) country-based restrictions, [29](#_bookmark111)

IP-based, [69](#_bookmark254)

RBAC, [126](#_bookmark441)

access logs, [147](#_bookmark523)-[149](#_bookmark529), [159](#_bookmark567)

Access-Control-Allow-Origin header, [71](#_bookmark258)

access\_log directive, [149](#_bookmark528), [150](#_bookmark535), [159](#_bookmark568) ACM (AWS Certificate Manager), [130](#_bookmark459) active health checks, [10](#_bookmark40), [21](#_bookmark86)-[23](#_bookmark90)

active-passive failover, [131](#_bookmark465)

activity monitoring, [139](#_bookmark500)-[144](#_bookmark517)

metrics collecting, [143](#_bookmark514)-[144](#_bookmark517)

monitoring dashboard, [140](#_bookmark506)-[141](#_bookmark508)

stub status, [139](#_bookmark502)

ADC (application delivery controller) (see NGINX Controller)

add\_header directive, [39](#_bookmark154), [71](#_bookmark259), [151](#_bookmark541) Adobe Adaptive Streaming, [97](#_bookmark345) Advanced F5 Application Security, [87](#_bookmark314) aggregating logs, [151](#_bookmark537)

alias directive, [7](#_bookmark28)

allow directive, [70](#_bookmark255)

Alpine Linux, [120](#_bookmark426)

Amazon EC2 UserData, [99](#_bookmark357), [100](#_bookmark360) Amazon EC2, load balancing on, [132](#_bookmark478) Amazon Machine Image (see AMI)

Amazon Route 53 DNS service, [101](#_bookmark364)-[102](#_bookmark366), [132](#_bookmark475) Amazon Web Services (see AWS)

AMI (Amazon Machine Image), [99](#_bookmark357), [104](#_bookmark375)

Ansible, [55](#_bookmark204)-[57](#_bookmark207), [171](#_bookmark609)

Ansible Galaxy, [57](#_bookmark206)

Ansible Tower, [57](#_bookmark206)

API gateway, using NGINX as, [114](#_bookmark404)-[118](#_bookmark416) App Protect Policy file, [85](#_bookmark312) app\_protect\_\* directives, [84](#_bookmark310)

app\_protect\_enable directive, [85](#_bookmark311)

app\_protect\_policy\_file directive, [85](#_bookmark311)

app\_protect\_security\_log directive, [85](#_bookmark311)

authentication, [59](#_bookmark215)-[67](#_bookmark248)

configuration synchronization, [134](#_bookmark488)

HTTP basic, [59](#_bookmark217)-[61](#_bookmark223)

JWKS, automatically obtaining and caching, [65](#_bookmark243)

JWKs, creating, [63](#_bookmark236)

JWTs, validating, [62](#_bookmark230), [64](#_bookmark239)

NGINX Secure Service Mesh (mTLS), [128](#_bookmark455)-[130](#_bookmark460)

OIDC identity provider, [66](#_bookmark246) preshared API keys, [116](#_bookmark410) satisfy directive, [82](#_bookmark302)

subrequests, [61](#_bookmark225)

auth\_basic directive, [60](#_bookmark220)

auth\_basic\_user\_file directive, [60](#_bookmark220)

auth\_jwt directive, [62](#_bookmark232), [64](#_bookmark240)

auth\_jwt\_key\_request directive, [65](#_bookmark244)

auth\_request directive, [61](#_bookmark226)

auth\_request\_set directive, [61](#_bookmark226)

Auto Scaling group, [102](#_bookmark368)-[104](#_bookmark371), [119](#_bookmark421), [133](#_bookmark479) automatic blocklist, building, [83](#_bookmark305)-[84](#_bookmark306) automatic failover, NGINX node routing, [101](#_bookmark364) automating tests with load drivers, [155](#_bookmark551) automation (see programmability)

availability zones, [133](#_bookmark480)

AWS (Amazon Web Service), auto- provisioning on, [99](#_bookmark356)-[101](#_bookmark361)

AWS ALB (AWS Application Load Balancer), [133](#_bookmark481)

AWS Certificate Manager (see ACM)

AWS EBS (AWS Elastic Block Storage), [168](#_bookmark600) AWS ELB (AWS Elastic Load Balancing), [31](#_bookmark117),

[81](#_bookmark296), [101](#_bookmark363)-[102](#_bookmark366), [127](#_bookmark444)

AWS Marketplace, deploying from, [104](#_bookmark374)-[104](#_bookmark376) AWS NLB (AWS Network Load Balancer),

[102](#_bookmark369)-[104](#_bookmark371), [133](#_bookmark480)

Azure (see Microsoft Azure)

#### B

backlog parameter, listen directive, [160](#_bookmark571) bandwidth, limiting, [34](#_bookmark129), [98](#_bookmark348)

bitrate-limiting support, MP4 media files, [98](#_bookmark349) buffer parameter, access\_log directive, [159](#_bookmark568) buffering access logs, [159](#_bookmark566)

buffering responses, [158](#_bookmark563) bypassing the cache, [38](#_bookmark149)

#### C

C modules, [53](#_bookmark196)

C programming language, [52](#_bookmark191) Cache Lock module, [41](#_bookmark164) Cache Slice module, [41](#_bookmark163)

Cache-Control response header, [39](#_bookmark154) caching, [35](#_bookmark135)-[41](#_bookmark165)

automatic caching of JWKS, [65](#_bookmark243) bypassing the cache, [38](#_bookmark149)

hash keys, [37](#_bookmark145)

invalidating objects, [39](#_bookmark156) locking the cache, [36](#_bookmark142) memory-cache zones, [35](#_bookmark138)

performance tuning, [40](#_bookmark160)

segmenting files to increase efficiency, [40](#_bookmark160) SSL session cache, [74](#_bookmark268)

canary release development, [26](#_bookmark102)

CDNs (content delivery networks), [35](#_bookmark136) CentOS, [2](#_bookmark9), [27](#_bookmark107), [49](#_bookmark185), [124](#_bookmark436), [133](#_bookmark485)

certificates

for client-side encryption, [72](#_bookmark262)-[74](#_bookmark270) monitoring with NGINX Instance Manager,

[165](#_bookmark591)

NGINX Controller installation, [168](#_bookmark603) NGINX Secure Service Mesh mTLS, [129](#_bookmark457),

[130](#_bookmark461)

for upstream encryption, [75](#_bookmark274) Chef, [54](#_bookmark199)-[55](#_bookmark202)

CIDR (classless inter-domain routing) range, [30](#_bookmark114)

client connections, keeping open, [156](#_bookmark554) client-side caching, [39](#_bookmark153)

client-side encryption, [72](#_bookmark262)-[74](#_bookmark270)

client/server binding, [16](#_bookmark68)-[18](#_bookmark72)

cloud deployments, [99](#_bookmark354)-[111](#_bookmark397)

auto-provisioning on AWS, [99](#_bookmark356)-[101](#_bookmark361) AWS Marketplace deployment, [104](#_bookmark373)-[104](#_bookmark376) with Google App Engine proxy, [110](#_bookmark392) Google Compute Engine, [109](#_bookmark388)

with Google Compute Image, [109](#_bookmark390)

load balancing over NGINX scale sets on Azure, [107](#_bookmark382)

Microsoft Azure Marketplace, [108](#_bookmark385) NLB sandwich, [102](#_bookmark369)-[104](#_bookmark371), [133](#_bookmark482)

routing to NGINX nodes without AWS ELB, [101](#_bookmark363)-[102](#_bookmark366)

with VM image on Azure, [105](#_bookmark379)-[107](#_bookmark380) cluster-aware key-value store, [47](#_bookmark176) cluster-aware rate limit, building, [83](#_bookmark305)-[84](#_bookmark306) code owners, [118](#_bookmark414)

commands, [5](#_bookmark21)

common vulnerabilities and exposures (see CVEs)

CONFIGPATHS parameter, configuration syn‐ chronization, [135](#_bookmark490)

configuration, [57](#_bookmark209)

(see also authentication; programmability) client-side encryption, [73](#_bookmark266)-[74](#_bookmark270) configuration management code, [100](#_bookmark358)

with Consul templating, [57](#_bookmark209)-[58](#_bookmark211) debugging configs, [176](#_bookmark622)

HTTP/2, [89](#_bookmark320)-[93](#_bookmark331)

includes for clean configs, [175](#_bookmark619) NGINX image from Docker Hub, [119](#_bookmark424)

NGINX Plus App Protect Module, [84](#_bookmark309)-[87](#_bookmark315) synchronization for HA deployment modes,

[133](#_bookmark484)-[135](#_bookmark491)

connection draining, [19](#_bookmark79), [44](#_bookmark170)-[47](#_bookmark172) connections

debugging, [176](#_bookmark622)

keeping client connections open, [156](#_bookmark554) keeping upstream connections open, [157](#_bookmark561) least connections load balancing, [15](#_bookmark61) limiting in traffic management, [31](#_bookmark119)

Consul templating, [57](#_bookmark209)-[58](#_bookmark211)

consul-template daemon, [57](#_bookmark210), [58](#_bookmark212) Consul’s interface for, [58](#_bookmark212) containers, [113](#_bookmark400)-[130](#_bookmark460)

API gateway, using NGINX as, [114](#_bookmark404)-[118](#_bookmark416) DNS SRV records, [118](#_bookmark419)

Docker image, building, [122](#_bookmark430)-[123](#_bookmark432) Dockerfile, creating, [120](#_bookmark428)

environment variables in NGINX, [124](#_bookmark434) Kubernetes ingress controller, [125](#_bookmark438)-[127](#_bookmark445) NGINX Secure Service Mesh with mTLS

enabled, [128](#_bookmark455)-[130](#_bookmark460) official NGINX image, [119](#_bookmark423)

Prometheus Exporter module, [127](#_bookmark448) content delivery networks (see CDNs) core dumps, enabling, [177](#_bookmark626)

CORS (cross-origin resource sharing), [70](#_bookmark257)-[71](#_bookmark260) curl command

collecting metrics, [143](#_bookmark515)-[144](#_bookmark516)

connection draining, [44](#_bookmark171)-[47](#_bookmark172) HTTP authentication test, [60](#_bookmark221) key-value store, [48](#_bookmark178)

NGINX Instance Manager agent, [164](#_bookmark586) request testing, [4](#_bookmark16)

CVEs (common vulnerabilities and exposures), [162](#_bookmark581)

#### D

daemon, starting NGINX as, [4](#_bookmark14) Debian, [1](#_bookmark6), [49](#_bookmark184), [134](#_bookmark487)

debug logging, [176](#_bookmark623)

debugging, [147](#_bookmark521)-[154](#_bookmark546)

access log configuration, [147](#_bookmark523)-[149](#_bookmark529) bypassing cache for, [38](#_bookmark151) configuration, [176](#_bookmark622)

error log configuration, [149](#_bookmark531) OpenTracing for NGINX, [152](#_bookmark543)-[154](#_bookmark546) request tracing, [151](#_bookmark539)-[154](#_bookmark546)

Syslog listener, forwarding to, [150](#_bookmark534) Syslog value for, [151](#_bookmark537)

debug\_connection directive, [176](#_bookmark624)

deny directive, [70](#_bookmark255)

Deployment versus DaemonSet methods, ingress controller, [126](#_bookmark442)

DevOps, [118](#_bookmark415) DNS

Amazon Route 53 service, [101](#_bookmark363)-[102](#_bookmark366), [132](#_bookmark475) distributing to NGINX nodes, [132](#_bookmark473) Google App Engine proxy, [110](#_bookmark394)

DNS SRV records, [14](#_bookmark55), [118](#_bookmark419)

Docker, [113](#_bookmark401)

image building, [122](#_bookmark430)-[123](#_bookmark432) Kubernetes ingress controller, [127](#_bookmark446) OpenTracing with, [153](#_bookmark544)

docker build command, [123](#_bookmark431) docker command, [120](#_bookmark425)

Docker Hub, NGINX image from, [119](#_bookmark424) Dockerfile, creating images, [120](#_bookmark428)-[123](#_bookmark432) Dovecot, [60](#_bookmark222)

drain parameter, [19](#_bookmark79)

dynamic DDoS mitigation, [83](#_bookmark304)-[84](#_bookmark306)

#### E

ECC (Elliptic Curve Cryptography) formatted keys, [74](#_bookmark269)

elastic load balancing (see AWS ELB) encryption, [59](#_bookmark219), [72](#_bookmark262)-[75](#_bookmark275), [90](#_bookmark321) enforcementMode, App Protect Policy, [85](#_bookmark313) environment variables in NGINX, [124](#_bookmark434) error logs, configuring, [149](#_bookmark531)

error\_log directive, [149](#_bookmark532), [150](#_bookmark535), [177](#_bookmark625) escape parameter, logging, [148](#_bookmark527)

EXCLUDE parameter, configuration synchro‐ nization, [135](#_bookmark490)

expect directive, match block, [22](#_bookmark89)

expire date, securing a location with, [77](#_bookmark285) expires directive, [39](#_bookmark154)

expiring link, generating, [78](#_bookmark288)-[80](#_bookmark291)

#### F

F4F module, [97](#_bookmark345)

f4f\_buffer\_size directive, [98](#_bookmark346) failover

active-passive failover, [131](#_bookmark465)

Amazon Route 53, [101](#_bookmark364)

automatic failover, NGINX node routing, [101](#_bookmark363)

with DNS load balancing, [132](#_bookmark476) files and directories, [5](#_bookmark20)-[6](#_bookmark22)

firewall, [69](#_bookmark251), [87](#_bookmark314), [171](#_bookmark611)-[173](#_bookmark615)

flush parameter, access\_log directive, [159](#_bookmark568) FLV (Flash Video) format, [95](#_bookmark338), [97](#_bookmark345)

Forwarded header, [30](#_bookmark115)

FQDN (fully qualified domain name), [11](#_bookmark45), [168](#_bookmark602)

#### G

generic hash load balancing, [16](#_bookmark63) geography, routing NGINX nodes by, [101](#_bookmark365)

GeoIP module, [27](#_bookmark105)-[28](#_bookmark109)

geoip\_city directive, [28](#_bookmark108)

geoip\_country directive, [28](#_bookmark108)

geoip\_proxy directive, [30](#_bookmark114)

geoip\_proxy\_recursive directive, [30](#_bookmark116) GoLang, Prometheus Exporter Module, [128](#_bookmark449) Google App Engine, [109](#_bookmark388), [110](#_bookmark392), [111](#_bookmark396)

Google Compute Cloud, [110](#_bookmark393) Google Compute Engine, [109](#_bookmark388), [110](#_bookmark393) Google Compute Image, [109](#_bookmark390) Google’s load balancer, [31](#_bookmark117)

gRPC method calls, [90](#_bookmark323)-[92](#_bookmark329) grpc\_pass directive, [90](#_bookmark324)

#### H

HA (High-Availability) deployment modes, [131](#_bookmark466)-[137](#_bookmark497)

configuration synchronization, [133](#_bookmark484)-[135](#_bookmark491) DNS, load-balancing load balancers with,

[132](#_bookmark473)

EC2, load balancing on, [132](#_bookmark478) NGINX plus HA mode, [131](#_bookmark469)

state sharing with NGINX Plus and zone sync, [136](#_bookmark494)

hash digest, [76](#_bookmark282)

hash directive, [16](#_bookmark63), [23](#_bookmark94)

hash keys, [37](#_bookmark145), [38](#_bookmark147)

hashlib library, Python, [77](#_bookmark283)

HDS (HTTP Dynamic Streaming), [95](#_bookmark336), [97](#_bookmark344)

health checks, [10](#_bookmark40)

active, [10](#_bookmark40), [21](#_bookmark86)-[23](#_bookmark90)

Amazon Route 53, [101](#_bookmark364)-[102](#_bookmark366) DNS load balancing, [132](#_bookmark475) EC2 load balancing, [133](#_bookmark480) HTTP, [141](#_bookmark509)

passive, [10](#_bookmark41), [20](#_bookmark82)

stream, [22](#_bookmark88)

TCP/UDP, [21](#_bookmark84)-[23](#_bookmark90), [141](#_bookmark510)

health\_check directive, [21](#_bookmark87) High Availability (see HA)

HLS (HTTP Live Stream) module, [96](#_bookmark340)-[97](#_bookmark341) hls\_buffers directive, [97](#_bookmark342)

horizontal scaling, [9](#_bookmark37), [119](#_bookmark421)

HSTS (HTTP Strict Transport Security), [81](#_bookmark298) HTML5 video, [41](#_bookmark163)

htpasswd command, [60](#_bookmark222) HTTP

authentication, [59](#_bookmark217)-[62](#_bookmark228)

health checks, [20](#_bookmark83)-[23](#_bookmark90), [141](#_bookmark509)

load balancing, [10](#_bookmark43)

proxy module SSL rules, [75](#_bookmark273) http block, [11](#_bookmark49)

HTTP Dynamic Streaming (see HDS) HTTP Live Stream (HLS) module (see HLS) http SSL module, [73](#_bookmark267)

HTTP Strict Transport Security (see HSTS)

HTTP/2, [89](#_bookmark318)-[93](#_bookmark331)

http2 parameter, listen directive, [90](#_bookmark321) http2\_push directive, [93](#_bookmark332)

HTTPS

redirects, [80](#_bookmark293)

upstream encryption, [75](#_bookmark276)

http\_auth\_request\_module, [61](#_bookmark225)

#### I

IaaS (infrastructure as a service), [99](#_bookmark354) IdP (identity provider), OIDC, [66](#_bookmark246)

inactive parameter, proxy\_cache\_path, [36](#_bookmark140) include directive, [115](#_bookmark405), [175](#_bookmark620)

index directive, [7](#_bookmark29)

infrastructure as a service (see IAAS) installation, [1](#_bookmark6)-[4](#_bookmark17)

with Ansible, [55](#_bookmark204)-[57](#_bookmark207)

with Chef, [54](#_bookmark199)-[55](#_bookmark202)

NGINX Controller, [167](#_bookmark597)-[169](#_bookmark604)

NGINX Instance Manager, [161](#_bookmark579), [163](#_bookmark585)

NGINX Plus, [3](#_bookmark11)

NGINX Plus App Protect Module, [84](#_bookmark309)-[87](#_bookmark315) NJS module, [49](#_bookmark183)-[51](#_bookmark188)

on Debian/Ubuntu, [1](#_bookmark6)

on RedHat/CentOS, [2](#_bookmark9)

verifying, [3](#_bookmark13)

invalidating objects, caching, [39](#_bookmark153) ip addr command, [134](#_bookmark489)

IP address

access based on, [69](#_bookmark253)

adding multiple addresses to DNS A record, [132](#_bookmark476)

connection debugging, [176](#_bookmark624) EC2 and virtual addresses, [133](#_bookmark482) finding original client, [30](#_bookmark113)

and keepalived, [131](#_bookmark471) limiting connections by, [31](#_bookmark119)

limiting request rate by, [32](#_bookmark124)-[34](#_bookmark127) load distribution via DNS, [132](#_bookmark476)

IP hash load balancing, [16](#_bookmark65) ip\_hash directive, [16](#_bookmark65), [23](#_bookmark94)

#### J

JavaScript, [49](#_bookmark183)-[51](#_bookmark188), [71](#_bookmark259)

Jinja2 templating language, [57](#_bookmark206) jq tool, NGINX Controller, [168](#_bookmark599) JSON Web Signature, [62](#_bookmark234), [65](#_bookmark241)

JWK (JSON Web Keys), [62](#_bookmark233), [63](#_bookmark236) JWKS (JSON Web Key Set), [65](#_bookmark243)

JWTs (JSON Web Tokens), [50](#_bookmark186), [51](#_bookmark187), [62](#_bookmark231), [64](#_bookmark239)

#### K

keepalive directive, [157](#_bookmark558)

keepalived, [131](#_bookmark470)

keepalive\_requests directive, [156](#_bookmark555)

keepalive\_timeout directive, [156](#_bookmark555)

key-value store, [47](#_bookmark176)-[49](#_bookmark179), [67](#_bookmark247), [84](#_bookmark307)

keyval directive, [48](#_bookmark177)

keyval\_zone directive, [84](#_bookmark307)

keyval\_zone directory, [48](#_bookmark177) kty attribute, JWK file, [64](#_bookmark237) Kubernetes, [125](#_bookmark438)-[127](#_bookmark445), [128](#_bookmark452)

#### L

LDAP (Lightweight Directory Access Protocol), [60](#_bookmark222)

least connections load balancing, [15](#_bookmark61) least time load balancing, [15](#_bookmark62) least\_conn directive, [15](#_bookmark61)

least\_time directive, [114](#_bookmark402)

levels parameter, proxy\_cache\_path, [36](#_bookmark140) Lightweight Directory Access Protocol (see

LDAP)

limiting bandwidth, [34](#_bookmark129), [98](#_bookmark348)

limiting connections, [31](#_bookmark119)

limiting rate of requests, [32](#_bookmark124)-[34](#_bookmark127), [117](#_bookmark411)

limit\_conn directive, [31](#_bookmark120)

limit\_rate directive, [34](#_bookmark130), [34](#_bookmark131)

limit\_rate\_after directive, [34](#_bookmark130), [34](#_bookmark131)

limit\_req directive, [32](#_bookmark126)

limit\_req\_dry\_run directive, [32](#_bookmark122), [32](#_bookmark122)

limit\_req\_zone directive, [84](#_bookmark307) Link response header, [93](#_bookmark333)

Linux distributions (see specific distributions) listen directive

access log configuration, [148](#_bookmark524) HTTP/2, [90](#_bookmark321)

OS tuning, [160](#_bookmark571)

serving static content, [7](#_bookmark26) UDP load balancing, [13](#_bookmark53), [14](#_bookmark56)

load balancing, [9](#_bookmark37)-[23](#_bookmark95)

on Amazon EC2, [132](#_bookmark478) with AWS ALB, [133](#_bookmark481)

with AWS ELB, [30](#_bookmark116), [81](#_bookmark296), [101](#_bookmark363)-[102](#_bookmark366), [127](#_bookmark444)

with AWS NLB, [102](#_bookmark368)-[104](#_bookmark371), [133](#_bookmark480)

with Azure, [107](#_bookmark382)

client/server binding, [16](#_bookmark68)-[18](#_bookmark72) and connection draining, [19](#_bookmark78)

in containerized environment, [114](#_bookmark402) gRPC calls, [92](#_bookmark327)

health checks, [20](#_bookmark82)-[23](#_bookmark90)

HTTP servers, [10](#_bookmark43)

methods, [14](#_bookmark59)-[16](#_bookmark66)

passive health checks, [20](#_bookmark82)

routing persistent session to upstream server, [18](#_bookmark75)

slow start, [23](#_bookmark92)

TCP servers, [11](#_bookmark47)-[13](#_bookmark50)

UDP servers, [13](#_bookmark52)

load drivers, automating tests with, [155](#_bookmark551) load-balancing pool, SRV record, [119](#_bookmark421) load-testing tools, [155](#_bookmark552)

LoadBalancer service type, Kubernetes, [126](#_bookmark443) load\_module directive, [84](#_bookmark310), [153](#_bookmark545)

location blocks, [7](#_bookmark30), [75](#_bookmark278), [116](#_bookmark408)

location directive, [7](#_bookmark29)

location directive, gRPC configuration, [91](#_bookmark326) locking the cache, [36](#_bookmark142)

log module, [147](#_bookmark523)-[149](#_bookmark529) logging

access logs, [147](#_bookmark523)-[149](#_bookmark529), [159](#_bookmark567)

aggregating logs, [151](#_bookmark537)

app\_protect\_security\_log directive, [85](#_bookmark311)

debug, [176](#_bookmark623)

error logs, [149](#_bookmark531), [177](#_bookmark625)

escape parameter, [148](#_bookmark527)

Syslog, [150](#_bookmark534)

log\_format directive, [148](#_bookmark527)

Lua module, [52](#_bookmark192)

#### M

match block for stream servers, [22](#_bookmark89)

max-size parameter, proxy\_cache\_path, [36](#_bookmark140) md5 hash, binary format of, [78](#_bookmark286), [78](#_bookmark289)

media streaming (see streaming media) memory-cache zones, [35](#_bookmark138)

microservice architectures, [117](#_bookmark413)

Microsoft Azure, [31](#_bookmark117), [105](#_bookmark379)-[108](#_bookmark386) Microsoft Azure Marketplace, [108](#_bookmark385)

monitoring, [139](#_bookmark500)-[144](#_bookmark517), [165](#_bookmark591) (see also logging) certificates, [165](#_bookmark591)

metrics collecting, [143](#_bookmark514)-[144](#_bookmark517)

NGINX dashboard, [140](#_bookmark506)-[141](#_bookmark508)

stub status, [139](#_bookmark502)

MP4 format, [95](#_bookmark338)-[97](#_bookmark341), [98](#_bookmark349)

mp4\_limit\_rate directive, [98](#_bookmark350)

mp4\_limit\_rate\_after directive, [98](#_bookmark350)

mTLS, [128](#_bookmark455)-[130](#_bookmark460)

multiple active failover, [131](#_bookmark467)

#### N

NAT (Network Address Translation), [31](#_bookmark121) net.core.somaxconn, [160](#_bookmark571)

network time protocol (NTP) servers (see NTP)

nghttp utility, [90](#_bookmark321)

NGINX, [1](#_bookmark4)-[8](#_bookmark34)

commands, [5](#_bookmark21)

extending with common programming lan‐ guage, [52](#_bookmark190)-[53](#_bookmark197)

files, directories, and commands, [4](#_bookmark19) installation, [1](#_bookmark6)-[4](#_bookmark17)

official NGINX image, using, [119](#_bookmark423) reloading configuration, [6](#_bookmark23), [7](#_bookmark32) serving static content, [6](#_bookmark25)

stub status, enabling, [139](#_bookmark502) NGINX Controller, [141](#_bookmark507), [167](#_bookmark595)-[173](#_bookmark615)

connecting NGINX Plus with, [169](#_bookmark606) driving controller with API, [170](#_bookmark608) setup, [167](#_bookmark597)-[169](#_bookmark604)

WAF, enabling, [171](#_bookmark611)-[173](#_bookmark615) NGINX GPG package, [2](#_bookmark7)

NGINX Instance Manager, [145](#_bookmark518), [161](#_bookmark576)-[166](#_bookmark592)

agent installation, [163](#_bookmark585)

automating discovery, configuration, and monitoring, [165](#_bookmark589)

setup, [161](#_bookmark578)-[163](#_bookmark583)

NGINX Instance Manager API, [145](#_bookmark518), [165](#_bookmark590)

NGINX Plus, [3](#_bookmark11)

Amazon Marketplace, [104](#_bookmark373)-[104](#_bookmark376) as API gateway, [114](#_bookmark404)-[118](#_bookmark416) authentication features, [62](#_bookmark230)-[67](#_bookmark248)

building Kubernetes ingress controller, [125](#_bookmark439) caching purging with, [39](#_bookmark156)

client/server binding, [16](#_bookmark68)-[18](#_bookmark72)

configuration synchronization, [133](#_bookmark484)-[135](#_bookmark491) configuring in dynamic environment, [43](#_bookmark169)-[47](#_bookmark173)

connecting with NGINX Controller, [169](#_bookmark606) connection draining, [19](#_bookmark78)

containers, [122](#_bookmark430)-[123](#_bookmark432) DNS SRV records, [118](#_bookmark419)

dynamic DDoS mitigation, [83](#_bookmark304)-[84](#_bookmark306) HDS, [97](#_bookmark344)

health checks, [10](#_bookmark40), [20](#_bookmark82), [21](#_bookmark86)-[23](#_bookmark90)

installation, [3](#_bookmark11)

invalidating objects, [39](#_bookmark156)

key-value store setup, [47](#_bookmark175)-[49](#_bookmark179) least time load balancing, [15](#_bookmark62) monitoring dashboard, [140](#_bookmark506)-[141](#_bookmark508)

nginx-plus-module-geoip, [27](#_bookmark106) Prometheus Exporter module, [128](#_bookmark451) routing persistent session to upstream

server, [18](#_bookmark75) NGINX Plus API, [43](#_bookmark169)

(see also programmability) connection draining with, [19](#_bookmark79), [44](#_bookmark170)-[47](#_bookmark172)

enabling adding and removing servers, [43](#_bookmark169)-[47](#_bookmark173)

metrics collecting with, [143](#_bookmark514)-[144](#_bookmark517) NGINX Instance Manager, [164](#_bookmark587) server statistics from, [128](#_bookmark453)

NGINX Plus App Protect Module, [84](#_bookmark309)-[87](#_bookmark315), [171](#_bookmark612) NGINX plus HA mode, [131](#_bookmark470)

NGINX Secure Service Mesh, enabling mTLS with, [128](#_bookmark455)-[130](#_bookmark460)

nginx-ha-keepalived package, [131](#_bookmark470)

nginx-meshcli tool, [129](#_bookmark456)

nginx-module-geoip package, [27](#_bookmark106)

nginx-plus-module-geoip package, [27](#_bookmark106)

nginx-sync package, [133](#_bookmark486)

nginx-sync.sh application, [135](#_bookmark492) nginx\_config resource, Chef, [54](#_bookmark200) nginx\_config role, Ansible, [56](#_bookmark205)

nginx\_ingress namespace and service account, [125](#_bookmark440)

nginx\_site resource, Chef, [54](#_bookmark201) ngx object, [53](#_bookmark193)

ngx\_http\_perl\_module, [124](#_bookmark435)

ngx\_http\_ssl\_module, [72](#_bookmark263)

ngx\_stream\_ssl\_module, [72](#_bookmark263) NJS (NGINX JavaScript), [52](#_bookmark191) NJS module, [49](#_bookmark183)-[51](#_bookmark188)

NLB (network load balancer) sandwich, [102](#_bookmark369)-[104](#_bookmark371), [133](#_bookmark480)

NodePort service type, Kubernetes load bal‐ ancer, [126](#_bookmark443)

NODES parameter, configuration synchroniza‐ tion, [135](#_bookmark490)

note for working in containerized environment, [113](#_bookmark401)

NTP (network time protocol) servers, [13](#_bookmark53)

#### O

OIDC (OpenID Connect) identity provider, [66](#_bookmark246) open file descriptors, raising number of, [160](#_bookmark572) openssl command, [76](#_bookmark282)

openssl passwd command, [60](#_bookmark222) OpenTracing for NGINX, [152](#_bookmark543)-[154](#_bookmark546) opentracing\_tag directive, [154](#_bookmark546)

OpenVPN service, [13](#_bookmark54)

original client IP address, finding, [30](#_bookmark113) OS tuning, [159](#_bookmark570)

#### P

Packer, HashiCorp, [100](#_bookmark359)

passive health checks, [10](#_bookmark41), [20](#_bookmark82)

passwords, authentication, [59](#_bookmark218)-[61](#_bookmark223)

performance tuning, [155](#_bookmark549)-[160](#_bookmark573)

automating tests with load drivers, [155](#_bookmark551) buffering access logs, [159](#_bookmark566)

buffering responses, [158](#_bookmark563)

caching, [40](#_bookmark160)

keeping client connections open, [156](#_bookmark555) keeping upstream connections open, [157](#_bookmark557) OS tuning, [159](#_bookmark570)

Perl module, [53](#_bookmark194)

perl\_set directive, [53](#_bookmark195), [124](#_bookmark435)

PKI (Public Key Infrastructure), [129](#_bookmark457) PostgreSQL database, NGINX Controller, [167](#_bookmark598) programmability

Chef installing and configuring, [54](#_bookmark199)-[55](#_bookmark202) Consul templating, automating configura‐

tions with, [57](#_bookmark209)-[58](#_bookmark211)

dynamic environment configuration for NGINX Plus, [43](#_bookmark169)-[47](#_bookmark173)

extending NGINX with common program‐ ming language, [52](#_bookmark190)-[53](#_bookmark197)

installing with Ansible, [55](#_bookmark204)-[57](#_bookmark207) key-value store setup, [47](#_bookmark175)-[49](#_bookmark179)

NJS module for exposing JavaScript func‐ tionality, [49](#_bookmark182)-[51](#_bookmark188)

Prometheus Exporter Module, [127](#_bookmark448) proxies

and finding original client IP address, [30](#_bookmark116) Google App Engine, [110](#_bookmark392)

HTTP proxy module SSL rules, [75](#_bookmark274) PROXY Protocol, [127](#_bookmark444)

PROXY Protocol header, [148](#_bookmark524) proxy\_buffering directive, [158](#_bookmark564)

proxy\_busy\_buffer\_size directive, [158](#_bookmark564)

proxy\_cache directive, [36](#_bookmark139)

proxy\_cache\_bypass directive, [38](#_bookmark150)

proxy\_cache\_key directive, [37](#_bookmark146), [41](#_bookmark162)

proxy\_cache\_lock directive, [36](#_bookmark143)

proxy\_cache\_path directive, [35](#_bookmark138)

proxy\_cache\_purge directive, [39](#_bookmark157)

proxy\_http\_version directive, [157](#_bookmark559)

proxy\_pass directive, [75](#_bookmark276), [110](#_bookmark395)

proxy\_pass\_request\_body directive, [61](#_bookmark227)

proxy\_protocol parameter, [148](#_bookmark524)

proxy\_response directive, [14](#_bookmark56)

proxy\_set\_header directive, [151](#_bookmark541), [157](#_bookmark560)

proxy\_ssl\_certificate directive, [75](#_bookmark276)

proxy\_ssl\_certificate\_key directive, [75](#_bookmark276)

proxy\_ssl\_crl directive, [75](#_bookmark276)

proxy\_timeout directive, [14](#_bookmark56)

ps command, [4](#_bookmark15)

Public Key Infrastructure (see PKI) purging cache, [39](#_bookmark156)

Python, [57](#_bookmark206), [77](#_bookmark283), [79](#_bookmark290)

#### R

random directive, [16](#_bookmark64), [23](#_bookmark94) random load balancing, [16](#_bookmark64)

rate-limiting module, [32](#_bookmark125)-[34](#_bookmark127), [117](#_bookmark412)

RBAC (Role-Based Access Control), [126](#_bookmark441) RedHat (see RHEL)

reload method, [7](#_bookmark33)

reloading configuration, [7](#_bookmark32)

request tracing, [151](#_bookmark539)-[154](#_bookmark546)

request\_id variable, [151](#_bookmark540)

resolve parameter, server directive, [119](#_bookmark420) resolver directive, [110](#_bookmark395), [119](#_bookmark420)

reuseport parameter, [13](#_bookmark54), [14](#_bookmark57)

rewrite directive, NGINX as API gateway, [116](#_bookmark409) rewrite\_log directive, [177](#_bookmark627)

RHEL (RedHat Enterprise Linux), [2](#_bookmark9), [27](#_bookmark107), [49](#_bookmark185), [133](#_bookmark485) Role-Based Access Control (see RBAC)

root directive, [7](#_bookmark28)

round robin load balancing, [15](#_bookmark60), [132](#_bookmark474)

Route 53 DNS service, [101](#_bookmark363)-[102](#_bookmark366), [132](#_bookmark475) RSA formatted key, [74](#_bookmark269)

#### S

satisfy directive, [82](#_bookmark302)

scale sets, virtual machine, [107](#_bookmark382) secrets, [75](#_bookmark278), [76](#_bookmark280)

secure link directive, [78](#_bookmark286) secure link module, [76](#_bookmark282) secure\_link\_md5 directive, [78](#_bookmark286)

secure\_link\_secret directive, [76](#_bookmark279)

security controls, [69](#_bookmark251)-[87](#_bookmark315) authentication (see authentication) client-side encryption, [72](#_bookmark262)-[74](#_bookmark270)

CORS, allowing, [70](#_bookmark257)-[71](#_bookmark260)

dynamic DDoS mitigation, [83](#_bookmark304)-[84](#_bookmark306) expire date to secure a location, [77](#_bookmark285) expiring link, generating, [78](#_bookmark288)-[80](#_bookmark291) HSTS, [81](#_bookmark298)

HTTPS redirects, [80](#_bookmark293)

IP address limits, [31](#_bookmark119)-[34](#_bookmark127), [69](#_bookmark253) NGINX as API gateway, [116](#_bookmark410)

NGINX Plus App Protect Module, [84](#_bookmark309)-[87](#_bookmark315) rate-limiting module, [32](#_bookmark126)

secrets, [75](#_bookmark278), [76](#_bookmark280)

security methods, satisfying multiple, [82](#_bookmark301) upstream encryption, [75](#_bookmark272)

segmenting files to increase efficiency, [40](#_bookmark160) send directive, match block, [22](#_bookmark89)

server block

serving static files over HTTP, [7](#_bookmark26)

setting upstream service outside of, [115](#_bookmark406) server directive, [11](#_bookmark45), [119](#_bookmark420)

server drain directive, [20](#_bookmark80) server slow\_start directive, [23](#_bookmark93)

servers, adding and removing with NGINX Plus, [19](#_bookmark78), [43](#_bookmark169)-[47](#_bookmark173)

server\_name directive, [7](#_bookmark27)

session state, [9](#_bookmark38), [16](#_bookmark65), [18](#_bookmark73), [20](#_bookmark80)

shared-memory zones, [117](#_bookmark411), [136](#_bookmark494)

sidecar pattern, NGINX Service Mesh, [129](#_bookmark458) slice directive, [40](#_bookmark161)

slicing, cache, [40](#_bookmark160)

slow start, load balancing, [23](#_bookmark92)

SMTP server, NGINX Controller installation, [168](#_bookmark601)

SPIRE, [130](#_bookmark462)

split\_clients module, [25](#_bookmark101)

SSL modules, [72](#_bookmark263), [73](#_bookmark267) SSL/TLS

client-side encryption, [72](#_bookmark262)-[74](#_bookmark270)

HTTP/2 consideration, [90](#_bookmark321), [91](#_bookmark325)

redirecting to HTTPS, [81](#_bookmark296) upstream encryption, [75](#_bookmark273)

ssl\_certificate directive, [72](#_bookmark264)

ssl\_certificate\_key directive, [72](#_bookmark264)

state sharing with NGINX Plus and zone sync, [136](#_bookmark494)

stateful versus stateless applications, [9](#_bookmark39) static content, serving, [6](#_bookmark25)

sticky cookie directive, [16](#_bookmark69) sticky learn directive, [17](#_bookmark71) sticky route directive, [18](#_bookmark76) stream health checks, [22](#_bookmark88) stream module, [11](#_bookmark48)-[13](#_bookmark50), [14](#_bookmark56) stream SSL module, [73](#_bookmark267) streaming media, [95](#_bookmark336)-[98](#_bookmark351)

HDS, [97](#_bookmark344)

MP4 and FLV, serving, [95](#_bookmark338)-[97](#_bookmark341) Strict-Transport-Security header, [81](#_bookmark299)

stub status information, Prometheus Exporter Module, [128](#_bookmark450)

stub status module, [139](#_bookmark503) stub\_status directive, [140](#_bookmark504)

subrequests, authentication, [61](#_bookmark225)

Swagger UI, [165](#_bookmark590)

sync parameter, DDoS attack mitigation, [84](#_bookmark307) sys.fs.file\_max kernel option, [160](#_bookmark572)

Syslog listener, forwarding to, [150](#_bookmark534) syslog parameter, [150](#_bookmark536)

#### T

TCP load balancing, [11](#_bookmark47)-[13](#_bookmark50) TCP/UDP health checks, [21](#_bookmark84)-[23](#_bookmark90), [141](#_bookmark510) TLS (see SSL/TLS)

traffic management, [25](#_bookmark98)-[34](#_bookmark132)

A/B testing, [25](#_bookmark100)

access restrictions based on country, [29](#_bookmark111) GeoIP module and database, [27](#_bookmark104)-[28](#_bookmark109) limiting bandwidth per client, [34](#_bookmark129), [98](#_bookmark348)

limiting connections, [31](#_bookmark119)

limiting rate of requests by predefined key, [32](#_bookmark124)-[34](#_bookmark127)

troubleshooting (see debugging) type parameter, key-value store, [49](#_bookmark180)

#### U

Ubuntu, [1](#_bookmark6), [49](#_bookmark184), [134](#_bookmark487)

UDP load balancing, [13](#_bookmark52), [21](#_bookmark84)-[23](#_bookmark90), [141](#_bookmark510)

upstream block, [92](#_bookmark328), [115](#_bookmark407)

upstream connections, keeping open, [157](#_bookmark561)

upstream encryption, [75](#_bookmark272)

upstream module, [11](#_bookmark44), [148](#_bookmark525)

URI (uniform resource identifier), [7](#_bookmark30), [118](#_bookmark417)

#### V

valid override parameter, resolver directive, [119](#_bookmark420) Virtual Router Redundancy Protocol (see

VRRP)

VM (virtual machine)

cloud configurations, [100](#_bookmark359)

creating NGINX image on Azure, [105](#_bookmark379)-[108](#_bookmark386) Google Compute features, [109](#_bookmark388)-[111](#_bookmark397)

VMSS (virtual machine scale set), [107](#_bookmark383)

VRRP (Virtual Router Redundancy Protocol), [131](#_bookmark471)

**W**

WAF (Web Application Firewall), [69](#_bookmark251), [87](#_bookmark314),

[171](#_bookmark611)-[173](#_bookmark615)

#### X

X-Forwarded-For header, [30](#_bookmark114), [148](#_bookmark526)

X-Forwarded-Proto header, [81](#_bookmark295)

#### Y

YAML, Ansible configuration, [57](#_bookmark206)

#### Z

zone synchronization, [136](#_bookmark495)

zone\_sync\_server directive, [136](#_bookmark496)

**About the Author**

**Derek DeJonghe** has a passion for technology. His background and experience in web development, system administration, and networking give him a well-rounded understanding of modern web architecture. Derek leads a team of site reliability and cloud solution engineers and produces self-healing, autoscaling infrastructure for numerous applications. While designing, building, and maintaining highly available applications for clients, he consults for larger organizations as they embark on their journey to the cloud. Derek and his team are on the forefront of a technology tidal wave and are engineering cloud best practices every day. With a proven track record for resilient cloud architecture, Derek pioneers cloud deployments for security and maintainability that are in the best interest of his clients.

### Colophon

The animal on the cover of *NGINX Cookbook* is the Eurasian lynx (*Lynx lynx*), the largest of the lynx species, found in a broad geographical range from Western Europe to Central Asia.

This wild cat has striking vertical tufts of dark fur atop its ears, with rough long hair on its face. Its fur is yellow-gray to gray-brown, with white coloring on the underbelly. This lynx is spattered with dark spots, with northern-dwelling variants tending to be grayer and less spotted than their southern counterparts.

Unlike other lynx species, the Eurasian lynx preys on larger ungulates—hooved animals—such as wild deer, moose, and even domesticated sheep. Adults require two to five pounds of meat each day and will feed on a single source of food for up to a week.

The Eurasian lynx came close to extinction in the mid-twentieth century but sub‐ sequent conservation efforts have brought the cat’s conservation status to Least Concern. Many of the animals on O’Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on a black and white engraving from Shaw’s *Zoology*. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag’s Ubuntu Mono.